A PROGRAMMER'S STORY

The Life of a Computer Pioneer

PER BRINCH HANSEN

FOR CHARLES HAYDEN

Per Brinch Hansen
5070 Pine Valley Drive, Fayetteville, NY 13066, USA

CONTENTS

Acknowledgments *v*

1 Learning to Read and Write 1938–57 *1*

Nobody ever writes two books – My parents – Hitler occupies Denmark – Talking in kindergarten – A visionary teacher – The class newspaper – "The topic" – An elite high school – Variety of teachers – Chemical experiments – Playing tennis with a champion – Listening to jazz – "Ulysses" and other novels.

2 Choosing a Career 1957–63 *17*

Advice from a professor – Technical University of Denmark – Ørsted's influence – Distant professors – Easter brew – Fired for being late – International exchange student – Masers and lasers – Radio talk — Graduation trip to Yugoslavia – An attractive tourist guide – Master of Science – Professional goals.

3 Learning from the Masters 1963–66 *35*

Regnecentralen – Algol 60 – Peter Naur and Jørn Jensen – Dask and Gier Algol – The mysterious Cobol 61 report – I join the compiler group – Playing roulette at Marienlyst resort – Jump-starting Siemens Cobol at Mogenstrup Inn – Negotiating salary – Compiler testing in Munich – Naur and Dijkstra smile in Stockholm – The Cobol compiler is finished – Milena and I are married in Slovenia.

4 Young Man in a Hurr 1966–70 *59*

Naur's vision of datalogy – Architect of the RC 4000 computer – Programming a real-time system – Working with Henning Isaksson, Peter Kraft, and Charles Simonyi – Edsger Dijkstra's influence – Head of software development – Risking my future at Hotel Marina – The RC 4000 multiprogramming system – I meet Edsger Dijkstra, Niklaus Wirth, and Tony Hoare – The genius of Niels Ivar Bech.

5 Shaping a New Field 1970–72 *81*

Alan Perlis invites me to spend a year at Carnegie-Mellon – Emigration to America – Niklaus Wirth defines Pascal – Driving home in a blizzard – Discussing the future of concurrent programming in Marktoberdorf and Belfast – Alan Perlis tells stories – Mad King Ludwig and Thomas Edison – The first modern book on Operating System Principles.

6 Inventing the Future 1972–76 *103*

Can you tell me, what is Caltech? – Sunshine and palm trees in February – Wine-tasting with the dean – Driving across America – A question of priority – Two baffling problems – That sounds easy – Concurrent Pascal and Solo – Al Hartmann's compiler – The art of compromise – Getting a sore throat in Bombay – Returning to Marktoberdorf – The distraction of grants – Leaving the magic kingdom – A passion for clear thinking.

7 The End of an Era 1976–84 *131*

The legendary Zohrab Kaprielian – Football game at the Rose Bowl – Creating a top department at USC – How Harvard grants tenure – The first book on concurrent programming – Doctor technices – Surviving the executive vice president – Designing the Edison multiprocessor for Mostek – United Technologies kills the project – Let no man complain to me – Brush fire and mud slides in Altadena – Magical simplicity – What we achieved.

8 Danish Interlude 1984–87 *153*

Student democracy and teaching in Denmark – Danish industry uses Concurrent Pascal – Consulting for GN Corporation – Rocking the boat.

9 Back in America 1987–2004 *163*

Distinguished professor at Syracuse – Birthday celebration in the former Danish West Indies – Becoming an American citizen – Parallel scientific computing – A personal supercomputer – Parallel cryptography – History of programming languages – The Computer Pioneer Award – Final words.

Sources *181*

Index *189*

ACKNOWLEDGMENTS

For helpful comments on my story, I thank Len Adleman, Birger Andersen, Sven Eriksen, Jon Fellows, Nissim Francez, Steve Goings, Sol Golomb, Christian Gram, Jonathan Greenfield, Al Hartmann, Nils Havsteen, Charles Hayden, Niels Zeuthen Heidam, Giorgio Ingargiola, Henning Isaksson, Konrad Jahn, Jørn Jensen, Anita Jones, Peter Kraft, Paul Lindgreen, Aage Melbye, Peter Naur, Børge Ring, Charles Simonyi, Leif Svalgaard, Fred Thompson, Villy Toft, Peter Villemoes, Paul Waltenburg, and Bill Wulf.

My special thanks go to Charles Hayden, who helped me publish this book on the internet, and Jim Horning, who read all the chapters. And to Børge Ring for translating excerpts from a Dutch trip report by Edsger Dijkstra.

I am grateful for permission to quote from the following material:

The personal letters of Jonathan Fellows, Steve Goings, Al Hartmann, Charles Hayden, Jim Horning, Giorgio Ingargiola, Anil Menon, Villy Toft, Niklaus Wirth, and Tom Zepko.

Klasseavisen ("The class newspaper") 1945–45, Konrad Jahn's class at Niels Ebbesensvej School, Frederiksberg, Denmark. (Excerpts translated from Danish by Per Brinch Hansen.)

E. W. Dijkstra, EWD316A, Reisverslag van Edsger W. Dijkstra aan Summer School, Marktoberdorf, juli 1971 (in Dutch). © 1971 by Maria C. Dijkstra-Debets. (Excerpts translated from Dutch by Børge Ring and Per Brinch Hansen.)

R. A. Maddux and H. D. Mills, Review of Per Brinch Hansen: The Architecture of Concurrent Programs. *IEEE Computer 12*, (May 1979), 102–103. © 1979 IEEE.

P. Naur, Review of Per Brinch Hansen: Operating System Principles. *BIT 15*, 1975, 455–457. © 1975 by BIT.

P. Naur, Notes for the official defense of Per Brinch Hansen's Dr. technices thesis at the Technical University of Denmark, January 23, 1978. (Excerpts translated from Danish by Per Brinch Hansen.)

P. Naur, Unpublished review of Per Brinch Hansen: Programming a Personal Computer, 1984.

A. J. Perlis, Talk on "Computing in the fifties." *ACM National Conference*. Nashville, TN, 1981. Transcript in J. A. N. Lee (ed.), *Computer Pioneers*, IEEE Computer Society Press, Los Alamito, CA, 1995, 545–556. © 1995 IEEE.

PER BRINCH HANSEN
Syracuse University

1

LEARNING TO READ AND WRITE 1938–57

Nobody ever writes two books – My parents – Hitler occupies Denmark – Talking in kindergarten – A visionary teacher – The class newspaper – "The topic" – An elite high school – Variety of teachers – Chemical experiments – Playing tennis with a champion – Listening to jazz – "Ulysses" and other novels.

Walking home from a Caltech party in the 1970s, I told Don Knuth that I was working on my second book. He turned to me and said "Nobody ever writes two books!" I should know—this is is my tenth book. It tells the story of my professional life and my impressions of the birth of modern programming with anecdotes about software pioneers I have known.

As a student of electrical engineering, I dreamt of making fundamental contributions to a new field. In 1963, I graduated from the Technical University of Denmark without any programming experience—it was not yet being taught. There were, as far as I remember, no textbooks available on programming languages, compilers or operating systems. That was my main reason for choosing to work in computing!

Over the next forty years I worked as a systems programmer in Denmark and a computer scientist in America. I witnessed computer programming change from an amateur activity into something resembling an engineering discipline, and was fortunate to contribute to the early development of operating systems and concurrent programming.

In this autobiography, I trace my school years, engineering studies, and the beginning of my career in Denmark. And I recount my exciting and frustrating years as a researcher at Carnegie-Mellon, Caltech, USC, University of Copenhagen, and Syracuse University.

I wrote the book for fun. I assume you know how to use a computer and are interested in programming. My story is mostly told in nontechnical detail. In a few places, where the story gets a bit technical, I explain the gist of the ideas.

⋆ ⋆ ⋆

You may well wonder why I describe my school days in a book about my professional life. Well, over the years, I have learned that, besides *intellect*, the most valuable asset of a programmer is the ability to *write clearly.* Needless to say, I wasn't born with a talent for writing well. But, thanks to my teachers, I learned to write nontrivial essays in elementary school through high school.

Now, if you don't write really well when you graduate from high school, you probably never will. Since I consider my writing skills to be far more important than my engineering background, I will begin my story by telling you how I learned to read and write.

After half a century, the memories of my childhood and youth are naturally somewhat fragmentary. And, like your life, mine did not follow a coherent script. I must also confess that some of my more "random" impressions are included simply because I find them amusing—as I hope you will too.

My parents, Jørgen Brinch Hansen and Elsebeth (née Ring), lived in Frederiksberg, a suburb of Copenhagen, Denmark. I was born on November 13, 1938, and nicknamed "Busser" after a popular cartoon character. (Americans know him as Blondie's husband Dagwood.)

My mother was the daughter of the Danish composer Oluf Ring. She was a charming, vivacious woman with a beautiful smile. Before marrying my father she worked in one of the best hairdressing salons in Copenhagen. On pictures from my childhood she is always well dressed and, even on the beach, her hairstyle looks perfect. When I was little, she liked to go shopping in the center of Copenhagen with me dressed in my best. I would rather have stayed at home playing with friends, but that was not an option.

When the neighbors complained about me (as they often did) my mother would try to hide it from my father. This was, however, impossible on the occasion when I threw a cobblestone through a basement window across the street. What a lovely sound that was! When my father heard about it (and the repair costs) he spanked me and sent me to bed without dinner. When that happened, my mother would usually sneak a sandwich into my bedroom.

People liked my mother and found her fun to be with. In her forties she visited Italy with her sister and brother-in-law. At an outdoor theater in Rome, some tourists were relaxing in front of an empty stage. My mother immediately walked up on the stage and danced across it to the applause of the audience.

My mother smoked constantly. Even when she was cooking, she held a cigarette in her mouth and tilted her head to avoid getting smoke in her eyes.

She also had a taste for strong coffee. I once offered to demonstrate to my graduate student, Jon Fellows, how coffee was meant to taste. First I made a portion of normal (weak) American coffee. After letting the coffee drip through a second portion of ground coffee, Jon and I enjoyed a cup of real Danish coffee. The next day he told me that his heart beat so fast he had to lie down at home.

My father was a tall, handsome man. On pictures he often looks stern and unsmiling. He was very intelligent, but rather silent. His mother died when he was only nine years old. His father then married a woman who talked incessantly. She left my father with a distaste for small talk. I suspect he often made people feel uncomfortable.

In 1935, he graduated in civil engineering from the Technical University of Denmark. My father apparently found one class a waste of his time. Paperback editions of Danish textbooks had to be cut open, page by page, before you could read them. At his oral exam in Road Construction, my father showed his disdain for the subject by bringing an uncut (unread) version of the professor's textbook to the examination table. The professor rewarded him with the extremely low grade of mdl+. This reduced my father's total grade point average on his diploma from ug– to mg+. (The Danish grades, ug–, mg+, and mdl+, correspond roughly to A–, B+, and D– in America.)

For twenty years, he worked for Christiani & Nielsen, a Danish engineering company that built harbors, docks, bridges, tunnels, airports, roads and railways all over the world. One of his first tasks was to design procedures for lowering nine tunnel elements to the bottom of the river Maas in Rotterdam, one of Europe's most heavily trafficked waterways. At the time it was the longest underwater tunnel in Europe. Each tunnel element, weighing 15,000 tons, was 180 feet long, 75 feet wide and 27 feet tall.

In May 1940, the German army invaded Holland. When the last tunnel piece had been lowered in December 1940, my father returned home on one of the last commercial flights from Rotterdam. The flight must have been somewhat unnerving: the Germans had painted the cabin windows white to prevent the passengers from discovering military secrets from the air.

After the war, my father became recognized as one of the world's leading experts in soil mechanics. In 1953, he was promoted to chief engineer at the C&N headquarters in Copenhagen. Two years later he accepted a professorship in Soil Mechanics and Foundation Engineering at the Technical University of Denmark.

A Norwegian colleague wrote:

> In several respects Brinch Hansen's personality was governed by his consistency and his requirement for a logical and rational approach to all problems. This gave him an appearance which could easily be interpreted by outsiders as reserved superiority. On the contrary, those who knew him learned to appreciate his ability to discuss frankly the facts in any problem and also his loyal and honest character. (Bjerrum 1969)

I vividly remember when my father asked his driving instructor to demonstrate parallel parking. My father thought it would be most convenient to do this on our street on a Sunday afternoon. So my parents ended up practicing this difficult maneuver right in front of our neighbors and their kids. I can only imagine what my mother must have thought.

If my father thought something made sense, he never hesitated to take an unpopular stand. Once he invited me to attend a talk at the Danish Academy of Engineering. After the talk, the members voted on some proposal. While my father voted against it, everybody else voted for it.

After a stormy marriage, my parents separated when I was sixteen years old. This was a great tragedy for my mother who still loved him and never had a relationship with another man. She provided a stable, loving home for my sister and me, until she died on December 7, 1962, at age 50, a month before I finished my engineering studies. My father died on May 27, 1969 at age 60, the year before I emigrated to the United States.

My books on *The Architecture of Concurrent Programs* (1977) and *Studies in Computational Science* (1995) are dedicated to my parents.

⋆ ⋆ ⋆

On April 9, 1940, Hitler invaded Denmark. As a sixteen-month old child, I watched German bombers circling low above the rooftops of Copenhagen dropping leaflets. Since Hitler's real target was Norway, the leaflets looked as if the Germans were trying to save money by writing in a mixture of Danish and Norwegian.

The German occupation did not play a major role in my early childhood. It was, after all, the only life I knew until I was seven years old. But the war years were not so easy for my parents. Most everyday necessities were rationed, including food, clothing, soap, and tobacco. There was a severe shortage of fuel for electricity, heating and transportation. The Germans imposed a curfew which forced people to stay overnight when they visited friends and relatives in the evening.

I remember a couple of violent episodes on our street. One day, a man was shot in front of me. The next day, I watched a street sweeper sweep a large pool of blood into the gutter. It was rumored that the victim was shot simultaneously from opposite ends of the street (which could not possibly have been true, unless the assassins were suicidal). On another occasion, a man was killed when Nazis blew up his villa in the middle of the night across the street from our apartment building. I believe he was a member of the resistance movement. After the war, an apartment complex, built in his garden, was named Bomhoffs Have in his memory.

Although the Germans inflicted more inhumane suffering on other countries, many lives were lost in Denmark. During the occupation, the Germans executed a hundred saboteurs, and sent 150 communists and 500 Jews to concentration camps. However, the Danes helped close to 7,000 Jews escape to Sweden on small fishing boats.

The war ended on May 4, 1945, when the German forces in Denmark capitulated to Fieldmarshal Montgomery. Had the Russians reached the Danish border before the English Army did, my life would have been very different.

⋆ ⋆ ⋆

In 1944, at age 6, I went to Miss Hansen's kindergarten on Lykkesholms Alle in Frederiksberg. She had her own way of keeping 33 little kids reasonably quiet. If you talked too much, you were placed on a tall chair during the lunch break facing the other children with your mouth covered by a towel. It doesn't seem to have worked in my case. I still talk too much instead of listening.

I don't remember if we learned the alphabet in kindergarten. But I have an early memory of crying in frustration, while my mother helped me tell the difference between the letters *h* and *k*.

⋆ ⋆ ⋆

On September 1, 1945, I began my elementary education at Niels Ebbesensvej school in Frederiksberg in a class of 39 boys. For the next five years, we had the same teacher in all subjects. His name was Konrad Jahn. He was the most important teacher in my life. The second week of class he promised that we would publish a weekly newspaper, as soon as we were able to write stories.

Two years later Mr. Jahn wrote:

> About five months later, we were able to publish the first issue of the Class Newspaper, which I immodestly believe is the first school paper written, typed, and duplicated by [Danish] kids in the first grade...By putting the greatest emphasis on Teaching Independence, the children have now in the middle of the third grade reached the point where they produce the paper completely on their own.

Here are a few stories from the class newspaper (dates are shown in the American abbreviated style—month/day/year):

– I destroyed the class newspaper, for I forgot to remove the pencil [from inside the stencil]. So Mr. Jahn had to retype the whole paper. (Per Just Sørensen, 1st grade, 6/12/1946)

– On Mondays we give talks, and on Fridays we also give talks. We also watch a movie about how an engine makes a car drive, and then Mr. Jahn asks for questions, and then one of us asked how one makes a cow glare! (Eskild Sørensen, 2nd grade, 1/20/1947)

– Nearly every day, Per Brinch is late. And I really believe he would like a chauffeur to drive him to school, And he also has an electric alarmclock, and every day when he has to go to school, a wire is loose or a tooth wheel is broken, and then he is full of stories and always has a letter about the alarmclock, the wires, and his parents ... In the morning when the alarm clock rings very softly, he wakes up, and then he is too lazy to get up. (Peter, 2nd grade, 4/12/1947)

– Yesterday we heard a talk in the basement and there were even slides shown. And Per Brinch stood and pointed at tunnel elements all the way across the river so the cars can drive underwater. (Erik Michaelsen, 3rd grade, 1/8/1948)

– On Easter Monday, I sat down at my uncle's desk to write a letter to my father and mother. Then two days later I come back to the desk, and the letter is still there (because there was no stamp on it). The day before I went home I go back in there, and it was still lying there. So I threw it in the wastepaper basket. (Per Brinch, 4th grade, 1949)

In the fourth grade each of us had to write a report on how a newspaper is produced, from the moment a journalist writes a story until the printed newspaper is distributed. After visiting the newspaper *Berlingske Tidende*, we read about *The Topic* (as Jahn called it), and cut pictures out of old newspapers. The class newspaper shows that we spent 14 hours on this project per week. I won the first prize of two kroner (roughly, a quarter) for my final report of 60 handwritten pages with 50 illustrations. Thanks to Mr. Jahn, I had written my first substantial report when I was eleven years old.

I found a copy of our class newspaper dated November 13, 1949 (which happened to be my eleventh birtday). The paper summarizes the opening meeting of our *classroom parliament.* Listen to this conversation among eleven year old boys:

– The teacher: I suggest that you elect a class council with three members who will be responsible for keeping silence and order in the classroom.

– Peter was elected as chair with 15 votes against Preben Møller 3, ... Per Brinch 1 [guess who voted for me], Klaus 1.

– Flemming: Suggests a council with six people.

– Eskild: Flemming's proposal is the best, because six people can get more ideas than three.

– Peter: Suggests that a new council is elected every day, so that we may get rid of it, if we are dissatisfied.

– Gunnar Bjerge: I propose that the council is elected for one week at a time, because otherwise we will waste too many hours.

– Kjeld: I suggest that the most intelligent should be the six members of the council.

– Per Stockholm: I think it's nuts that it's always the most clever ones who are elected for something.

– Erik N: Yes, I would like, on behalf of the class, to welcome everybody!

– Teacher: I am pleased that somebody remembers to welcome us!

We soon learned to appreciate the advantages of a democratic society. A proposal to turn the three strongest boys into a "police force" was promptly and soundly defeated. In fact we only agreed on two rather mild, but very effective penalties: being asked to stand in the corner of the gym, and being excluded from interesting lessons and meetings.

The council wanted a written declaration from Mr. Jahn regarding its authority. So he wrote:

> I hereby hand over my authority *within* the four walls of the classroom to the new class council. This authority applies to the laws of the class and regulations concerning discipline, what is to be taught etc., but excludes everything concerning the disciplinary rules of the school.
>
> November 11, 1949. Signed K. Jahn

The teacher's private thoughts: "During the lunch break the council chair asked me to tell the class to keep quiet, so that we could eat in peace. But it's evident that the children have understood that I have handed over my authority to the council, because they are noisy, like steam pouring out of a container that has kept it bottled up!"

So it went for days. Some of us said pure nonsense, and, for many, many days in the beginning we decided to paint the whole day long. But Mr. Jahn insisted that a painting started one day should be finished before you began another painting. If you know how impatient children are, you will understand why, after some time, we asked to have lessons in writing and arithmetic as well.

Out of this experiment came something schools and colleges rarely teach: We gained faith in our ability to take responsibilty and make decisions. I owe much of my ability to work independently to my early school years with Mr. Jahn.

When I graduated in the fifth grade, he wrote the following evaluation of me:

> In the elementary school, Per Brinch Hansen has shown unusual potential. His diligence has to some extent depended on his interest in the subject matter, but when the interest was there, his achievements were exceptional (The Topic; Helping with the school play). His written works have often been characterized by a restless, artistic untidiness, but can be well written. Behavior: Always extremely good. Independent and helpful.

Since Mr. Jahn was decades ahead of his time, his revolutionary ideas were often met with skepticism from other teachers and criticism from parents, whose children probably would not have done better under another teacher.

However, my father recognized genius when he saw it. When my little sister Eva graduated from Konrad Jahn's class, my father and I bicycled one evening through the dark streets of Copenhagen, visiting the homes of her class mates, asking the parents for donations for a farewell present. On the last day of class, my father made a short speech in front of the class and gave Mr. Jahn a handsome leather briefcase. It was a very emotional occasion.

In 1985, I dedicated my book *On Pascal Compilers* "To my teacher Konrad Jahn."

⋆ ⋆ ⋆

In 1950, at the age of twelve, I was accepted by St. Jørgens Gymnasium, an elite high school on Filippavej in Frederiksberg. For the next seven years I obtained a thorough grounding in Languages (Danish, Swedish, English, German, French, and Latin), History (Ancient, Medieval, and Modern), Science (Physics, Astronomy, Chemistry, Geography, and Natural History), and Mathematics (Algebra, Geometry, and Precalculus).

This was my first encounter with traditional education. Although my teachers did not strike me as visionary, they were highly educated, and some of them had written textbooks in their fields. Several of them are included in the 1959 edition of *Kraks Blå Bog* ("Who's Who in Denmark"): Frode Andersen (Physics), Morten Borup (Literature), Peter Ilsøe (History), Aage Kampp (Geography), Jan Neiiendam (History), and Just Rahbek (Literature).

The headmaster, Peter Ilsøe, visited every class room four times a year. He would ask each of us to come forward and receive our grade report, after announcing our class standing to everyone. I remember him as a jovial man in his early sixties. But he was apparently not well liked by the faculty. One teacher turned his back to the headmaster whenever he entered the class room and stood by the window until he left again.

Mr. Ilsøe was an interesting teacher of classic Greek history. From time to time, he would sing short verses he had written about historical events. This was a very effective teaching method. Fifty years later, I still remember one of them:

Fire hundrede firti ni	(In four hundred and forty nine)
Perserkrigen er forbi.	(the Persian war ends.)
Perikles for styret står	(Perichles runs the government)
i Athen i tyve år.	(in Athens for twenty years.)

My favorite history teacher was Jan Neiiendam. He rarely asked questions about our daily assignments. Instead, he told fascinating stories about historical figures. I remember one about the mad Danish king, Christian 7 (1766–1808), walking drunk through the streets of Copenhagen after a visit to a woman known as "Bootie-Cathrine." At night, the king walked around town with her and a group of young officers, picking fights with night watchmen and beating up girls in whorehouses.

A grey-haired woman, Edith Hintz, taught geography and handwriting. It was not her fault that my handwriting remains illegible to this day. She taught me a lesson I still remember. Out of boredom, I would sometimes use my key to drill a small hole in a tree in the schoolyard. When sap started oozing out of the hole, a biology teacher inspected the tree and declared that it was doomed. Miss Hintz sentenced me to write one hundred times:

Drilling holes in trees is vandalism!
Drilling holes in trees is vandalism!
. . .

The last time I saw the tree, it was still alive and doing well. So much for expert opinion.

Our math and science teacher, Knud Steenberg Sørensen was a very nervous man. You could not help getting nervous yourself when he quizzed you in front of the class. If you made a mistake, he would break down crying and shout: "You are not only getting a zero—you are getting a double-zero!"

On the last day of class before Christmas, he tried to entertain us. One year, he asked a student to imagine an object, which he called a "half-moon triangle." At that age, we were not trained to reason about the abstract properties of non-existing objects. And sure enough, pretty soon, the student and Mr. Steenberg Sørensen were both crying, and the Christmas entertainment turned into harassment. Fortunately, we were saved when the school bell rang. But that was not the end of it. When classes resumed after New Year, the teacher called upon the same kid and continued the mental torture for another hour.

One of my English teachers (who shall remain nameless) was a sadist. There is no other way of describing him. In the school yard, one of the boys had shouted his first name loudly behind his back. During our next English lesson, the teacher asked the boy, again and again: "Who gives you the right to call me by first name?" No matter what the kid said, the teacher slapped him hard. This went on for a long time, while the victim cried and the rest of us watched in horror.

In his novel *The Neglected Spring*, the Danish author Hans Scherfig (1940) describes such a teacher:

> And the teacher is an educated man. He has scientific interests. He has written a fine dictionary and many editions of excellent textbooks. He has traveled abroad and has acquired culture and fine manners.
>
> And the well-informed teacher has assumed responsibility for hitting those who are late. Perhaps his job could have been performed by a man with less scientific education. But it could not have been performed more conscientiously.

Most of our teachers were, however, decent human beings.

Students who were late in the morning were met at the entrance to the school by an elderly gentleman, senior master Just Rahbek. He would write your name in his note book and sentence you to arrive fifteen minutes *early* for the next three days. To me he looked like an eternal bachelor. I was surprised recently to discover a romantic side to his life: When he was 55 years old, he married a Croatian woman, Alexandrine Sisacki, from the town of Sisak.

As a teenager, I had a chemical laboratory in a corner of my mother's tiny kitchen. After school I would go downtown to Struer's Chemical Laboratory and buy small amounts of chemicals for experiments.

Once I tied a thread around a piece of sodium and lowered it slowly into mercury. This produced a small explosion with yellow flames. "Run mother!" I shouted. In the evening she complained to my father. But, in this case, he was on my side.

My father had an odd collection of old chemistry books, including an 1855 edition of *Die Schule der Chemie oder Erster Unterricht in der Chemie versinnlicht durch einfache Experimente* by Professor Julius Adolph Stőckhardt. It was printed in German Gothic type and was not an easy book for a beginner.

Fortunately, my classmate Peter Schoubye had a copy of Paul Bergsøe's wonderful book *Kemi på en anden måde* ("Chemistry in a different way").

On a snowy afternoon, I walked home from Peter's apartment carrying a testtube with a liquid we had produced. I wasn't sure what it was I was carrying. It was quite hot to touch and, from time to time, the stuff would bubble up and threaten to overflow the testtube. So I would place it in a snowdrift and wait for a few minutes, to let it cool down. The next day I found out it was an extremely toxic substance.

When boys experiment with chemistry, the temptation to make gunpowder is irresistable. My father lost his hair when the gunpowder he was making exploded in his face. His sister remembered him entering their kitchen, moaning quietly and putting his blackened head under the water faucet.

My cousin, Ole Bak, also made gunpowder, the usual way, by mixing sulphur, charcoal, and potassium nitrate. Since nitrate absorbs vapor from the air, gunpowder must be dried before it can be used. Ole got the bright idea of drying gunpowder in his mother's baking oven. It burned furiously, emitting noxious, poisonous gasses that forced his parents to leave the kitchen quickly and retreat to the other end of their house for several hours.

One day, Peter Schoubye's mother called me on the phone and told me that he had blown off one of his fingers and was in the emergency room. If I still had any of Peter's gunpowder, would I please throw it away. I did, and that was the last time I experimented with explosives.

⋆ ⋆ ⋆

After graduating from junior high school in the ninth grade, I was admitted to senior high in 1954. This was the first time I went to school with girls. I spent many pleasant afternoons with my friends, Sven Gundel, Ann Harsen, Sven Husum, Leif Christensen, Hanne Andersen, Bent Vang Olsen, and Jørgen Albertsen. We drank tea and talked about dating, school, movies, and books (but rarely politics).

Sven Gundel had a weird sense of humor. Although we were not very good at it, we enjoyed playing tennis. On one occasion, without telling me, he had invited the whole class to watch us play. After a while, he asked me: “Why don’t you play against Birgit?” So I did. Not only did she return every shot—she hit the ball so hard I ended up playing with my back against the fence. I had forgotten that Birgit Jensen had won a junior championship in women’s tennis. Everybody but me thought this was very funny.

Sven’s father, Leif Gundel, was the editor-in-chief of the Danish communist newspaper *Land og Folk* (“The nation and the people”). When we graduated from high school, Sven gave me a copy of the official history of the Russian communist party. I thanked him and asked wryly: “Don’t you think I should wait for the next revised edition?”

Sven Husum and I spent countless hours playing jazz records. Listening to jazz has remained an important part of my life during the long hours when I study or write.

I owe my love of jazz to my uncle, Børge Ring, a well-known Danish bass player. As a teenager, I visited Børge and his wife Nanny in Amsterdam, where he still works as a cartoon animator. In a record store we listened to a long-playing record with pianist Oscar Peterson and bassist Ray Brown. I didn’t really understand why my uncle liked it. But I admired him so much that I spent all my pocket money on that record. The appreciation came later, when I had played it numerous times at home.

In high school I also learned to appreciate literature. (As Groucho Marx said: “Outside of a dog, a book is man’s best friend. Inside of a dog, it’s too dark to read.”) From my friend, Jørgen Albertsen, I borrowed Henry Miller’s novel *The Tropic of Capricorn.* When our teacher, Morten Borup, mentioned that Jacob Paludan’s Danish novel, *Jørgen Stein*, was the “bible” of his generation, I held up Miller’s novel in class and said: “This is our bible.” The old man smiled, so he must have read it too.

I read more widely than at any other time in my life. When I graduated from high school, I had, of course, read the classic works in Danish literature—our teachers saw to that.

On my own, I read a fair amount of English and American literature (in English): most of Graham Greene, Ernest Hermingway, and John Steinbeck. One summer evening, I was reading Steinbeck’s *East of Eden*, when three girls came to our summer cottage and asked me to join them at a local dance. Well, I was so absorbed by the novel that I declined the invitation. You may think I made the wrong choice. Maybe. I don’t remember who these girls were. But I still remember Steinbeck's gripping story.

I read Danish translations of Norwegian, Swedish, German, French, and Italian novels; and, of course, the great Russian novels: Leo Tolstoy's *Anna Karenina* and Fyodor Dostoevsky's *The Brothers Karamazov* (Time Magazine once wrote: "These brothers need a keeper!"). You get the idea: I read all the time.

James Joyce's *Ulysses* was a challenge I could not resist. This 700-page novel describes a day in the life of an advertising agent, Leopold Bloom, as he wanders through the streets of Dublin as a modern Ulysses. It is regarded as one of the most difficult works of fiction, embedded, as it is, in multiple layers of meaning. Every chapter not only mirrors an episode in Homer's epic poem—it also centers around a place, a human organ, a science, a color, and so on. Chapter 14, for example, corresponds to the episode of The Oxen of the Sun—an ancient symbol of fertility. The place is a hospital, the organ is the womb, the science is medicine, and the color is white. The chapter traces the nine months of a pregnancy and, in parallel, the historical development of the English language. The opening of the chapter is written in the earliest English prose. When the child is born, Joyce switches to modern English.

Joyce plays dirty tricks with the reader. Early in the novel, Leopold Bloom is leaving his home: "On the doorstep he felt his hip pocket for the latchkey. Not there. In the trousers I left off. Must get it. Potato I have." The allusion to the potato becomes clear only 400 pages later: "Spud again the rheumatiz? All poppycock, you'll scuse me saying."

As an eighteen year old I decided to climb this intellectual mountain. Every Saturday, I spent the whole night reading one chapter of Ulysses, writing down notes and questions. A week later, I read the same chapter again. It took me six months to finish the novel this way.

In my final year of high school, I passed an exam, where we had six hours to write an essay. I wrote about Ulysses from memory. A leading newspaper published the essays of all students who got As. Since I got an A–, my essay was not published.

Today I think Joyce misunderstood the essence of creativity. A written work should not be a labyrinth. The genius of Isaac Newton was not to make physics incomprehensible. On the contrary, his unique contribution was to make it possible for others to understand what only he could describe concisely.

I admit there is room for disagreement here. People still gather in a Syracuse café on "Bloom's day" (June 16) and take turns reading the last chapter of Ulysses. But I have lost my taste for difficult writing.

In June 1957, I graduated with honors from St. Jørgens Gymnasium. I was now a broadly educated student of the arts and sciences, who had learned to read widely and write well. It was time to choose a more narrow path towards a professional career.

2

CHOOSING A CAREER 1957–63

Advice from a professor – Technical University of Denmark – Ørsted's influence – Distant professors – Easter brew – Fired for being late – International exchange student – Masers and lasers – Radio talk — Graduation trip to Yugoslavia – An attractive tourist guide – Master of Science – Professional goals.

There was never any doubt in my mind that I wanted to become an engineer like my father. But what kind of engineer? I didn't think it would be a good idea to enter civil engineering where my father already had made a name for himself. My choice of electrical engineering may have been influenced by the recent popularity of television in Denmark following the coronation of England's Queen Elisabeth II in 1953.

In senior high school, I was usually the best student in my class. But my chemistry experiments were just a hobby. I did not make any scientific discoveries. And, my interest in literature did not inspire me to write original works of art. Although I was reasonably intelligent, I was not a prodigy.

Since my mother did not have a high school education, she worried about whether I would be able to complete an engineering education. So she persuaded me to call my father's colleague, Helge Lundgren, professor of Harbor Construction at the Technical University. "What was your final math grade in high school?" he asked. "A−," I answered. "Well," he said, "then it is, of course, impossible to say anything with certainty." No, professor Lundgren was not joking—he was absolutely serious. Ah well, I thought, I will just do the best I can. So I enrolled as a student of electrical engineering.

In 1957, the Technical University was still known as Polyteknisk Læreanstalt ("Polytechnic University"). It was founded in 1829 at the initiative of Hans Christian Ørsted, the Danish discoverer of electromagnetism. Since he believed that a well-rounded engineer should master the fundamentals of all fields of engineering, the students were called *polytechnicians* (from the

Greek word *poly* meaning "many"). Ørsted remained the first chancellor until his death in 1851.

I began my studies in a quadrangle of massive buildings from 1890 in Sølvgade, not too far from downtown Copenhagen. They were situated in a corner of the Botanical Garden, across the street from the Eastern Park ("Østre Anlæg"). These beautiful parks were the remnants of the ramparts around the medieval Copenhagen, which were leveled in the 1860s to make room for the growing city.

The Technical University had no dormitories. Most students from Copenhagen lived at home with their parents. Students from other towns rented rooms nearby or lived in public residence halls. Some students could not handle the freedom of living away from home for the first time. They would skip classes and play cards in the student cafeteria.

There was no parking lot for students. Nobody that I knew owned a car. Like most students, I used my bicycle to get from our apartment to the university.

Thanks to Ørsted, all engineering students took the same classes for the first two years: Mathematics, Geometry, Physics, Chemistry, Applied Mathematics, Theoretical Mechanics, Structural Engineering, Material Science, and Geometric Drawing.

By the 1950s, this noble ideal was beginning to look somewhat impractical. Even I could see that an A− in Structural Engineering did not qualify me, as an electrical engineer, to design the transmission tower for a television station.

Material Science was ridiculous: you had to learn a great many physical constants by heart, such as the electric conductivity of copper with eight decimals. (Fortunately, some of the leading digits were zeros.) Before the written exam, I wrote all these constants on a huge sheet of paper. After memorizing them for days on the balcony of my mother's apartment, I passed the exam with a B+. By the end of the summer, I had forgotten most of them.

When you work as an engineer, you soon learn to remember the most important constants in your field; the rest you look up when you need them. It serves no purpose to learn them by heart as a student. My friend Niels Zeuthen Heidam agreed that Material Science was a waste of time. As soon as we had passed the course, we sold our textbooks and used the money to buy rum and coke.

We were now attending lectures in a huge auditorium with a hundred students, or more. In the winter, the auditorium was heated by enormous radiators covered with wooden boards. When I came late to class (as I often did) all the seats were already occupied. The only place to sit down was on top of a radiator. After a while, the rising heat made me drowsy. So I would stand on the floor until my feet got tired. Then I would sit on a radiator again, and so on.

I felt somewhat lonely among all these students I didn't know. And the professors were distant figures on the podium whom I never saw outside class.

I did not attend lectures regularly. I preferred to study at home, until I reached the point, where I was unable to do the homework. Then I would borrow notes from a friend and attend the lectures until I caught up again. Although our homework was graded by teaching assistents, their grades served only to remind us how we were doing. Real grades were only given for final exams at the end of the year. If you needed help, you had to find and pay for a private tutor. I didn't need a tutor and would eventually graduate with a grade point average that made me number 3 of 47 students.

The technical university was undoubtedly an elite institution. It was highly selective and demanding. It made no attempts to retain those who dropped out. I don't think it can be done any other way if your primary goal is to educate competent engineers who can design reliable bridges, airplanes, and computers. (The poor reliability of software shows that programming is still not taught as a rigorous engineering discipline.)

The way it was taught, chemistry became dry as dust, and mathematics became understandable, but intimidating. Erling Følner was professor of mathematics. He married the daughter of the famous mathematician, Harald Bohr. In the days before women's liberation, this brought to mind the old German saying that mathematical talent is passed from father to son-in-law. He would walk into the auditorium, turn his back to us, and fill six huge blackboard in minute handwritting with formulas, copied verbatim from the textbook, and walk out again.

Judging from our textbooks, mathematicians seemed to have the ability to see into the future. In order to prove an important result ("a theorem") they would often start by proving a minor result ("a lemma"), which they would then use to prove the big one. I could not, for the life of me, understand, how they could possibly have known in advance that they would need this lemma to prove that theorem. I was so awed by this gift of foresight, which I did not seem to have, that the thought of becoming a mathematician never entered my mind.

Years later, I discovered that mathematicians are faking it. They do *not* prove theorems the way they present them in textbooks. They fumble with ideas, just like the rest of us, until they realize that some lemma may simplify their proof. Then they turn around and present their results, as if they were discovered by sleepwalking geniuses, who never make mistakes or enter blind alleys. This orderly method of presentation is technically correct, but misleading, since it hides a crucial insight from students: how do mathematicians *really* think?

I enjoyed my physics courses on Mechanics, Thermodynamics, Light, Electricity, Magnetism, and Quantum Theory. However, my diploma shows that I got higher grades in Mathematics. I don't know why.

On October 4, 1957, the Soviet Union launched *Sputnik*, the world's first artificial satellite. Weighing 184 pounds, it was about the size of a football and carried a radio transmitter. It circled the Earth for several months.

This historic achievement gave professor Henning Højgaard Jensen a well-come opportunity to compute the initial speed a satellite needs to escape gravity and stay up there. On Earth, the *escape velocity* turns out to be roughly 7 miles per second (about 11 km/s).

I remember professor Richard Petersen fondly. "Little P," as we called him, was a small, energetic man in his mid-sixties with a friendly smile. He was the only professor I can remember, who knew the names of all his students, no matter how large his class was. He taught applied mathematics: probability theory, queuing theory and Laplace transforms. If that sounds boring, let me assure you that it's not.

Little P once made a minor mistake on the blackboard and ended up with the wrong result. He looked at his formulas for a long time and finally said: "Jeg har et problem med min potens." In English this just means, "I have a problem with my exponent." But, in Danish, it also means "I have a problem with my potency." He looked puzzled when everybody burst out laughing.

Technical drawing was boring to me. (I often tell my students: "When you say something is boring, it is a statement about *you*—and *not* about the subject matter.") The instructor, Helge Christensen, demanded absolute perfection. We were expected to draw geometric figures in India ink with an accuracy of 1/10 of a millimeter (that is, 1/250 of an inch). My friend, Nils Havsteen had to redraw a gateway motive ("porten") ten times before it was accepted. If it was tedious, it was also invaluable for future engineers. Most university students already possess intelligence (otherwise they wouldn't be there). What they lack is the professional discipline required to get minute details right. Helge Christensen taught us that. Fifteen years later, I came to appreciate this lesson in precision: when my first textbook was in production, I told my publisher that the redrawing of my figures was too inaccurate and had to be redone.

On the last day before Easter, the student cafeteria sold a strong local beer, known as *Easter brew*. On that day, wise professors cancelled their lectures. But there was always someone who forgot this. I remember a professor who entered a lecture hall full of drunk students (I was one of them). So the professor starts lecturing, while a hundred students talk loudly. After a while, the professor shouts "Silence!" For a brief moment, the only sound is that of beer bottles rolling down the steps of the auditorium. Then the talking resumes. When an hour has passed, the students do not wait for the professor to announce that the lecture is over. We all shout "Time's up!" and leave. Outside, I find a friend sleeping on the cold pavement. I wake him up, and together we somehow make it to the street and take a taxi home.

In the spring of 1959, after two years of study, I had completed the first half of my engineering education. I was looking forward to the second half, where I would concentrate on electrical engineering, specializing in electronics and telecommunication. These subjects were taught in an extension of the Technical University in Østervoldgade, built around 1940.

My excellent professors included Jørgen Rybner, who, in his late fifties, was obviously an authority on Electric Circuits and Telephony. However, the major breakthroughs in these areas were made long before my time.

Hans Lottrup Knudsen taught a demanding (but rewarding) theoretical course on Electromagnetic Fields and Antennas. At the oral exam, sitting alone in front of him, without any books, I was asked to use Maxwell's equations to derive the radiation pattern of a linear array of antennas. Such an arrangement has the ability to concentrate electromagnetic radiation in a single plane.

Recent technology was also taught by Jens Rasmus Jensen (Servomechanisms), Per Gert Jensen (Digital Electronics) and Georg Bruun (Transistor Electronics).

For my degree project, I used tunnel diodes, invented by Leo Esaki at the beginning of my studies. Although I got an A−, I no longer remember anything about it. Long after my graduation, I used to dream that the university had just informed me that my diploma was invalid, because I had failed to complete my project!

⋆ ⋆ ⋆

Before graduation, all students were required to complete industrial practice working for a participating employer. The intention was to teach future engineers to get along with workers. I worked in the machine shop of a small company, Danish Servo Technology, owned by Søren T. Lyngsø. I admired the professionel skill of the machinists, but my own work was trivial beyond compare: drilling holes in aluminum boxes and cleaning machines with kerosene. I did, however, enjoy playing pool with the workers after hours. After a while, I left the machine shop to work in the company's lab. Here I designed the only electronic device in my entire career: a transistor circuit that could detect if a weaving machine attempted to tie more than two threads together at a time.

At night, I read books about interesting subjects outside my university curriculum: Quantum Theory, Solid State Physics, Semiconductors, and Signal Flow Graphs.[1] But I did not get enough sleep and would often be late for work. Mr. Lyngsø could not very well expect his workers to show up on time, if a student did not do the same. So, eventually, he fired me. This is the only time that has ever happened to me.

He was right, of course, but I was now in a precarious situation. The university administrator in charge of industrial practice pointed out how important it was for him to maintain good relations with Søren Lyngsø. He was hesitant to recommend me to another participating employer. This was serious, since I needed to complete my practical experience to graduate.

It turned out to be one of those unnerving moments in life, when your situation looks hopeless, and then it turns out to be the beginning of an unexpected piece of luck: the university decided to let me continue my industrial practice abroad as an international exchange student.

⋆ ⋆ ⋆

My father's career in soil mechanics inspired me to look for an area that was still in its pioneering phase. If a subject was being taught, it was probably already too late to make fundamental contributions, I thought. Throughout

[1]My reading list included D. Bohm, *Quantum Theory*, 1951; C. Kittel, *Introduction to Solid State Physics*, 1956; E. Spenke, *Electronic Semiconductors*, 1958; and S. J. Mason and H. J. Zimmermann, *Electronic Circuits, Signals, and Systems*, 1960.

my studies, I continued to look for something that was not yet being offered by our professors. At the beginning of my studies, I read about television in Frederick Terman's classic text, *Electronic and Radio Engineering* (1955). A few years later, the excitement seemed to be in transistor circuits. However, by 1960, it was obvious to me that electronic digital computers was the technology of the future.

My university did not offer courses about computers. Christian Rovsing, president of the engineering students' union (1959–60), gave me some idea of the nature of computer programming. For three evenings, he taught a handful of students the rules of the first programming language Fortran.

A milestone in computing, Fortran was developed by John Backus, at IBM headquarters on Madison Avenue in New York City, and introduced for the IBM 704 computer in April 1957. Fortran programs were written in a notation that resembles ordinary algebra.

On the last evening of Rovsing's short course, I said: "Christian, I understand most of what you have explained, but *what does it all mean*?" You see, without access to a computer, the rules were just formalities to me, without any connection to practical reality.

Christian Rovsing went on to a distinguished career. After graduating, he worked a couple of years for IBM in Sweden and France, before starting his own computer company in Denmark. In 1984 he received the IEEE Centennial Medal.

⋆ ⋆ ⋆

So far I have written from memory supported by a few documents. At this point, my story is supported by letters to my parents and my future wife, Milena.

In the summer of 1961, I spent seven weeks as an international exchange student at IBM's Hursley Laboratory, outside Winchester in Southern England.

On the way to Winchester, I visited London as a tourist. I traveled on a student ticket, which I only received shortly before the train left the central station in Copenhagen. A student representative handed me a bunch of forms and explained that he would like me to be the guide for 75 students traveling to London that day. In return he promised to pay my train ticket, which sounded reasonable to me.

In the southern Danish town of Gedser, I was supposed to help students through the passport control and board a ferry to Germany. I quickly learned to hide my ignorance. When the students asked questions, like: "Is it far to the train in Grossenbrode?" I would improvise: "No, only a five minute walk from the ferry!"

The next morning, when we changed trains in Amersfoort, Holland, I almost lost a group of Swedish girls, who just stood next to their suitcases on the platform, while the rest of us walked decisively in the right direction. I didn't notice them until the conductor signaled the departure of our connecting train. I rushed over, picked up the nearest suitcase and got them moving fast. We boarded the train just in time.

Rotterdam, which had been demolished by German bombers on May 14, 1940, was completely rebuilt and topped by a forest of TV antennas. The silhuette of the enormous harbor was impressive. Behind the apartment buildings, it extended from one end of the city to the other.

In Hook van Holland, we boarded a ferry to England on a sunny day. Sitting on the top deck, looking at the quiet sea, I felt like Onassis on a Mediterranean cruise.

At Liverpool station in London, I left the students in the care of a Norwegian student representative and took the subway (known as the "tube") to Archway. My lodging was one of many similar townhouses: Nice outside, but rather neglected inside, due to the many lodgers who had lived there. When I arrived, there were about ten of us, all young men. The landlady, Mrs. Sheridan, who was rather nice, understandably preferred not to spend her time talking with lodgers.

I shared a small, unattractive room with an English batchelor, who was interested in science and very talkative. Since my roommate went to bed early and turned out the lights, I got up at eight in the morning and left for the center of the city at ten.

Walking around London without any plan, I "discovered" streets and buildings I recognized from movies and books: Oxford Street, Marble Arch, Piccadilly Circus, Charing Cross, Trafalgar Square (full of pigeons you practically had to brush off), and Big Ben in floodlight.

At the British Museum, I was awed by the Egyptian and Assyrian collections I had read about in Carl Grimberg's *The World History* (1959). How marvelous to find an enormous exhibition hall displaying a relief showing the Assyrian king Assurbanipal almost 4,000 years ago, on foot, killing a lion with a dagger. To recognize details of the relief and know that the king once stood before this monument!

Another hunting scene carried the inscription: "I am Assurbanipal, king of the World, king of Assyria. With my strength I held, alone on foot, one of the ferocious desert lions by the ears, and, with the help of Assur and Ishtar, I ran my spear through it."

My strongest impression of London was the striking contrast between the depressing working-class neighborhoods, and the impressive center with its beautiful buildings and multitude of monuments. A dirty city, without the charm of Paris, but fascinating with its endless traffic and people of so many nationalities.

The parks of London were enormous and beautifully landscaped. I walked in Hyde Park, saw Kensington Park from a rowing boat on the Serpentine, and enjoyed Green Park, close to the Queen Victoria Memorial.

The main railroad stations were huge: Paddington, Waterloo, St. Pancras, Victoria station—noisier and dirtier than anything else I saw.

Winchester was completely different with a population of about 30,000, and much cleaner than London. The town had a historical continuity without parallel. Outside the town was a hill crowned by a gigantic rampart from the Iron Age. Nearer the city, sections of the town wall dated back to Roman times. Between the 9th and 13th centuries, Winchester was the capital of England. The Danish viking king, Canute the Great (about 995–1035) ruled here and is buried in Winchester Cathedral. "Canute had two sons, Halfacanute and Partacanute, and two other offspring, Rathacanute and Hardlicanute" (Sellar 1964). Two city gates from the 13th century were still standing, and the oldest houses were from the same period. Half-timbered houses from the 16th century were common.

I rented a room just outside town, in a former farmhouse on Andover Road. The room was a huge improvement over the one in London: 18 by 24 feet, with a large armchair, a radio, and a view of the garden.

My landlady, Mrs. Early, was kind and helpful. The first evening, she made me a big dinner with meat, potatoes, and bread pudding. At home, my mother had taught me that you must eat everything on your plate. Otherwise, your hostess will think that you don't like her cooking. She would have been shocked by the American habit of eating a steak, leaving the fat on the plate. So I ate it all. Mrs. Early apparently thought that this young man must be starved. The next day she gave me twice as much food for dinner. Again, I did what my mother had taught me, and, with some difficulty, ate the whole thing. On the third day I gave up, when she offered me even more food.

In the room next to mine, her grandchild watched TV every day, for hours. On my side of the wall, I would mostly hear gunshots and horses neighing. I saw this as a premonition of the childhood of my future children. When my own children grew up, we only had a TV in the living room, and their viewing time was restricted. As far as I can tell, this discipline had no lasting impact.

Hursley Park was a former manor with enormous oak-paneled rooms, which IBM had turned into a research center. It was 3 miles outside Winchester. On my first day, I was shown around and introduced to many people. The English struck me as incredibly helpful and charming.

I did what I could to improve my English. I listened to BBC on the radio, read newspapers, wrote technical notes in English, and talked to technicians during the lunch break.

Weekends I spent with a Danish engineer, named Svend, who had bought a Volkswagen on credit (known in England as the "never-never"). After my first week in Winchester, we drove back to London for a one-day visit.

The purpose of our trip was to see the recently opened Soviet Exhibition. The astronaut, Juri Gargarin, had just visited England. In April 1961, he became the first man to orbit Earth in a spacecraft. Since the British government had no protocol for visiting astronauts, it took three weeks to decide who should receive him.

First we went to a Russian fashion show. The models were rather hefty by western standards, and the few party dresses looked clumsy. But the everyday clothes looked quite attractive.

The space exhibition was located in a fantastic planetarium, where movies where shown simultaneously on five screens, accompanied by electronic music. Walt Disney could hardly have done it better!

The rest of the exhibition was somewhat disappointing. The huge number of visitors, mostly Asians and Africans, made it difficult to see the exhibits. And there was very little written information about the machines and instruments shown.

We stayed at the exhibition for most of the afternoon and returned to the city in the evening. Piccadilly swarmed with people, and it was nice to be there again. Back in Winchester, at 1 a.m., all lights were out.

A month later, Svend and I went on a three-day vacation to Cornwall. On the way back, we visited Stonehenge, Europe's most famous Stone Age monument. This mysterious circle of huge stones, was built between 3000 and 1900 BC, about the time when the Egyptian pyramids were built. The average stone, weighing about twenty-six metric tons, was about 18 feet tall.

The original thirty stones had supported a circle of curved lintels.

Every morning, a technical assistant, named Dave, would give me a lift from Winchester to Hursley Park. The English engineers were very relaxed. In the morning, they would usually chat for the first half hour. At ten o'clock, they took a coffee break and, at noon, they went to lunch. After lunch, Dave and I might go for a long walk, or I might play chess with a technical assistant. At three in the afternoon, there was a tea break and, at 5:30, everybody went home.

The lab was not quite what I had imagined. They did not waste time on "unnecessary research." They all worked on two computer projects. The largest employed about 75 people, including 25 engineers, for several years.

My knowledge of semiconductor physics turned out to be irrelevant. If the engineers knew *what* a transistor did, it didn't interest them *how* it did it. I concluded that the difference between transistor physics and computer organization is similar to the difference between biochemistry and physiology.

The first couple of days, I was completely lost. The engineers hardly had time to answer my questions, and I discovered that, without a certain amount of knowledge, one cannot even ask relevant questions.

The machine, they were working on was, of course, not yet described anywhere. So, initially, I spent most of my time at work reading about computers. However, when the lab started assembling its computer, I was given a necessary, but monotonous task.

The computer memory consisted of hundreds of thousands of tiny ferromagnetic rings (known as "cores"). Each core stored one bit of information (a zero or a one). The cores were arranged in two-dimensional arrays with two perpendicular wires passing through each core. A particular core was selected by sending currents simultanously through both wires that passed through it. The magnetic state of the selected core would then be sensed by means of a third wire passing through all the cores.

My task was to test the memory, one bit at a time, by observing, on an oscilloscope, how each core reacted when it was selected. Although I performed this task carefully, my lack of enthusiasm must have been obvious.

The design team had an unusual organizational structure. It was headed by two people: a technical leader and a management leader, who, as far as I could tell, got along well. It was interesting to watch what happened when the management guy took a week's vacation: after a couple of days, the technical leader and the rest of the engineers slowly stopped working, waiting for the manager to come back.

In 1960, IBM became concerned that their computers were incompatible with each other (Bashe 1986). At that time, IBM was selling eight transistorized computers, six of which were incapable of executing programs written for the others. A year later, a companywide task force recommended that IBM create a family of compatible computers with the same architecture. This was the beginning of the IBM/360 computers.

It was also the end of the computers developed at Hursley. I was there the day a manager announced that the development of their small computer, called SCAMP, would be stopped. This was devastating for the engineers, who had put their whole effort and creativity into a project that went nowhere. Twenty years later, the same thing happened to me, when an American company built a multiprocessor to my specifications. But I am getting ahead of my story.

After seven weeks, my enjoyable summer at IBM came to an end. One of the managers wrote a report (Fig. 2.1) for IAESTE (International Association for the Exchange of Students for Technical Experience).

I returned to Denmark with the impression that I knew very little about computers, but would like to dedicate my career to these miraculous machines.

Employer's Report:

Name of Student: Brinch Hansen, Per
Period of practical experience in weeks: 7 weeks
Rate of payment per week: £7
Gross payment made to student: £50 8s. 0d.
Conduct: Excellent. Time-keeping: Good.
Observations on the student's general aptitude:
Worked on a variety of constructional and checking jobs.
He showed himself to be a keen, intelligent, theoretician
with a more limited aptitude for practical work.
Date: 7th November, 1961. Signature: (unreadable)
On behalf of IBM World Trade Laboratories (G.B.)

Figure 2.1 The IBM manager's report.

⋆ ⋆ ⋆

As a student, I wrote my first technical article—about the exciting American inventions of the laser and the maser. Not that I felt competent to write about it, being only a student, and not a physicist. But, even though American technical magazines frequently mentioned the laser, and had done so for some years, nobody else in Denmark had written about it.

The laser is able to create light 100 million times stronger than the light on the surface on the sun. The maser, which emits microwaves instead of light, was used, in July 1962, to amplify the extremely weak signals from the first television satellite, Telstar.

In Niels Bohr's classic model of the atom, electrons orbit around a nucleus at discrete distances. When atoms absorb energy from their surroundings, electrons jump from inner orbits with low energy to outer orbits with higher energy. As the electrons fall back to their inner orbits, light is emitted in discrete amounts. This *spontaneous emission* of light is random and diffuse in nature. That's why you can only focus a flashlight beam at short distances.

In 1917, Einstein predicted that light emission could be increased dramatically by shining light of the right frequency on atoms. If the atoms start with a lot of electrons in their outer orbits, the presence of light starts a chain reaction, where some atoms emit light, which then stimulates the emission of light from other atoms, and so on, until all the atoms emit light at practically the same time.

The *stimulated emission* of light, which does not occur in nature, is the principle behind the laser. The laser uses an artificial, oblong ruby coated with silver at both ends. When the ruby is stimulated by light, it emits spontaneous light in all directions. The light that happens to be reflected from the ends of the ruby will travel back and forth numerous times, stimulating further light emission from the atoms. This creates a strong ray of red light, which is pencil-thin and remains focussed at enormous distances.

On May 1, 1962, the monthly magazine for Danish engineers, *Ingeniøren*, published my article on *Maser—a new amplifier element that makes communication at optical frequencies possible.* (I had not yet mastered the art of using titles of five words or less.) A professor of physics had been kind enough to check that my derivation of Einstein's laws of electromagnetic radiation was correct.

I was now invited to record a popular talk on the magic ruby for the Danish Broadcasting Service. Perhaps someone older, and more experienced than me, ought to have done it. On the other hand, I thought, this is my chance, and I can't afford to pass it up.

During the recording of my talk, I was nervous and kept talking, without breathing, until I had to stop and gasp for air. Fortunately, the radio technicians were able to remove this sound effect from the tape. On March 14, 1962, the Danish radio broadcast my program on *The Red Ray of the Ruby.* Notice the alliteration, as in "James Joyce" or "Buddy Bolden's Blues."

The editor of the science program was very enthusiastic about my manuscript. He called the editor of the popular science magazine, *Vor Viden* ("Our Knowledge") and encouraged him to print it. It appeared in print on May 31 and June 14, 1963.

⋆ ⋆ ⋆

In the summer of 1962, our graduating class traveled abroad with a professor, visiting foreign companies and relaxing before final exams. The organizing committee consisted of Johannes Arboe Brøndum, Paul Waltenburg and me. Paul had spent two months in Yugoslavia as an international exchange student. His brother Carl had just married a woman, named Renata Stankovič, from Zagreb. So we decided to invite Professor Georg Bruun to accompany us on a graduation trip to Yugoslavia.

We traveled by train from Copenhagen to Munich, Germany, a twelve hour journey. Whenever we had to change trains, the forty students (or so) lined up on the platform with their luggage, while one of us ran ahead and found out where to go from there.

After another ten hours by train through Austria, we arrived after dark in Ljubljana, the capital of the Yugoslav republic Slovenia. At the railroad station, we were met by two female guides and driven in an ancient, dark bus to a student dormitory.

The next day, we were given a guided tour of a large manufacturing company, named Litostroj. Afterwards, we had the opportunity to ask questions of the management. I remember asking: "How much does a worker and a manager earn?" One of our guides translated the question into Slovene. When a manager responded to my question in Slovene, there was a brief discussion between the guide and the manager (still in Slovene). She finally translated the answer to my question. Later, she explained to me: "When the manager first answered your question, I told him, it will leave a bad impression on these foreigners, if we tell them how much a manager makes." After some discussion, they agreed among themselves that perhaps the figures ought to be lowered. Their final answer turned out to be much lower than anything I knew from Denmark.

On the third day, we traveled by bus to the famous Postojna caves, south of Ljubljana. A local tourist guide showed us a small white salamander that lived in the caves. In heavily accented English, he explained: "The human feesh, it valks on de ground and svims in de vater."

I now had a chance to look more closely at one of our tourist guides from Ljubljana. She was a small blonde woman, with the figure of a model, walking elegantly in high heeled shoes with a deliberate little twist of the instep. Her name was Milena Hrastar. She had a master's degree in English and German from the University of Ljubljana. During the summer, she worked as a tourist guide for foreign students. On our last evening in Ljubljana, I invited her out for dinner at restaurant "Rio."

The next morning, all of us left Ljubljana and took the train to Belgrade, the capital of Yugoslavia. However, it was not Milena, but another guide, named Zorka, who accompanied us.

In Belgrade we visited the wrong factory, a small place that produced loudspeakers. The manager quickly recovered from his surprise and gracefully gave us a tour of the factory. When we left, professor Bruun said: "That was today's visit. The rest of the afternoon is off." My friends went to a large outdoor swimming pool, while I walked around Belgrade on my own. I remember a poor neighborhood, where a Turk in baggy, black pants was walking with a barrel on his shoulders.

From Belgrade we flew to Dubrovnik, on the Adriatic coast. This is the most beautiful city I have ever seen. Inside the walls, which surround this medieval city completely, cars were not allowed. In the evening, the only sounds you heard on the main street, were the shuffling of feet on the marble sidewalk, and voices echoing from the white sandstone houses.

Further north, in Split, I remember sitting in an outdoor cafe in the middle of the ruins of the palace of the Roman emperor Diocletian. The emperor retired to this palace on the Dalmatian coast in 305. I found it charming that the modern inhabitants stretched clotheslines between the antique columns and the nearby apartment buildings.

On the train to Belgrade, I discovered that I had forgotten my raincoat in Ljubljana. That gave me an excuse to leave my friends in Zagreb, a couple of days later, and take the morning train back to Ljubljana, where Milena was waiting with my coat. We spent the afternoon together. After a nice dinner at restaurant Šestica, we walked through the quiet, rainy streets to the railroad station. The evening train from Zagreb to Munich had already arrived at the station, and my friends were anxiously waiting for me to turn up before the train left. They were all standing at the windows cheering when I kissed Milena goodbye. My friends teased me, as I walked through the train looking for my luggage (which they had taken care of). As we approached Austria, the dark flat country of Slovenia changed into the rocky silhouettes of the Julian Alps.

Back in Munich, I had two hours to eat breakfast, buy cigars, and send Milena a postcard. Then the long train ride through Germany. In Hamburg: another card for Milena, and, early in the morning, back in Copenhagen and pretty tired.

⋆ ⋆ ⋆

In the last year of my education, my professional goals became much clearer to me. I was very interested in the construction of computers. The few available books, I had read, looked at computer design from the point of view of an electronic engineer. These books explained the electronic design of switching circuits, arithmetic units, memories and peripheral devices.[2]

And then I read a completely different book that described computer organization from a programmer's point of view.[3] It was a revelation to me.

Project Stretch was an experimental supercomputer designed by IBM in the late 1950s. Although it was a commercial failure, this ambitious effort was a milestone in computing. The Stretch designers introduced the term *computer architecture* to describe the functional aspects of a computer that are of interest to programmers only. These aspects are independent of the underlaying electronic circuit technology.

I had never before seen a reasoned essay on the choice of a character set. In the 1960s, this was a problem of considerable practical importance in computing. Inside a computer, the letters of the alphabet are represented by numbers, as are the digits and special characters. The trouble was that different computers used different character codes. In order to process text, output by one computer, on another computer, you first had to replace each character code by a different one. The problem of character incompatibility would disappear in the late 1960s, when computer manufacturers adopted the ASCII character set, which is now standard on all computers.

The Stretch book discussed other general issues in computer architecture: Should computer arithmetic be decimal or binary? How should computer

[2]My own book collection included A. P. Speiser, *Digitale Rechenanlagen*, 1961; and H. D. Huskey and G. A. Korn, *Computer Handbook*, 1962.

[3]W. Buchholz, *Planning a Computer System: Project Stretch*, 1962.

instructions address their data operands? How can program execution continue during input/output operations? How can a computer execute several programs at the same time?

I knew I wanted, someday, to be able to understand this book completely and become a computer architect.

3

LEARNING FROM THE MASTERS 1963–66

Regnecentralen – Algol 60 – Peter Naur and Jørn Jensen – Dask and Gier Algol – The mysterious Cobol 61 report – I join the compiler group – Playing roulette at Marienlyst resort – Jump-starting Siemens Cobol at Mogenstrup Inn – Negotiating salary – Compiler testing in Munich – Naur and Dijkstra smile in Stockholm – The Cobol compiler is finished – Milena and I are married in Slovenia.

On January 31, 1963, I graduated from The Technical University of Denmark with a master's degree in electronic engineering. Shortly before, I started looking for my first job as an electronic engineer:

> I want to be sure I get a good job—one concerned with electronic computers, and the main thing is not my wages, but rather that I constantly learn something new. The question about what to learn is quite tricky. First I wanted to learn "everything" about computers, but lately a professor at our technical university has convinced me that there is the danger, that I will spend my most productive years merely trying to understand, what others have done, without having time to contribute anything myself. So the question comes up: When and what to specialize in? Anyhow I'm going to have a talk with the manager at our biggest computing center on January 3rd. (Letter to Milena, January 1, 1963.)

Actually, I did have some idea about my professional goals. I just didn't know, if I could pursue them in Denmark.

The only place in Denmark that developed computers was Regnecentralen, a research institution under The Danish Academy of Technical Sciences. In 1957, Regnecentralen completed the first Danish computer, Dask, in an old villa on Bjerregaardsvej 5, in Valby, a suburb of Copenhagen. Built under the leadership of Bent Scharøe Petersen, Dask used thousands of vacuum tubes in its electronic circuits and tens of thousands of magnetic cores in its memory. It executed 18,000 instructions per second.

Only one copy of Dask was built. Weighing three and a half metric tons, it was installed in the former dining room of the villa. The oak parquet floor had to be reinforced to support this computational monster. A large cooling and ventilation system was installed in the basement.

The power supply of Dask emitted a sharp blue light that was visible from the street. An elderly lady, with a vivid imagination, complained that she felt a prickly sensation from "these electrons and atoms" whenever she walked past the villa.

In November 1961, Regnecentralen finished a small, transistorized computer, named Gier. Housed in a wardrobe-sized closet with teak paneling, Gier looked like a piece of modern Danish furniture. It had a core store of 1024 words and a drum of 12800 words (about 5K and 60K bytes). Eventually about fifty Giers were produced.

My job interview at Regnecentralen started in the Rialto Center, a new office building on Falkoner Alle 1, within walking distance of the apartment I shared with my sister in Frederiksberg.

I talked briefly with the director, Niels Ivar Bech, a charming, dynamic man, who asked me: "Where will you be in ten years?" With tongue in cheek, I said: "In your chair!" He smiled—that was the kind of answer he liked. Looking back, my answer was absurd. There was no chance that I would ever be able to replace Bech's inspired leadership. But I didn't know that at the time.

For the next six hours I had unscheduled meetings with various department heads. Whenever they realized I was looking for something else, they would suggest that I visit another department.

I spoke to Aage Melbye about administrative data processing. His people programmed some of the most demanding computer applications. The main problem was to update large files efficiently and reliably. Since drums and disks were still small, the files were stored on magnetic tapes. To avoid wasting computer time after a tape failure, it was necessary to include restart facilities in these programs. A few years later, I would gain first-hand knowledge of these problems, when I programmed the input/output system for the Siemens Cobol compiler.

My next stop was the hardware group in Valby, headed by Henning Isaksson. Two years earlier, they had finished the Gier computer. I explained my interest in computer architecture and mentioned that I would prefer a job that would constantly teach me something new. Henning made it clear that, if he needed two flip flops, I would have to do the same thing twice. This made sense from his point of view (but not mine). I could not have predicted that Henning eventually would make my dream of becoming a computer architect come true. However, on that day, he suggested that I go back to the Rialto center and talk to the compiler group.

On the fifth floor, I met the leaders of Regnecentralen's compiler group: Peter Naur, a tall man with a serious expression and a full beard, and Jørn Jensen, a short man with a friendly smile and an unruly mop of hair. When I had explained my interest in understanding the relationship between programming languages and computer architecture, they handed me a thick yellow report with a devious smile and said: "Come back next week if you understand this." The report was entitled *Cobol-1961, Report to Conference on Data Systems Languages* (U.S. Department of Defense 1961).

James Joyce would have given the Cobol 61 report high marks for unreadability (but low marks for consistency). I did not understand a word of it. Fortunately, nobody asked me about it when I joined Regnecentralen's compiler group. To Milena, I wrote: "At last I found the right thing—a group working on advanced problems in computer languages."

⋆ ⋆ ⋆

Peter Naur was educated as an astronomer. He joined Regnecentralen in 1959 and became heavily involved in the international development of the programming language Algol 60.

The invention of programming languages is surely one of the most significant milestones in the history of computing. The science writer, Isaac Asimov (1976), put it this way:

> I strongly suspect that the advance of science or any branch of it depends upon the development of a simple and standardized language into which its concepts can be put. Only in this manner can one scientist understand another in his field.

Now, a programming language can only serve as a *standard* if it is concisely defined in a *language report.* In practice, however, most language definitions rely heavily on the reader's ability to fill in gaps and remove inconsistencies by educated guessing. I believe there is a reason for this sad state of affairs:

The task of writing a report that defines a programming language with complete clarity to its implementors and users may look deceptively easy to someone who hasn't done it before. But in reality it is one of the most difficult intellectual tasks in the field of programming.

The programming language Algol 60 introduced recursive procedures, block structure, scope rules, and type declarations in imperative programming. It was developed by an international committee that included John Backus (the developer of Fortran), Fritz Bauer and Klaus Samelson (who, together, developed the stack method of expression evaluation), John McCarthy (the inventor of LISP and one of the founding fathers of artificial intelligence), Alan Perlis (a pioneer of compiler development and the first chair of computer science at Carnegie-Mellon), and Peter Naur (whose contribution to Algol would be a landmark in computing).

Now, it is one thing to have a group of smart people sitting around a table discussing clever ideas. It is quite another thing for these people to describe their best ideas concisely in writing.

In 1959, at the initiative of Peter Naur, the *Algol Bulletin* was issued, which served as an international forum for discussing the development of the language. The bulletin was published by Regnecentralen.

For a meeting of the committee in January 1960, Naur prepared an unsolicited draft of the Algol report. Throughout the draft, he used a recent notation introduced by John Backus to define the syntax of *all possible* Algol programs! Naur's improvements of Backus's notation became known as *BNF notation* (or *Backus-Naur form*).

This was a huge step forward compared to the Fortran report, that defined the programming language by examples only. The problem with this informal method is illustrated by the old joke that "French is easy: 'horse' is *cheval*, 'dog' is *chien*,... and so on."

John Backus (1981) acknowledged Naur as the driving intellectual force behind the definition of Algol 60:

> Peter Naur's conduct of the Algol Bulletin and his incredible preparation for that [January 1960] meeting in which Algol was all written down already in his notebook—he changed it a little bit in accordance with the wishes of the committee, but it was that stuff that really made Algol 60 the language that it is, and it wouldn't have even come about, I don't think, had he not done that.

After seeing his draft, the committee asked Naur to be the editor of the official *Algol 60 report.*[1] Twelve years later, the Dutch computer scientist Edsger Dijkstra (1972) wrote:

> The famous Report on the Algorithmic Language Algol 60 is the fruit of a genuine effort to carry abstraction a vital step further and to define a programming language in an implementation-independent way. . . The report gloriously demonstrated the power of the formal method BNF, now fairly known as Backus-Naur-Form, and the power of carefully phrased English, at least when used by someone as brilliant as Peter Naur. I think that it is fair to say that only very few documents as short as this have had an equally profound influence on the computing community.

⋆ ⋆ ⋆

A computer program, written in a programming language, like Algol 60, is just another *text.* You can print it and edit it, but a computer cannot execute it as it stands. Before an Algol program can be executed, it must be "translated" into numeric machine code for a particular computer. The system program, that performs this translation, is called a *compiler.*

The small core memories of the mid 1960s made it impractical to write a program, as large as an Algol compiler, in a programming language, such as Fortran. Why? Because machine code generated by a Fortran compiler (or any other compiler) occupied significantly more memory than hand-written code. To use a small memory efficiently, a compiler had to be written in *assembly language*—a cryptic notation that required a programmer to specify each machine instruction in the code. A large program written in assembly language usually only made sense to the person who wrote it.

After the completion of the Algol 60 report, Regnecentralen's next challenge was to design Algol compilers for Dask and Gier. In this effort, Jørn Jensen's genius for machine coding would play a key role.

The American computer scientist, Alan Perlis (1981), left this impression of Jørn:

[1]P. Naur (ed.), J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauer, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger, Report on the algorithmic language Algol 60, *Communications of the ACM,* May 1960.

> When the [first Bendix G-20 computer] arrived at Carnegie it came with a full load of software: one binary load routine used by the engineers for testing memory. That was the sole extent of the software.
>
> How could one build a compiler [for Perlis's language, named Gate] quickly? We were fortunate at that time to have with us a visitor from Denmark, Jørn Jensen, who was with the Regnecentralen. Jørn was a magnificent programmer.
>
> Jensen sat at one desk; he was building the assembler. Arthur Evans sat at another desk; he was building the parser. Harold van Zoren sat at the third desk; he was building the code generator. All three were being defined simultaneously.
>
> The method of construction worked as follows. Jensen would decide that a certain construction ought to be in assembly language, and he would broadcast to the other two...When they decided...what changes would be required, working in good code-team fashion, they suspended and broadcast back to Jensen their proposals. Jensen would drop what he was doing and start another independent process. The amount of code that each wrote turned out to be of the order of about 2,000-3,000 machine language instructions. It turned out that at that size of code, such a technique worked magnificently. Each of the programmers could keep two to four processes in the air simultaneously, and changes progressed very fast. All three parts were completed at the same time. Jensen debugged the assembly language on the computer, simultaneously with the debugging of the parser and the code generator.

In June 1960, the Dutch computer scientists, Edsger Dijkstra and Jaap Zonneveld, completed the world's first Algol 60 compiler. At Regnecentralen, Jørn Jensen, Toke Jensen, Per Mondrup, and Peter Naur completed an Algol compiler for Dask in late 1961. This was followed by the much more elegant compiler for the Gier, which Dijkstra called "a masterpiece." To appreciate the achievements of the Dutch and Danish software pioneers, you need to know that there were no textbooks on compilers at the time. In 1964, Brian Randell and Lawford John Russell would publish the first book on *Algol 60 Implementation.*

Regnecentralen's second compiler implemented Algol 60 on the Gier computer with a memory of only 1K words and a drum of 12K words. Inspired by the Atlas computer at Manchester University in England, the run-time system implemented "demand paging" of compiled code—without any hardware support! The demand paging used the drum to simulate a "virtual memory" that was much larger than core memory, and, in most cases, almost as fast.

The programs and tables of the Gier compiler occupied about 5,200 memory words. The machine programming was done almost entirely by Jørn Jensen. It is amazing that a human being could comprehend such a large program written in unreadable assembly language!

Gier Algol (Naur 1963a) introduced several innovations in compiler technology, which I cannot go into here. The compiler was checked by using it to compile small Algol programs constructed specifically to ensure that each instruction of the compiler would be executed at least once. This systematic approach to testing made the compiler virtually error-free.

Regnecentralen planned to demonstrate Gier Algol at the IFIP Congress in Munich, Germany, in August 1962. As luck would have it, the Gier, that was shipped to Munich by truck, was shaken to pieces. Bech immediately borrowed another Gier, that had already been delivered to a customer, and sent it to Munich (again by truck).

The demonstration of Gier Algol in Munich was a success, and Regnecentralen got a contract with Siemens to develop a Cobol compiler for the Siemens 3003 computer.

This is where things stood, when I joined Regnecentralen's compiler group on March 1, 1963.

⋆ ⋆ ⋆

At Control Data Corporation in Minnesota, Seymour Craig designed the most powerful computers of the 1960s. The CDC 1604 was a large computer intended for scientific customers. It had been used to provide on-line quotations of stock prices from the New York Stock Exchange.

In 1963, Regnecentralen acquired a CDC 1604A computer for its main service center. Many years later, I had dinner with a former CDC executive, who remembered Niels Ivar Bech. He had never forgotten this Dane who traveled to the corporate headquarters in Minneapolis and tried to convince a roomful of rugged executives to give Regnecentralen a CDC 1604A computer, in return for a Cobol compiler—with Danish keywords! When that didn't work, Bech negotiated an agreement with the Danish shipbuilding yard, Burmeister & Wain, to buy a quarter of the computer time.

Regnecentralen was a lively place. On the afternoon of March 28, 1963, seventy people gathered in the small cafeteria at the Rialto center to celebrate the news of the CDC computer. An hour later, after liberal consumption of the strong beer, known as Easter brew, the room was very noisy. At dinner time, I followed this happy group of people on a tour of local restaurants. Around midnight, I walked home with a splitting headache.

In August 1963, the CDC machine was installed in the Rialto center. Bent Bagger and Henning Bernhard Hansen would replace the English keywords in the CDC Cobol compiler with Danish words. The creative part of this task was to suggest Danish terminology for data processing concepts. Once they had settled on the terminology, the actual replacement of keywords in the CDC compiler must have been straightforward, compared to the task of building a complete Cobol compiler from scratch (as the compiler group had to do).

Peter Naur was also interested in the development of computer terminology. One day he entered the office, in which Paul Lindgreen and I were working, and started talking about finding Danish words for computing. Since computers are not just number crunchers, he felt that "computer science" was a misnomer. He had decided to call the discipline *datalogy* (in Danish: "datalogi"). The architect of Gier, Bjarner Svejgaard, would later remark, that English may not be good Danish, but apparently a mixture of Latin and Greek is all right.

Naur was now trying to decide what a "computer" ought to be called. In a facetious mood, I suggested calling it a *datamaton* (Danish: "datamat"). I didn't tell Naur that my inspiration was a self-service laundry in Frederiksberg, named "laundromat" (Danish: "vascomat"). On the spot, Paul Lindgreen added the word *datamatics* (Danish: "datamatik") to denote automatic data processing (Naur 1966b). (Paul Lindgreen's recollection is that he was the one who asked me to suggest a Danish word for a computer.)

This is the only time, I have added a new word to my own language. For many years, everybody in Denmark called it a "datamat," until a new generation of PC users decided that smart people speak "Denglish." So, nobody says "datamat" anymore. It is now "computeren"—pronounced with a heavy Danish accent.

⋆ ⋆ ⋆

Before I could contribute anything to a Cobol compiler, I obviously needed to teach myself to write small computer programs.

One Sunday, my father wanted to invite me to the horse races, but, since I didn't care which horses won, I preferred to stay at home. The following Sunday, he proposed that we try our luck at the casino. After a pleasant dinner, we drove to the Marienlyst resort, north of Copenhagen.

At the roulette, my father followed a simple strategy for postponing the inevitable loss of his money, by slowly increasing his bets, until he reached the maximum amount permitted by the resort. When that happened, he would begin another round of bets, starting with the smallest possible bet. If he won anything, he would immediately start another round. When his total losses exceeded the modest amount, he was prepared to lose for the evening, he quit.

To my great surprise, I won a small amount of money that evening. The same thing happened to me on another occasion. Of course, something *must* be wrong, I thought—every roulette is designed to have only one winner in the long run: the owner! This reminds me of the classic exchange in the movie *Never give a sucker an even break* (1941): "Is this a game of chance?" asks the patsy. "Not the way I play it!" responds the card shark. So I decided to write an Algol program that would make the Gier act like a roulette.

Since Gier had no operating system, I signed up for a block of time, say 15 minutes, and had the machine all to myself. Most of the time, the computer was idle, while I input my program text from paper tape, typed user commands, or printed my output on the typewriter terminal.

For larger computers, the inefficiency of open shop operation inspired computer manufacturers to develop batch processing or multiprogramming systems. However, for the small Gier computer, the manual operation was not a serious problem.

At first, my computer roulette was not very random: it stopped twice as often on odd numbers as it did on even ones. But, after a while, it worked:

> The other day I invited my father to Regnecentralen, because I had succeeded in making the computer Gier play roulette. My father was very amused indeed. The computer is connected to a typewriter, and the roulette-program was made so, that the computer started the performance by asking: "How many games are you prepared to risk?" I typed: "10000 games." Then the computer started playing the many games, saying BZZ, BZZ… for nearly three minutes. And, finally it typed: "Sorry!—you have lost 45980 Dinars." [For the benefit of Milena, I replaced

> Danish kroner with Yugoslav dinars.] (Letter to Milena, June 6, 1963.)

Using elementary probability theory, it was easy to verify the results of my simulation. Once I understood the exact nature of the game, I lost all interest in playing roulette.

In my book, *Programming for Everyone in Java* (1999), I used roulette simulation as a beginner's exercise.

Milena felt that my simulation was a frivolous way of using an expensive computer. I explained that it was an example of the Monte Carlo Method of computing, named after the famous resort town in Monaco, which has the world's oldest casino.

Thirty years later, I used simulation on a supercomputer with 48 processors to find the shortest tour through 100 cities in two minutes (Brinch Hansen 1995). This is obviously a problem of some practical importance. (It is also a much harder problem than roulette simulation.)

⋆ ⋆ ⋆

The programming language Cobol was designed about the same time as Algol. At the 1978 conference on the History of Programming Languages (HOPL), Joe Wegstein, National Bureau of Standards, commented (Cobol Discussion 1981):

> The Cobol Committee had these people from various manufacturers who had a lot of vested interest, and were very intense about that sort of thing in connection with everything being done. Whereas, the Algol Committee had a bunch of senior professors of Europe and an oddball collection from the U.S., and—all very temperamental and intense about mathematical aspects of programming.

Now, Algol was designed for numeric computations. The only data structures supported by the language were tables ("arrays") of numbers.

Cobol, on the other hand, was designed for business data processing. The most important contribution of Cobol was the introduction of data records and sequential files, which were needed to process data on punched cards and magnetic tapes.

Ten years later, Niklaus Wirth combined both forms of computing by including records and files in his Algol-like language, Pascal.

In his History of Modern Computing, Paul Ceruzzi (2003) writes:

> Cobol became one of the first languages to be standardized to a point where the same program could run on different computers from different vendors and produce the same results.

Alas, this worthy goal was *not* reached. After completing the Siemens Cobol compiler, Regnecentralen concluded that:

> The major problem of implementation turned out to be the numerous definition problems created by the vagueness of the official Cobol report. (Brinch Hansen 1966)

Compared to Algol 60, Cobol was poorly defined. In places, where the Cobol report was incomprehensible, Regnecentralen's compiler group had to *guess* what the intention of the Cobol committee *might* have been. More than likely, other compiler groups interpreted the report differently and implemented incompatible variants of the language.

A peculiar feature of Cobol was its attempt to replace well-known algebraic notation by verbose English: whereas you might write a/b in Algol, this became DIVIDE A BY B in Cobol (or even DIVIDE B INTO A). This was supposed to make it easier for managers to read programs.

At the HOPL conference, the Cobol notation provoked the following exchanges of questions and answers:

> *Question*: Did the participants in the original Cobol development sincerely believe that the use of an English-like language would enable nonprogrammers (e.g. managers) to understand programs.
>
> *Answer*: Yes. We sincerely believed managers would be able to read the programs and that more people would find them easier to write.
>
> *Question*: Did the Cobol committee seriously believe that the users could not handle grade school operators $+, -, \times, /$?
>
> *Answer*: Quite seriously, there was a strong sentiment...that the users did not want to use algebraic symbols in the normal course of writing an arithmetic expression.

How can one explain that Cobol remained the most widely used programming language on the planet for decades? Well, in 1960, the U.S. Department of Defense announced that it would only use computers that supported Cobol. That guaranteed the commercial success of Cobol—independent of its merits!

Needless to say, the government could not dictate the opinions of computer scientists:

> *Question*: [Cobol] continues to be viewed with great disdain, as is data processing in general, by many computer scientists. It is rarely taught in "prestigious" computer science departments, where it is still regarded as an abomination. Have you any comments?
>
> *Answer*: I think Cobol ought to be taught because there are concepts in there which are important and which are useful, and business data processing has a large significant, intellectual component. But most of the senior key computer science people don't agree.

This was the programming language that Regnecentralen's compiler group would be responsible for implementing for Siemens in Munich.

⋆ ⋆ ⋆

Peter Naur and Jørn Jensen worked so closely together that they hardly needed to say anything to solve a problem. I remember a discussion where Peter was writing something on the blackboard, when Jørn suddenly said "but Peter ..." and immediately was interrupted with the reply "yes, of course, Jørn." I swear that nothing else was said. It made quite an impression on me, especially since I didn't even know what the discussion was about in the first place.

As an electronic engineer, I was used to circuit diagrams showing resistors, capacitors, and transistors. What the compiler guys did was completely different. On the blackboard, they would illustrate their ideas with small Algol 60 fragments. Since Algol was not a natural language for thinking about data structures, they would also draw complicated pictures of tables linked together in mysterious ways by arrows.

However, in truth, the Cobol compiler was progressing very slowly (if at all). Naur and Jensen had already finished their second compiler for the elegant Algol language. Now they had to do it again with a far less attractive language. It seemed to me that their hearts were not in it.

After three months, I began to catch on. On May 23, 1963, I wrote to Milena:

> We have common meetings sometimes on Fridays, just to coordinate things and settle issues of doubt. You see, the normal situation is that everyone gets a small part of the project to work on. First, everyone will work enthusiastically for a few weeks or so, independent of the others, but gradually the tempo slows down for a lot of psychological reasons—some details cannot be solved, before you know what the others are doing, and other problems you simply close your eyes to (and put them in a drawer).
>
> So every now and then, Peter Naur calls for a meeting to make us face the problems. You can't imagine, how I enjoy the atmosphere of a group of people, who have to convince each other, defend their views, and reach decisions.
>
> Sometimes I get permission to work at home for several days—mainly, when I have to write a report on what I have been doing lately. (There are too many distracting factors at work: noise from the street below, and the temptation to chat with the others.) [Throughout my professional career, I would continue to do all my writing at home.]
>
> Well, the latest crazy and wonderful idea is, that the whole department of some ten engineers is going to work "at home" for one week to jump-start the project. From Monday, the 21st of October, until Saturday, the 26th, we are moving to a small inn [Mogenstrup Kro] in the southern part of Zealand, far away from any big, noisy town. [Zealand, on which Copenhagen is also situated, is the largest island in Denmark.] Each of us will have his own small room to work in, and often we will gather for a common discussion. In the evenings, we can walk in the woods and get to know each other outside the office. I find it a splendid idea.

The Mogenstrup meeting clarified many things: The Cobol compiler would be divided into ten phases (known as "passes"). Since the Siemens computer had no drum or disk, the compiler would use three magnetic tapes. The compiler would be input, one pass at a time, from a system tape. The other two tapes would be used as scratch tapes during compilation.

Pass 1 would input a Cobol program from punched cards, perform a partial compilation and output intermediate code on one of the scratch tapes. Pass 2 would then input the intermediate code from tape, perform another partial compilation, and output slightly more detailed code on the other scratch tape, and so on. In this way, the compiler passes would be loaded, one at a time, from the system tape, while the compiled code would move back and forth between the scratch tapes, being gradually refined. The last pass would leave final code on a scratch tape, from which it could be loaded and executed.

Since every pass performed a single scan of the original Cobol program (or the intermediate code), this scheme was known as multipass compilation. Multipass compilation made it possible to use a compiler that was much larger than the available core memory. The compiler group had already used multipass compilation of Algol programs on Dask and Gier.

Peter Naur and Jørn Jensen would be responsible for the overall design of the Cobol compiler. However, in reality, Peter Villemoes became the project leader. The design, programming, and testing of the individual passes would be done by Sven Eriksen, Roger House, Jørn Jensen, Peter Kraft, Paul Lindgreen, Ole Riis, Peter Villemoes, and me. Naur's cousin, Berta Kiær, would be our secretary.

Back in the Rialto center, my first task was to program the parser, a compiler phase that would check if the syntax of a Cobol program (that is, the sequence of programming symbols) was correct.

Instead of having a few basic constituents, that could be used in many contexts (as in Algol 60), Cobol 61 consisted of a large number of unrelated clauses, each of which required a special piece of code in each pass. This complexity made it impractical to perform syntax analysis the same way it was done in Gier Algol (by simulating a so-called "finite state machine").

I invented a different method of representing the Cobol syntax by linked lists of symbols. The parser would input a Cobol program, one symbol at a time, and use the linked lists to check the syntax. The parser would also erase all clauses with illegal syntax. This was my first (modest) invention in system programming.

When it was finished, the machine code and fixed tables of the syntax analyser occupied about 5,000 lines in assembly language—a fairly hefty program for a beginner. Other members of the group programmed compiler passes that were more complicated than the parser. However, after forty years, I no longer remember exactly who did what.

As I mentioned earlier, assembly language is extremely difficult to understand. Even after a short vacation, you may find it difficult to remember the meaning of your own assembly language program. I solved the problem of program documentation by adopting a brilliant method used to document the Gier Algol compiler: I divided each program page into two halves. The left half defined the program in assembly language, while the right half defined the same program in Algol 60. The assembler treated the Algol 60 statements as comments to be ignored. However, these comments simplified my job tremendously, since it was fairly easy to determine if a sequence of assembly language instructions implemented an Algol 60 statement correctly.

Now, if a program and its description are two separate documents, a programmer may not always remember to update the description, every time the program is modified. However, since the documentation method combined an assembly language program and its definition in Algol 60 in one document, it became natural for me to update both parts simultaneously.

My yearly salary of 22,900 kroner (about $3,300) was not a lot of money in 1963. So at the end of my first year at Regnecentralen, I asked Jørn for a raise. I told him, that I liked my job and would hate to give it up at this point. On the other hand, I felt obliged to take the consequences of my request—otherwise, how could I expect him to take it seriously? So I asked him to reach a decision within a fortnight. If I got no raise, I would find myself another job. Jørn smiled and said: "This is a viewpoint I can only respect. I will talk to Bech and tell you, whether you will get a raise or lose your job."

On April 5, 1964, I wrote to Milena: "Don't be nervous: I got my raise the following day."

A month later, serious testing of the Cobol compiler began. The compiler passes were tested, one at a time, in their natural sequence (pass 1 was debugged first, then pass 2, and so on). The compiler was tested by letting it compile small test programs written in Cobol—a method borrowed from Gier Algol.

The parser was the second compiler pass. When I began my tests, pass 1 had already been tested and was therefore able to compile my test programs into correct input for pass 2. In each test run, the compiler printed the test program, that was being compiled, followed by the output produced by pass 2. By comparing the test program and the corresponding output from the parser, it was easy to see which symbols it handled incorrectly. I would then correct the parser and repeat the same test case, until it worked.

You must remember, that the compiler was being programmed in one country and tested in another. In Munich, Siemens was still testing various aspects of the hardware. The machine was in so much demand, that we also had to use it in the evenings and during weekends, when the Germans went home. Let me tell you, walking towards Siemens on Hofmannstrasse 51, at 4 in the afternoon, while 10,000 workers walked the other way, was an experience!

With our limited access to the computer, there was no opportunity to experiment with incomplete programs. We took turns arriving in Munich with a complete compiler pass and a set of test programs, that had already been punched on cards and proofread in Copenhagen. The compiler passes were so carefully planned that few (if any of them) had logical flaws. The main purpose of our systematic testing was to remove the inevitable clerical errors.

I continued to use this method of program development for forty years. In my experience the combination of careful design, proof reading, and systematic testing can make programs more reliable than the hardware they run on.

Of course, this glib description does not reveal my early frustration with the parser, when nothing worked, and nothing was printed! The only thing I could do in that situation, was to read the beginning of my program, instruction by instruction, until I figured out why it produced no output.

From then on, my testing went as planned:

> May 26, 1964
>
> Dear Father—My program works! Believe me, it is an experience, finally to work on a large computer. I have been to Yugoslavia twice on my weekends.
>
> Soberly yours, Per

Although I now 'knew' how a computer worked, I still found it unbelievable that a machine would follow thousands of instructions I had written in pencil. It *is* pure magic that human beings have learned to construct computational processes by combining electricity, transistors, circuits, computer architecture, assemblers, compilers, operating systems, and user programs. If you don't share that feeling of awe, you haven't really understood the miracle of computing.

In Munich, we stayed at Hotel Daniel, Karlsplatz 15, close to the main railroad station. Here is a letter to Milena, mailed from the hotel on June 8, 1964:

> You are quite right, we had troubles with the program. In such a large program, there are always bound to be some errors. In fact, we are only here to detect and correct such 'bugs' and it has been quite a tricky task. But it works now, and I think I can fly back to Copenhagen by the middle of this week. However, first I must have some talks with the Germans about my next program.
>
> We are running to and from the computer all day long, from 9 in the morning till 7 in the evening. In the begining, the Germans were a bit puzzled by our unsocial behavior: we never spent much time chatting with them and would often criticize the way they had designed their computer.
>
> Last week, however, we had occasion to repair this impression. We were invited to join them in their monthly 'lab evening' ('Labor-Abend'). This is an evening where they go with their spouses to a restaurant and talk about anything but their work. When we arrived at the outdoor restaurant in Schwabing, the Germans had already been drinking for three hours and were in high spirits. One shy fellow was making speeches for all the girls at Siemens. That evening, artists exhibited their paintings by candlelight, while a group of teenagers played guitar and sang the blues. An endless stream of people crowded the pavement and the outdoor restaurants.
>
> We had several bottles of a not-so-famous white wine, labeled 'No 1a,' and engaged the Germans in the conversation they had missed. The evening ended in some strange restaurant at 3 a.m. The next day, I felt like a dying man in the computer room.

⋆ ⋆ ⋆

We were young and cocky and not always as polite to our German hosts, as we should have been.

The Siemens 3003 had a hardware feature that was meant to prevent its operating system from being destroyed by incorrect (or malicious) user programs. The operating system resided in a fixed part of memory. When you flipped a switch on the computer, it became impossible to change any memory location within the protected area. This certainly guaranteed that the operating system code could not be changed during program execution.

However, the computer architect had overlooked one thing: an operating system must be able to record various data about running programs to function correctly. Since it was impossible to update memory locations in the protected area, the operating system had to keep its variables in unprotected memory locations, where they were completely at the mercy of user programs. If programs made arbitrary changes to these locations, it would soon crash the operating system.

In short, the so-called "memory protection" was a hacker's dream. The members of the compiler group knew this. When the Germans demonstrated the machine for us, it was a favorite joke of our American programmer, Roger House, to say: "Excuse me, you forgot to turn the protection switch on!"

⋆ ⋆ ⋆

Once in a while, the computer broke down, leaving us with a perfect excuse to relax:

> Last friday, our computer broke down completely, so we have had a quiet weekend without any work whatsoever. It is very hot in Munich, so yesterday we took a trip to the countryside to a small mountain lake, where we ate a tremendous dinner. Afterwards, we rented a rowing boat and drifted around the lake. Our 'real' work has been delayed quite a bit by repeated computer failures, so I will probably have to stay here at least another week. Not that I mind, since time passes easily in a city like Munich: we go to the theater and concerts, and eat mostly in 'foreign' restaurants—Chinese, Hungarian, Bosnian, Russian, Spanish, and Italian. (Letter to my father, June 11, 1964)

Breakfast was included in our hotel bill, and Siemens gave us a free lunch. Since Regnecentralen allowed us to spend a fixed daily amount for meals, we had plenty left for sumptuous dinners around town. We became connoisseurs of Munich's restaurant scene, and knew, for example, which of the two Russian restaurants was the best one. I soon learned that putting on weight is much easier than losing it again (which I never did).

⋆ ⋆ ⋆

On August, 21, 1964, I presented my first scientific paper at the NordSAM conference in Stockholm, Sweden. It described a method that made the evaluation of logical expressions slightly faster during the execution of a compiled Cobol program.

A logical expression of the form [IF] A GREATER B AND LESS C ..., is evaluated in three steps: (1) First, check if it is true (or false) that A is greater than B; (2) Then, check if it is it is true (or false) that A is less than C; (3) Finally, check if both conditions turned out to be true (or not). Since the GREATER and LESS relations are evaluated before the AND relation, GREATER and LESS are said to be operators of *higher priority* than AND.

However, if it turns out in step 1 that A is not greater than B, then step 2 is superfluous, and can be skipped. That was the whole idea behind my code optimization.

To me, this was an elegant compilation technique. But, looking back, I don't think it served any practical purpose. Before you go to the trouble of optimizing compiled code, you should conduct an experiment to find out, if it has any measurable effect. Otherwise, you are just increasing the size of your compiler for no good reason.

I cannot imagine that the efficiency of business data processing will ever depend on the speed at which a computer evaluates logical expressions. However, this minor optimization (which I do not claim to have invented) would later be included as a language feature in C and Java.

Today, it is still common for programmers to confidently recommend a method, because "it is faster" than another one—without offering any performance measurements to document the magnitude of the improvement.

Anyhow, here I was at my first computer conference lecturing in English to an international audience that included Peter Naur and Edsger Dijkstra. I was very encouraged to see both of them smiling broadly during my presentation. Afterwards, I discovered why: In my talk, I constantly said "the priority of this operator is higher than the priority of that one." Since English was not my native language, I mispronounced the word "higher" as "hi-ger" (with a hard "g" as in "good"). Everytime I did that, Naur and Dijkstra smiled.

⋆ ⋆ ⋆

My most difficult programming task was to write the input/output procedures for files stored on punched cards, magnetic tapes, and line printer forms. This file system would be used during the execution of compiled Cobol programs.

Each tape station was about the size of a small closet. While a tape was being read or written, it moved from one reel across a magnetic head to another reel. To prevent the fast moving tape from breaking during frequent starts and stops, it also moved through two vacuum chambers, which held enough loose tape to absorb the forces of acceleration and deceleration.

The tape stations on the Siemens computer were rather unreliable. Jørn Jensen witnessed a faulty tape station jam a tape by rewinding both reels at the same time! Dust particles on the tape caused transient input/output errors. I handled these by transferring the same block of data again. This could, of course, have been done simply by backspacing over the last block and reading (or writing) it again. Instead I backspaced ten blocks—enough to move the bad block into the nearest vacuum chamber, where the air current would blow the dust off the tape. I would then upspace nine blocks and transfer the same block again.

A more serious problem was the permanent errors caused by spots of missing oxide on the tape. The only thing you could do with a bad spot was to ignore it and write the same block again after the spot. To facilitate error recovery, each block was output with a block number and a block length. A block that did not have the correct number and expected length during input was assumed to be a bad spot and was skipped.

When Jørn Jensen first told Siemens about the need to extend data blocks on user tapes with two additional words, they would not agree to this. And who could blame them. It would be a major problem for Siemens to ask its customers to adopt a new tape format, that would make their existing tapes unreadable.

I then asked Jørn to tell the Germans that Regnecentralen could not be responsible for the reliability of tape input/output, unless they agreed to our proposal. That did it! They agreed, and the tapes worked fine.

Peter Villemoes had developed the techniques for dealing with tape errors during compilation. However, the more general file system I was writing for the execution of Cobol programs, posed additional challenges.

Compiled Cobol programs were supposed to run in a core memory of 8K words only. When a program opened a file, it was assigned a buffer space in memory. To make the best use of the small memory, the buffer space was reclaimed, when the file was closed. Over time, the memory ended up being full of active buffers separated by gaps of unused space left behind by closed files. If a buffer space could not be found for a new file, the gaps between the buffers were closed by moving all the buffers to one end of the memory.

The trickiest feature was probably the ability to restart a Cobol program from a previous point of execution, after a hardware failure. At regular intervals, my input/output procedures would stop the running program briefly and output restart data on a tape. When the hardware had been fixed, the most recent restart data were used to instruct the operator to mount the same tapes, one at a time. The tapes would then automatically move to the same spots, where they had been, when the restart data were written, and the computation would resume, as if nothing had happened. This was fun—and awesome—to watch!

I regard the run-time filing system as my graduation project from the compiler group. I now understand that it was really a small operating system, I had programmed. However, in the mid 1960s, the dividing line between language implementation and operating systems was still not clearly understood.

⋆ ⋆ ⋆

After a total effort of 15 man-years, Regnecentralen delivered a complete Cobol implementation of 39,000 instructions to Siemens in July 1965. We had used about 600 hours of computer time to assemble and test the system. In human time, it took about 45 minutes to program each instruction and less than 1 minute to test it.

The Siemens Cobol compiler was eight times faster than the fastest American compiler evaluated by the Bureau of Ships (Siegel 1962). After a basic input/output time of 45 sec, our compiler translated a Cobol program at the rate of 250 cards per minute, generating final machine code.

When the compiler was completed, Sven Eriksen joined Siemens in Munich and became responsible for maintaining the compiler. He was incredibly well organized. We mailed every compiler change to him as a deck of punched cards with a separate test program to verify that the correction worked. He kept the punched cards of our original Cobol implementation, and all subsequent modifications, in chronological order in a card filing cabinet.

About a year after the delivery of the Cobol system, a user reported that my filing system did not work correctly. Since Siemens had modified the Cobol system in places, it could have been a nightmare for me to travel to Munich and determine what the error was and who was responsible for correcting it.

Instead I asked Eriksen to reestablish the compiler exactly as it was a year ago. He was able to do that, and demonstrate that the customer's program worked under the original file system, we had delivered. That got me off the hook and left Siemens with the problem of figuring out, what they had done wrong after the delivery.

I wonder, how many software developers today treat system updates in the same professional manner?

⋆ ⋆ ⋆

Peter Naur encouraged Roger House, whose native language was English, to write a paper about the Cobol compiler. Since this idea got nowhere, I wrote the paper with helpful comments from Roger. It was published in the Scandinavian journal BIT in 1966.

Today, few people (if any) have access to the Cobol compiler for the Siemens 3003 computer. But anyone, who is interested, can still read about it in BIT. In the long run, it seems to me, the most important aspect of programming is the description of interesting ideas in readable papers. The programs themselves are merely useful by-products of this effort. Besides intellect, the most valuable asset of a programmer is the ability to write clearly! Needless to say, this viewpoint is not popular among students, who prefer free-style "coding" without the burden of documentation.

⋆ ⋆ ⋆

Before joining Regnecentralen in 1963, I met Milena in Slovenia. During my trips to Munich over the next two years, I visited her ten times. This was the most romantic episode in my life. On March 27, 1965, we were married in the townhall of Ljubljana.

The compiler group sent us a telegram with amusing comments on the wisdom of marrying (Fig. 3.1).

27 3 65

MILENA AND PER BRINCH HANSEN
HOTEL BELVEDERE
IZOLA ISTRIA

LUCKY TEST BERTA STOP
NO PROTEST RIIS STOP
POOR YOU BUT YET GERDA KRAFT STOP
LUCKY YOU LINDGREEN STOP
AND ALL THAT JAZZ DIANE STOP
MONDRUP TOKE JOHANSEN STOP
BELIEVE US IT IS NOT TOO BAD ROGER AND JEANNE STOP
A CHALLENGE BUT WORTH IT NAUR STOP
A HUGE GRATULATION FROM THE ABSENT GUYS

Figure 3.1 Wedding telegram from the compiler group.

4

YOUNG MAN IN A HURRY 1966–70

Naur's vision of datalogy – Architect of the RC 4000 computer – Programming a real-time system – Working with Henning Isaksson, Peter Kraft, and Charles Simonyi – Edsger Dijkstra's influence – Head of software development – Risking my future at Hotel Marina – The RC 4000 multiprogramming system – I meet Edsger Dijkstra, Niklaus Wirth, and Tony Hoare – The genius of Niels Ivar Bech.

I was fortunate to start my programming career at Regnecentralen. For almost three years, I had participated in the design, programming, testing, and documentation of a large compiler. I knew it was time to leave the compiler group and try something else. Niels Ivar Bech had something in mind—but I had other ideas.

In a brilliant paper, Peter Naur (1966a) viewed compilation as a general data processing problem that involves more fundamental programming methods, which he felt should be taught as part of a core of computer science. At a time, when compiler contruction was still regarded as a fundamental subject in its own right, Naur's insight was ahead of its time.

In 1966, Bech invited me to join a working group consisting of Peter Naur, Christian Gram, Henning Bernhard Hansen, Jens Hald, and Alan Wessel. Their goal was to develop a systematic text on datalogy (as Naur called it). This was an exciting idea—but it was not mine. My answer to Bech was honest: "Thank you, but I prefer to wait until I am writing my own book."

Naur (1968) proceeded to outline a complete core course on computer science based on fundamental principles. His vision of computer science was published in the same year as Donald Knuth's famous Volume 1 of *The Art of Computer Programming* (1968).

For various reasons, the working group never finished its ambitious project (although parts of it was published in Danish). A short English version of the complete text was published in 1974, with Peter Naur as the only author.

⋆ ⋆ ⋆

After my three-year apprenticeship at Regnecentralen, Milena and I were talking about living abroad for a while. After my graduation, IBM had encouraged me to keep in touch, in case I would be interested, after some years, in working at one of their labs in Sweden, England, or the United States. That sounded promising after my enjoyable experience at the IBM Hursley Laboratory in the summer of 1961.

Henning Isaksson had asked Niels Ivar Bech for a systems programmer for quite some time. Since I was thinking of leaving anyhow, Bech suggested that I might join Isaksson's hardware group in Valby.

In the Polish city of Pulawy, the Danish engineering company, Haldor Topsøe, was designing the largest fertilizer plant in Europe. The company signed a contract with Regnecentralen to deliver a small computer with data logging software. The system was supposed to demonstrate that the plant satisfied the specifications guaranteed by Topsøe, and would also help management with maintenance of the plant.

From the beginning, Henning and I viewed the Pulawy-project as an opportunity to develop Regnecentralen's third computer. However, Bech did not see it that way. He strongly encouraged Henning to use the recent CDC 1700 computer. Bech was not known for cautious decisions. On this occasion, he may have been influenced by Regnecentralen's reorganization in 1964 as a limited company with shareholders.

Henning finally said: "Look, if we use CDC software in our process control programs, our customers will expect us to help them with the problems of software, that we have not developed." This argument persuaded Bech that we would be better off developing our own process control computer. He remembered only too well how Regnecentralen had been forced to use Fortran on its large CDC 1604 computer, because the Algol compiler from Control Data turned out to be unreliable (Isaksson 1976). In 1978, I gave the same answer, when the chairman of Mostek Corporation asked me, if I thought it would be a good idea to put an IBM mainframe computer on a chip.

Now, if something has a name, it obviously must exist. My favorite example is the medical term "essential hypertension." With a name like that, who could doubt that medical doctors know, what they are talking about. On the contrary, if doctors don't have a clue about the cause of hypertension, they call it "essential." So, although Gier's successor was still on the drawing board, we needed a name for it. The natural choice would have been calling it the RC 3. However, Regnecentralen was already marketing the RC 3000, a special-purpose device for data conversion. So I suggested calling the new computer the RC 4000, since "who would buy an RC 3 for a million kroner, when you can buy an RC 3000 for a lot less?" So RC 4000 it was.

⋆ ⋆ ⋆

Henning was an efficient manager and very pleasant to work for. At my request, he persuaded Bech to let Peter Kraft join us from the compiler group. Peter was an experienced programmer who had learned his craft during the Gier Algol project. Of average height, with a receding hairline and large, dark-rimmed glasses, his face was usually lit up by a brilliant smile. We worked well together, perhaps because he provided a calming influence on my own forceful personality. The hardware engineer, Villy Toft, wrote this portrait of Peter:

> Cheerful, humourous and positive, he had a constructive approach to problem solving, unhampered by potential problems. A gifted problem solver with an analytical talent, who showed no signs of stress, he worked quietly and calmly with other members of the group as a catalyst and inspirator. His modesty gave him a tendency to downplay his own achievements, even though others valued them highly.

Most of us should be so lucky to be remembered like that!

When it became obvious that Peter and I would need another programmer, Bech sent us a Hungarian teenager! At age 14, Charles Simonyi wrote his first program for a huge Russian Ural II computer in Budapest, where his father was professor of electrical engineering. He desperately wanted to leave his communist country and emigrate to the United States. During a demonstration of the Gier in Budapest, Charles met Bech, who offered him a one-year job in Denmark. After completing high school, he left Hungary in the summer of 1966 and came to Denmark without an education. He was 17 years old, had a Beatles haircut and spoke limited English with a Hungarian accent. When he heard a Caravelle jet outside, he would open the window, stick his head out, and shout: "Vonderful Caravelle." At the end of the Pulawy project, Charles had saved enough money to go to California. At Berkeley, he paid for tuition by working as a programmer in the university computing center. After graduating in 1972, he joined Xerox Palo Alto Research Center, where his former professor, Butler Lampson, and others were pioneering personal computing on Alto personal computers with graphic interfaces, mice, laser printers, and ethernets. For the Alto, Butler and Charles designed Bravo, the first graphic text editor. When Simonyi joined Microsoft in 1981, he brought his knowledge of Bravo and turned it into Microsoft Word. Today, Charles is a member of the U.S. National Academy of Engineering. He is also very rich.

⋆ ⋆ ⋆

Before Simonyi joined us, Peter Kraft and I had already defined the architecture of the RC 4000, and the prototype for Pulawy was being assembled in a small lab right below our office.

With 1K words of memory only (about 5K bytes), the Gier computer had been designed for clever handcoding of compact machine code. Experienced programmers had been known to stare for days at small fragments of the Gier Algol compiler, trying to figure out what Jørn Jensen's code was doing.

However, by 1965, it seemed safe to predict that, by the end of the decade, most programs would be written in high-level languages for computers with large memories. Most machine code would then be generated by compilers (and not by programmers).

When programmers write compact code, they take advantage of all kinds of programming tricks. It is not so easy to write a compiler that does the same kind of optimization. From a compiler writer's point of view, the ideal computer should have a systematic instruction set that makes code generation straightforward (but not necessarily optimal). Faster processors and larger memories would soon make this approach practical.

In defining the instruction set of the RC 4000, our goal was to simplify program compilation (instead of hand-coding). Whereas most computers had several instruction formats, the RC 4000 would have only one. This meant that any instruction could use the full set of addressing modes.

Every instruction defined an operation between a memory location and a register. However, by addressing the registers as the first four words of memory, you could operate on any pair of registers. It was even possible to jump to a register and execute its value as an instruction. This feature was used to autoload an initial program into an empty memory.

The basic addressing modes were extended by an instruction, called Modify Next Address, which used its own address to increment the address part of the next instruction. (The operation changed only the effective address of the next instruction but left its displacement field unchanged.) This instruction made it possible to use any memory location as an index register. A sequence of these instructions could modify an instruction with the sum of several registers or perform multiple-levels of indirect adressing.

Since the hardware and software engineers lived in different worlds, I faced the problem of describing the architecture in a formal language that made sense to both groups. Although pictures with informal explanations were helpful, they could not always convey the finer details accurately.

I settled the issue by using Algol 60 as a hardware definition language. Before the computer was built, I wrote a reference manual that defined the instruction set completely by an Algol program. This program simulated the execution of RC 4000 machine code using simple and indexed variables to represent hardware registers and memory locations. It defined how operands were addressed in memory, and how arithmetic results were computed, bit by bit, with overflow detection. It also defined the instruction fetch cycle, the memory protection system, the interrupt system, and the function of the power-on and reset keys.

Peter Kraft still remembers that if we discussed some aspect of the architecture, which at first looked like a detail only, I would often go home and work throughout the night, revising and rewriting the description of the architecture one more time.

Inspired by my use of Algol 60 as a hardware definition language, one of Isaksson's engineers, Allan Giese, extended Algol and used it to describe the internal structure of the RC 4000 (the microprogram).

At some point, Niels Ivar Bech called a meeting with Christian Gram, Jørn Jensen, Peter Naur, Bjarner Svejgaard, Henning Isaksson and me to discuss the proposed RC 4000 architecture. This was a valuable opportunity to benefit from the comments of Regnecentralen's senior people. It was also a "final exam" I had to pass, before Isaksson would get the green light to build the machine.

In preparation for the meeting, I distributed a detailed draft of the RC 4000 reference manual. At the meeting, the architecture was accepted without much discussion. I may have learned something from Naur's performance in the Algol 60 committee. It may also have helped that I had seventeen years of experience in writing essays.

This was a group of people who (quite correctly) insisted on concise writing. I still chuckle when I remember what Christian Gram said at that meeting, almost forty years ago. In my draft, I wrote, "A special autoload instruction is used for initial program loading." Christian's response was: "All instructions are special." Bingo!

In April 1967, Regnecentralen published the first official edition of my *RC 4000 Computer Reference Manual*. Two years later, Christian Gram extended the manual with complete definitions of floating-point arithmetic (Brinch Hansen 1969b). At that point, it was no doubt the only reference manual in the world that made it possible for programmers to predict the result, bit by bit, of dividing two non-normalized floating-point numbers!

⋆ ⋆ ⋆

I don't mean to drag you through the details of fertilizer production and real-time programming. But I would like you to understand the gist of what we did, since this was my first encounter with a major revolution in computer programming that became the focus of my professional work for thirty years: *concurrent* or *parallel programming*—the art of making a computer execute several programs at the same time.

The three units of the Pulawy plant produced ammonia, nitric acid, and ammonium nitrate. The plant was operated manually under supervision of the RC 4000 prototype, which had a core memory of 4K words (about 12K bytes), but no drum or disk.

John Saietz, a chemical engineer from Haldor Topsøe, worked with Peter Kraft to specify the process control tasks:

The RC 4000 would count the production of fertilizer and the consumption of electricity by sampling digital signals from bag-filling devices and kilowatt-hour meters every second. Every five minutes, the computer would input about 350 analog measurements of pressures, temperatures, and material flow rates, checking that they remained within certain alarm limits. Every hour, two snapshots of the plant operation would be printed, listing some 550 analog measurements. When one shift of workers was replaced by another, the machine would print a report of material balances showing the production and energy consumption over the past eight hours.

Topsøe also wanted the operators to be able to make the computer perform some tasks more frequently, when units of the plant were being restarted after repair.

It was now up to us to translate the chemical engineer's specification into real-time control software.

Ignoring for the moment the engineering details of fertilizer production, a programmer might summarize the project as follows: a small computer has to perform a fixed number of cyclical tasks with frequencies determined by plant operators. These tasks must be able to share data tables and input/output devices (including an analog/digital converter and various printing devices).

It seemed natural to write a separate program for each control task. However, we could not expect to fulfill the real-time requirements by executing one task program at a time: two task programs might need to be started at the same time, and the time required for a single execution of a task program might also be longer than the time interval between successive executions of other task programs.

Ideally, we would have liked to be able to run task programs in parallel. However, since the computer could only execute one instruction at a time, we had to settle for a pseudo-parallel mode of execution, known as *multiprogramming.*

Four hundred times a second, an electronic timer interrupted the running task program and transfered control to a scheduling program, known as the *monitor.* The monitor then resumed the execution of another task program for 2.5 msec, and so on. In this way, the computer was shared cyclically among the active tasks.

The use of clock interrupts to simulate concurrent execution of programs was pionered on the Atlas computer by Tom Kilburn and David Howarth (1961). Multiprogramming is still the principle behind time-sharing operating systems (such as Unix or Windows).

Switching a single computer among multiple tasks is similar to a waiter (the computer) serving several tables (the tasks). As long as the waiter only spends a fraction of his time at each table, most of the customers will be able to eat at the same time.

In the Pulawy system, each task used only a few percent of the computer time. The rest of the time, the tasks would wait for slow peripheral devices. As soon as a task started waiting for the completion of input/output, the monitor switched to another task. So although the tasks never executed instructions simultaneously, the typewriters would nevertheless print at the same time.

In our real-time system, we needed some form of *synchronization* to prevent several tasks from using the same printer (or data table) at the same time. But what kind of synchronization?

The early multiprogramming systems were programmed in assembly language without any conceptual foundation. The slightest programming mistake could make these systems behave in a completely erratic manner that made program testing nearly impossible.

A common synchronization technique at the time was to suspend a task in a queue until it was resumed by another task. The trouble was that resumption had no effect if the queue was empty. This happened if resumption was attempted *before* a task was suspended. (This pitfall reminds me of a mailman who throws away your letters if you are not at home when he attempts to deliver them!)

This mechanism was unreliable because it made a seemingly innocent assumption about the relative timing of parallel events: A task must *never* attempt to resume another task that is not suspended.

Since the Pulawy operators could change the frequencies of individual tasks (and even stop some of them indefinitely), we could not make any assumptions about the relative (or absolute) speeds of the tasks. Time-dependent event queues would have been a disastrous choice for our real-time system. Around 1965 IBM's PL/I language included event queues of this kind. Surprisingly, the suspend and resume primitives are also included in the recent Java language.

Regnecentralen had no experience with multiprogramming. Fortunately, Edsger Dijkstra was kind enough to send me a copy of his 1965 monograph *Cooperating Sequential Processes*, with a personal dedication: "Especially made for graceful reading!" (I still have it.) One of the great works in computer programming, this masterpiece laid the conceptual foundation for concurrent programming.

It began by making the crucial assumption about *speed independence*:

> We have stipulated that processes should be connected loosely; by this we mean that apart from the (rare) moments of explicit intercommunication, the individual processes themselves are to be regarded as completely independent of each other. In particular, we disallow any assumption about the relative speeds of the different processes. [In Dijkstra's terminology, individual tasks were known as *processes.*]

Dijkstra proceeded to illustrate how concurrent processes can synchronize themselves correctly by sending timing signals through *semaphore variables* (as he called them).

Using semaphores, I was able to program the *RC 4000 real-time monitor* in only 400 words of memory, leaving 3,700 words for the data logging tasks that would be implemented by Peter Kraft and Charles Simonyi.

To reduce the size of the task programs, Peter and Charles defined a special-purpose computer that made it easy to write compact code for the complicated engineering computations. The code for this computer was executed by a small interpreter written in RC 4000 machine code. Such a machine, implemented in software (rather than hardware), is known as an "abstract machine." (In the 1980s, interpreted code would also be used in the first versions of the Microsoft Word and Excel programs for the Macintosh computers.)

Interpreted code has also been a marvelous tool for language implementation. The idea of letting a compiler generate code for an abstract machine tailored to a programming language goes back (at least) to LISP in the early 1960s. A major advantage of interpreted code is that it is "portable"— it can run on any computer, if you reprogram the interpreter in the machine code of the target machine. In the 1970s, abstract machines would be used to implement portable compilers for Pascal and Concurrent Pascal. Twenty years later, the same technique would be used to make the Java language "platform-independent."

Now back to the RC 4000 real-time system. The final system occupied 98 percent of the 4K word memory. How was it possible to predict the size of the software that precisely? Well, John Saietz simply gave us a list of desirable features that could be omitted, if necessary. We just kept adding more of these until the memory was full.

At the NordSAM conference in Oslo, Norway, on June 13, 1967, Peter Kraft presented two papers, written by me, on the RC 4000 computer and its real-time control system (Brinch Hansen 1967a, 1967b). According to *Electronic News*:

> Some 400 representatives from the Scandinavian countries participated in NordSAM 67...The RC 4000 system developed by A/S Regnecentralen of Copenhagen, a general-purpose computer especially suited for real-time control, was held up as perhaps the most promising development for international markets. (July 10, 1967)

In the same month, Regnecentralen began installing the RC 4000 prototype and its real-time system in Poland. The communist government was responsible for construction of the plant. Unfortunately, under Wladislaw Gomulka's regime, efficiency was not a top priority. When Villy Toft and Peter Kraft arrived in Pulawy, the computer room still left something to be desired: it had neither doors nor electricity. It required patience and diplomacy to get the room finished. However, a week later, these problems had been fixed, and the installation could begin. The final operational tests of the plant and its real-time datalogging system took place in the spring of 1968.

Later, I heard that the Polish government had not built enough railroad capacity in Pulaway, and was unable to ship the fertilizer as fast as it was being produced to prevent it from piling up. While I cannot confirm this story, I do know Villy had to deal with mice and rats, who ate the backplane wiring and relieved themselves on the circuit boards.

Strangely enough, I never visited the chemical plant in Poland, probably because I didn't like flying. I did travel once by train from Copenhagen to a meeting in Warsaw, enjoying warm tea from a samovar in the rear of the train. Returning through East Germany, I saw armed border guards using huge mirrors to look under the train for refugees trying to escape the communist regime.

⋆ ⋆ ⋆

In June 1967, I returned from Regnecentralen's hardware group in Valby to the Rialto Center as head of RC 4000 software development. From Aage Melbye's group, Peter Kraft and I were joined by Christian Gram, who would define floating-point arithmetic and numerical procedures, and by Søren Lauesen, who would be responsible for developing an Algol compiler.

I was now officially in charge of more projects than I could hope to participate in. However, it still seemed important to me to continue doing, what I enjoyed most—designing and documenting systems programs. I therefore decided to head an operating system group consisting of Jørn Jensen, Peter Kraft, Søren, and me.

In Pulawy, we had tailored a small real-time system to the specific needs of a single customer. We were now hoping to develop a more general monitor program for the RC 4000. It remained to be seen what kind of generality we were looking for.

First we proposed a system with fixed memory partitions for simultaneous execution of three programs, with runtimes of the order of seconds, minutes, and hours, respectively. But, we soon realized that this ad hoc system was not a "general" solution to anything.

Finally, I went to Bech and said: "Look, we aren't getting anywhere. Is it all right with you if Jørn, Søren, Peter, and I stay at a country inn for a weekend?" Bech immediately agreed (he had done the same thing when Regnecentralen's Cobol project had come to a standstill).

I wanted us to discuss the software issues in depth in cozy surroundings to give ourselves one last chance. We had already agreed that we would either return with new ideas or give up and settle for copying the best ideas we could find elsewhere.

How on earth did I have the nerve, at age 29, to gamble my career on a single weekend? Never underestimate the power of the dreams of youth (and its blissful ignorance of "the real world")!

Anyhow, on October 28, 1967, we checked in at Hotel Marina by the seashore, north of Copenhagen. We talked for two days, drank coffee with French cognac, and enjoyed fine dinners. One evening, Jørn and I saw a black-and-white western at a local movie theater—just what we needed to relax after intense discussions.

And it worked! The thought of returning to Regnecentralen without new ideas was simply unacceptable to us. Out of that weekend came the first seminal ideas for the *RC 4000 multiprogramming system.*

Over the next four months, our discussions moved beyond known concepts to the cutting edge of operating system design. I will not attempt to describe how our ideas slowly emerged in daily discussions (I remember some, but not all, of it). Instead I will explain how one thing led to another until everything fit together nicely. Just keep in mind that our conclusions did not emerge nearly as orderly as I present them here.

Regnecentralen was already using computers for software development and business data processing. The Pulawy project added real-time applications to this mix. Unfortunately, real-time software is often unique for each application. And, Regnecentralen simply did not have enough system programmers to develop a different operating system for each RC 4000 installation.

To avoid that trap, we had to look at operating systems in a new way. Computer manufacturers were still developing different operating systems for batch processing, time-sharing, and real-time scheduling. Our hope was to develop a monitor program that would provide the necessary mechanisms to implement *all forms* of multiprogramming.

Regnecentralen's main service center updated large files on magnetic tapes. Early on, we decided to simplify the operator's task in such an installation. Instead of using device numbers, programs would refer to tapes by device-independent *names.* This convention would allow an operator to mount a tape on any available unit and give it a temporary name.

From Dijkstra, we had learned to regard the execution of a program as a sequential process. Now, when you think about it, a sequence of input/output operations on a magnetic tape is also a sequential process. This insight led us to regard program execution and input/output as different kinds of processes—we called them *internal* and *external processes.*

Since we had already decided to assign names to peripheral devices, it was more or less inevitable that we would also end up giving names to internal processes. The beauty of this idea was that programs could refer to processes by their names without knowing where in memory they were located.

We still had to deal with the problem of process synchronization. Semaphores were not robust enough for our purposes. If a program used semaphores incorrectly, it could crash any operating system. Instead, the monitor would implement a reliable form of *message passing.* When one process sent a message to another process, the message was copied inside the monitor and linked to a message queue associated with the receiver. The memory protection guaranteed that the message would remain intact until it had been safely delivered to the receiver.

So we now had named processes communicating by messages. You didn't have to be a genius to suggest that it would be a neat idea to let internal processes request input/output by sending messages to external processes. The monitor would, of course, have to maintain a message queue for each device.

However, since an input/output operation could fail, it would be necessary to return an acknowledgment to the process that sent a command to a device. The way we handled this problem was elegant (but not so obvious). In the end, every communication with a peripheral device consisted of an exchange of a message and an answer between an internal process and an external process. Devices received input/output commands as messages and returned acknowledgments as answers. Needless to say, we soon decided to let internal processes communicate the same way.

Every communication could now be viewed as a procedure call from one process to another: a message identified the procedure and supplied its input parameters; the corresponding answer returned the results of the procedure call. In distributed systems, this form of communication is now known as *remote procedure calls*.

Our final idea was to let internal processes form a tree structure in memory. In this tree, every process would function as the operating system for its children, who, in turn, would control their own children, and so forth. The leaves of the tree would be user processes. The idea of running several operating systems at the same time is, of course, beautiful, but who needs it? Over the years, I have learned not to worry about such questions. If an idea is elegant, you will, sooner or later, find an (unexpected) use for it. The generality of a process tree does, for example, provide an orderly way of switching between different operating systems. It also enables you to test a new operating system on top of an old one, which is a lot easier than developing it on an empty machine.

In February 1968, before programming the system, I described our design philosophy, which drastically generalized the concept of an operating system:

> The system has no built-in assumptions about program scheduling and resource allocation; it allows any program to initiate other programs in a hierarchal manner. Thus, the system provides a general frame[work] for different scheduling strategies, such as batch processing, multiple console conversation, real-time scheduling, etc. [Here I obviously meant "processes" rather than "programs."]

The RC 4000 multiprogramming system was not a complete operating system, but a small *kernel* upon which operating systems for different purposes could be built in an orderly manner The kernel provided the basic mechanisms for creating a tree of parallel processes that communicated by messages.

This radical idea was probably the most important contribution of the RC 4000 system to operating system technology. If the kernel concept seems obvious today, it is only because it has passed into the general stock of knowledge about system design. According to the IEEE Computer Society (2002):

> The RC 4000 multiprogramming system introduced the now-standard concept of an operating system kernel and the separation of policy and mechanism in operating system design. The

microkernels and remote procedure calls used in modern operating systems can trace their roots back to the RC 4000 system.

A well-documented reliable version of the RC 4000 multiprogramming system was running in the spring of 1969. At that point, I described it in a 5-page journal paper.[1] I then used this paper as an outline of the 160-page system manual by expanding each section of the paper.[2]

⋆ ⋆ ⋆

Regnecentralen built several operating systems on top of the RC 4000 kernel. Some of them used dynamic swapping of processes between main memory and backing storage. As usual, the kernel only provided a mechanism for doing this, but left the policy of how and when it was used to an operating system. The latter would ask the kernel to stop a running process and its descendants temporarily. The operating system would then output the memory image of the process to a backing store, and use the same memory segment to reload the image of another process, that had been stopped earlier. The operating system would then ask the kernel to restart that process and its descendants.

In the early 1970s, Regnecentralen developed RC 4000 software for two Danish power plants, Vestkraft and Nordkraft. Villy Toft was again the system installation manager. Working with Niels Nedergaard from Vestkraft, Peter Kraft and Otto Vinter designed a process control system that combined real-time tasks with swapping of Algol programs running as background jobs. Later Regnecentralen's Einar Mossin joined forces with Peter and Niels in designing and programming a real-time system for Nordkraft. One of the challenges here was to record the avalance of alarms, that occurs when the plant is close to a breakdown.

I hired a student, Leif Svalgaard, who became so absorbed in programming the RC 4000 for a Danish Weather Bureau, that he forgot to take his final exam. Leif wrote an operating system that received data and plotted weather maps in real-time. This operating system coexisted with another one that used swapping to perform scientific computations in parallel. According to Leif: "The RC 4000 kernel made all this safe, efficient, and easy,

[1]P. Brinch Hansen, The nucleus of a multiprogramming system, *Communications of the ACM 13*, April 1970.

[2]P. Brinch Hansen, *RC 4000 Computer Software: Multiprogramming System*, Regnecentralen, Copenhagen, Denmark, April 1969.

allowing us to concentrate on meteorological problems instead of fighting the operating system."

Søren Lauesen (1975) described an ambitious operating system designed to handle batch processing, remote job entry, time sharing, jobs generated internally by other jobs, as well as process control simultaneously. It used over a hundred parallel activities, one for every peripheral device and job process. Since the RC 4000 multiprogramming system was limited to 23 concurrent processes, the "Boss 2" system (as it was called) simulated another level of multiprogramming inside a single RC 4000 process. The additional processes were known as "coroutines" (a programming concept that goes back to the early 1960s). Using induction, Søren proved that his system was deadlock-free and guaranteed to complete any request for service. It was implemented and tested by four to six people over a period of two years:

> When we started the Boss 2 design, we knew that the RC 4000 software was extremely reliable. In a university environment, the system typically ran under the simple [manual] operating system for three months without crashes...The crashes present were possibly due to transient hardware errors ...During the first year of operation, the [Boss 2] system typically ran for weeks without crashes. Today it seems to be error free.

⋆ ⋆ ⋆

My descriptions of the RC 4000 multiprogramming system caught the attention of leading computer scientists in Europe and America.

The accolade was a letter from Edsger Dijkstra, professor at the Technological University of Eindhoven in the Netherlands:

> I would like to express my gratitude and admiration for the manual of the multiprogramming system for the RC 4000. It is admirable! You wrote "We present our system as a systematic and practical solution..." and I have the feeling that you are fully right in doing so: it strikes me as a convincing demonstration that it is worthwhile to do a clean job and that it pays to be elegant. My appreciation is equally divided between what the manual describes and the way in which you have described it: it was a pleasure to read it! (Letter from Edsger Dijkstra, August 1, 1969).

After receiving a copy of the RC 4000 multiprogramming system manual, the Swiss computer scientist, Niklaus Wirth, wrote:

> I am much impressed by the clarity of the multiple process concept, and even more so by the fact that a computer manufacturer adopts it as the basis of one of its products. I have come to the same conclusion with regard to semaphores, namely that they are not suitable for higher level languages. Instead, the natural synchronization events are exchanges of message. (Letter from Niklaus Wirth, July 14, 1969)

For almost forty years, Wirth developed innovative programming languages, such as Euler, Algol W, PL 360, Pascal, Modula, Modula-2, and Oberon. When I first met him, he had returned from Stanford, after ten years in Canada and the United States, and was now an assistant professor at the ETH (The Federal Institute of Technology) in Zurich, Switzerland.

At a meeting at the ETH, two years before "Algol" became synonymous with Algol 60, European and American computer scientists had outlined an earlier version, called Algol 58. In May 1968, when the ETH celebrated the Tenth Anniversary of Algol 58, I was invited to participate in a panel discussion on operating systems. The panel, chaired by Niklaus Wirth, consisted of Alfred Schai (Switzerland), Michael Griffith (France), Brian Randell (England), Edsger Dijkstra (The Netherlands), Hans Rudolf Wiehle (Germany), and me (Denmark). About 50 computer scientists attended the discussion in an auditorium with terrible acoustics that made it difficult to hear anything. I don't remember much about the meeting, except that I met Niklaus Wirth for the first time.

In August 1968, Peter Naur, Paul Lindgreen, Søren Lauesen and I attended the IFIP Congress in Edinburgh. Tony Hoare, the inventor of the famous Quicksort algorithm, presented a paper on data structures in a two-level store. In the lobby of his hotel, he listened patiently, while I explained the concepts behind our multiprogramming system.

The small group of Danes at IFIP 68 soon became regulars at the conference bar. If I showed up early, the bartender would say: "Your friends are not here yet." Here I met David Howarth, the designer of the Atlas supervisor that pioneered multiprogramming and demand paging. After explaining that the Scots drink their best whisky and export the rest, David introduced me to Crawford's Five Star whisky. Sure enough, when I asked for this de luxe whisky in a liquor store in London, the owner explained that he, unfortunately, only carried Crawford's Three Star.

1968 was also the year in which the first Nato Conference on Software Engineering was held in Garmisch, Germany. Dijkstra (1999) viewed this as a turning point in the history of computer programming:

> It was there and then that the so-called "Software Crisis" was admitted and the condition was created under which programming as such could become a topic of academic interest. The latter, not surprisingly, turned programming from an intuitive activity into a formal one.

In October 1969 I attended the 2nd Nato Conference on Software Engineering in Rome, Italy (Naur 1969). About sixty people from eleven countries attended. Looking like a Who's Who in Programming, the list of participants included: Fritz Bauer, Bob Barton, Edsger Dijkstra, Tony Hoare, Butler Lampson, Roger Needham, Alan Perlis, Brian Randell, John Reynolds, Doug Ross, Jules Schwartz, Christopher Strachey, Niklaus Wirth, and Mike Woodger.

Niklaus Wirth's working paper on "The programming language Pascal and its design criteria" was my introduction to the first secure programming language that was powerful enough to implement its own compiler.

Edsger Dijkstra talked about "Structured Programming." This method of stepwise programming boils down to breaking a program into small abstract programs that can be divided further, until you reach a level of detail supported by the programming language. At that point, you turn around and combine the small, final pieces into a complete program. The method is similar to the mathematician's way of dividing a theorem into lemmas that can be verified separately and then used to prove the theorem.

What I remember most clearly is Butler Lampson from the Berkeley Computer Corporation. Butler talked like a machine gun. For those who don't know Butler: The rate of human speech is measured in "millilampsons." Butler is the only one who has reached the absolute limit of "1 lampson."

⋆ ⋆ ⋆

Years ago, I wrote an autobiographical sketch, entitled "The programmer as a young dog" (Brinch Hansen 1976d), about my time at Regnecentralen. The title was inspired by James Joyce's "A portrait of the artist as a young man" and Dylan Thomas's "Portrait of the artist as a young dog." When

I showed it to my colleague, Skip Mattson, at Syracuse University, he said: "I would like to know more about your Danish boss." All right then, I will tell you what I know about him.

Niels Bech was born in 1920 in Lemvig, a small Danish town in Northern Jutland. As a child and youth, he would stutter helplessly when he tried to pronounce the combination of an "s" and a "b" in his name. He invented the middle name, Ivar, to be able to say his own name. Growing up in a small, provincial city, he must have been at the receiving end of many cruel jokes.

Bech was a tall guy. At a movie theater in his hometown, a man behind him repeatedly asked him to sit down. Finally, Bech had enough—he stood up and turned around to show how tall he really was. At that point, somebody shouted: "My God, now he is standing on the seat!"

After the completion of Dask in 1957, Niels Ivar Bech became managing director of Regnecentralen. When I first met him, he was 43 years old. A man of many contradictions, he did not hesitate to make bold decisions that put his tiny company at financial risk. Yet, because he was afraid of taking exams at the university, he never completed a higher education.

He trusted his coworkers implicitly and let them pursue their own ideas with minimal intervention. He even tolerated that his hardware development groups, headed by Bent Scharøe Petersen and Henning Isaksson, used different standards of documentation. However, to achieve his goals, he would sometimes bypass his group managers. Bech's underground style of managing through unofficial channels was known as "moling." On his fiftieth birthday, his staff gave him a stuffed mole in a glass cage.

Sometimes, Bech's moling worked brilliantly. One of Scharøe Petersen's electronic engineers, Kurt Henning Andersen, wanted to develop the world's fastest tape reader, using an electronic buffer to stop the paper tape gradually without breaking it. Scharøe did not support the idea, correctly pointing out that Regnecentralen had no expertise in the development of electromechanical devices. However, when Bech heard about the idea, he gave Kurt enough money to develop the paper tape reader in his own kitchen. When the RC 2000 paper tape reader was presented in the fall of 1963, it read paper tape at the unbelievable rate of 2,000 char/sec. With a speed of 15 feet/sec, the tape emerged from the reader like exhaust from a jet plane during take-off, and landed in a waste basket eight feet away. Over a ten-year period, Regnecentralen sold about 1,200 RC 2000s.

But Bech's interference could also be frustrating. At a meeting with a potential customer, he once asked me how long it would take to finish the Algol compiler for the RC 4000. My realistic estimate was twelve months. That was not the answer Bech wanted to hear. So he turned to Jørn Jensen and asked him: "Don't you think we can do it in six months?" Taking the hint, Jørn said "Sure, we can." It made me angry that Bech undermined my credibility as software manager in front of a customer. As it turned out, it took eighteen months to finish the compiler.

In the end, I believe that Bech's unorthodox management style limited Regnecentralen's potential for future growth. His moling worked when Regnecentralen was small. But, eventually, he would have needed a growing staff of professional managers, who would not have tolerated his habit of bypassing them, whenever he found it convenient to do so.

However, in all other aspects, Bech was an inspiring leader. His directive for the RC 4000 software development was rather amazing. His only request to me was: "I need something new in multiprogramming!"

In my opinion Niels Ivar Bech was somewhat of a gambler and showman. He could rarely resist the temptation to do the unexpected. I once participated in a negotiation between Bech and a customer about the sale of an RC 4000 in the middle of a noisy discotheque. Perhaps it is true that unconventional acts rarely succeed in business (we did not sell a machine that evening), but they almost always work in research.

Research is gambling at the highest level. A cautious effort only leads to uninteresting results. A research director must have a sense of which problem to attack next and the courage to give his collaborators the freedom to solve it without imposing narrow constraints. The talent for inspiring his associates to create new things of world-wide renown was one that Bech possessed in the highest degree. Once you have known a leader with this intellectual courage, it is quite depressing to realize how extremely rare this quality is.

Niels Ivar Bech was a dreamer in the most creative sense of the word. His time scale was longer than the one I adopted as a young, impatient engineer. I found it unreasonable that he gave some of his associates time to write textbooks on computer science, without considering how this would influence the immediate needs of the company. That was short-sighted of me. While Bech gave younger colleagues the chance to create new things, he gave his senior people the opportunity to lay the foundation of computer science education in Denmark.

Denmark has made four world-class contributions to computer technology: the Algol 60 report, the Gier Algol compiler, the RC 2000 paper tape reader, and the RC 4000 multiprogramming system. Each of these products combined radically new ideas, which were years ahead of their time (and therefore could not be motivated by an immediate "need"). Without Niels Ivar Bech's brilliant sense of innovation, a small Danish company could probably not have attracted so many outstanding young engineers and be at the cutting edge of programming technology for more than a decade.

Bech's drive and vision went far beyond his job at Regnecentralen. From 1959, he was instrumental in organizing the Nordic Symposiums on Computing, known as NordSAM. In 1960, he became one of the founding members of the International Federation for Information Proccessing (IFIP). For his contributions to IFIP, Bech received the Silver Core Award in 1974.

The decision to publish a Scandinavian journal of computing was made over a glass of beer in Bech's office in 1960. Bech provided economic support and offered to let Regnecentralen handle the administration and distribution. The first issue of the journal BIT gave readers the following choice:

1. *Yes*, I want to subscribe.
2. *No*, I do not hesitate. Put me on your subscription list.
3. *I don't know* of any good reason why I should not subscribe.

From its start, Peter Naur served as co-editor of BIT. His seminal papers on the Gier Algol compiler, elimination of go to statements, type checking, program assertions ("general snapshots"), and modular programming ("action clusters") all appeared in BIT. Naur's paper on *Go to statements and good Algol style* (1963b) appeared five years before Dijkstra's more widely publicized *Go to statements considered harmful* (1968a).

My early papers on the Siemens Cobol compiler, the RC 4000 architecture and the real-time system at Pulawy were also published in BIT.

During the cold war, American companies were not allowed to sell computers in Eastern Europe. This gave Bech a unique opportunity to sell Regnecentralen's equipment in Poland, Czechoslovakia, Hungary, Bulgaria, Rumania, East Germany, and Yugoslavia.

If a communist country was short on western currency, Bech was not above a little horse trading (seriously). On one occasion he apparently delivered computer equipment to Poland in return for a shipment of horseflesh, which he somehow managed to sell in Denmark.

His boundless energy and visionary thinking made it inevitable that some of his efforts would meet resistance. Around 1960, the Technical University of Denmark and the National Engineering and Science Foundation (Statens teknisk-videnskabelige fond) recommended that the government support the use of Gier computers for research and education at Danish universities. This idea was successfully opposed by Willy Olsen, manager of the government's own computing center, Datacentralen, opened in 1959 at the initiative of Viggo Kampmann, minister of finance. I once asked Kampmann, who was married to my father's cousin, why he supported the creation of Datacentralen. He said he thought competition would be good for Regnecentralen. Willy Olsen, apparently, did not share Kampmann's belief in competition.

Throughout the 1960s, Bech tried unsuccessfully to persuade Scandinavian computer manufacturers to merge. Once, Bent Scharøe Petersen and I accompanied him to a meeting with DataSaab in Linkőping, Sweden. That evening, we returned to Copenhagen on a tiny airplane, flying through a blizzard with zero visibility. Sitting in front with the pilot, Scharøe mentioned that we had lost all radio contact. That did not seem to worry Bech in the least.

On another flight in Bulgaria, passengers started screaming and praying when an engine caught fire. The story goes that Bech calmly ordered beer for everybody.

Towards the end of the 1960s, it became increasingly clear that the pioneering era of the Danish computer industry was coming to an end. In 1970, I left Regnecentralen and moved to the United States. At that time, Niels Ivar Bech was already showing signs of illness. Since then I only saw him briefly at the IFIP 71 Congress in Milena's hometown, Ljubljana, in Slovenia.

In 1971, Bech was fired by Regnecentralen's board of directors. I don't know why he was dismissed. In his dealings with business leaders and government employees, he probably had the misfortune of thinking big among people who were not used to thinking big. Since he was decades ahead of his time, Bech undoubtedly would have found some of his board members shortsighted. They, in turn, would almost certainly have found him unrealistic (as I did, on the occasion when Bech told me that his goal was "to push IBM back into the Atlantic Ocean"). Perhaps, in the words of Vartan Gregorian, "He did not and could not serve people he did not respect, especially those who were political hacks, men without integrity, mission or vision, empty suits." Who knows?

On July 25, 1975, he died of a heart attack at age 55. With Niels Ivar Bech's death, Denmark lost its leading role in the development of programming technology. Four years later, Regnecentralen ceased to exist. However, by then it hardly mattered. Bech had already made a lasting contribution to his country by training the first generation of computer pioneeers and laying the groundwork for computer science education in Denmark. Over the years, a number of Regnecentralen's senior people became faculty members at Danish universities: Peter Naur, Aage Melbye, Ole Møller, Poul Sveistrup, Christian Andersen, Henning Isaksson, Henning Bernhard Hansen, Christian Gram, Peter Kraft, Søren Lauesen, Paul Lindgreen, and I (for a short time).

Those of us, who were privileged to start our careers under Bech's visionary leadership, will always remember Regnecentralen as the lost paradise.

In 1983, I dedicated my book, *Programming a Personal Computer*, to the memory of Niels Ivar Bech.

⋆ ⋆ ⋆

My years at Regnecentralen were some of the happiest years of my professional life. I had worked in compilers, computer architecture, and operating systems. And I had met four computer scientists, who would influence my future work: Peter Naur, Edsger Dijkstra, Niklaus Wirth, and Tony Hoare.

It was time to move on. I was now planning to go abroad and write the first systematic textbook on operating system principles.

5

SHAPING A NEW FIELD 1970–72

Alan Perlis invites me to spend a year at Carnegie-Mellon – Emigration to America – Niklaus Wirth defines Pascal – Driving home in a blizzard – Discussing the future of concurrent programming in Marktoberdorf and Belfast – Alan Perlis tells stories – Mad King Ludwig and Thomas Edison – The first modern book on Operating System Principles.

Alan Perlis was an early pioneer in the development of programming languages and compilers. He was involved in the definition of the programming languages Algol 58 and Algol 60. In a retrospective talk, he said (Perlis 1981):

> One of the things we learned about computing in the 1950s was that there are no bounds to the subject. It cannot be put into a tidy receptacle. Everywhere that computing has been embedded in some other discipline, it has not flowered. Computing is not part of electrical engineering; it is not part of mathematics; it is not part of industrial administration. Computing belongs to itself. The reason this *is* the computer age is precisely because of that.

In 1962, Perlis was forty years old and director of the computation center at Carnegie Institute of Technology in Pittsburgh. He believed that "the programming and using of computers deserve an early appearance in the university curriculum for the educated man." At an MIT symposium on "Computers and the World of the Future," he added (Perlis 1962):

> Perhaps I may have been misunderstood as to the purpose of my proposed first course in programming. It is not to teach people how to program a specific computer, nor is it to teach some new

> language. The purpose of a course in programming is to teach people how to construct and analyze processes. I know of no other course that the student gets in his first year in a university which has this as its sole purpose.

Today, when every department of computing and engineering offers programming courses, it may be difficult to appreciate just how farsighted Perlis was—until you realize that he took it for granted that every student of the arts and sciences eventually would be required to take such a course. Forty years later, that still hasn't happened.

In 1965, Carnegie Tech became one of the first universities to create a graduate department of computer science. Alan Perlis, Allen Newell, and Herbert Simon were the driving forces behind the establishment of computer science as an independent discipline at Carnegie.

The appointment of the visionary MIT professor, Joseph Carl Robnett Licklider, as director of information processing research at ARPA (The Advanced Research Projects Agency at the US Department of Defense) was a major piece of luck for Carnegie Tech. Licklider selected MIT and Carnegie as the first two ARPA centers of excellence. While MIT would develop time-sharing, Carnegie would have complete freedom to explore the fundamentals of computing. I cannot think of any other country and research agency that would distribute government funds in such an informal way.

Of course, money by itself does not explain how Carnegie was able to establish one of the finest schools of computer science in the world (Perlis 1981):

> I think computers have flowered in this country because of our national style of accomplishing things. This country has always supported enterpreneurial activities—people who have ideas and are willing to sweat to bring them out. Computers flower in such an environment. . .The Soviet Union, having a large centralized society, needs computers much worse than we do. Yet they are totally unable to produce them, totally unable to apply them in anywhere near the profusion that we find here. They have the wrong kind of society for the instrument they most badly need.

Perlis was the founding editor of *Communications of the ACM*, a past president of the Association for Computing Machinery, and now the first chair of computer science at Carnegie. His administrative tasks cannot have left him much time for concentrated intellectual work. Apart from Peter Naur's Algol 60 report I find few references to his work in textbooks.

Alan was completely bald and had no eyebrows. He looked (and was) extremely intelligent. He was immensely charming with an endless supply of anecdotes and jokes. And highly original in his thinking. They said that if a student met him in the elevator, Al would propose several thesis topics on the spot. The trouble was that only one of them was worth pursuing and the student had no idea which one that might be.

His faculty and students loved him. At parties they would gather around him and listen to his anecdotes and words of wisdom. He was famous for his wise and amusing sayings (Perlis 1982):

- Syntactic sugar causes cancer of the semicolon.
- In the long run every program becomes rococo—then rubble.
- Simplicity does not precede complexity, but follows it.
- Structured Programming supports the law of the excluded muddle.
- A LISP programmer knows the value of everything, but the cost of nothing.
- Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.
- When we write programs that "learn," it turns out we do and they don't.
- Because of its vitality, the computing field is always in desperate need of new cliches: Banality soothes our nerves.
- Editing is a rewording activity.

He had sound advice on how to deal with university administration (Perlis 1981):

> My attitude has always been that, if you are right, the administrators will accede to your wishes. I haven't been disappointed. By and large, administrators are always looking for people to tell them what they ought to be doing, rather than being confronted with a decision that they have to make on which they have no

> information with which to make that decision, so the natural technique is to postpone or form another committee. Instead, one really ought to go to them and say, "Do this because it's right."

Alan Perlis reminded me of Niels Ivar Bech. Both of them were visionary thinkers and brilliant leaders, who were ahead of their time in recognizing the need for research and education in computing. Like Bech at Regnecentralen, Perlis's mission in life was to create a place where creative people could flourish.

In an interview after Perlis's death, in 1990, Allen Newell said: "People would go to conferences and listen to every word he said, because every time he talked about a topic, he was absolutely right about the way it should be. That's why he was the first Turing Award winner. Not at all for any technical contribution, really, but because he epitomized the nature of computer science."

⋆ ⋆ ⋆

Dijkstra's first doctoral student, Arie Nicolaas Habermann, was a former math teacher in the Netherlands. In 1968, after receiving his PhD, Nico joined the computer science department at Carnegie-Mellon University (as it was now called).

At the Technological University of Eindhoven (THE) in the Netherlands, he had participated in the implementation of Dijkstra's famous THE multiprogramming system. This system consisted of several "program layers," which gradually transformed the physical machine into an ideal abstract machine for multiprogramming. The program layers could be designed and tested one at a time.

At a time, when the term "software crisis" was coined to describe the sad state of operating systems, Dijkstra (1968b) made a startling claim:

> We have found that it is possible to design a refined multiprogramming system in such a way that its logical soundness can be proved a priori and its implementation can admit exhaustive testing. The only errors that showed up during testing were trivial coding errors . . . the resulting system is guaranteed to be flawless.

Dijkstra proved that the THE system was deadlock-free. (A deadlock is a situation in which concurrent processes wait indefinitely for events that will never occur.) In his thesis, Habermann extended Dijkstra's proof to a layered system with any number of levels.

I first met Nico at the NordSAM conference in Stockholm, Sweden, in August 1964. Over the years, we kept in contact. Like his thesis advisor, Edsger Dijkstra, he was well aware of the RC 4000 multiprogramming system. On January 2, 1969, he invited me to visit Perlis's department at Carnegie-Mellon:

> It is my pleasure to invite you officially to spend some time during the spring in Pittsburgh and to visit our department. We would like you to spend about two weeks with us, during which we expect you to tell us about your work.

I arrived in Pittsburgh in March 1970 and stayed with Nico, his wife Marta, and their children, Eveline, Irene, Marianne, and Frits. Every morning, Nico and I drove in his Volkswagen from the suburb of Mount Lebanon to the university in Schenley Park.

In a classroom at Carnegie, I lectured on the RC 4000 multiprogramming system. The audience of about 30 people included Alan Perlis, Gordon Bell, Bill Wulf, and, of course, Nico. Among the graduate students, I remember Anita Jones and Rudy Krutar.

During my visit, I mentioned that I was planning to spend a year abroad writing a textbook on operating systems. Perlis immediately said: "I am as greedy as the next man—why don't you write it at Carnegie?" Since Nico was also planning to write such a book, we agreed to make it a joint effort.

A month later Perlis sent me an official invitation to spend a year in Pittsburgh (Fig. 5.1). As you can tell, he could charm your socks off! This letter would change my life dramatically.

⋆ ⋆ ⋆

On November 1, 1970, Milena and I left Denmark with our children, Mette (age 4) and Thomas (age 3), and traveled to Pittsburgh on a Lufthansa 707 jetplane. In Mount Lebanon, we rented a small house on 603 Oxford Boulevard. Our rented furniture included a sofa stuffed with sawdust that slowly accumulated on the floor. Upstairs, we slept on mattresses on the floor.

Carnegie-Mellon University
Department of Computer Science
Schenley Park
Pittsburgh, Pennsylvania 15213

April 17, 1970

Civ.ing. Per Brinch Hansen
Hostrups Have 32/6
1954 Copenhagen V
Denmark

Dear Per:

With this letter I should like to formally invite you to spend either 10 months or 12 months as a research associate in the Department of Computer Science. It is my understanding that you would like to commence on or about October 1, 1970, and this is perfectly satisfactory with the department.

The salary will be $15,000 for 10 months or $18,000 for 12 months. Furthermore, the department agrees to pay 1/2 of your reasonable traveling expenses from Copenhagen to Pittsburgh and return.

We are pleased to make this offer since the talent and experience you would provide would be of great value to the research program of this department. Furthermore, the department is pleased that a person of your exceptional ability will be collaborating with members of our faculty in important research to further our understanding of the nature of complex computer operating systems.

Please keep me informed of the progress in your application for visa so that we may provide whatever assistance is needed from this end.

Needless to say the entire faculty is looking forward to having you with us this coming school year.

Sincerely yours,

Dr. Alan J. Perlis, Head
Department of Computer Science

Figure 5.1 Invitation from Alan Perlis.

From Marcus Motor downtown, I bought a new Volvo for $3,400. In Denmark, where sales taxes doubled the price of a car, this was considered a large car. But, in America our neighbors referred to it as "your cute little car" and Perlis offered the opinion that "it rides like a truck." It turned out to be a lemon that stalled if you tried to turn left at a traffic signal. To avoid this in the middle of heavy traffic, I would drive around the block turning right three times instead.

In Denmark the kids attended a nursery school that did not teach religion. When I asked Perlis, who was Jewish, for advice on how to chose a similar one in Pittsburgh, he suggested putting them in a Jewish nursery school. We took his advice and it worked out well.

It was a lonely time for Milena. Her H-4 visa did not permit her to work in America. When I was at work and the children at nursery school, she was alone in the house without a car.

⋆ ⋆ ⋆

After the first visit to America, I began outlining my book and collect papers on operating systems from Regnecentralen's excellent library. As soon as I started writing, it became clear that I needed a programming language to express operating system functions concisely without unnecessary trivia. I began using Pascal, which had a much richer set of data structures than Algol 60. Niklaus Wirth had just published the Pascal report, while three of his coworkers completed the first compiler (for the CDC 6600 computer). In hindsight, my choice of Pascal was an inspired gamble. I had no way of knowing whether this new language would ever be widely available and used for teaching!

The idea of writing the book jointly with Nico Habermann turned out to be a grave mistake. At Regnecentralen, I was used to an atmosphere of frank, critical discussion. If I thought Nico's ideas were problematic, I would tell him so. Habermann was equally critical of my approach. He hated Pascal with a passion and would eventually write a scathing criticism of the language, which the editor of Acta Informatica asked him to tone down (Habermann 1973). He had a point—in its lack of conciseness, the Pascal report was a step backwards compared to Peter Naur's brilliant Algol 60 report. Nevertheless, the language features of Pascal were (in my opinion) a significant contribution to programming language design.

Coming from the Netherlands, Nico must have felt under great pressure to live up to the expectations of a leading department in a foreign country.

Throughout his academic career, he would remain true to the high standards of software design he had learned from Dijkstra. However, as a PhD student he naturally stood in Dijkstra's shadow. When he joined Carnegie as a visiting researcher, Nico was 36 years old. His challenge was to start independent research at an age when most research careers already have peaked.

I, on the other hand, had already had some success with the RC 4000 project. Whereas Habermann had to teach and advise students, I had all the time in the world to work on my book. I made rapid progress, and he did not. Eventually, the pressure became too much for him, and one day, he left my house in a huff. The next day, I was extremely relieved to find that Perlis reacted calmly to the news of our breakup: "We are going to need many books on operating systems. Nico and you should write two separate books on the subject."

⋆ ⋆ ⋆

I admired the way American computer scientists kept themselves informed about ongoing research at other universities and research centers. When I started working at Carnegie-Mellon, I was immediately invited to give seminars at leading universities and research centers:

Toronto (November 30, 1970).
Princeton (December 16, 1970).
IBM Research Center (March 1, 1971).
Cornell (April 12, 1971).
MIT (May 17, 1971).
Berkeley (October 13, 1971).
Michigan (December 13, 1971).

These talks not only gave me a chance to meet leading computer scientists, they were also an opportunity for Milena and me to drive with our children around the eastern part of North America.

In Toronto I met members of the Sue operating system group: Rick Holt, Jim Horning, and Dennis Tsichritzis. They were building an operating system, named Sue, for the IBM 360 computers using Dijkstra's layered approach.

This was our first visit to Niagara Falls. "Niagara" is an American Indian word meaning "thundering waters." Every second, 350,000 gallons of water flow over the 2,300 feet wide falls and crash some 180 feet below. During the day, it is truly one of the natural wonders of the world. I was less enchanted by the view at night, when the falls are illuminated by floodlights in pink, green and white.

One evening we were driving home from a visit (I don't remember which one) to Boston. As we reached Interstate 80 in Pennsylvania, it started snowing heavily as dark fell. This highway runs east-west in the Allegheny Mountains. I should have noticed that there was a reason why truck drivers were leaving the highway, one by one. Although we were now alone on this snowy road, I continued stubbornly. At one point, the snow fell so heavily that I had to roll down the front window and wipe the windshield manually while I was driving. After driving 600 miles, we reached Pittsburgh early in the morning. Later, I read that people froze to death every year when their cars got stuck in the snowy mountains.

⋆ ⋆ ⋆

In the summer of 1971, I attended two scientific meetings in Europe that would have a deep influence on my work. The first one was the International Summer School on Program Structures and Fundamental Concepts of Programming, July 19–30. It was organized by Professor Fritz Bauer, Technical University of Munich. The meeting was held in Marktoberdorf, a small town in Southern Germany. I was one of seven invited speakers:

– Rudy Bayer, On the Structure of Data and Application Problems.

– Per Brinch Hansen, Operating System Principles.

– Edsger Dijkstra, Hierarchical Organization of Sequential Processes.

– Tony Hoare, Proof of Programs.

– Alan Perlis, Conversational Languages.

– John Reynolds, Lattice Theoretic Models for Programming Language Definition.

– Niklaus Wirth, The Representation, Implementation and Application of Fundamental Concepts of Programming.

Over the course of the Summer School, each of us gave about ten lectures. The 100 (or so) participants had submitted applications to attend and were selected by the directors of the Summer School. The speakers attended every lecture and participated in the lively discussion. After dinner in the local Hotel Sepp, the lecturers and students would continue the discussion over a glass of beer.

In my recollections of the Summer School I have included some of Dijkstra's (1971b) impressions of the speakers (translated from Dutch).

With deep insight, Dijkstra (1971a) explained his layered approach to operating system design in greater detail. He also demonstrated how to prove the correctness of fundamental algorithms with semaphores. He introduced and solved the scheduling problem of the "dining philosophers," which poses subtle dangers of deadlock and unfairness —described in flamboyant terminology as "deadly embrace" and "starvation." (A scheduling policy is "unfair" if it consistently favors some processes over others.)

In his lectures, Hoare presented his method for formal verification of sequential programs. This was still unfamiliar territory to me. I remember Tony asking me how I would prove the correctness of concurrent programs with message communication. I had no idea what he meant by the question.

> Among the speakers, C. A. R. Hoare undoubtedly stole the show: He spoke every morning after breakfast. (He had asked to speak "early." However, instead of lecturing at the beginning of the Summer School, he ended up being the first speaker every day!) He talked about axiomatising the current concepts of programming. The interesting thing was that he not only combined this with correctness proofs of programs but also with elements of programming languages. It was further commendable to observe that he expressed many reservations about the limitations of his subject: he made it clear that, in all sorts of cases he did not regard it as realistic to supply a formal correctness proof. Sort of like "If you are challenged to supply a correctness proof, you should be able to do that, but it is not something you should do all the time."

Niklaus Wirth described Pascal and explained how his group succeeded in writing the first compiler in Pascal and making it compile itself:

> During the first week, N. Wirth from Zurich spoke for eight hours about his creation, the programming language Pascal. . .The Pascal implementation is a beautiful work that confirms my impression of him as one of Europe's most competent programmers.

> The Pascal compiler was first written in Pascal. It was then in a couple of weeks translated by hand into a low-level language. Two weeks after they first gained access to the [CDC 6600] computer, the Pascal compiler compiled itself.

Dijkstra gave a mixed review of my performance:

> The status of Pascal was not only enhanced by Hoare's experiment with axiomatization—Hoare has played an important role in the Pascal draft—but also because Per Brinch Hansen used it to describe operating system principles. In the past year, Brinch Hansen has. . .worked on a book about "Operating System Principles" which he summarized in his talks. The text was fine, but was not well presented. As a speaker he is a bit monotonous, and he did not feel at ease: it obviously bothered him that he had to borrow so much from me and appeared in his search for his own position somewhat argumentative.

That was not quite right. I admired Dijkstra and was delighted to acknowledge his fundamental ideas. I was simply an inexperienced speaker who felt nervous about lecturing in English to an audience that included some of the world's leading computer scientists. When I returned to Marktoberdorf two years later, my beginner's problems had disappeared.

Perlis was in fine form. During one of his lectures, he told a hilarious story about a guy, who wanted to be buried in his Cadillac. As the car with the dead man was lowered into the enormous grave, one mourner said to another: "Man, that's living!"

The contrast between the rigorous scientist, Edsger Dijkstra, and the easygoing Alan Perlis was striking:

> I found the great Alan J. Perlis very poor. He speaks with incredible ease if not to say bluff; his jokes are the most carefully prepared part of his presentation, and it is a pleasure to listen to him, as long as you don't listen too carefully to what he says. Unfortunately, part of his audience did just that. . .His actual subject was LCC, a conversational language developed at Carnegie-Mellon. But this was especially difficult to defend in this company, since it flies in the face of the ideas of program structuring preached by Wirth, Hoare and me. . .I have never seen

> him so insecure; perhaps it had to do with his departure from Carnegie-Mellon the previous week to join Yale University. . .and start a new life. Professionally he did not make a good impression.

In the interest of fairness, I can't resist quoting another epigram by Perlis:

> When a professor insists that computer science is X but not Y, have compassion for his graduate students.

Over the years, it became a tradition at the Summer Schools to visit Neuschwanstein Castle, built in the style of a Medieval castle by King Ludwig II of Bavaria (1845–86). (Cinderella's castle in Disneyland is a small replica of Neuschwannstein.) In 1864, Ludvig became king at the age of 18. This shy young man tended to neglect the affairs of state in the capital Munich, preferring instead to live in the Alps, where he spent a fortune building castles. Eventually, a group of politicians decided to have the king declared insane and replace him by his uncle. However, the doctors who declared Ludvig to be mentally ill never examined him. On Sunday, June 13, 1886, the king was found drowned in Starnberg Lake. Walking through his castle, I was looking at a massive, handcarved wooden horse sleigh, when it suddenly struck me: In 1882, this romantic dreamer was building a fairy tale castle, while Edison was demonstrating his electric light in Munich, 50 miles away!

⋆ ⋆ ⋆

The Marktoberdorf Summer School was followed by a Seminar on Operating Systems Techniques held at the Queen's University in Belfast, Northern Ireland, between August 30 and September 3. The speakers included some of England's leading operating system designers. David Hartley gave a reappraisal of the Atlas I system, which had pioneered multiprogramming ten years earlier. Sandy Fraser and Roger Needham spoke about the Atlas II system (known as Titan). Tony Hoare and his coworker Mike McKeag presented survey papers on various aspects of operating systems. And I outlined my operating system text again (Brinch Hansen 1971).

The threat of terrorist bombings had discouraged several contributors, including Dijkstra, from attending the seminar. At the time, bombings were frequent in downtown Belfast. In my hotel room, I heard several faint explosions at night. During a presentation at the seminar, a female student entered the small lecture room, put her briefcase on the floor and left the room. I thought she might have left it there to go to the bathroom. But Tony immediately rushed over, looked inside the briefcase, picked it up and ran after her. I remember thinking: "He is awfully nosy." Then it dawned on me that he was worried she might be a terrorist trying to blow us up. On another occasion, we were driving in a tourbus outside the city when the radio warned listeners not to be worried about sonic booms that might occur that day during a testflight of the Concorde supersonic plane. The bus erupted in nervous laughter.

At an official reception in the City Hall, we were all waiting in line to shake hands with The Right Honourable Lord Mayor of Belfast. I was impressed by his technique: as soon as the mayor grabbed my hand, he pulled me forward to keep the line moving.

At the final banquet, I was seated next to Tony Hoare. After welcoming everybody he sat down, and I looked forward to a relaxing dinner conversation after a long day of discussions. However, as soon as he sat down, Tony started writing on a paper napkin, saying to me: "Let P be a process which satisfies the following condition..." I remember thinking: Let this be a lesson to anyone who doesn't realize that the leaders of their field never take a break—they think all the time, and, if you don't do the same, you will never be one of them.

In Marktoberdorf and Belfast, Tony and I discussed the future of programming languages. The trend was clear. Since 1960, high-level programming languages had replaced machine coding in one application after another: scientific computing (Fortran and Algol 60), business data processing (Cobol), and compiler design (Pascal). It was obvious to us that the next challenge was to invent programming notation for concurrent programs.

Now, in order to cooperate on common tasks, concurrent processes must be able to exchange data and timing signals through shared variables in memory. From the beginning, operating system designers recognized that multiprogramming will not work unless operations on shared data structures take place strictly one at a time. The reason is obvious: To understand what happens when, say, a process sorts an array, you have to assume that the sorting takes place without interference from other processes. If other processes can make arbitrary changes to the same array, while it is being sorted, it would indeed be a miracle if it ever ends up being sorted.

This is known as the *mutual exclusion* problem. In the RC 4000 multiprogramming system, we solved it by the well-known technique of disabling interrupts during the execution of monitor procedures. This guaranteed that the computer would never interrupt an indivisible operation once it was started.

Dijkstra had shown how to implement these *critical sections* (as he called them) using semaphores. (Hoare would rename them *critical regions.*) However, if semaphores are omitted or used incorrectly, concurrent processes will interfere with one another in time-dependent ways that depend on the relative speeds of the processes. And those speeds can be influenced by completely unrelated factors, such as the speed at which terminal users respond to messages, or the occasional delays when printers run out of paper. Every time, you execute a time-dependent program, it will produce different results—even if the input is the same. It behaves like a universe in which Newton's laws have been replaced by random, unpredictable events. Since you cannot reproduce the output of such a program, it is often impossible to locate time-dependent programming errors by testing.

At the Belfast symposium, Hoare (1971) presented a first attempt to extend programming languages with abstract notation for process synchronization. He proposed that the error-prone semaphores should be replaced by language notation that would enable a compiler to recognize shared variables and check that they are used inside critical regions only.

Since it is awkward to talk about notation without showing it, I will yield to temptation and show you what he did. Take, for example, a mailbox that can transmit one integer at a time from one process to another. In a Pascal-like notation, the mailbox is declared as a shared data record with two components—a message slot of type integer and a boolean indicating whether the mailbox currently is empty or not:

```
var mailbox: shared record
               slot: integer; empty: boolean
             end;
```

A sending process waits until the mailbox is empty before putting a message into the mailbox:

```
with mailbox when empty do
begin slot := message; empty := false end
```

A receiving process waits until the mailbox is non-empty before removing a message from the mailbox:

```
with mailbox when not empty do
begin received := slot; empty := true end
```

Hoare's notation suppresses implementation details (such as the synchronization mechanism and scheduling policy used) and shows only the essence of the problem: sending and receiving are critical regions on the shared mailbox. Since these operations are delayed until the mailbox is in a suitable state (empty or full), I named them *conditional critical regions.*

Hoare's paper was as an eye-opener for me: It was my introduction to *the difficult art of language design.* The idea of preventing time-dependent errors by compile-time checking struck me as magical at a time when multiprogramming systems relied exclusively on run-time checking of variable access.

I had already received an earlier draft of Hoare's paper and had used conditional critical regions in my own book manuscript. However, at the Belfast symposium, I expressed some reservations from a software designer's point of view (Brinch Hansen 1971):

> The conceptual simplicity of simple and conditional critical regions is achieved by ignoring the sequence in which waiting processes enter these regions. This abstraction is unrealistic for heavily used resources. In such cases, the operating system must be able to identify competing processes and control the scheduling of resources among them. This can be done by means of a *monitor*—a set of shared procedures which can delay and activate individual processes and perform operations on shared data.

In his presentation, Hoare responded to my criticism (Discussions 1971):

> As a result of discussions with Brinch Hansen and Dijkstra, I feel that this proposal is not suitable for operating system implementation. . .My proposed method encourages the programmer to ignore the question of which of several outstanding requests for a resource should be granted.

Hoare's proposal was based on the programming style used in Dijkstra's THE system, where critical regions were scattered throughout the program text. At Marktoberdorf, Dijkstra briefly outlined an alternative model, where a shared variable and its critical regions were combined into a separate process, known as a "secretary." This server process would own the shared variable and execute the critical regions on request from client processes.

Dijkstra had implemented his multiprogramming system as cooperating processes communicating through *shared variables* in *unprotected memory.* From his point of view, the idea of combining shared variables and critical regions into server processes was a new approach to resource scheduling.

However, this idea was obvious to the designers of the RC 4000 multiprogramming system, based as it was, on a paradigm of processes with *disjoint memories* communicating through *messages* only. There was simply no other way of using the RC 4000 system!

The secretary concept, which Dijkstra sketched informally, had already been used since 1969 in the RC 4000 system under the name of a "conversational process" (Brinch Hansen 1969a). This was a server process that could be in the middle of conversations with several client processes at the same time. In the RC 4000 system, the basic operating system was a conversational process that spawned other processes in response to messages from operator consoles. If the basic system temporarily was unable to honor a request, it would postpone the action by delaying its receipt of the message. In the meantime, it would serve other clients. Since it was impossible to predict when the system would respond to a request, a conversational process was an early example of the use of *nondeterministic communication.*

Dijkstra may well have been influenced by my RC 4000 manual, which explained the purpose and use of conversational processes. Mike McKeag (1972–73) would demonstrate the similarity of these ideas by using the RC 4000 message primitives to outline simple secretaries for the well-known synchronization problems, known as the "bounded buffer," the "dining philosophers," and the "readers and writers."

Be that as it may. The important thing was that a *resource manager* was a well-known programming technique in the form of a *basic monitor*, invoked by supervisor calls, or a *conversational process* (a "secretary"), invoked by message passing. Our future efforts would be focussed on extending programming languages with an elegant notation for this *monitor concept* (as I called it).

⋆ ⋆ ⋆

I now returned to Carnegie-Mellon to finish my book. Today, when all operating system texts cover the same material, it may look like a well-defined task. But I had no models to imitate. It wasn't even obvious to me, what the sequence of chapters should be. To write the first systematic book on operating systems I would have to shape the field.

The implementation techniques of operating systems were reasonably well understood in the late 1960s. But most systems were too large and poorly described to be studied in detail. All of them were written either in assembly language or in sequential programming languages extended with assembly language features. Most of the literature on operating systems emphasized low-level implementation details of particular systems rather than general concepts. The terminology was unsystematic and incomplete.

Before the invention of abstract concurrent programming, it was impractical to include algorithms in operating system descriptions. Technical writers mixed informal prose with unstructured flowcharts and complicated pictures of linked lists and state transitions.

In 1971, when I had been working on my book for almost a year, the National Academy of Engineering summarized the state of affairs at the time:

> The subject of computer operating systems, if taught at all, is typically a descriptive study of some specific operating system, with little attention being given to emphasizing the relevant basic concepts and principles. To worsen matters, it has been difficult for most university departments to develop a new course stressing operating systems principles. . .There are essentially no suitable textbooks on the subject. The best source material is found in technical papers that frequently are hard to locate, understand, or correlate. (Cosine Report 1971)

Indeed! I still remember one instance of the slow, methodical task of trying to make sense of the available sources. From Regnecentralen's excellent library I brought a one-foot high stack of photocopies of papers that included descriptions of memory allocation techniques used in various systems. In our house in Pittsburgh, I read all of it and eliminated half of it as poorly written or of minor interest. Then I read the remaining half again and cut that in half, and so on, until I finally had reduced it to a dozen papers. At that point, my mind was so saturated with the subject that I began to see a reasonable pattern for my chapter on memory management.

By the time I outlined my book in Belfast, I had written drafts of half of the eight chapters. And, I had reached the conclusion that *operating systems are not radically different from other programs. They are just large programs based on the principles of a more fundamental subject: parallel programming.*

Starting from a concise definition of the purpose of an operating system, I divided the subject into five major areas. First, I presented the principles of parallel programming as the essence of operating systems. Then I described processor management, memory management, scheduling algorithms and resource protection as techniques for implementing parallel processes.

I defined operating system concepts by abstract algorithms written in Pascal extended with a notation for *structured multiprogramming.* My (unimplemented) programming notation included concurrent statements, semaphores, conditional critical regions, message buffers, and monitors. These programming concepts are now discussed in all operating system texts.

The book includes a *concise vocabulary* of operating system terminology, which is used consistently throughout the text. The vocabulary includes the following terms:

concurrent processes, *processes* that overlap in time; concurrent processes are called **disjoint** if each of them only refers to **private data**; they are called **interacting** if they refer to **common data**.

synchronization, a general term for any constraint on the order in which *operations* are carried out; a synchronization rule can, for example, specify the precedence, priority, or mutual exclusion in time of operations.

monitor, a *common data* structure and a set of meaningful *operations* on it that exclude one another in time and control the *synchronization* of *concurrent processes.*

⋆ ⋆ ⋆

In the spring of 1972 it was show time! I sent my finished manuscript to Karl Karlstrom, the famous computer science editor at Prentice-Hall, who asked Jim Horning, University of Toronto, what he thought of it. On May 1, Jim answered:

> Dear Karl:
>
> This is a preliminary report on Operating System Principles by Per Brinch Hansen, that I am reading with great interest. I think you should definitely try to get the book, and to give it the speedy publication which Brinch Hansen wants.
>
> This is a good book, perhaps even a great book, better than anything I have seen on operating systems. Its style and clarity makes it suitable for third-year undergraduate students, yet it is sufficiently comprehensive for use in a graduate course. I

> have not noted any technical errors, and remarkably few errors of presentation.
>
> I am eager to see the book come out soon, but not at the cost of cutting corners. This book will be around for a long time, and it should be handsomely done.
>
> Jim Horning

Eleven days later, I signed a publishing agreement. I also sent complete copies of the manuscript to Edsger Dijkstra and Tony Hoare.

In July 1973, Prentice-Hall published *Operating System Principles* (Brinch Hansen 1973a), and I was again invited to lecture at the Summer School in Marktoberdorf. This time, Dijkstra (1973) wrote:

> Personal reasons prevented M. Griffiths of Grenoble from speaking as scheduled, but we were lucky in having Per Brinch Hansen (California Institute of Technology) as a substitute. He went through the highlights of his recently published book "Operating System Principles" and he did that much, much better than two years ago, when he covered the same material in a very biased and even aggressive manner. Now he gave a neat, balanced survey. It is a pity that he has a very monotonous voice; it is really soporific and now I cannot even read one of his publications without hearing it.

Two years later, Peter Naur reviewed my book. Please, forgive me for indulging myself. It was, after all, Peter who had shown me that computer programming should be guided by sound principles and clear writing. In BIT 15 (1975), he wrote:

> It would be improper not to start this review with a warning: the reviewer is biased in favour of this book. In fact as mentioned in several remarks in the book, the work and attitude embodied in it have been influenced positively by the work with projects around 1964 for which your present reviewer was partially responsible. Your reviewer is unable to suppress a feeling of joy that the seeds sown at that time have borne this book as its fruit.
>
> It needs hardly to be said that the subject of this book, operating systems, are complex constructions. They are the outcome of the efforts of computer designers and programmers to apply

their powerful tools to their own central problem: to control the use of computers in a flexible and efficient manner. True to its title, this book is a deliberate attempt to get to grips with this complexity through a concentration on principles. For this reason the book addresses the advanced reader, who is already familiar with at least some of the existing operating systems. In fact, the general character of the problem is not described in depth, but merely reviewed in the 22 pages of chapter 1.

The presentation is generally at a very high level of clarity, and gives evidence of deep insight. In pursuing his principal aim, the establishment of a coherent set of basic principles for the field, the author is highly successful. The principles are supported by algorithms written in Pascal, extended where necessary with carefully described primitives. Close attention is paid to the thorny question of terminology. The main terms used in the text are explained in a vocabulary at the end of the book.

As said explicitly in the introduction, the book presents one man's view of the subject. Thus, in particular, only one actual system, the RC 4000 multiprogramming system, is described in detail. Even so the author is very open to the ideas of others, and the text is scattered with references to work of others. The author must also be credited that he is able to maintain a critica1 attitude even to his own insight and solutions. Thus the discussion of the RC 4000 multiprogramming system ends with a detailed critical review of both advantages and disadvantages of the system. As a whole the book is not of the kind that is designed to provide the complete, closed basis for a course, but rather of the kind that tries to open the student's mind to the open ends, the unsolved problems, and the divergence of views, as expressed in other literature.

In summary, this book treats the problems lying within its limitations with deep insight, a keen sense of underlying, general principles, and excellent clarity of exposition. It will be suitable for use in advanced computer science education and for the systems programmer and research worker. Highly recommended.

Peter Naur

Believe it, or not, while writing my first book, it never occurred to me that I had any competition in this country of 250 million people. Once more, youthful ignorance worked to my advantage. A year after my book came out, there were already several other texts on operating systems. Now, of course, there must be hundreds. Over the years, Karlstrom contracted with foreign publishers to translate my book into Japanese (1976), German (1977), Czech (1979), Polish (1979), and Serbo-Croatian (1982).

6

INVENTING THE FUTURE 1972–76

Can you tell me, what is Caltech? – Sunshine and palm trees in February – Wine-tasting with the dean – Driving across America – A question of priority – Two baffling problems – That sounds easy – Concurrent Pascal and Solo – Al Hartmann's compiler – The art of compromise – Getting a sore throat in Bombay – Returning to Marktoberdorf – The distraction of grants – Leaving the magic kingdom – A passion for clear thinking.

After the completion of my operating system book, it was time to decide whether we should return to Europe or stay in the United States. One day, I received a long-distance phone call from Pasadena in Southern California. The caller said: "This is Gilbert McCann from Caltech. Would you be interested in being interviewed for a faculty position?" There was a long pause at the other end of the phone when I answered: "Certainly, but can you tell me, what is Caltech?"

California Institute of Technology is probably America's most demanding college of science and engineering. With about 900 undergraduates, 900 graduates and 300 faculty, it has one professor for every three undergraduates. The students are among the top 1% in the nation. Some of them are handpicked by professors who visit high-schools around the country.

Caltech has had a major effect on the aerospace industry in Southern California. The Jet Propulsion Laboratory, America's first center for space research, is staffed and managed by Caltech. After a flight of 380 million kilometers in 167 days, the Mariner 9 spacecraft was placed in orbit around Mars on November 13, 1971. By the end of June 1972, it had taken over 7,000 pictures and mapped the entire surface of the planet.

In his yearly report for 1971/72, Caltech president Harold Brown wrote: "Our traditions and capabilities at Caltech are strongly focussed on the most fundamental matters, at the leading edge of knowledge in each discipline.

Few of us would be at Caltech if we did not believe that such efforts are of surpassing value." I remember one professor telling me, "If MIT is working on something, we are not interested. Caltech cannot do everything. So we concentrate on areas in which we are unique." This philosophy has certainly paid off. In the first century of the Nobel Prize, 27 Nobel laureates have been associated with Caltech.

Caltech students are famous for their imaginative pranks. California Boulevard divides the campus into two parts. To walk across it, you have to push a button on a traffic signal and wait for a green light. On one occasion, students unscrewed the light cover, put the green glass at the top and the red one at the bottom of the signal. Drivers now had to get out of their cars and push the button to get a (brief) green signal. During rush hour, a mile-long line of cars moved at snail's pace. Since the city administration knew that Caltech students are very intelligent, they assumed that the wiring of the signal had been changed in some way. So they sent an electrician over to check it out. He was very puzzled when he couldn't find anything wrong—until a Caltech student walked up to him and said: "Excuse me, I thought the red light was supposed to be on top of the green one." Great ideas are often simple!

On "Ditch Day," Caltech seniors ditch their classes and vanish from campus. Any senior found on campus risks being caught and tied to a tree with duct-tape. Senior students have secured their doors in elaborate ways. Underclass students must then try to get past these "stacks" and into the seniors' rooms. In one memorable instance, a senior had filled his room completely with an enormous water balloon. When some students cut a hole in the balloon, a flood of water swept them down the corridor.

⋆ ⋆ ⋆

It was my luck to arrive at Caltech just as they were starting up a department of Information Science. In September 1971, Caltech dedicated the Jorgensen Laboratory for information and computer science. The building was a gift of Earle M. Jorgensen, a Trustee of the Institute since 1957, and his wife. An elite institution like Caltech, financed by wealthy donors, would (unfortunately) be unthinkable in most European countries, due to the prevailing egalitarian attitudes and high levels of taxation.

On Wednesday, February 16, 1972, at 4 p.m., I gave a seminar at Caltech on "Structured multiprogramming."[1] I described conditional critical regions

[1]P. Brinch Hansen, Structured multiprogramming. *Communications of the ACM 15*,

and explained how they waste unpredictable amounts of processing time by reevaluating boolean conditions until they are true. I then showed how to eliminate this inefficiency by extending a programming language with queuing variables, which give the programmer complete control of process scheduling within conditional critical regions. This idea became an essential ingredient of the future monitor notation.

My trip to Southern California was a welcome break from the winter in Pittsburgh. I enjoyed walking around Caltech in February, without a coat, looking at the palm trees and the beautiful San Gabriel mountains, a few miles north of Pasadena.

Two days later, I was invited to a wine-tasting dinner at the Athenaeum, Caltech's magnificent faculty club, built in Mediterranean style with beautiful landscaping and tennis courts. This elegant building was also a gift from private donors. The first formal dinner, in February 1931, was attended by three Nobel Prize winners: Albert Einstein, Robert Millikan and Albert Michelson.

The wine-tasting was held in a large, oak-panelled dining room. My host was the sixty-year old Francis Clauser, the charming chairman of the Division of Engineering and Applied Science. He had done brilliant research in aeronautics and was a member of the National Academy of Engineering. During the dinner, we tasted German white wines and California red wines from Caltech's private wine cellar. The wines were discussed by the wine chairman, professor Harold Wayland.

Within a month, Caltech obtained letters of recommendation for me from Tony Hoare (Belfast), Don Knuth (Stanford), Butler Lampson (Xerox Parc), Roger Needham (Cambridge), Alan Perlis (Carnegie-Mellon), and Niklaus Wirth (ETH Zurich).

In March 1972, I was invited to visit Caltech again, this time with Milena and our two small children. At the Los Angeles Airport, Caltech's private limousine was waiting for us. When the driver saw Milena's winter coat, he said "You won't be needin' that here, Ma'm!" After driving thirty miles north on the Harbor and Pasadena Freeways, we reached Colorado Boulevard in Pasadena, where Caltech had reserved a motel room for us.

In the evening, Gilbert McCann and his wife Betty took us out to dinner at a family restaurant. I remember Betty asking me: "Isn't it a handicap not to have a PhD?" I looked surprised and said: "No, I don't think so." The only degree offered by the Technical University of Denmark was a five-year

July 1972.

Master's. I still have reservations about the PhD. It seems to me, that it is not a good idea to ask young people to spend some of their most creative years taking more courses, passing final exams, and doing research that fits into the ideas (and grants) of their professors.

The Caltech physicist, Richard Feynman, was even more blunt about his role as a PhD advisor (Gleick 1992):

> I do not like to suggest a problem and suggest a method for its solution and feel responsible after the student is unable to work out the problem by the suggested method...What happens is that I find that I do not suggest any method that I do not know will work and the only way I know it works is by having tried it out at home previously, so I find the old saying that "A Ph.D. thesis is research done by a professor under particularly trying circumstances" is for me the dead truth.

Back at Carnegie-Mellon, I wrote a letter informing family and friends that "California Institute of Technology has offered me a faculty position as an Associate Professor of Computer Science...I have now accepted this offer and will start working [at Caltech] on July 1."

⋆ ⋆ ⋆

Our trip across the United States and our life in California are described in Christmas letters to family and friends. We left Pittsburgh on June 1 and drove 3,100 miles (about 5,000 km) across America to Los Angeles. First, we went to Montreal, where I chaired a session on the design of operating systems at the Canadian Computer Conference. On the way, we got a last glimpse of Niagara Falls. From then on, we drove west through the states of Quebec, Ontario, Michigan, Illinois, Iowa, South Dakota, Wyoming, Utah, Nevada and California. Fifteen days in a row we traveled west, every evening towards a radiant sunset.

South Dakota made the greatest impression. Badlands: desert-like rock formations eroded by rivers below the surrounding flat country for millions of years. Mount Rushmore, where Gutzon Borglum, son of a Danish immigrant, carved 60-feet tall sculptures of George Washington, Thomas Jefferson, Abraham Lincoln, and Theodore Roosevelt on a granite mountain. The day we left Mount Rushmore, a dam collapsed, flooding the entire area and killing many people.

South Dakota is right in the middle of the old Wild West. This is where Buffalo Bill came from, and the Sioux Indians still live here. In Custer State Park we went by Jeep to see a flock of buffalos on the prairie. In Deadwood City we were shown around the Broken Boot gold mine by an old man who remembered both Calamity Jane and Wild Bill Hickock from his childhood. On Boot Hill, high above the city, we found their tombstones.

Yellowstone National Park in Wyoming is larger than Delaware and Rhode Island combined. It has an incredible collection of natural wonders in one place: hot springs, geysers, waterfalls, mountains and lakes. At one point, Milena got out of the car to photograph a black bear sitting beside the road. The bear didn't like that, so she had to run back to the car. Inside the park, we stayed overnight in a five-story alpine lodge.

The next day, we headed south, through the Grand Teton National Park. In Salt Lake City, Utah, we saw the Mormon Temple with the statue of Christ by the Danish sculptor, Bertel Thorvaldsen. In Salt Lake City, we bought a brand new set of radial tires, guaranteed for 40,000 miles. Sixty miles later, we had a flat tire in the middle of the Great Salt Desert. After mounting a worn-down spare, we crawled though Nevada until we reached Reno where we bought another tire (which later was punctured in Los Angeles).

In San Francisco, I wanted to show my family the Golden Gate bridge. But as we drove across it in the late afternoon the fog drifted in from the ocean and hid everything. We now drove south along the Pacific Ocean on the Cabrillo Highway though Monterey, portrayed in Steinbeck's novels. Finally we reached Los Angeles.

At Caltech we were met by a lady who handed us the key to a house we could live in over the summer. It was a white, two-story house on Boulder Road in the suburb of Altadena. On the second floor, a covered balcony extended from one end of the house to the other.

Two days later, I flew back to the east coast in four hours to lecture on the RC 4000 multiprogramming system at a Summer Institute of Computer Science at University of Maryland.

Greater Los Angeles: 8 million people in one place. Two murders a day and several thousand robberies. Brilliant weather (if you dare to breathe the air). The wind from the ocean blows the smog from downtown towards Pasadena, where the mountains stop it. Sometimes you could not see the Sierra Madre mountains, a few miles north of Caltech.

On the positive side, Southern California had so many attractions for the children: Los Angeles Zoo, Universal Studios, Magic Mountain, Farmers

Market, Knott's Berry Farm, Lion Country Safari, Marineland, Japanese Village, Busch Gardens, La Brea Tar Pits, Mount Wilson Observatory (and its Skyline Park with llamas, deer, goats and turkeys), the Queen Mary (the world's largest ocean liner), Disneyland, and, of course, the ocean beaches.

⋆ ⋆ ⋆

At age 27, Gilbert McCann earned a PhD in electrical engineering from Caltech. As a graduate student, a two-million-volt stroke from a surge generator paralyzed all his outer nerves and muscles for 24 hours and damaged one of his eyes permanently.

During World War II, he designed an analog computer that made it possible to shoot down most of the German V-1 rockets when they reached the coast of England. After the war, he became a faculty member at Caltech and started building a huge analog computer. By the 1950s, McCann's lab served every aircraft company in America and Europe. When the workload became too much for Caltech to handle, they spun off a commercial company, Computer Engineering Associates, with McCann as the largest shareholder.

When I met him in 1972, he was sixty years old. His manners were somewhat brusque. Since Americans habitually abbreviate first names (which we don't in Denmark), I wasn't sure what to call him. So I asked him: "Do you want me to call you Gilbert or McCann?" In an annoyed tone, he answered "I respond to either," which was not very helpful to me.

In his efforts to dominate his department, McCann could be ruthless towards faculty members whose research and financial support were independent of his own. This was obvious in his relationship with professor Fred Thompson, who had been the most promising student of the famous logician Alfred Tarski. At Caltech, Fred made a truly courageous gamble on the future of computing by working on the problem of using English for human interaction with computers. After several years of hard work, he had finished a software system in assembly language for a particular computer. At that crucial stage, McCann used his influence to support an offer from IBM to replace that computer with another one that was unable to execute Fred's program. Years of programming were wasted and Fred had to start all over again.

⋆ ⋆ ⋆

In 1961, Fernando Corbató pioneered timesharing at MIT. Ten years later, computing at Caltech was still a cumbersome affair based on old-fashioned batch processing. First, you used a noisy machine, the size of a small desk, to punch your program on IBM cards. Then you carried your deck of cards to the neighboring Booth Center for Computing, and gave it to an operator behind a counter. Several hours later, you walked back to the center and picked up your punched cards and printed output from one of the small "pigeonholes" arranged alphabetically by user names.

The overriding concern was to keep Caltech's *mainframe* computer running efficiently with as little human intervention as possible. You were not allowed anywhere near the computer equipment. Operators collected decks of punched cards from users and used a small computer to input a batch of jobs from punched cards to a magnetic tape. This tape was then mounted on a tape station connected to the mainframe computer. The jobs were now input and run one at a time in their order of appearance on the tape. The running jobs output data on another tape. The output tape was moved to a small computer and printed on a line printer. While the mainframe computer executed a batch of jobs, the small computers simultaneously printed a previous output tape and produced the next input tape. The final task of the operators was to separate the printed output manually and place it in the correct pidgeonholes.

Batch processing was severely limited by the sequential nature of magnetic tapes and early computers. Although tapes could be rewound, they were only efficient when they were accessed sequentially. And most computers could only execute one program at a time. It was therefore necessary to run a complete batch of jobs at a time and print the output in "first-come, first-served" order.

To reduce the computer time that was lost while operators changed magnetic tapes on the mainframe computer, it was essential to batch many jobs on the same tape. Unfortunately, large batches greatly increased service times from the users' point of view. It would typically take hours (or even a day or two) before you received the output of a single job. If the job involved a program compilation, the only output for that day might be an error message caused by a misplaced semicolon!

On campus there was an economic conflict between students, who needed to compile and run small programs with reasonable turn-around times to meet their deadlines for homework, and researchers, who ran large computations supported by research grants. The compromise adopted was to run large jobs at night or on weekends.

McCann directed the computing center for seven years. In his allocation of computer time for research, he apparently favored some faculty over others. Finally, a group of well-funded faculty went to the chairman of engineering and threatened to buy computer time outside campus, unless he replaced McCann, which he did.

⋆ ⋆ ⋆

I posed no threat to McCann's power and found him quite supportive of my work at Caltech. His main interest was now using computers to study the nervous system of the fly. He left it to younger faculty, including Giorgio Ingargiola and me, to develop academic courses.

Giorgio spoke English with a pronounced Italian accent interrupted frequently by an infectious laugh. His office was next door to mine. At noon, we would walk across the sunny campus and enjoy lunch at the Athenaeum. He taught a course on formal models of computation and directed a programming laboratory with student projects.

Caltech had a trimester system. In the first trimester, I taught structured programming, followed by compiler design in the second trimester, and ending with operating systems in the third trimester. These courses could be taken by students from any department. The enrollment in each course was 60–70 students, which was a large class at Caltech. I enjoyed teaching these smart kids, who raised many questions in class and often came up to me after class to continue our discussions.

The Caltech students published a booklet with candid comments about the teaching abilities of the faculty. About one professor, they wrote: "He obviously knows his stuff, and so would you—if only you could stay awake in his class!" They described my courses as "An easy way to get an A" (a viewpoint not shared by many students at Syracuse University).

My experience at Regnecentralen had taught me that professional programming is not a form of unsystematic trial-and-error. You need to think deeply until you understand exactly what you want your program to do. Before you compile a program for the first time, you should proofread it for logical consistency. And before you run it, you need to prepare a systematic testcase with output that demonstrates that every line of the program has been executed.

In this view of programming, thinking time is much more important than computer time. Programming takes place at a desk away from any computer. Today, when I see faculty and students spending hours at computer terminals, I wonder: Are they really thinking deeply, or are they just typing? Modern computing has turned us into amateur typists.

Thirty years ago, when we had to use batch processing with slow turnaround, the idea of using the computer as little as possible made all the more sense to me. In those days, students needed written permission from their instructors to set up personal accounts with limited amounts of computer time. To encourage my students to think more and compute less, I gave them less computer time than they needed to complete the compiler project. When their accounts ran out, I gave them 50 percent more, then 25 percent and so on.

At one point, Francis Clauser informed me that my well-intentioned policy had the unexpected side-effect of making some students "borrow" computer time from the accounts of other students. This was a clear violation of Caltech's Honor Code which states that no member of the community shall take unfair advantage of any other member of the community. As soon as I heard that, I sacrificed my miserly approach and gave everybody as much computer time as they needed.

The Honor Code gave us all remarkable freedom. You could, for example, tell students to go home and solve exercise 2.2.6.8 in Knuth's book on "Fundamental Algorithms"—without looking in the answers section! However, human nature being what it is, it was not easy for students to live up to the Honor Code. A survey showed that while most of them strongly supported the Honor Code, few were prepared to turn in their friends for violating it.

⋆ ⋆ ⋆

In the spring of 1972 I read about the *class* concept invented by the Norwegians Ole-Johan Dahl and Kristen Nygaard for their programming language Simula 67. Although Simula was *not* a concurrent programming language, it inspired me in the following way: So far I had thought of a monitor as a program module that defines all operations on a *single* data structure. From Simula I learned to regard a program module as the definition of a *class* of data structures accessed by the same procedures.

This was a moment of truth for me. Within a few days I wrote a chapter on resource protection for my operating system book. I proposed to represent monitors by *shared classes*. My book included a single monitor for a message buffer. Figure 1 shows it in a slightly simplified form. The shared class is a program module that combines three things: (1) the data representation of a message buffer, (2) the send and receive procedures, which define the only possible operations on a buffer, and (3) a statement that defines the initial buffer state as empty.

```
shared class buffer =
  slot: integer; empty: boolean;

procedure send(message: integer)
begin
  await empty;
  slot := message;
  empty := false;
end;

procedure receive(var message: integer);
begin
  await not empty;
  message := slot;
  empty := true;
end;

begin empty := true end;
```

Figure 6.1 The first monitor notation.

The key idea is that processes only have indirect access to the variables of the shared class. They can call the send and receive procedures, which operate on the buffer variables, but they do not have direct access to these variables. This scope rule has an important implication for program reliability: Once you have programmed and tested a shared class, it remains correct, and cannot easily be corrupted by other parts of the program.

A shared class is a notation that explicitly restricts the operations on a shared data structure and enables a compiler to check that these restrictions are obeyed. It also indicates that all operations on a particular instance must be executed as *critical regions*. In short, a shared class is a monitor type. My decision to use *await statements* in the first monitor proposal was a matter of taste. I might just as well have used the *queuing variables*, which I had proposed in 1972.

⋆ ⋆ ⋆

In the spring of 1972, I had sent Tony Hoare a copy of my book manuscript which included my monitor concept. Six month later, he submitted a paper on "A structured paging system," which was published in the fall of 1973, one month after the publication of my book. In this paper, Hoare used my shared classes and queuing variables, with minor changes, to outline an unimplemented demand paging system.

As an engineer, I had serious reservations about this paper. Nobody can have confidence in a theoretical specification of something as complicated as a demand-paging system—unless the validity of the model has been tested in an actual implementation. At Regnecentralen I had defined the instruction set of the RC 4000 computer completely by an Algol 60 program. Had we not built this computer, my hardware specification would have remained an unpublished, academic exercise.

On a personal level, I was surprised and hurt to find that, instead of citing my book as the original published source of the monitor concept, Hoare thanked me (and others) vaguely "for ideas, discussion, inspiration, and criticism on points too numerous to recall." When I pointed out that this was unacceptable, he acknowledged my invention of monitors in a tutorial, published in the following year (Hoare 1974a). However, the damage had been done, and, for years, people would continue to call them "Hoare's monitors."

Looking back, it was, of course, naive of me to publish the monitor concept in a textbook, instead of a professional journal. But I was young and idealistic and felt that my first book should include at least one original idea. It did not occur to me that researchers rarely look for original ideas in undergraduate textbooks.

At that point, I considered it premature to write a tutorial on the monitor concept. My professional standards were deeply influenced by Naur and Jensen's Gier Algol compiler, Dijkstra's THE multiprogramming system, Regnecentralen's RC 4000 multiprogramming system, and Wirth's Pascal compiler. Every one of these systems had been implemented *before* it was described in a professional journal. Since this was my standard of software research, I decided to implement monitors before writing more about them.

⋆ ⋆ ⋆

At Caltech, I started thinking about defining a programming language with concurrent processes and monitors. To reduce the effort, I decided to include these concepts in an existing sequential language. Pascal was an obvious choice for me, since I had used the language in my operating system book. I named the new language *Concurrent Pascal*. Apart from that, nothing else was obvious.

With a notation for monitors now in hand, you would think it would be easy to include it in Pascal. I had no idea of how to do this. I remember sitting in my garden in Altadena, day after day, staring at a blank piece of paper and feeling like a complete failure.

I faced two baffling problems for the first time: (1) how can you make a concurrent programming language *secure* from time-dependent behavior by using extensive compilation checks and minimal run-time checks? (2) When concurrent processes terminate, is it possible to reclaim and reuse their memory spaces without resorting to slow "garbage collection?" It took me almost two years to find reasonable solutions to the first problem and make compromises that enabled me to ignore the second one.

In September 1973, I sent Mike McKeag "a copy of a preliminary working document that describes my suggestions for an extension of Pascal with concurrent processes and monitors." This is the earliest evidence of Concurrent Pascal. In April 1974, I distributed a report on "Concurrent Pascal: a programming language for operating system design."

Concurrent Pascal extends Pascal with program modules defining monitor, process, and class types. (Since class types are related to sequential rather than concurrent programming, I will ignore them here.)

The monitor shown in Fig. 6.2 defines a single-slot buffer as a new data type. If a process tries to receive a message from an empty buffer, the monitor delays that process in a queuing variable. When another process sends a message through the same buffer, the monitor immediately continues the execution of the delayed process within the receive procedure. The process that performs the continue operation automatically returns from the send procedure. This *context switch* ensures mutual exclusion of monitor calls. Sending is similar to receiving.

Figure 6.3 defines a (trivial) process type that copies an endless stream of integers from one buffer to another.

The syntax clearly shows that each module defines a data structure and all the possible operations on it. The compiler must check that (1) every process and monitor only refers to its own variables; (2) processes interact through monitor procedures only; and (3) processes do not deadlock by calling monitors recursively (either directly or indirectly).

I now understood what I was doing. One day the president of Caltech, Harold Brown, came to my office and asked me to explain my research. After listening for half an hour, he said, "That sounds easy." I agreed because that was how I felt at the time. Caltech sure was different! This was the only time

```
type buffer =
monitor
slot: integer; empty: boolean;
sender, receiver: queue;

procedure entry send(message: integer);
begin
  if not empty then delay(sender);
  slot := message;
  continue(receiver);
end;

procedure entry receive(var message: integer);
begin
  if empty then delay(receiver);
  message := slot;
  continue(sender);
end;

begin empty := true end;
```

Figure 6.2 A monitor type.

in my life, I had the opportunity to discuss my research with a university president.

⋆ ⋆ ⋆

More than anyone else, Gordon Bell was the driving force behind the minicomputer revolution. At Digital Equipment Corporation, he was the main architect of the PDP 11, the first minicomputer that was powerful enough to support modern programming languages. When I first met him, he was spending a sabbatical year at Carnegie-Mellon. By 1975, as vice president of engineering at DEC, Gordon and his team had designed the 32-bit VAX computer, which became the standard computer for science and engineering. In 1983 he started Encore Computer, which built the Encore Multimax, a multiprocessor that I would later use for parallel programming at Syracuse University. In 1991, president George Bush awarded Gordon Bell the National Medal of Technology.

```
type copyprocess =
process(inp, out: buffer);
value: integer;
begin
  cycle
    inp.receive(inp, value);
    out.send(out, value);
  end
end;
```

Figure 6.3 A process type.

In 1970, when the first PDP 11s were delivered, over 170,000 were sold. At Caltech, McCann acquired a PDP 11/45 for his lab. Since it cost only a fraction of a mainframe computer, it was operated in open shop mode (just like Regnecentralen's Gier computer had been).

I had already made Pascal available for students on Caltech's mainframe computer. In this effort, I was assisted by Robert Deverill, a professional programmer working for McCann. At the time, no minicomputer supported Pascal. So we had to program the Concurrent Pascal compiler in Pascal and test it on Caltech's mainframe computer before moving it laboriously to the PDP 11.

An early six-pass compiler was never released. Although it worked perfectly, I found it too complicated. Each pass was written by a different student who had difficulty understanding the rest of the compiler.

From June through September 1974 my first PhD student, Al Hartmann, wrote another Concurrent Pascal compiler. His goal was to be able to compile small operating systems on a PDP 11/45 with at least 32 K bytes of memory and a slow, removable disk (about two feet in diameter). The compiler was divided into seven passes to fit into the small memory. It consisted of 8,300 lines written in Pascal and could be completely understood by one person. Systematic testing of the compiler took three months, from October through December 1974.

The Concurrent Pascal compiler was used from January 1975 without problems. It was described in Hartmann's PhD thesis (1975), later published as a monograph.

In another month Al Hartmann derived a compiler for a Pascal subset, which we called *Sequential Pascal* (Brinch Hansen 1975b). On the PDP 11, it compiled the largest pass of the Concurrent Pascal compiler in 3 min. The compilation speed was limited mostly by the slow disk.

The Concurrent Pascal compiler generated code for a simple machine tailored to the language. I borrowed this idea from a portable Pascal compiler distributed by Wirth's group (Nori 1974). My main concern was to simplify code generation. The portability of Concurrent Pascal was just a useful by-product of this decision. Twenty years later, the Java language would resurrect the idea of "platform-independent" concurrent programs. Unfortunately, Java replaced the secure monitor concept of Concurrent Pascal with *insecure* shortcuts (Brinch Hansen 1999b).

The Concurrent Pascal machine was simulated by a kernel of 8 K bytes written in assembly language. The kernel multiprogrammed the PDP 11/45 processor among concurrent processes and executed them using an efficient technique known as *threaded code* (Bell 1973). It also performed basic input/output from a typewriter, a disk, a magnetic tape, a line printer, and a card reader.

I defined the kernel in Pascal (extended with classes). Tom Zepko, a Caltech undergraduate, helped Bob Deverill hand-translate the kernel into assembly language for the PDP 11. It was completed in January 1975 and described in a report (Brinch Hansen 1975d).

The programming tricks of assembly language were impossible in Concurrent Pascal: there were no typeless memory words, registers, and addresses in the language. The programmer was not even aware of the existence of physical processors and interrupts. *The language was so secure that concurrent processes ran without any form of memory protection.*

In defining Concurrent Pascal, I made major compromises to make program execution as efficient as possible on a minicomputer that could only address two small memory segments simultaneously: (1) All procedures were non-recursive, (2) All processes, monitors, and classes existed forever, and (3) All processes and monitors were activated by an initial process.

These compromises made memory allocation trivial. The first rule enabled the compiler to determine the memory requirements of each module. The first two rules made static memory allocation possible. The third rule made it possible to combine the kernel, the program code, and all monitor variables into a single memory segment that was included in the address space of every process. This prevented fragmentation of the limited address space and made monitor calls almost as fast as simple procedure calls.

By putting simplicity and efficiency first we undoubtedly lost generality. But the psychological effect of these compromises was phenomenal. Suddenly an overwhelming task seemed manageable.

⋆ ⋆ ⋆

In January 1975, Milena and I traveled to India. The United Nations had donated funds for the country to open a center of software research and acquire a large computer at the Tata Institute of Fundamental Research in Bombay. To celebrate this event, a conference was organized for all computer science teachers in the country. Bill Wulf from Carnegie-Mellon, Rod Burstall from the University of Edinburgh, and I were invited to lecture.

Since Bombay is on the opposite side of the globe, we broke the long flight from Los Angeles in half by stopping overnight in Frankfurt, Germany. In Bombay we stayed at the famous Taj Mahal hotel. The place was swarming with international guests: oil sheiks in white garments with golden stripes, Indian women in colorful saris, Japanese tourists—and us. The hotel had French, Indian, and Chinese restaurants, as well as some sort of cafeteria. In addition there was room service twenty-four hours a day.

The first couple of days, I ate Indian breakfasts. Since I couldn't read the menu, I started from the top and ordered a new dish every morning. A typical dish consisted of some very spicy curry balls. After a few days, I had to stop this diet and see the hotel's doctor about my sore throat.

Right outside our luxury hotel, the poor were sleeping on the pavement. It was difficult to accept that small children of the same age as our children were running ahead of us begging.

After a week of lecturing, all of us flew to Aurangabad to see the famous Ajanta caves—a row of temples cut into massive rock centuries ago. Later, Milena and I traveled inland to Hyderabad and visited a company that produced a minicomputer similar to the PDP 11. I noticed that most of their peripheral devices came from communist countries in Eastern Europe. When we returned to our hotel room after dinner and turned on the lights, an army of well-fed cockroaches scampered under the bed. We called room service and they sent an employee who sprayed the room with kerosene. That night we slept with the lights on, breathing the smelly fumes.

At Caltech, I had an Indian graduate student, named Sriram Udupa. One evening, Milena and I visited his family in Bombay. They were orthodox Brahmins. They served rice and thin fine bread on tin plates covered with palm leaves. We sat in a circle on the floor and ate with our fingers, since

Brahmins regard knives and forks as unclean. The dinner was intended for the men and their sons only. The women served and watched us eat. However, they made an exception for Milena and allowed her to eat with us. It was all very dignified and made a deep impression.

After the conference in Bombay, Milena and I flew to New Delhi and on to Agra to see the famed Taj Mahal mausoleum, built by the Mogul emperor Shah Jahan (1592–1666) for his favorite wife, Mumtaz Mahal. The only thing we did that day was sit in the park in front of this master piece built of white marble with inlaid semiprecious stones. To me, Taj Mahal was as unique as Michelangelo's sculptures in Florence, Italy—one of those rare miracles, which human beings create once every five hundred years. In the basement under the building there was a marble casket with a big hole in the lid. The small detail that was missing was the 160-carat Kohinoor diamond, which an Indian prince gave England's Queen Victoria. It is now part of the British crown jewels.

The Taj Mahal reminded me of one of my favorite quotes (Bronowski 1973):

> The most powerful drive in the ascent of man is his pleasure in his own skill. He loves to do what he does well and, having done it well, he loves to do it better. You see it in his science. You see it in the magnificence with which he carves and builds, the loving care, the gaiety, the effrontery. The monuments are supposed to commemorate kings and religions, heroes, dogmas, but in the end the man they commemorate is the builder.

In spite of the exotic sights, it was depressing to visit India after growing up in a Scandinavian welfare state. Right outside the Taj Mahal, there was a small, stinking village. They said that American pilots slept in their airplanes in Calcutta to avoid seeing the hell, which was called life there. And Bombay smelled like a garbage dump everywhere (even inside the Taj Mahal hotel). Life seemed depressing even for the well-educated middle class of engineers and researchers we met.

After another stop in Frankfurt we returned home to our children, who had enjoyed staying with friends and never missed us.

⋆ ⋆ ⋆

After returning to Caltech, I wrote three model operating systems in Concurrent Pascal to evaluate the language. The modular concurrency had a dramatic (and unexpected) impact on my style of programming.

It was the first time I had programmed in a language that enabled me to divide programs into modules that could be programmed and tested separately. The creative part was clearly the initial selection of modules and the combination of modules into hierarchical structures. The programming of each module was often trivial. I soon adopted the rule that each module should consist of no more than one page of text. This discipline made programs far more readable and reliable than traditional programs that operate on global data structures.

In May 1975 I finished the *Solo* system, a single-user operating system for the development of Concurrent and Sequential Pascal programs on a PDP 11/45. The operating system was written in Concurrent Pascal. All other programs, including the Concurrent and Sequential Pascal compilers, were written in Sequential Pascal. The heart of Solo was a job process that compiled and ran programs stored on a removable user disk. Two additional processes performed input and output simultaneously. System commands enabled the user to replace Solo with any other Concurrent Pascal program stored on disk, or to restart Solo again. Al Hartmann had already written the compilers. I wrote the operating system and its utility programs in three months. Wolfgang Franzen measured and improved the performance of the disk allocation algorithm.

The Solo system was the first major example of a concurrent program consisting of processes, monitors, and classes (Brinch Hansen 1975c). It enabled us to use Sequential and Concurrent Pascal on the PDP 11/45 without going through the cumbersome batch processing at Caltech's computing center.

At Regnecentralen we had used the RC 4000 computer to implement process control programs for a chemical plant, two power plants, and a weather bureau. These real-time applications had one thing in common: each was unique in its software requirements. Consequently the programs were expensive to develop.

When the cost of a large program cannot be shared by many users, the only practical way of reducing cost is to give process control engineers a high-level language for concurrent programming. I illustrated this point by means of a real-time scheduler, which had been programmed in assembly language at Regnecentralen. I now reprogrammed the same scheduler in Concurrent Pascal.

The *real-time scheduler* executed a fixed number of task processes with frequencies chosen by an operator. I wrote it in three days. It took 3 hours of machine time to test it systematically. Writing a description took another couple of days. So the whole program was developed in less than a week (Brinch Hansen 1975e).

At the end of 1975 I wrote a *job-stream system* that compiled and executed short Pascal programs input from a card reader and output on a line printer. Input, execution, and output took place simultaneously using buffers stored on a disk. A user job was preempted if its compilation and execution time exceeded 1 minute. I designed, programmed, and tested the system in 10 days. When the system was finished, it ran short jobs continuously at the speed of the line printer (Brinch Hansen 1976a).

Each model operating system was a Concurrent Pascal program of about 1,000 lines of text divided into 15–25 modules. A module was roughly one page of text (50–60 lines) with about 5 procedures of 10–15 lines each (Table 1).

Table 1 Model operating systems.

	Solo	Job stream	Real time
Lines	1,300	1,400	600
Modules	23	24	13
Lines/module	57	58	46
Procedures/module	5	4	4
Lines/procedure	11	15	12

These examples showed that it was possible to build nontrivial concurrent programs from very simple modules that could be studied page by page (Brinch Hansen 1977).

Compared to assembly language, Concurrent Pascal reduced my programming effort by an order of magnitude and made concurrent programs so simple that a software journal could publish the entire 1,300 lines of the Solo program text (Brinch Hansen 1975c).

I tested the modules of a concurrent program one at a time starting with those that did not depend on other modules. In each test run, the initial process was replaced by a short test process that called the top module and made it execute all its statements at least once. When a module worked, another one was tested on top of it.

Dijkstra had used a similar procedure to test the THE multiprogramming system, which was written in assembly language. However, Concurrent Pascal made bottom-up testing secure. The compilation checks of access rights ensured that new (untested) modules could not make old (tested) modules fail. My experience was that a well-designed concurrent program of one thousand lines required a couple of compilations followed by one test run per module. And then it worked (Brinch Hansen 1977).

⋆ ⋆ ⋆

In his book, "Advice to a Young Scientist," the Nobel Laureate Peter Medawar (1979) wrote:

> Ever since Bacon's day experimentation has been thought to be so deeply and so very necessarily a part of science that exploratory activities that are not experimental are often denied the right to be classified as sciences at all.

Unfortunately, this obvious requirement has often been ignored in academic research on software design. In a guest editorial introducing the Solo papers, I commented on the sad state of my profession (Brinch Hansen 1976c):

> It is not uncommon for a computer scientist to make a proposal without testing whether it is any good in practice. After spending 3 days writing up the monitor proposal and 3 years implementing it, I can very well understand this temptation. It is perhaps also sometimes a human response to the tremendous pressure on university professors to get funding and recognition fast.
>
> Nevertheless, we must remember that only one thing counts in engineering: Does it work?...What would we think of mathematicians if most of their papers contained conjectures only? Sometimes an educated guess can be a great source of inspiration. But we must surely hope that the editors of computer journals will reject most proposals until they have been tried at least experimentally.

There was no doubt in my mind, that it was essential to put monitors to a realistic test before I could recommend them as a proven tool for software engineering. That was the whole purpose of developing Concurrent Pascal and Solo.

In July 1975, I described Concurrent Pascal and Solo at the International Summer School in Marktoberdorf, Germany. After presenting our system, that had been working for three months, I found it odd to hear Tony Hoare present an outline of an *unimplemented* operating system, which would be published in the proceedings of the Summer School (Hoare 1976b).

The first operating system written in Concurrent Pascal (called *Deamy*) was used only to evaluate the expressive power of the language and was never built (Brinch Hansen 1974a). The second one (called *Pilot*) was used for several months but was too slow. They were described in internal working documents only.

In a collection of his best papers, Hoare (1989) wrote:

> The ultimate test of an idea, and the one that deserves the most trust, is when it has been applied successfully in some important project...These more substantial tests have always been left to my readers.

He was apparently looking for a "royal road" to software research that would save him from being personally involved in the completion of his "model operating system." It would be another four years before Hoare's coworkers completed their own monitor language, Pascal Plus (Welsh 1979). They never developed an operating system in Pascal Plus capable of compiling and executing real programs. None of this detracts from Hoare's accomplishments as a theoretician. But as a software developer, he was obviously not in the same class as Peter Naur, Edsger Dijkstra, and Niklaus Wirth.

In a paper on programming languages for real-time control, Tony Hoare (1976a) had this to say about Concurrent Pascal:

> This is one of the few successful extensions of Pascal, and includes well structured capabilities for parallel processing, for exclusion and for synchronization. It was tested before publication in the construction of a small operating system, which promises well for its suitability for real-time programming. Although it does not claim to offer a final solution of the problem it tackles, it is an outstanding example of the best of academic research in this area.

At Marktoberdorf, Bill Wulf talked about his Hydra operating system, which used run-time checking of access rights (called "capabilities"). In a summary of the Summer School, Dijkstra (1975) wrote:

> Bill Wulf (Carnegie Mellon) and Per Brinch Hansen (Cal.Tech.) reported both on their development projects (the Hydra system and a pilot model to try out the applicability of Concurrent Pascal, respectively). Both gave eight lectures, and it was a pity that their subject were so similar: sometimes all the details became rather boring and the relative importance of operating system design became overstressed.

I remember thinking, has Dijkstra forgotten that it was his development of a *working* system that gave us confidence in the ideas behind the THE multiprogramming system?

Alas, by 1975 Dijkstra had already formed the dogmatic opinion that programming is a mathematical discipline in which there is no place for concise informal reasoning supported by other means of documentation, such as pictures, "operational" explanations (as he called them), and systematic test cases. He was no longer interested in programs that were too large to be proven mathematically correct. (This pretty much ruled out any program of more than a couple of pages).

However, Niklaus Wirth was not in doubt about what we had done:

> I thank you very much for sending me the two reports on Concurrent Pascal and on the Solo operating system. They are truly encouraging and describe solid engineering progress. This is extremely refreshing after the large heaps of papers that flood the literature and which only present new, abstract ideas, and usually make things more complicated than they were before. May I ask you to kindly send me a second copy of these valuable reports for our library. (Letter from Wirth, October 14, 1975.)

⋆ ⋆ ⋆

I have always regarded research proposals as a distraction from my work. By the time I get a small grant, I have already done so much work, that I hardly need the money! The problem is that funding agencies cannot afford to admit that awarding research grants is like active management of investment funds: program directors like to believe that they are making rational choices, but in reality they just take turns being lucky!

Most professors will never make a major discovery. So why do we grant them tenure? Because we have no way of knowing which ones will make the fundamental contributions! So we gamble on all of them. And, if only one in ten researchers change their fields, it is still an excellent investment from society's point of view. That's the reality of research. However, if funding agencies respected this fact, they would have to award grants on a random basis. Since it is difficult to acknowledge this inconvenient truth, the charade begins: faculty members make promises, they know they can't keep. And funding agencies shy away from the lone inventor and express a preference for grandiose "multi-disciplinary" research involving several departments (or even universities).

As a new faculty member at Caltech, I had applied for a grant, but was unsuccessful. One anonymous reviewer wrote: "What the world needs is parallel computers, not parallel languages!" When Tony Hoare visited me in January 1974, I asked him "what's wrong with my research proposal?" He looked at it and gave me some worldly advice: "Instead of saying that your ideas are a great improvement over those of professor X, why don't you say: This work builds on the foundation established by professor X." I followed his advice and, in September 1974, was awarded a grant of $71,200 by the National Science Foundation. Of course, by then I had already worked on Concurrent Pascal for two years without external support and would finish the research in another nine months.

I used some of the money to pay McCann for computer time on the PDP 11/45. Towards the end of my project, I decided to cut my computer time in half. The next day, McCann's secretary, Evelyn Johnson, informed me that McCann had just doubled the hourly rate for computing. However, that is the only time I personally felt that McCann misused his power. I am grateful to him for letting me use a minicomputer that made my work accessible thoughout the world.

At Caltech we prepared a distribution tape with the source text and portable code of the Solo system, including the Concurrent and Sequential Pascal compilers. The system reports were supplemented by implementation notes (Brinch Hansen 1976b).

I used part of my grant to hire a secretary, named Barbara. For the job interview, she wore a dress, high heels, and war paint. As soon as she reported to work, she wore jeans, like everybody else. By the spring of 1976, she had distributed the system to 75 companies and 100 universities in 21 countries: Australia, Austria, Belgium, Canada, Denmark, Finland, France, Germany, Great Britain, Holland, India, Ireland, Italy, Japan, Norway, South Africa, the Soviet Union, Spain, Sweden, Switzerland, and the

United States.

We charged around $100 to pay for the expenses of shipping the system tape and manuals. After a while, we had accumulated a small surplus. When I left Caltech, a check for this amount was issued to NSF. In response, the program director wrote: *"You are supposed to spend our money—not return it!"*

⋆ ⋆ ⋆

Shortly after my arrival at Caltech, Robert Cannon replaced Francis Clauser as chairman of engineering and applied science. He would move computer science at Caltech in a new direction and end McCann's dominance of the field.

Bob Cannon wanted to know how Caltech could become unique in computing. So he put together a committee that included Carver Mead (Applied Physics), Herb Keller (Applied Mathematics), John Pierce (former Head of Bell Labs and inventor of the communications satellite), Gilbert McCann and me (Information Science). He met with us regularly and kept asking the same question: "Where is the gold buried in computing?"

At our suggestion, he invited leading computer scientists from other universities to meet with the committee at Caltech. The visitors included John McCarthy (creator of the programming language LISP), Ivan Sutherland (a pioneer in computer graphics), and (at my suggestion) Tony Hoare. Mead and Sutherland hit it off immediately and started traveling around the country asking researchers the same question as Bob Cannon. They came back and told Cannon: "VLSI technology is the future of computing!" Carver Mead predicted that by 2001 transistor sizes would shrink by a factor of 100, and he was absolutely right.

In 1976, Caltech hired Sutherland to lead computer science at Caltech. Only then did Cannon deal with the political problem of McCann: He informed McCann that the Jorgenson Laboratory would now house two separate departments, named Bioinformation Systems (headed by McCann) and Computer Science (headed by Sutherland).

As an untenured faculty member I was now caught in the middle of a power struggle that was beyond my control. Carver Mead would indeed put Caltech at the cutting edge in hardware technology and would eventually receive the National Medal of Technology in a White House ceremony. Unfortunately, as far as I could tell, he had absolutely no appreciation of modern programming. He believed that programming, as we knew it then, would become superfluous once you could put a million transistors on a chip.

As soon as Mead and Sutherland decided to concentrate on VLSI technology, Caltech was no longer interested in the fundamental ideas of programming explored by Dijkstra, Hoare, Wirth, and me. Under those circumstances, I considered it professional suicide to apply for tenure at Caltech. After five exciting years, I decided to leave the magic kingdom in Pasadena. On April 30, 1976, I submitted my letter of resignation to Bob Cannon (Fig. 6.4).

⋆ ⋆ ⋆

When I left Caltech, I was 38 years old and had just completed some of my best work. In a historical paper (Brinch Hansen 1993), my colleague, Giorgio Ingargiola, described his impression of me at Caltech:

> You had this tremendous clarity about what you were doing in concurrency and languages; you made restrictive choices usually on the basis of efficiency (you list a number of such choices in your paper). You stated something like "start with as few and simple mechanisms as possible; add later only if it becomes necessary."
>
> At least in your discussions and lectures, you built programs from English statements, making explicit the invariants and refining these statements, usually not modifying them, until the program was done.
>
> I was amazed at how slowly you developed code when lecturing, and, by contrast, how fast you got debugged running code for the Concurrent Pascal compiler, and for various concurrent programs and the Solo OS.
>
> You had very little interest in computer science topics outside of the area in which you were doing research. You made polite noises, you indicated interest, but your span of attention was minimal.

My PhD student, Al Hartmann, contributed this amusing portrait:

> There are really two histories interwoven in this paper—the history of the development of concurrent modular programming, and the history of one man's ruthless quest for simplicity in design and programming. The former topic is indifferent to whether

California Institute of Technology
Information Science 286-80
Pasadena, California 91125

30 April 1976

Dr. Robert H. Cannon, Jr.
Division Chairman
Engineering and Applied Science
Caltech 104-44

Dear Bob,

I have decided to leave Caltech as an Associate Professor of Computer Science on August 31, 1976.

Computer Science at Caltech is now changing completely as indeed it should. I am sure that Ivan Sutherland will give Caltech strength in computer applications.

To make an outstanding contribution to computer engineering you will, of course also need some of the most creative minds in computer design, programming, and theory.

Although the Computer Science Committee initially declined to make offers to three of the most outstanding computer scientists: Tony Hoare, Edsger Dijkstra, and Niklaus Wirth, I hope that you eventually will reconsider this decision. The combination of any one of them and myself would have given Caltech a strength in programming that would have been unequaled anywhere else.

I have enjoyed working with Caltech students for the past four years. Together we have developed the first abstract programming language for Concurrent programming. It has now been distributed to about 60 companies and 85 universities throughout the world.

I will be very pleased to serve on the Computer Science Program Committee until I leave campus.

Yours sincerely,

Per Brinch Hansen
Associate Professor of Computer Science

Figure 6.4 My resignation letter.

one chooses to develop concurrency mechanisms for greater expressive power and more complex functionality, or, as you have chosen, to radically shorten and simplify the design of common concurrent systems. The Solo operating system is downright primitive in the sparseness of its features, representing a countercultural current against ever-increasing operating system complexity. Your style and taste in programming run almost counter to the second law of thermodynamics, that all closed systems tend towards increasing entropy and disorder.

In a world of Brinch Hansens (which may exist in some parallel dimension to ours), all systems tend towards reduced entropy over time and toward a blissful state of ultimate simplicity. Each new release of the operating system for one's personal workstation is smaller than the previous release, consumes fewer system resources, runs faster on simpler hardware, provides a reduced set of easier to use features than the last release, and carries a lower price tag. Hardware designers espousing the same philosophy produce successive single-chip microprocessors with exponentially declining transistor counts from generation to generation, dramatically shrinking die sizes, and reducing process steps by resorting to fewer, simpler device types. No one would need to "invent" RISC computing in this world, since reduced feature sets would be an inexorable law of nature.

The Concurrent Pascal project had a profound influence on Tom Zepko:

Part of the history you describe is an important part of my own history. At the time I was involved with Concurrent Pascal, I was an undergraduate and not so much concerned with the conceptual significance of the language as with learning how to build a language system from the ground up. I got the practical experience I wanted by working on the Concurrent Pascal compiler, the threaded code interpreter, and the operating system kernel. I have continued to do this same kind of work for the last fifteen years.

The concepts behind the Concurrent Pascal, the evolution of the ideas as you describe them, are clearer to me now than they were as a student. The needs you were addressing do require some years of experience to appreciate. But even as a student,

> some things left a lasting impression. What I learned from you, beyond specific programming techniques, is what I can only describe as a passion for clear thinking. This was obvious in the way you approached program design, and it was obviously the driving force behind the design of the Concurrent Pascal language.
>
> Some of the ideas embodied in Concurrent Pascal were radical at the time. That they seem less so now is a tribute to the trailblazing nature of your work. Your approach to programming and to language design now has many advocates. Structured programming, modular design, strong typing, data encapsulation, and so on, are all considered essential elements of modern programming and have found their way into a wide variety of languages. I'm thankful to have played a part in this work.

Although I did not seek tenure at Caltech, I still treasure a Christmas card from my student, Bart Locanthi, that simply said: "*Being my teacher is a tenured position.*"

7

THE END OF AN ERA 1976–84

The legendary Zohrab Kaprielian – Football game at the Rose Bowl – Creating a top department at USC – How Harvard grants tenure – The first book on concurrent programming – Doctor technices – Surviving the executive vice president – Designing the Edison multiprocessor for Mostek – United Technologies kills the project – Let no man complain to me – Brush fire and mud slides in Altadena – Magical simplicity – What we achieved.

In the fall of 1975, I started looking for a permanent job as a tenured professor. I was encouraged to apply for a new professorship in datalogy at the Technical University of Denmark, but the deadline was too short for Milena and me to decide to fold our tent and return to Europe.

I visited universities in Washington, Utah, California, Colorado, Wisconsin, North Carolina, New York and Ontario. At each university, I stayed for two days, gave a talk and spent the rest of the time meeting individually with local faculty members and joining them for lunch and dinner. These were interesting, but exhausting trips since I was constantly being evaluated by my peers.

At the beginning of 1976, these universities received letters of recommendation from Edsger Dijkstra, Don Knuth, Butler Lampson, Bill Lynch, Harlan Mills, Peter Naur, John Reynolds, and Niklaus Wirth. Of the five offers I received, I chose the University of Southern California (USC) in downtown Los Angeles.

After living in rented houses for five years, Milena and I had finally bought an idyllic ranch house in Altadena. My decision to join USC was heavily influenced by our desire to stay in Altadena and create a home of our own where our children could grow up.

At the time, computer science at USC was just a program in electrical engineering. The tenured faculty consisted of Seymour Ginsburg, an early pioneer in formal languages, and Ellis Horowitz, who was becoming a prolific writer of textbooks. In addition, there were a few assistant professors. With such a small faculty, USC had a unique opportunity to develop a first-rate department from scratch. But, to do that, they would need new leadership. The program was headed by Jack Munushian, one of the nicest people I ever met. However, as a professor of material science, he was not an effective leader of computer science.

Before accepting an offer from USC, I met with the legendary Zohrab Kaprielian, who had turned USC into a major research institution. Kaprielian joined electrical engineering in 1958. After four years, he became chairman of electrical engineering. By 1970, he was dean of engineering, and, two years later, promoted to senior vice president. Shortly thereafter, he became executive vice president of the university.

Every time Kaprielian was promoted to higher office, he kept all his previous positions. When I first met him, he was in his early fifties and was clearly in charge of the university. Solomon Golomb, professor of electrical engineering at USC, commented on the five levels of administration between himself and the president of the university: "All of them were Zohrab Kaprielian. We operated on the principle of one man, one vote, and Kaprielian was the one man who had the one vote."

Kaprielian's visionary leadership showed what Regnecentralen's Niels Ivar Bech might have achieved if he had lived in the United States. But, in contrast to Bech, Kaprielian was a ruthless politician who made many enimies, which didn't seem to bother him (as they say in Washington, "If you want a friend in this town, get a dog!"). I remember an instance, where a delegation of university administrators and senior faculty from USC visited China. One of the participants was the head of the university's news bureau. For some reason, this small office was not controlled by Kaprielian. As soon as the plane left Los Angeles International Airport, Kaprielian announced that he had "reorganized" the news bureau. From now on, it would work directly under his office. When the plane landed in Beijing many hours later, the announcement had already appeared in the Los Angeles Times, and it was too late for the former head to do anything about it.

In 1968, Kaprielian pioneered distance-education by establishing an instructional television network under Jack Munushian's leadership. The broadcasting of lectures via television made it possible for engineers to complete graduate class work at the corporate offices of Hughes Aircraft and other companies. Today, USC broadcasts over one hundred engineering courses by satellite and internet webcast to 1,000 graduate students.

Kaprielian had an uncanny talent for recognizing opportunity and making fast decisions. In 1972, he was approached by Keith Uncapher, who was director of the computer science division at RAND Corporation in Santa Monica. His research on packet-switching led to the military's Arpanet and then the Internet. He was now trying to create a university-based research institute in Southern California. The University of California at Los Angeles (UCLA) told him it would take 18 months to work out a deal. However, Kaprielian jumped at the chance, and, within a week, Uncapher was able to start the Information Sciences Institute (ISI) at USC.

My meeting with Kaprielian took place in his large office, which was kept in semi-darkness by heavy drapes. He was a short man who looked supremely confident, with a smile that was both friendly and slightly devious. He spoke so softly that you had to be very quiet to hear what he said. He asked what it would take to make me join USC. He immediately offered me a tenured position as full professor of computer science and agreed to fund a minicomputer lab with a PDP 11/55 computer, so I could continue my work with Concurrent Pascal without interruption.

He would have preferred to let computer science continue for a while as a program headed by his friend Munushian. However, under that scenario, I could not see USC becoming a national leader. He then agreed to let my appointment coincide with the establishment of a computer science department chaired by me. The department would be housed in a new building financed by Henry Salvatori, a prominent Southern California industrialist. At the end of my fifteen minute meeting with Kaprielian, I came away with a promise of two endowed chairs and fifteen professorships for computer science.

On September 1, 1976, I started working at USC. Walking around campus with my son during the Christmas break, we met Kaprielian, who gave us two tickets to the New Year's college football game at the Rose Bowl Stadium in Pasadena, which seated 100,000 spectators. At that time, these tickets were selling for $100 each on the street. My ten-year old son, Thomas, who had grown up with American football, loved every minute of the game, as we watched the USC "Trojans" defeat the Michigan "Wolverines" 14–7. As a young man in Denmark, I enjoyed watching soccer and tennis on TV. But I didn't have a clue about the rules of American football. To me it looked like each team first stuck their heads together and shouted something. Then the two teams proceeded to tackle each other and fall. The cycle of shouting and stumbling then started all over. Having an inquisitive nature, I asked my son what was going on. Thomas, who was embarrassed by his ignorant immigrant father, studiously ignored me. Twenty years later, he gave me "The Complete Idiot's Guide to Understanding Football." I struggled bravely with it before giving up. You have to grow up with a sport to appreciate the subtleties of the game.

It was now up to Ginsburg, Horowitz, and me to create a top department. Although we were hardly Caltech, I decided to act as if we were. I am sure my colleagues thought I was nuts when I said that "North America is too small a continent for our recruiting!"

We started systematic recruiting of new faculty by asking leading departments in the United States and Europe to name their best PhD students. Based on letters of recommendation from their advisors, we invited perhaps five out of twenty candidates to visit USC and give seminars about their research and be interviewed by the faculty. We then asked the dean (who, of course, was Kaprielian) to send offers to one or two candidates.

By talking to colleagues at other universities, we got a clear idea that all of us were competing for the same top candidates. The department spent years selecting faculty candidates in this time-consuming fashion. Most of them, of course, accepted offers from universities like Stanford, MIT, and Berkeley. Nevertheless, there were limits to how many positions these few universities had to offer. As long as USC tried a little harder than other departments, we would usually hire one person per year.

I remember calling Tony Hoare at Queen's University in Belfast, who recommended one of his research associates, Nissim Francez. With Hoare and others, he had done research on the semantics of concurrency. He would eventually publish a book about the tricky question of fair scheduling of concurrent processes. In 1977, Nissim became an assistant professor at USC. He is now a professor at the Technion in Israel.

In 1980, we were fortunate to attract Len Adleman, who was an assistant professor at MIT. In 2002, Rivest, Shamir, and Adleman received the Turing Award for their invention of the RSA cryptosystem.

At one point, we offered one of our endowed chairs to Zohar Manna, who was being considered for tenure at Stanford. My senior colleagues felt that we had no chance of attracting him to USC. My response was that that was for *him* to tell us! So, we wined and dined Zohar, who turned us down when Stanford offered him tenure. I believe I was correct in assuming that Zohar would add to our reputation by telling his colleagues to watch these ambitious guys at USC.

Now, hiring first-rate people is difficult enough. But evaluating them for tenure is even harder. If you start promoting weak researchers, you will some day find that most of your faculty belong to this category. Once that happens, a department has no future. Unproductive researchers naturally judge other researchers by their own standards and often see outstanding researchers as a threat to their own local reputation and influence. So average researchers have a tendency to hire and promote other researchers of the same kind.

To avoid falling into that trap, we did not consider anybody for tenure unless they had letters of recommendation from the top ten people in the world in their field. During the first seven years, we only promoted one assistant professor. In the same period, we probably hired ten others who left the department without tenure.

One of our assistant professors—I will call him Joe—was a difficult promotion case. His PhD thesis had attracted international attention. However, at the end of his term as assistant professor, his thesis still remained his best work. Since we had hired him based on his PhD work, I did not think we should also grant him tenure for the same ideas. My decision to deny Joe tenure upset Ginsburg and Horowitz, who felt that he met the standards of tenure at USC and did not want to vote against a friend they had known for several years. The human conflict between friendship and professional standards often prevents faculty from reaching tenure decisions that are in the best interest of the university.

Towards the end of my years at USC, I learned how Harvard deals with this dilemma. Sometime in 1983, I received a letter from Henry Rosovsky, the renowned dean of Arts and Sciences at Harvard University, inviting me to serve on an ad hoc committee to advise president Derek Bok on a proposed tenure appointment in Computer Science. The department had already obtained letters from leading computer scientists ranking the candidate as one of the top specialists in the world. As an experienced dean, Rosovsky (1990) knew "all too well that departments frequently present a somewhat misleading impression of enthusiasm and unanimity. . .Private and confidential letters [from each member of the departmental committee] provide a superb check on the extravagances of official case statements."

The ad hoc committee, chaired by president Bok, included dean Rosovsky and two faculty members from other departments at Harvard. I was invited as one of three computer scientists from other universities. At the ad hoc meeting, members of Harvard's computer science department faced the committee, one at a time, for about half an hour each. Prior to the meeting, the identities of the panel members had been kept confidential.

According to dean Rosovsky, "It is not at all unusual for positive witnesses to turn slighly negative under the stress of interrogation." When I was there, Derek Bok asked each witness the same question: "Is the candidate, in your opinion, a major intellect?" The witnesses were obviously well prepared to answer questions about the significance of the candidate's work. However, when asked this embarrassing question by Harvard's president, every one of them admitted that he was not a major intellect. During lunch, Derek Bok asked each of us to offer his or her personal advice. The final decision was his. In this case, he denied the department's recommendation of tenure.

Dean Rosovsky makes it clear why Harvard succeeds where lesser universities fail: "Few, if any university presidents play as great a role in the appointment process. Harvard's Derek Bok considers this role to be the most important and interesting part of his work. It is the most direct way for him to control the quality of the faculty...We seek the best scholar-teachers, and if they happen to have abominable personalities, why then we claim joyfully to suffer in the name of learning."

Five years before my visit to Harvard, I had already reached the same conclusion as the bestselling author Nevil Shute, who started a small airplane manufacturing company, named Airspeed, in the 1930s. In his fascinating "Autobiography of an Engineer" (1954), he writes:

> I would divide the senior executives of the engineering world into two categories, the starters and the runners, the men with a creative instinct who can start a new venture and the men who can run it to make it show a profit. They are very seldom combined in the same person...I was a starter and useless as a runner.

When it became evident that we were on the right track towards becoming first-rate, I discovered that I did not much like the administrative aspects of my job. I wanted to do research instead of meeting the mother of a promising high-school senior from Pasadena. And my strong desire to chart the future course of the department undoubtedly made me insensitive to the personal agendas of some of my colleagues.

In 1978 I stepped down as department head. Two years later, a survey conducted by the National Research Council in 1980 ranked computer science at USC as one of the top ten departments in the country in terms of reputation and faculty publications. Our reputation was based on the opinions of 5,000 faculty members at 228 universities (Computerworld 1983).

Seymour Ginsburg said that the main effect of my tenure as the first chair had been to raise the standards of the department and make bold decisions. As long as I was in charge, he was a strong supporter of my relentless drive for success. He was then in his early fifties and had been at USC for ten years. He had the highest possible standards in his research. In private, he would make surprisingly sharp observations about his colleagues. I remember him saying: "Professor X can't tell the difference between the great and near-great." Ginsburg was not interested in succeeding me as department chair. He was content to influence the department as a gray eminence behind the scenes. When I stepped down, he supported a succession of acting chairs of less and less academic stature and vision.

Over the years, USC dropped to a still respectable position in the second rank of computer science departments.

⋆ ⋆ ⋆

Before joining USC, I had developed the programming language Concurrent Pascal at Caltech. I knew the time was now ripe for a book on the principles of abstract parallel programming. My second book, *The Architecture of Concurrent Programs*, included the complete text of the model operating systems I had written in Concurrent Pascal (Brinch Hansen 1977). Thanks to my editor, Karl Karlstrom, it was also translated and published in Japanese (1980) and German (1981).

The mathematician Harlan Mills, who was well-known for his efforts to introduce structured programming at IBM, and his associate Roy Maddux studied my book carefully. In a review they wrote (Maddux 1979):

> This is, as far as we know, the first book published on concurrent programming. Previously, this topic has been included in books on operating systems, a closely related but different subject. Books on operating systems usually consist of a survey of such topics as processor allocation, memory management, interrupts, I/O, file systems, process synchronization, batch and multiprogramming systems, scheduling, deadlock, and protection.

> Even after reading several books of this nature, the reader is left feeling that he has been exposed to a number of complex problems yet has learned very little about designing and implementing even a modest operating system. If you have shared these feelings with us, you will welcome Brinch Hansen's most recent book.

I agree with this criticism of operating system texts. Over the years they have often been reduced to the level of "Popular Mechanics." By making such superficial courses "required," universities have a convenient excuse to lower their standards and attract marginal students. I finally stopped teaching the subject ten years ago.

Maddux and Mills were particularly pleased with the Solo system:

> Here, an entire operating system is visible, with every line of program open to scrutiny. There is no hidden mystery, and after studying such extensive examples, the reader feels that he could tackle similar jobs and that he could change the system at will. Never before have we seen an operating system shown in such detail and in a manner so amenable to modification.

In conclusion, they wrote:

> The book cannot be called a textbook; it is, rather, a thorough technical monograph that requires sustained concentration. The importance of Concurrent Pascal as the first language for concurrent programming makes the effort worthwhile.

While the book was still in production, I submitted the manuscript to my alma mater, the Technical University of Denmark, as a thesis for the Doctor Technices degree. This Danish degree (which requires no course work) is awarded about once a year to a researcher who has moved engineering and applied science a significant step forward.

After twenty years in civil engineering, my father, Jørgen Brinch Hansen, earned the Dr. techn. degree in 1953. He was then chief engineer at the internationally known engineering firm, Christiani & Nielsen. His doctoral thesis, Earth Pressure Calculation, developed the first generally applicable method for the solution of most earth pressure problems in practice.

Now, a quarter of a century later, it was my turn. In September 1976, the Technical University appointed a committee to read my thesis. The committee consisted of three distinguished Scandinavian professors of computer science: Ole-Johan Dahl (University of Oslo), Christian Gram (Technical University of Denmark), and Peter Naur (University of Copenhagen). Seven months later, they submitted a five-page evaluation to the university recommending that it be accepted for the defense of the technical doctoral degree.

In January 1977, I satisfied one of the official requirements for the degree by asking Prentice Hall to ship 200 copies of the thesis to the Technical University, for general distribution to various places (I have no idea where they went).

The leading Danish newspapers, Berlingske Tidende and Politiken, interviewed me about the practical significance of my work and announced the time and place of the official defense of my thesis. This event took place on January 23, 1978, in the largest auditorium at the Technical University of Denmark. It began at 2 p.m. and lasted about four hours. The Swedish computer magazine Data wrote (February 1, 1978):

> All [three opponents, Dahl, Gram, and Naur] gave the doctoral candidate an extremely positive reception. In the auditorium, where no less than 400 people were present, the spirit of Niels Ivar Bech seemed to be present, while his associates engaged in discussion at a higher level. (English translation by me.)

Each opponent gave a summary and evaluation of my thesis. In his remarks, Peter Naur said:

> The text is characterized by great clarity and convincing arguments. In my opinion, it culminates in the description of the Solo operating system, the job stream problem, and the real-time scheduler. In these chapters, the description proceeds fluently with an apparent ease that is quite overwhelming. Here, above all, Per Brinch Hansen demonstrates his mastery. Everything looks so easy, as it always does in the hands of the master. For those who will attempt to do the same, it will probably turn out to be fraught with problems and traps, but as a source of inspiration, these sections will be of enormous value. For the work as a whole, the significance of these sections is that they demonstrate the value of the new programming language concepts they are based on. The discussion of these new concepts [monitor and process types] must also be praised for its convincing clarity.

So far, so good. However, as an opponent, Naur was also expected to point out weaknesses of my thesis. It was not by chance that he focussed on the definition of the programming language Concurrent Pascal. He said:

> I see that on page 245 you define a process type as a form of data type, while on page 236 you define a data type as a set of values. Can you tell me in what sense a process is a set of values?

I was well aware that my first language report was the weakest part of my work. Since I have always made it a rule never to defend the indefensible, I turned to Peter and said:

> The English computer scientist Tony Hoare once said that the Algol 60 report, which you wrote, was a considerable improvement over its successors. Well, my report is one of the successors.

Everybody laughed and Peter smiled saying: "You got it!"

It would be another seven years before Tony Hoare (1985) introduced a mathematical model which identifies a process with all the possible sequences of actions (known as "traces") in which it can participate. In 1974, Roy Campbell and Nico Habermann had introduced an early notation, called "path expressions," for this idea.

If I had to single out an event that marked the peak of my research career, it would be that day in 1978 when I became the first computer scientist to receive the Dr. techn. degree. I was then 39 years old and had worked at the cutting edge of operating systems and concurrent programming for ten years. Never again would I have a similar streak of luck.

In most instances, scientific creativity peaks around age forty. Nobody knows why it should be so. In his study of *Genius, Creativity, and Leadership*, Simonton (1984) suggests that "True creativity demands the right combination of enthusiasm and experience. . .Enthusiasm tends to peak rather early in life and then steadily decline, whereas experience gradually increases with age. . .Thus the age-40 floruit is a consequence of this uniquely balanced juxtaposition of youth's rapture and maturity's sagacity."

Speaking of honors: When I joined USC, Kaprielian offered me an endowed chair. I didn't think that my first act as department head should be to accept an honor for myself. So I suggested that he offer it to Seymour Ginsburg. In the spring of 1978, Ginsburg became the first Fletcher Jones Professor of Computer Science. Kaprielian, who was not used to being turned down, never offered me another endowed chair.

Three years later, the new president of USC, James Zumberge, removed Kaprielian from his position as executive vice president. Driving home from a New Year's party, Kaprielian suffered a fatal heart attack and crashed through the living room of a house in Beverly Hills. The following year, on September 15, 1982, I was named the first Henry Salvatori Professor of Computer Science at USC.

My last act as chair of computer science was to nominate Tony Hoare for an honorary doctorate at USC. My colleagues would have preferred to honor an American computer scientist, but, since I was the chair, they went along with my nomination. In 1979, Hoare became the first computer scientist to be awarded the degree of Doctor of Sciences *Honoris Causa* by an American University.

⋆ ⋆ ⋆

In 1978, L. J. Sevin, chairman of Mostek Corporation in Dallas, and one of his young engineers, Steve Goings, paid me a visit at USC. Mostek was then the world's largest manufacturer of semiconductor memories. Goings, who had read my book, "The Architecture of Concurrent Programs," had suggested to L.J. that they should meet with me to discuss what Mostek should be doing in computing.

Mostek predicted that VLSI technology soon would make it possible to put an IBM/360 mainframe computer on a single chip! So L.J. wanted to know if I thought it would be a good idea for Mostek to develop such a powerful chip. As far as I could see, the problem was not the chip, but the notoriously unreliable IBM software that would run on it. Once they had sold a large number of System/360 microprocessors, I feared their customers would expect them to correct errors in OS/360—a task that taxed even the expertise of IBM itself.

At the 1969 Nato Conference on Software Engineering, Martin Hopkins, IBM, admitted that, "We face a fantastic problem in big systems. For instance, in OS/360 we have about 1000 errors in each release and this number seems to be reasonably constant."

In a letter to me, thirty years later, Dijkstra wrote:

> I always felt that IBM's inability to make a decent operating system for its own hardware played a significant role in the recognition of the "software crisis" in 1968. In that sense, OS/360 has been significant.

Before our meeting, Steve Goings had already told L. J. Sevin that he did not think there was much future in designing chips that emulate obsolete computer architectures:

> I urged that we abandon the IBM emulator, and create a microprocessor that could work effectively in a multiprocessor architecture, and provide for more direct support of high level programming languages. Further, we needed the help of top level software engineers in the field, He asked me if I had anyone in mind. My reponse was, "I do not know him personally, but I have a high regard for the publications of Per Brinch Hansen." (Letter from Steve Goings, July 19, 2004.)

That is when they decided to see me. During our conversation, I said, "While you are here, I would like to tell you about an inexpensive multiprocessor I have proposed."

A multiprocessor consists of identical processors that run in parallel and communicate through common memory. The challenge is to make sure that the common memory does not become a serious bottleneck for the processors. When Bill Wulf and Gordon Bell (1972) developed their pioneering C.mmp multiprocessor at Carnegie-Mellon, they made the bold decision of making every memory location accessible to every processor. They did this by connecting sixteen PDP 11 minicomputers to sixteen independent memory modules via a crossbar switch. Since a switch that connects n processors to n memory modules has a hardware complexity of order n^2, this is a rather expensive solution.

What I outlined was a simpler multiprocessor with two to ten microprocessors. The processors would have their own local memories and would share a single common memory (Brinch Hansen 1978b). This architecture was intended for dedicated real-time applications programmed in a language with concurrent processes and monitors. Each processor and its local memory would be dedicated to the execution of a single process. The processes would communicate by means of monitors stored in the common memory.

Since a monitor only performs one operation at a time, it is per definition a bottleneck in a concurrent program. To make a concurrent program as fast as possible, a wise programmer will make sure that each process spends most of its time accessing its own code and local variables and uses as little time as possible inside monitors. If that assumption was correct, it would make sense to replace the crossbar switch with a single common memory module.

This would make the complexity of the multiprocessor proportional to the number of processors.

While I was explaining all of that, L. J. Sevin looked immensely bored. When I was finished, he turned to Goings and said: "I think we ought to build his machine!"

In the fall of 1978, Mostek started a research project managed by Steve Goings. As project consultant, I would be responsible for designing a concurrent programming language and a multiprocessor architecture tailored to the language. A team of Mostek engineers, headed by Nick Matelan, would be responsible for the hardware implementation. Several times a year, Goings, Matelan, and I would spend a weekend at the Pasadena Hilton Hotel discussing technical details.

My first task was to develop a concurrent programming language, named Edison, which included concurrent statements and conditional critical regions. It was as powerful as the combination of Pascal and Concurrent Pascal, but much simpler (Brinch Hansen 1981). At my suggestion, Mostek signed a consulting agreement with Peter Naur to review my definition of Edison. Naur made almost no comments about my choice and design of language features. His main concern was the clarity of the language report. I would write a complete draft of the report and Naur would then point out what the weaknesses were and suggest broadly how they might be removed in my next draft. Between January 1979 and September 1980, I wrote four versions of the Edison report from scratch. About the second version, Naur wrote (Brinch Hansen 1981):

> The report is a vast improvement over the previous version in clarity, consistency, and completeness. The remaining weaknesses, described below in detail, are to a large extent concerned merely with finer matters of conceptual clarity.

After this pleasant introduction, he went on to enumerate 79 conceptual problems. The writing of the Edison report was far more difficult and time consuming than the selection of language features and the design of the first compiler.

A key element in the development of the Edison multiprocessor was our decision from the beginning to define the function of every piece of hardware by an equivalent Edison program. Such a description was far more precise than a mixture of circuit diagrams, timing examples and prose. Not only could an Edison algorithm be subject to compile-time checking of consistency, but it could also be tested on an existing computer. More importantly, Edison would serve as a formal specification language that was understood by both hardware and software engineers.

In the spring and summer of 1979, I wrote a report (revised after discussions with Nick Matelan) that defined the multiprocessor architecture by Edison algorithms. These algorithms closely mirrored the exchange of data and signals that took place in interactions between processors, memories, busses, peripheral devices, and arbiters. We also specified how to build a distributed system as a cluster of multiprocessors.

The specification of hardware by means of algorithms was not yet widely used in the computer industry and certainly not for something as complex as a multiprocessor with a hierarchy of bus lines. Our Edison algorithms enabled the hardware engineers to discover several logical errors in my original proposals of buslines *before* the circuits were implemented. In at least one case, they also made a hardware designer realize that an intermittent error in a circuit design was caused by his deviation from my Edison specification of what the circuit was supposed to do.

In a letter to me, Matelan wrote: "We got a 4-node Edison multiprocessor working, Monday morning, May 12 [1980]." During the summer, I wrote the first Edison compiler in Edison and tested it on a PDP 11/55 computer at USC. When I demonstrated this compiler for Mostek in November, L. J. Sevin agreed buy it for $100,000. However, before I had a chance to see the multiprocessor or sell my compiler, United Technologies bought Mostek and cancelled the multiprocessor project. Many years later, Steve Goings told me that the technical documents for the Edison project disappeared on that unfortunate occasion.

A few years ago, I discovered that Nick Matelan had started his own company, Flexible Computer Corporation, which developed a multiprocessor called the Flex/32. Since Matelan (1985) neither acknowledged Mostek nor me, it is difficult to say how much this machine owed to the Edison multiprocessor. At one point, Purdue had a 7-processor Flex configuration, while a 20-processor machine was installed at the NASA Langley Research Center in Virginia. Matelan's company no longer exists.

At the beginning of the Edison multiprocessor project, L. J. Sevin told me that Mostek was funding a dozen research projects that gambled on future computer technology. He expected most of them to fail. Even though nothing came of our project, I am glad I met L.J. who always thought big. During a dinner at a Dallas restaurant, he asked if I would be interested in starting a software research center for Mostek. "Can I do it anywhere in the world?" I asked. "We hope you will do it in Texas," he said, "but, if you prefer, you can also build it in Denmark." I said I would need to think about it before I gave up my tenured university position. L.J. responded with the immortal words, "Let no man complain to me about the size of his balls!"

In the end, L.J. did all right. When United Technologies bought Mostek, he joined a venture capital partnership and invested his money in a little-known manufacturer of PCs named Compaq. Steve Goings did some consulting work for L.J. The first day he walked into this new business, L.J. introduced him around and said, "Steve is a pioneer. You can always tell who the pioneers are. They are the ones with all the arrows in their backs."

⋆ ⋆ ⋆

Our one-story house on 1351 Pleasant Ridge, Altadena, was located at the top of a steep street in the foothills of the Sierra Madre mountains, at the entrance to a steep, narrow canyon covered with oak trees and brush. The house was built in Spanish style with hardwood floors and beam ceilings. The living room was dominated by a large fireplace embedded in a brick wall. It had full-length floor-to-ceiling windows facing south. On the opposite wall, sliding doors led to a covered patio facing the canyon. After we had furnished it with pine furniture and rya rugs, our home looked like a cozy hunting cabin. On a clear night, we had a panoramic view of the endless carpet of lights in Pasadena and beyond. On a smoggy day, it looked as if we lived above the clouds.

Our patio was like a zoo with frogs, lizards, rabbits, hummingbirds, and a small pond with Japanese koi fish. Sometimes a deer would come all the way down from the canyon, and at night packs of coyotes would howl nearby in the mountains. One evening, Milena went to bed without closing the screen door to the patio. I found her sleeping while a tarantula the size of a child's hand was crawling on the floor. Although they look dangerous, tarantulas are fairly harmless and won't bite if you leave them alone. Their painful bite is apparently no worse than a bee sting.

We were much more concerned about the rattlesnakes that occasionally found their way into our garden looking for water and mice. The previous owner was a doctor who kept snake serum in his refrigerator. He said, "Don't worry about snakes—the kids always spot them first!" And he was right, they did. Before getting into my car in the garage, I always knelt down to see if a rattler was hiding under the car. I killed several small ones by cutting their heads off with a shovel. However, when I found a big rattler, as thick as my arm, on the driveway, I called the fire department. The fire fighters drove up to our house in a huge fire engine with flashing lights and sirens on, killed the snake and cut the rattle off as a present to my son.

Although we learned to live with the snakes, I could never get used to the black widows—the poisonous spiders that were found throughout the house, in our potted plants, behind book shelves, and underneath our beds. Fortunately, none of us was ever bitten by one.

The most dramatic event we experienced in Southern California was a natural catastrophy that nearly ruined us and almost killed me. (However, as the Danes say, "nearly" and "almost" never threw anybody off his horse.)

Sometime in September 1979, children set fire to a trash can several miles from our home. This started the largest brush fire in the area in forty years. The mountains were covered with dry vegetation that burns like torches. When it starts burning, there isn't much anyone can do other than waiting until it burns itself out.

Our house was completely surrounded by brush that grew right down to the edge of the patio. On the first day of the fire, we saw smoke rising above the mountains east of our house. During the night, the flames reached the top of the mountains and started creeping down towards the homes of our neighbors. Soon the whole mountain side was burning with a faint crackle lighting the terrain with a deep red color. It was both beautiful and cozy unless you lived right next to it.

The next day, the fire slowly burned away from us and moved up into a large canyon that was separated from our small canyon by a mountain ridge. In the middle of the night I woke up and saw, for the first time, the sun rising in the north with a shiny glow high up in our canyon.

On the morning of the third day, the fire was burning through our canyon towards our house. For several days, the air above Los Angeles had been stagnant making the air pollution worse and worse. However, since the fire burned downhill in the still air, it moved relatively slowly. Late in the afternoon, the fire reached our property. On this quiet day, it was no problem for the firefighters to prevent it from spreading to our house. Two fire fighters spent the night in lawn chairs in our driveway, guarding a tree that, if it were to catch fire, would explode and burn like a torch.

When it was finally contained, the brush fire had destroyed 30,000 acres and had occupied 3,000 fire fighters for a week. Where our house had been surrounded by green hillsides we saw only scorched ridges covered with soil and gravel.

In California, it often rains for days in December and January, as storms move in from the Pacific. Where we lived, most of the runoff water from the hills had been stopped by brush and tree roots before it reached the bottom of our canyon. However, when it rained heavily, some water would flow through a small storm drain under our house. This storm drain was owned by the county who was responsible for maintaining it.

However, since the mountains were now stripped of vegetation and covered with debris, it was a virtual certainty that our house would be destroyed by mudslides during heavy rain. The only hope of saving the house was to build a deflector wall that would direct the debris flow across the patio continuing past the garage and through the driveway onto the street (Fig. 7.1). But we were unable to find an engineering firm that would help us calculate the dimensions of such a wall. The consulting firm of Alderman, Swift & Lewis described their main concern:

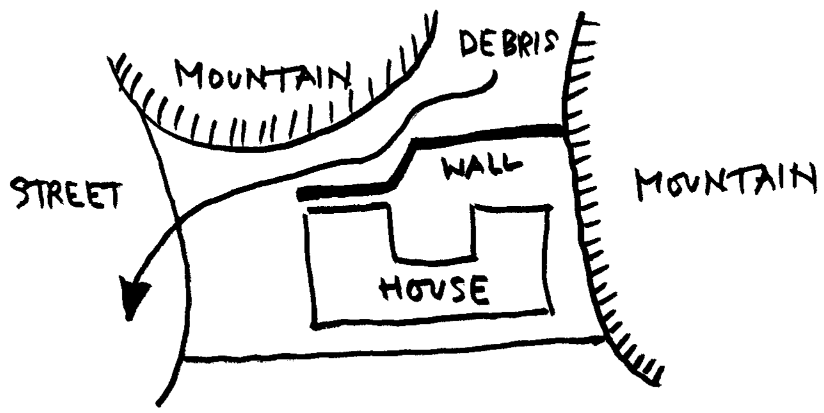

Figure 7.1 Debris flow in Altadena.

> Because of the location of your house, patio and garage, it would be necessary for a diversion wall to alter the direction of flow nearly 90 degrees in a very short distance. Unlike clear flow, debris flow cannot be diverted this quick. As a result, the diversion wall may be topped.

Two professors of civil engineering at USC volunteered to help us for free without any guarantees. In the meantime I started calling local building contractors and discovered again that none of them were willing to take the risk of being sued. I finally found a Danish bricklayer, Knud Balling, who put me in contact with an American builder, Ed Sylvis. With no contract other than a handshake I agreed to pay this man and his Mexican crew on an hourly basis, without knowing in advance what it would end up costing.

Since there was no time left to worry about minor details, such as a building permit, we decided to call our wall a “timber and pipe fence.” When it was finished in late November, it was 150 feet long and 6 feet tall. It was built of 2×6-inch lumber bolted to fifty 12 foot steel posts with a diameter of 6 inches, embedded in six feet of reinforced concrete. After I gave it several coats of dark red paint, it didn't look all that bad. However, we were still listed by the police as one of a dozen families who would need to be evacuated to save our lives during a big storm. In the last forty years, such storms had typically occurred twice a year.

Nothing happened in December. But in January 1980 it started raining for days. Early one morning, Milena was driving home after taking the children to school, when the house suddenly started shaking, as if a large helicopter was hovering right above it. From the bedroom window, I saw that our red wall suddenly had turned grey on the side facing the house. I ran outside in the rain. On the other side of the wall, an avalanche of mud and boulders from the canyon had piled up close to the top of the wall and was flowing through our driveway and down the steep Rubio Vista Drive into the gardens of a dozen other homes. Milena called from the local police station and told me that she was unable to drive up the street.

Although the Flood Control District had no funds to build our wall, they were still technically responsible for keeping their tiny (useless) storm drain open. They used this as an official excuse to send one of their bulldozers and an army of dump trucks to prepare us for the next storm. For twenty four hours, the bulldozer and the trucks worked continuously to remove debris behind the wall.

The storm had dumped two inches of rain. While this went on, an even bigger storm (which occurs about once every three years) was moving towards Southern California from the Pacific. Students and faculty from USC came to our house and piled hundreds of sandbags around the house inside the wall, while three carpenters covered all doors and windows with plywood. At lunch time, I went to the drive-in entrance of Burger King and casually ordered “20 whoppers, please.”

The next morning, January 11, the headline on the frontpage of the Pasadena Star-News read “Altadenans may evacuate.” The article included two photos showing how “Altadenan Per Brinch Hansen readies for rain” while “Storm clouds hang over L.A. as seen from Pleasantridge Drive in Altadena.” According to Bill Hardy of the Flood Control District, “If any house in Altadena is in danger, it is 1351 Pleasant Ridge, which is directly in the mouth of a gorge.”

We now got four inches of rain, and again the wall held up. This happened several times over the next four months. When a big storm was forecast, we spent the night in a motel, while flood control workers protected our property against looters. When spring finally came, we had survived the worst rainy season in ten years. In May, we were able to remove the plywood and half-rotten sandbags and let the daylight into our rooms. After some minor repair, the house looked as good as new.

Had we not built the wall, I would almost certainly have been killed in the ruins of our house during the first mudslide and my family would have been ruined. When the wall was finished, Milena and I cooked steaks on the patio for Ed Sylvis and his crew. Over a beer, I said to Ed: “You knew I was completely at your mercy—how come you didn’t take advantage of me?” He answered: “I can always make more money, but I can only lose my reputation once!”

⋆ ⋆ ⋆

While these natural catastrophies threatened our home, the pioneering era of concurrent programming was coming to an end. It is time to look at what we had achieved.

In the first survey paper on concurrent programming I had cited 11 papers only, written by four researchers. None of them described a concurrent programming language (Brinch Hansen 1973b). The development of monitors and Concurrent Pascal started a wave of research in concurrent programming languages. Fifteen years later, there were close to 20 monitor languages and 100 languages for distributed computing (Brinch Hansen 1993, Bal 1989).

Two of my former Ph.D. students recalled their experience of working with Concurrent Pascal at USC (Brinch Hansen 1993):

> *Jon Fellows*: The beauty of the structures you created using Concurrent Pascal created an aura of magical simplicity. While

> working with my own programs and those of other graduate students, I soon learned that ordinary, even ugly, programs could also be written in Concurrent Pascal... My current feeling is that the level of intellectual effort required to create a beautiful program structure cannot be reduced by programming language features, but that these features can more easily reveal a program's beauty to others who need to understand it.
>
> *Charles Hayden*: I think the significance of the system was ... that one could provide a protected environment for concurrent programming—a high-level language environment which could maintain the illusion that there was no "machine" level. It was remarkable that through compile time restrictions and virtual machine error checking ... you could understand the program behavior by looking at the Pascal, not at the machine's registers and memory. It was remarkable that the machine could retain its integrity while programs were being developed, without hardware memory protection.

In the fall of 1981, when Microsoft had just implemented DOS in assembly language for the first IBM Personal Computer, my students and I had already used high-level languages for seven years to write portable single-user operating systems for minis and micros, and had published the complete program text of some of these systems. Charles Hayden wrote no less than three operating systems on his own: a single-user system, a multiuser system, and another one with a Unix-style I/O system.

During the summer of 1981, I tested a single-user operating system for a PDP 11/23 microcomputer, written in the programming language Edison. The Edison system was able to compile itself and its compiler in 56K bytes of memory using two 8-inch floppy diskettes of 250K bytes each as the only form of backing store.

In the fall of 1982, I moved the Edison system to the IBM PC by rewriting a kernel of 4K bytes in assembly language. The Edison-PC system compiled itself in a 64K byte memory using dual $5\frac{1}{4}$-inch floppy diskettes.

In 1983, I published a book about the Edison system, entitled *Programming a Personal Computer*. According to Peter Naur (1984):

> In this book the author carries through an entirely fresh attack on the problem of programming language and operating system

> design, the incentive being the availability of microcomputers. Within the compass of the 388 pages of the book, the author manages to present in every detail: Edison, a new programming language suitable for concurrent programming; Edison system, an operating system; Edison code, an intermediate language designed to be suitable as intermediary between Edison and the machine languages of microcomputers; Alva, an assembly language for PDP 11 computers specially designed for supporting Edison; the complete programs for implementing each of these languages and systems; extensive discussions of the argument that lie behind the designs adopted throughout. While most of the detailed argumentation of the presentation is found similarly in the author's earlier work, the new development serves as a convincing demonstration of the power of the principles and methods employed in solving a problem having basically new constraints, those of a microcomputer.

It was now obvious to any casual observer that a programming revolution had taken place in programming languages and operating systems.

Looking back, what am I most proud of? The answer is simple: We did something that had not been done before! We demonstrated that it is possible to write nontrivial concurrent programs exclusively in a secure programming language.

In retrospect, the monitor concept was the first example of *object-oriented concurrent programming* (although I never used that term). However, the particular paradigm we chose (monitors) was a detail only. The important thing was to discover if it was possible to add a new dimension to programming languages: *modular concurrency.*

Every revolution in programming language technology introduces abstract programming concepts for a new application domain. Fortran and Algol 60 were the first abstract languages for numerical computation. Pascal was used to implement its own compiler. Simula 67 introduced the class concept for simulation.

Before Concurrent Pascal it was not known whether operating systems could be written in secure programming languages without machine-dependent features. The discovery that this was indeed possible for small operating systems and real-time systems was far more important (I think) than the introduction of monitors.

Monitors made process communication abstract and secure. That was, of course, a breakthrough in the art of concurrent programming. However, the monitor concept was a detail in the sense that it was only one possible solution to the problem of making communication secure. Today we have three major communication paradigms: monitors, remote procedures, and message passing.

The development of abstract language notation for concurrent programming started in 1971. Fifteen years later Judy Bishop (1986) concluded:

> It is evident that the realm of concurrency is now firmly within the ambit of reliable languages and that future designs will provide for concurrent processing as a matter of course.

So passed an exciting era.

8

DANISH INTERLUDE 1984–87

Student democracy and teaching in Denmark – Danish industry uses Concurrent Pascal – Consulting for GN Corporation – Rocking the boat.

After fourteen years in America I suffered a first-class attack of homesickness for Denmark. I had written the book on operating systems that was my original reason for coming to the United States. And, through my work in operating systems and concurrent programming, I had fulfilled my dream of making fundamental contributions to a new field. As Americans say, "Been there—done that!" Now what?

At this point in my life, I longed to return to Denmark and continue the life I had left behind. If you think I was deceiving myself, why, you are absolutely right. But did I listen to reason? Nope, I just had to go back and discover for myself that my lost youth was indeed, well, lost. Milena, who had no wish to leave the United States, was surprisingly understanding. She told our children, Mette and Thomas: "Dad is unhappy. He has to get Denmark out of his system." Looking back, I find it unforgiveable that I turned their lives upside down, just as my son was close to graduating from high school, and my daughter was ready to start her university education.

In the fall of 1983, I applied for a new professorship in datalogy at the University of Copenhagen. Danish universities do not ask for confidential letters of recommendation from colleagues at other universities. Instead I was asked to submit copies of my best books and papers to an ad hoc committee consisting of professors Peter Naur and Peter Johansen from University of Copenhagen and Kees Koster from the Catholic University of Nijmegen, The Netherlands. After studying my work for four months, the committee submitted a six page summary of my career and scientific contributions recommending that I be appointed to the vacant professorship.

In an egalitarian society like Denmark, you were constantly aware, as Vartan Gregorian put it, "that people think they are equal, and that whether or not that is accurate is irrelevant." All professors received the same salary independent of their achievements and the terms of their appointments were not negotiable.

Nevertheless, I had one concern before I was ready to accept a position in Denmark: At USC I had recently created a personal computer lab with 40 IBM PCs that enabled me to teach a course in which students wrote single-user operating systems in the programming language Edison. Would it be possible for me to establish a smaller PC lab at the University of Copenhagen?

Since I was used to negotiating directly with high-level administrators in the United States, I wrote a letter to Bertel Haarder, Danish minister of education, asking for his help in providing funds for a handful of IBM PCs. During a visit to Denmark, I also met him privately. Haarder, who after all was a politician, promised to find money for the PCs.

I soon learned that bypassing the normal channels of communication just isn't done at Danish universities. On April 27, 1984, the following item appeared in the newspaper Politiken (translated into English by me):

> *Gets job in spite of concerns.* This summer the Institute of Datalogy at University of Copenhagen (DIKU) will increase the number of professors from two to three. The favorite for the position is one of the leaders of the international world of computing, dr. techn. Per Brinch Hansen. He is currently professor of computer science at University of Southern California.
>
> The 45-year old Danish engineer is number one on the institute's confidential list of preferred applicants. . .However, it raised some concern at DIKU, when Per Brinch Hansen as a condition for his appointment practically demanded that the institute make six IBM computers of a specific type available. DIKU, which has one of them, now plans to acquire two more in the immediate future and three more later on.
>
> In 1978, the internationally known Danish computer scientist wrote—as the first in the world—a doctoral thesis about the special problems posed by computer programs that execute many tasks at the same time.
>
> Further down on the institute's list are people from DIKU's present staff.

Three months later, I received official notification that "We, Margrethe the Second, Queen of Denmark, by the grace of God, makes it known that We hereby, from August 1, 1984, appoint professor, doctor technices Per Brinch Hansen, who is a Danish citizen, as professor with permanent appointment."

DIKU was a child of the student uprising in the late 1960s. For generations, every university department had been headed by a single professor who had the final say in all matters concerning curriculum, examinations, research, and appointments. Inspired by the student riots in the United States and France, Danish students and instructors staged demonstrations in the spring of 1968, shouting "Down with the tyranny of professors," "Student participation NOW," and "Research for the people."

Everybody agreed that something had to be done. The politicians, who knew how to count votes, were well aware that the number of students far exceeded the number of professors. In May 1970, the Danish parliament passed a new Statute of Administration ("styrelsesloven"). From now on, every university department would be governed by a council ("institutråd") with equal representation of teachers, students, and staff members.

In 1969, Peter Naur became the first Danish professor of computer science ("datalogy"). The following year, associate professor Edda Sveinsdottir became the first democratically elected head of DIKU. The new department couldn't have made a better choice: Edda, who was friendly and helpful, attacked problems with boundless energy. Her innovative research in three-dimensional scanning of the brain ("computerized tomography") and Peter Naur's pioneering work in compiler development helped establish DIKU's tradition as a center of applied computer science. In 1987, Edda became professor of datalogy at Roskilde University.

In my time, DIKU's council consisted of sixty teachers, students, and staff members. Once a month, the council met for a couple of hours. Consequently, every teacher had only a couple of minutes to express personal opinions about departmental issues and had only one of sixty votes, independent of personal ability and achievements. Since teachers tended to vote as individuals, students often prevailed by voting unanimously.

When a group of people have the right to decide anything in democratic fashion, they naturally concentrate on problems they understand and ignore less familiar issues. At one departmental meeting, the discussion centered around problems with our printing office, the sun shades in the cafeteria, the reliability of the elevator, the use of office space, and the absence of teachers from these meetings "which they are obliged to attend." Under the new system of governance, the focus of discussion was no longer innovative teaching and research, but job satisfaction.

My first impression was that a handful of teachers did interesting research, while the majority were unproductive. Looking through the Science Citation Index for the previous five years, I found that worldwide only one third of our faculty was cited in the works of other researchers.

As the Harvard dean, Henry Rosovsky (1990), has pointed out "Not everything is improved by making it more democratic:"

> The limiting case exists in some European universities where the practice of "parity" was born in the 1960s. Power over virtually all decisions came to be equally shared between students, faculty, and employees—and not infrequently the government. The educational results have been disastrous. Academic standards declined and a sense of mission was lost.

However, Edda Sveinsdottir felt that the endless discussions, that were necessary to resolve even minor issues, were worthwhile because they often led to decisions that everyone could live with. Personally, I found it pointless to listen to student representatives, who had never done research, but nevertheless believed that the problems of society could be solved by trying to control the unpredictable nature of research. If people had told Thomas Edison what to do, he might have invented a faster telegraph key instead of the electric light.

Needless to say, the reality did not always resemble the politically correct utopia. When I suggested to my colleagues that I would like to revise the operating system course completely, their immediate reaction was "Please don't! If we change even one course, the militant students will use that as an excuse to demand a revision of the entire curriculum." In the end, I simply had to accept that the Danes had developed an educational system that valued cooperation and peace of mind more than individual pursuit of excellence.

It was a joy to teach Danish undergraduates. It would, of course, be unfair to compare them to Caltech students. But, thanks to the excellent Danish high schools, they were, on the whole, better prepared than most American undergraduates at USC or Syracuse University.

With the help of an army of teaching assistants (TAs), I taught a course on compiler design for a class with over 200 students. In the days before email, it was impractical for me to help that many students individually. Instead I selected my student, Birger Andersen, to be my chief TA. Birger, who is now Associate Professor at the Copenhagen University College of

Engineering, gave me feedback from 15 regular TAs, each of whom was responsible for helping 15 students. This arrangement worked fine. Although the TAs were well-meaning, their democratic desire to be involved in all decisions was annoying to someone who had twenty years of experience in compiler design. My indirect communication with TAs, through Birger, saved me from having weekly discussions with fifteen people about my choice of textbook and philosophy of teaching.

I used my own textbook, with the modest title *Brinch Hansen on Pascal Compilers* (1985), to explain how a Pascal compiler works. Each student then used Pascal to write a complete compiler for a small programming language. The compiler project was divided into six phases, each corresponding to a chapter in the textbook. After reading a chapter, the students were ready to program the corresponding part of the project.

After one of my lectures, three female students came up to me and said: "Thanks for an interesting lecture—it happens so rarely." Before I could ask for their names, they walked away. Twenty years later, I still remember this nice compliment.

⋆ ⋆ ⋆

While I was still in Southern California, I heard that Danish universities began using Concurrent Pascal as soon as it became available from Caltech. But I didn't know it was also being used by high-tech companies, such as Brüel & Kjær, Elbau, GNT-Automatic, and ITT Standard Electric Kirk. The widespread use of Concurrent Pascal in Denmark was partly due to DIKU's requirement that every student had to solve a non-trivial programming problem for a company before graduating.

On November 6, 1985, I participated in a one-day conference on "Concurrent Pascal—Perspectives and Experience" at the Eremitage Hotel in Lyngby. The meeting was arranged by the Center for Electronics (Elektronikcentralen) and the Association for Microprocessor Electronics" (SMT). The fifty attendants represented thirty companies and research centers. I opened the conference with a talk on "Edison—the successor of Concurrent Pascal."

Risto Petersen described Elbau's use of Concurrent Pascal in dedicated microprocessor systems for farmers, dentists, and foundries.

Niels Holm Pedersen summarized Brüel & Kjærs experience using Concurrent Pascal to program electronic measurement instruments. On the morning of the conference, the newspaper Berlingske Tidende published an interview with Niels and me under the headline: "Industry and academia speak the same language."

Richard Whiffen, president of the company Enertec in Pennsylvania, gave an interesting talk about a Concurrent Pascal subset for microcomputers, called mCP. A version delivered to McDonnell Douglas in St. Louis was used to produce a digital flight controller for the F15 Eagle aircraft. The original compiler from Caltech was able to determine the memory requirement of any Concurrent Pascal program before it was executed. As Whiffen pointed out: Knowing that it was impossible to get a memory overflow in a fighter jet flying at twice the speed of sound was comforting to the pilot!

⋆ ⋆ ⋆

During a preliminary visit to Copenhagen, I had lunch with René Tang Jespersen, chairman of the GN Corporation, to discuss the possibility of consulting for him.

GN was the last remnant of the Great Northern Telegraph Company established in 1869 by the Danish Titan of industry, Carl Frederik Tietgen (1829–1901). On October 20, 1871, the Danish frigate "Tordenskiold" landed one end of a telegraph cable in Deepwater Bay, Hongkong. The other end would be landed in Shanghai. By 1894, the telegraph cables of Great Northern extended from England across Scandinavia and Russia to Japan and China. Tietgen was also a driving force behind Danish banking (Privatbanken, 1857), shipping (Det forenede Dampskibsselskab, 1866), and sugar production (De danske Sukkerfabrikker, 1872).

The telegraph was the first invention that made it possible to reach people quickly anywhere in the world. Since you paid for every word you sent, telegrams were usually brief and to the point—even in emergencies. When the Danish brig "Ane" was shipwrecked near Laguna de Terminos in the Gulf of Mexico on the morning of February 13, 1871, my great-grandfather's brother, Captain Peter Brinch, sent the following telegram (in English) to his family on the Danish island of Fanø:

ANE WRECKED NEAR LAGUNA STOP CREW SAVED

A hundred years later, war and revolution had reduced Tietgen's global empire to a handful of small, innovative companies, which included GN Batteries, GN Danavox, GN Data, GN Elmi, GN Netcom, and GN Telematic.

My meeting with Tang Jespersen took place in Great Northern's corporate headquarters on Kongens Nytorv, opposite the Royal Theater. The building was crowned by a "statue of liberty" (named Electra), who held a light globe that was turned on at night. René had worked ten years for the computer manufacturer Honeywell in Scandinavia and Belgium. After a hectic life as director of finance and administration for Honeywell Europe, he returned to Denmark as vice president of GN. When I met him in November 1983, we were both 45 years old. During our lunch, he offered me a three-year contract as an independent advisor to GN's board of directors.

In August 1984, Ernst Hede, president of GN Elmi, introduced me to one of his young engineers, Anders Raasted, who was in charge of an interesting project. They were developing a *multicomputer in a briefcase* that would be used to measure the reliability of telephone lines. The briefcase would be connected to one end of a telephone line and left alone for weeks transmitting test signals down the line. At the other end of the line, a second briefcase would receive the signals and return them to the first briefcase, which would collect data about the frequency of transmission errors. Telephone technicians would be able to inspect the measurements at any time without interrupting the real-time data collection.

Elmi asked me to design a special-purpose operating system with parallel processes for this real-time application. As I started working on the problem, the number of parallel activities soon reached a point where I found myself unable to write a clear description of what I was doing. So I asked Raasted to be patient while I designed yet another parallel programming language for real-time design.

I had already invented parallel programming languages which included monitors (Concurrent Pascal, 1975), remote procedure calls (Distributed processes, 1978), and conditional critical regions (Edison, 1981). This time I used a minimal subset of Pascal to design a secure parallel language, named *Joyce* (Brinch Hansen 1987a, 1987c). The new language was based on Hoare's idea of communicating sequential processes (CSP) which exchange messages through synchronous channels without automatic buffering. Joyce removed a major limitation of CSP by introducing *parallel recursion* in the form of processes that spawn copies of themselves.

To experiment with the new language, I developed a portable implementation of Joyce on an IBM PC (Brinch Hansen, 1987b). This was apparently the first recursive CSP language implemented on a computer. The Joyce compiler checked that parallel processes never referred to the same variables. This was essential since the multicomputer would consist of microprocessors without shared memory. The compiler also checked that every message sent from one process to another was received in a variable of the same data type as the message itself. This turned out to be one of the most frequently detected programming errors in my Joyce programs.

My next step was to use Joyce to simulate a simplified version of Elmi's real-time system. Working with Raasted's group, I defined the process structure and communication patterns of the actual real-time system. I then wrote a Joyce program with the same number of processes and communication channels. My Joyce model was, of course, greatly simplified. The circuit board that generated test signals was represented by a ten-line process that sent a sequence of integers through an output channel. Transmission errors were simulated by occasionally outputting the wrong values. Other processes simulated a real-time clock, a simple filing system that collected measurements, and a console used by technicians to inspect selected measurements.

This Joyce program was an *executable model* of the real-time system that could be studied and tested by programmers to reveal systemwide flaws, such as deadlocks, in the process structure. Today, this design method would be considered an example of "rapid prototyping."

Using my Joyce program as a model, Elmi was able to fill in the missing details and implement the final software product as a set of Pascal programs running in parallel. It turned out to be the first time Elmi had delivered a new product to its customers *ahead of schedule.*

⋆ ⋆ ⋆

Returning to Denmark was not as easy for me as I had thought. In C. P. Snow's novel "Last Things," a Jewish tycoon, Azik Schiff remarks that "coming to England as an exile, he had felt one irremovable strain: you had to think consciously about actions which, in your own country, you performed as instinctively as breathing."

I had the same experience when I lived in America. Now, many years later, I had it again in Denmark. At DIKU it was definitely not kosher for the faculty to have close ties to industry. This taboo surprised me since I had grown up in Copenhagen with a father, who had worked both in industry as a civil engineer and in academia as a professor of engineering. I remember an instance where DIKU had agreed to let IBM borrow our PCs for use in a summer course for unemployed people. My colleagues were horrified when I proposed to ask IBM in return to let us borrow one of their newest work stations.

On January 1, 1985, as the first Danish computer scientist, I was elected a Fellow of The Institute of Electrical and Electronics Engineers "For contributions to concurrent programming and operating systems." IBM graciously agreed to host the event and serve refreshments for the audience. At the award ceremony I demonstrated my Edison system for the IBM PC on a huge screen display at the IBM auditorium in Lyngby. I don't remember what my colleagues thought of this untraditional arrangement.

On another occasion, I rocked the boat as chair of a faculty committee that considered one of DIKU's PhD candidates for an assistant professorship. For obvious reasons, leading American universities rarely hire their own graduates. PhDs who have studied in the same department for years are not likely to move it in a new direction and risk offending their former advisors and their colleagues.

By the Danish rules of the game, my committee was only supposed to answer one question: Is the candidate minimally qualified to become an assistant professor? Instead, I asked my colleagues if they thought this appointment would improve the department in any way. Caught by surprise, they agreed with me that he should not be appointed. It caused a stink when the rest of the faculty heard what happened. They had all taken it for granted that he would be appointed. After that, I was viewed as someone who threatened the harmony of the department.

I regard Denmark as one of the most civilized countries on Earth with its low rates of poverty and crime, universal health care, and free education. To pay for this level of welfare, Danes pay extremely high taxes (my income taxes were 2/3 of my salary). I still consider that an acceptable price to pay for a just society.

The computer scientist, Alan Perlis, told me an amusing story about the difference between American and Danish mentality. During a visit to Regnecentralen, he asked my boss, Niels Ivar Bech, to show him a slum area in Copenhagen. So Bech drove him to a neighborhood that was poor by Danish standards. However, when they got there, Perlis said: "Niels Ivar, this is not a slum—where *are* the slums of Copenhagen?" Bech answered: "If we had any, this is where they would be!"

Like so many other foreigners, Milena and I found Americans to be some of the nicest and most hospitable people you can imagine. Shortly after we moved to Southern California, two of our neighbors, Eileen and Bob Harder, invited us over for Thanksgiving dinner. This openness towards strangers is rare in Denmark, where most people stick to their own friends.

Foreigners, who live in Denmark, have often described how difficult it is to find close friends among the Danes. When I worked for Regnecentralen, one of my colleagues was a young American, named Roger House, who married a Danish women, named Jeanne. After a couple of years, Roger returned to the United States. On his last day at Regnecentralen, he quietly asked us: "How come none of you ever invited me to your homes?" We looked at him and said: "But we didn't think you would be interested!"

After our return to Denmark, Milena and I discovered that we were no longer part of the circle of our former friends. They were happy to come to our return party—but few of them invited us back. Those who did had no interest in our stories about life in America. This feeling of alienation is the price you pay for leaving your country in search of adventure: in the end you don't belong in either country—but you have lived an exciting life. So be it!

8

DANISH INTERLUDE 1984–87

Student democracy and teaching in Denmark – Danish industry uses Concurrent Pascal – Consulting for GN Corporation – Rocking the boat.

After fourteen years in America I suffered a first-class attack of homesickness for Denmark. I had written the book on operating systems that was my original reason for coming to the United States. And, through my work in operating systems and concurrent programming, I had fulfilled my dream of making fundamental contributions to a new field. As Americans say, "Been there—done that!" Now what?

At this point in my life, I longed to return to Denmark and continue the life I had left behind. If you think I was deceiving myself, why, you are absolutely right. But did I listen to reason? Nope, I just had to go back and discover for myself that my lost youth was indeed, well, lost. Milena, who had no wish to leave the United States, was surprisingly understanding. She told our children, Mette and Thomas: "Dad is unhappy. He has to get Denmark out of his system." Looking back, I find it unforgiveable that I turned their lives upside down, just as my son was close to graduating from high school, and my daughter was ready to start her university education.

In the fall of 1983, I applied for a new professorship in datalogy at the University of Copenhagen. Danish universities do not ask for confidential letters of recommendation from colleagues at other universities. Instead I was asked to submit copies of my best books and papers to an ad hoc committee consisting of professors Peter Naur and Peter Johansen from University of Copenhagen and Kees Koster from the Catholic University of Nijmegen, The Netherlands. After studying my work for four months, the committee submitted a six page summary of my career and scientific contributions recommending that I be appointed to the vacant professorship.

In an egalitarian society like Denmark, you were constantly aware, as Vartan Gregorian put it, "that people think they are equal, and that whether or not that is accurate is irrelevant." All professors received the same salary independent of their achievements and the terms of their appointments were not negotiable.

Nevertheless, I had one concern before I was ready to accept a position in Denmark: At USC I had recently created a personal computer lab with 40 IBM PCs that enabled me to teach a course in which students wrote single-user operating systems in the programming language Edison. Would it be possible for me to establish a smaller PC lab at the University of Copenhagen?

Since I was used to negotiating directly with high-level administrators in the United States, I wrote a letter to Bertel Haarder, Danish minister of education, asking for his help in providing funds for a handful of IBM PCs. During a visit to Denmark, I also met him privately. Haarder, who after all was a politician, promised to find money for the PCs.

I soon learned that bypassing the normal channels of communication just isn't done at Danish universities. On April 27, 1984, the following item appeared in the newspaper Politiken (translated into English by me):

> *Gets job in spite of concerns.* This summer the Institute of Datalogy at University of Copenhagen (DIKU) will increase the number of professors from two to three. The favorite for the position is one of the leaders of the international world of computing, dr. techn. Per Brinch Hansen. He is currently professor of computer science at University of Southern California.
>
> The 45-year old Danish engineer is number one on the institute's confidential list of preferred applicants. . .However, it raised some concern at DIKU, when Per Brinch Hansen as a condition for his appointment practically demanded that the institute make six IBM computers of a specific type available. DIKU, which has one of them, now plans to acquire two more in the immediate future and three more later on.
>
> In 1978, the internationally known Danish computer scientist wrote—as the first in the world—a doctoral thesis about the special problems posed by computer programs that execute many tasks at the same time.
>
> Further down on the institute's list are people from DIKU's present staff.

Three months later, I received official notification that "We, Margrethe the Second, Queen of Denmark, by the grace of God, makes it known that We hereby, from August 1, 1984, appoint professor, doctor technices Per Brinch Hansen, who is a Danish citizen, as professor with permanent appointment."

DIKU was a child of the student uprising in the late 1960s. For generations, every university department had been headed by a single professor who had the final say in all matters concerning curriculum, examinations, research, and appointments. Inspired by the student riots in the United States and France, Danish students and instructors staged demonstrations in the spring of 1968, shouting "Down with the tyranny of professors," "Student participation NOW," and "Research for the people."

Everybody agreed that something had to be done. The politicians, who knew how to count votes, were well aware that the number of students far exceeded the number of professors. In May 1970, the Danish parliament passed a new Statute of Administration ("styrelsesloven"). From now on, every university department would be governed by a council ("institutråd") with equal representation of teachers, students, and staff members.

In 1969, Peter Naur became the first Danish professor of computer science ("datalogy"). The following year, associate professor Edda Sveinsdottir became the first democratically elected head of DIKU. The new department couldn't have made a better choice: Edda, who was friendly and helpful, attacked problems with boundless energy. Her innovative research in three-dimensional scanning of the brain ("computerized tomography") and Peter Naur's pioneering work in compiler development helped establish DIKU's tradition as a center of applied computer science. In 1987, Edda became professor of datalogy at Roskilde University.

In my time, DIKU's council consisted of sixty teachers, students, and staff members. Once a month, the council met for a couple of hours. Consequently, every teacher had only a couple of minutes to express personal opinions about departmental issues and had only one of sixty votes, independent of personal ability and achievements. Since teachers tended to vote as individuals, students often prevailed by voting unanimously.

When a group of people have the right to decide anything in democratic fashion, they naturally concentrate on problems they understand and ignore less familiar issues. At one departmental meeting, the discussion centered around problems with our printing office, the sun shades in the cafeteria, the reliability of the elevator, the use of office space, and the absence of teachers from these meetings "which they are obliged to attend." Under the new system of governance, the focus of discussion was no longer innovative teaching and research, but job satisfaction.

My first impression was that a handful of teachers did interesting research, while the majority were unproductive. Looking through the Science Citation Index for the previous five years, I found that worldwide only one third of our faculty was cited in the works of other researchers.

As the Harvard dean, Henry Rosovsky (1990), has pointed out "Not everything is improved by making it more democratic:"

> The limiting case exists in some European universities where the practice of "parity" was born in the 1960s. Power over virtually all decisions came to be equally shared between students, faculty, and employees—and not infrequently the government. The educational results have been disastrous. Academic standards declined and a sense of mission was lost.

However, Edda Sveinsdottir felt that the endless discussions, that were necessary to resolve even minor issues, were worthwhile because they often led to decisions that everyone could live with. Personally, I found it pointless to listen to student representatives, who had never done research, but nevertheless believed that the problems of society could be solved by trying to control the unpredictable nature of research. If people had told Thomas Edison what to do, he might have invented a faster telegraph key instead of the electric light.

Needless to say, the reality did not always resemble the politically correct utopia. When I suggested to my colleagues that I would like to revise the operating system course completely, their immediate reaction was "Please don't! If we change even one course, the militant students will use that as an excuse to demand a revision of the entire curriculum." In the end, I simply had to accept that the Danes had developed an educational system that valued cooperation and peace of mind more than individual pursuit of excellence.

It was a joy to teach Danish undergraduates. It would, of course, be unfair to compare them to Caltech students. But, thanks to the excellent Danish high schools, they were, on the whole, better prepared than most American undergraduates at USC or Syracuse University.

With the help of an army of teaching assistants (TAs), I taught a course on compiler design for a class with over 200 students. In the days before email, it was impractical for me to help that many students individually. Instead I selected my student, Birger Andersen, to be my chief TA. Birger, who is now Associate Professor at the Copenhagen University College of

Engineering, gave me feedback from 15 regular TAs, each of whom was responsible for helping 15 students. This arrangement worked fine. Although the TAs were well-meaning, their democratic desire to be involved in all decisions was annoying to someone who had twenty years of experience in compiler design. My indirect communication with TAs, through Birger, saved me from having weekly discussions with fifteen people about my choice of textbook and philosophy of teaching.

I used my own textbook, with the modest title *Brinch Hansen on Pascal Compilers* (1985), to explain how a Pascal compiler works. Each student then used Pascal to write a complete compiler for a small programming language. The compiler project was divided into six phases, each corresponding to a chapter in the textbook. After reading a chapter, the students were ready to program the corresponding part of the project.

After one of my lectures, three female students came up to me and said: "Thanks for an interesting lecture—it happens so rarely." Before I could ask for their names, they walked away. Twenty years later, I still remember this nice compliment.

⋆ ⋆ ⋆

While I was still in Southern California, I heard that Danish universities began using Concurrent Pascal as soon as it became available from Caltech. But I didn't know it was also being used by high-tech companies, such as Brüel & Kjær, Elbau, GNT-Automatic, and ITT Standard Electric Kirk. The widespread use of Concurrent Pascal in Denmark was partly due to DIKU's requirement that every student had to solve a non-trivial programming problem for a company before graduating.

On November 6, 1985, I participated in a one-day conference on "Concurrent Pascal—Perspectives and Experience" at the Eremitage Hotel in Lyngby. The meeting was arranged by the Center for Electronics (Elektronikcentralen) and the Association for Microprocessor Electronics" (SMT). The fifty attendants represented thirty companies and research centers. I opened the conference with a talk on "Edison—the successor of Concurrent Pascal."

Risto Petersen described Elbau's use of Concurrent Pascal in dedicated microprocessor systems for farmers, dentists, and foundries.

Niels Holm Pedersen summarized Brüel & Kjærs experience using Concurrent Pascal to program electronic measurement instruments. On the morning of the conference, the newspaper Berlingske Tidende published an interview with Niels and me under the headline: "Industry and academia speak the same language."

Richard Whiffen, president of the company Enertec in Pennsylvania, gave an interesting talk about a Concurrent Pascal subset for microcomputers, called mCP. A version delivered to McDonnell Douglas in St. Louis was used to produce a digital flight controller for the F15 Eagle aircraft. The original compiler from Caltech was able to determine the memory requirement of any Concurrent Pascal program before it was executed. As Whiffen pointed out: Knowing that it was impossible to get a memory overflow in a fighter jet flying at twice the speed of sound was comforting to the pilot!

⋆ ⋆ ⋆

During a preliminary visit to Copenhagen, I had lunch with René Tang Jespersen, chairman of the GN Corporation, to discuss the possibility of consulting for him.

GN was the last remnant of the Great Northern Telegraph Company established in 1869 by the Danish Titan of industry, Carl Frederik Tietgen (1829–1901). On October 20, 1871, the Danish frigate "Tordenskiold" landed one end of a telegraph cable in Deepwater Bay, Hongkong. The other end would be landed in Shanghai. By 1894, the telegraph cables of Great Northern extended from England across Scandinavia and Russia to Japan and China. Tietgen was also a driving force behind Danish banking (Privatbanken, 1857), shipping (Det forenede Dampskibsselskab, 1866), and sugar production (De danske Sukkerfabrikker, 1872).

The telegraph was the first invention that made it possible to reach people quickly anywhere in the world. Since you paid for every word you sent, telegrams were usually brief and to the point—even in emergencies. When the Danish brig "Ane" was shipwrecked near Laguna de Terminos in the Gulf of Mexico on the morning of February 13, 1871, my great-grandfather's brother, Captain Peter Brinch, sent the following telegram (in English) to his family on the Danish island of Fanø:

ANE WRECKED NEAR LAGUNA STOP CREW SAVED

A hundred years later, war and revolution had reduced Tietgen's global empire to a handful of small, innovative companies, which included GN Batteries, GN Danavox, GN Data, GN Elmi, GN Netcom, and GN Telematic.

My meeting with Tang Jespersen took place in Great Northern's corporate headquarters on Kongens Nytorv, opposite the Royal Theater. The building was crowned by a “statue of liberty” (named Electra), who held a light globe that was turned on at night. René had worked ten years for the computer manufacturer Honeywell in Scandinavia and Belgium. After a hectic life as director of finance and administration for Honeywell Europe, he returned to Denmark as vice president of GN. When I met him in November 1983, we were both 45 years old. During our lunch, he offered me a three-year contract as an independent advisor to GN’s board of directors.

In August 1984, Ernst Hede, president of GN Elmi, introduced me to one of his young engineers, Anders Raasted, who was in charge of an interesting project. They were developing a *multicomputer in a briefcase* that would be used to measure the reliability of telephone lines. The briefcase would be connected to one end of a telephone line and left alone for weeks transmitting test signals down the line. At the other end of the line, a second briefcase would receive the signals and return them to the first briefcase, which would collect data about the frequency of transmission errors. Telephone technicians would be able to inspect the measurements at any time without interrupting the real-time data collection.

Elmi asked me to design a special-purpose operating system with parallel processes for this real-time application. As I started working on the problem, the number of parallel activities soon reached a point where I found myself unable to write a clear description of what I was doing. So I asked Raasted to be patient while I designed yet another parallel programming language for real-time design.

I had already invented parallel programming languages which included monitors (Concurrent Pascal, 1975), remote procedure calls (Distributed processes, 1978), and conditional critical regions (Edison, 1981). This time I used a minimal subset of Pascal to design a secure parallel language, named *Joyce* (Brinch Hansen 1987a, 1987c). The new language was based on Hoare’s idea of communicating sequential processes (CSP) which exchange messages through synchronous channels without automatic buffering. Joyce removed a major limitation of CSP by introducing *parallel recursion* in the form of processes that spawn copies of themselves.

To experiment with the new language, I developed a portable implementation of Joyce on an IBM PC (Brinch Hansen, 1987b). This was apparently the first recursive CSP language implemented on a computer. The Joyce compiler checked that parallel processes never referred to the same variables. This was essential since the multicomputer would consist of microprocessors without shared memory. The compiler also checked that every message sent from one process to another was received in a variable of the same data type as the message itself. This turned out to be one of the most frequently detected programming errors in my Joyce programs.

My next step was to use Joyce to simulate a simplified version of Elmi's real-time system. Working with Raasted's group, I defined the process structure and communication patterns of the actual real-time system. I then wrote a Joyce program with the same number of processes and communication channels. My Joyce model was, of course, greatly simplified. The circuit board that generated test signals was represented by a ten-line process that sent a sequence of integers through an output channel. Transmission errors were simulated by occasionally outputting the wrong values. Other processes simulated a real-time clock, a simple filing system that collected measurements, and a console used by technicians to inspect selected measurements.

This Joyce program was an *executable model* of the real-time system that could be studied and tested by programmers to reveal systemwide flaws, such as deadlocks, in the process structure. Today, this design method would be considered an example of "rapid prototyping."

Using my Joyce program as a model, Elmi was able to fill in the missing details and implement the final software product as a set of Pascal programs running in parallel. It turned out to be the first time Elmi had delivered a new product to its customers *ahead of schedule.*

⋆ ⋆ ⋆

Returning to Denmark was not as easy for me as I had thought. In C. P. Snow's novel "Last Things," a Jewish tycoon, Azik Schiff remarks that "coming to England as an exile, he had felt one irremovable strain: you had to think consciously about actions which, in your own country, you performed as instinctively as breathing."

I had the same experience when I lived in America. Now, many years later, I had it again in Denmark. At DIKU it was definitely not kosher for the faculty to have close ties to industry. This taboo surprised me since I had grown up in Copenhagen with a father, who had worked both in industry as a civil engineer and in academia as a professor of engineering. I remember an instance where DIKU had agreed to let IBM borrow our PCs for use in a summer course for unemployed people. My colleagues were horrified when I proposed to ask IBM in return to let us borrow one of their newest work stations.

On January 1, 1985, as the first Danish computer scientist, I was elected a Fellow of The Institute of Electrical and Electronics Engineers "For contributions to concurrent programming and operating systems." IBM graciously agreed to host the event and serve refreshments for the audience. At the award ceremony I demonstrated my Edison system for the IBM PC on a huge screen display at the IBM auditorium in Lyngby. I don't remember what my colleagues thought of this untraditional arrangement.

On another occasion, I rocked the boat as chair of a faculty committee that considered one of DIKU's PhD candidates for an assistant professorship. For obvious reasons, leading American universities rarely hire their own graduates. PhDs who have studied in the same department for years are not likely to move it in a new direction and risk offending their former advisors and their colleagues.

By the Danish rules of the game, my committee was only supposed to answer one question: Is the candidate minimally qualified to become an assistant professor? Instead, I asked my colleagues if they thought this appointment would improve the department in any way. Caught by surprise, they agreed with me that he should not be appointed. It caused a stink when the rest of the faculty heard what happened. They had all taken it for granted that he would be appointed. After that, I was viewed as someone who threatened the harmony of the department.

I regard Denmark as one of the most civilized countries on Earth with its low rates of poverty and crime, universal health care, and free education. To pay for this level of welfare, Danes pay extremely high taxes (my income taxes were 2/3 of my salary). I still consider that an acceptable price to pay for a just society.

The computer scientist, Alan Perlis, told me an amusing story about the difference between American and Danish mentality. During a visit to Regnecentralen, he asked my boss, Niels Ivar Bech, to show him a slum area in Copenhagen. So Bech drove him to a neighborhood that was poor by Danish standards. However, when they got there, Perlis said: "Niels Ivar, this is not a slum—where *are* the slums of Copenhagen?" Bech answered: "If we had any, this is where they would be!"

Like so many other foreigners, Milena and I found Americans to be some of the nicest and most hospitable people you can imagine. Shortly after we moved to Southern California, two of our neighbors, Eileen and Bob Harder, invited us over for Thanksgiving dinner. This openness towards strangers is rare in Denmark, where most people stick to their own friends.

Foreigners, who live in Denmark, have often described how difficult it is to find close friends among the Danes. When I worked for Regnecentralen, one of my colleagues was a young American, named Roger House, who married a Danish women, named Jeanne. After a couple of years, Roger returned to the United States. On his last day at Regnecentralen, he quietly asked us: "How come none of you ever invited me to your homes?" We looked at him and said: "But we didn't think you would be interested!"

After our return to Denmark, Milena and I discovered that we were no longer part of the circle of our former friends. They were happy to come to our return party—but few of them invited us back. Those who did had no interest in our stories about life in America. This feeling of alienation is the price you pay for leaving your country in search of adventure: in the end you don't belong in either country—but you have lived an exciting life. So be it!

9

BACK IN AMERICA 1987–2004

Distinguished professor at Syracuse – Birthday celebration in the former Danish West Indies – Becoming an American citizen – Parallel scientific computing – A personal supercomputer – Parallel cryptography – History of programming languages – The Computer Pioneer Award – Final words.

After the first year in Denmark, Milena and I knew that our family now belonged in the United States. Our children did not feel at home in Denmark. In May 1986, our daughter Mette announced that she was going back, no matter what! From then on, things happened quickly. The American Embassy in Copenhagen informed me that our residence permits (known as "green cards") had expired. We would only be allowed to return if we obtained new immigrant visas by the end of the year and returned to the United States no later than April 1987.

I immediately called Syracuse University (SU), in Central New York, where I knew the computer scientist John Reynolds, and asked if it would be possible to appoint me as full professor within two months—a process that normally takes six months. In September, Milena and I flew to Syracuse where I gave a talk. While Milena looked at houses, I met with the academic vice chancellor, Gershon Vincow, and the faculty of the School of Computer and Information Science. Two weeks later, the interim dean, Ernie Sibert, offered me an appointment as distinguished professor, an honorary title that had only been bestowed on two other professors at the university.

By February 1987, all four of us had returned to America. Milena and I moved into a white colonial on 5070 Pine Valley Drive in the small village of Fayetteville, a short drive from the university. In our backyard we have a large swimming pool (27 by 60 feet). The house lies in a beautiful valley surrounded by tree-topped hills. The unfenced lawns with trees make the neighborhood look like a park.

Syracuse is a city of about 160,000 people in the center of New York state, a five-hour drive from New York City. Until the 1920s the 363-mile long Erie Canal, extending from the Hudson river at Albany to Lake Erie at Buffalo, passed through downtown Syracuse. The climate is similar to the Danish one with tons of snow during the winter and plenty of rain during the spring and summer. The story goes that a Syracuse professor missed the summer one year—it fell on a Tuesday, and he was out of town.

After moving to the East Coast, we spent many vacations in the Caribbean. In November 1988, Milena and I celebrated our 50th birthdays in the U.S. Virgin Islands. We stayed a week at the Morningstar Beach Club on St Thomas. From 1666 to 1917, these islands were known as the Danish West Indies until the United States bought them from Denmark for 25 million dollars. Since the US is not supposed to have colonies, the islands now have the status of "unincorporated territory." In Charlotte Amalie you can still see Jørgen Iversen's Red Fort (1680), built by the first Danish settlers near King's Wharf, the Governor's House (1747) on Kongens Gade, and the Lutheran Church (1820) on Nørregade. Every day, cruise ships arrive at the West Indian Company Dock, and the tourists all head for the duty-free shops on Dronningens Gade.

One day we took the ferry from Redhook Bay to Cruz Bay, the only town on St. John, and hired a tour guide to drive us along Kongevejen through the tropical forest to Coral Bay, where King June and his last followers killed themselves after killing 76 whites and destroying 48 plantations during the slave rebellion in 1733. At Annaberg Plantation, the ruins of a Danish sugar mill, built in the 1780s, have been partially restored. On the north shore are some of the most beautiful beaches in the Caribbean.

We flew by seaplane to St. Croix. On the way from Christiansted to Frederiksted we walked through St. George Village, a tropical garden, landscaped around the ruins of a plantation, built by Governor General Peter Oxhold around 1815. The Whim Greathouse is another plantation, from 1803, completely restored with mahogany furniture and crystal chandeliers. Near the pier in Frederiksted lies Fort Frederik, completed in 1776. Here Governor von Scholten liberated the slaves on July 3, 1848. In Christiansted, we saw the Governor's Residence, Fort Christian, and the Danish Scale House. These yellow-and-white buildings reminded me of the old houses in Frederiksberg, Denmark.

Four years later, on May 19, 1992, I took the oath of citizenship of the United States at the Onondaga County Courthouse in Syracuse.

⋆ ⋆ ⋆

In the 1980s, the early programming problems of operating systems surfaced again in *parallel scientific computing* (also known as *computational science*): there was a serious need for machine-independent programming languages and algorithms. To understand this challenge, I spent five years writing portable parallel programs for typical programs in science and engineering.

As a first step, my student Anand Rangachari and I moved the parallel programming language Joyce from an IBM PC to an Encore Multimax 320, a multiprocessor with 18 processors and 128 Mbytes of shared memory (Brinch Hansen 1989). This machine was designed a few years before I joined SU. It was owned and operated by the Northeast Parallel Architectures Center (NPAC) at Syracuse University.

The only valid reason for using parallel programming in scientific computing is to tackle problems that require more computing power than you can get from a single processor. From that point of view, our experiments with the Multimax were somewhat academic. The Joyce compiler generated portable code which was interpreted by a kernel of 2,300 lines written in assembly language. In theory, the multiprocessor had the potential of making programs eighteen times faster. However, most of the potential speedup was wasted by the portable code, which was an order of magnitude slower than machine code.

Nevertheless, I learned a great deal from this first experiment about the problems of implementing a parallel programming language on a multiprocessor. The main decision issues were: (1) *load balancing*—the number of process scheduling queues required to balance the computational load evenly among the processors, (2) *synchronous communication*—the implementation details of processes exchanging messages through unbuffered channels, and (3) *mutual exclusion*—the number of software locks needed to prevent multiple processors from accessing the same queue or channel at the same time (without slowing the processors unnecessarily down). We settled these issues by performance measurements (Brinch Hansen 1988).

The Achilles heel of the multiprocessor concept was the empirical observation by Intel cofounder Gordon Moore (1979) that the density of integrated circuits had doubled every year since 1958. Moore's law predicted that by 1992 you would be able to buy 1,000 processors for the same price as 10 processors in 1985. And, since nobody believed that a shared memory machine could support that many processors efficiently, multiprocessor architectures appeared to have no future. The catch phrase at the time was that "multiprocessors do not scale up."

One way out of this dilemma was to give up the simple idea of a multiprocessor with shared memory in favor of a *multicomputer with distributed memory.* Such a parallel architecture consists of a bunch of microcomputers, each with its own local memory. The processor nodes communicate by sending messages to their nearest neighbors only through communication links. Each link is a "point-to-point" connection between exactly two nodes. The removal of the bottleneck created by shared memory greatly increased the performance of parallel computers. However, the occasional need to route some messages though a sequence of intermediate nodes made multicomputers far more difficult to program than multiprocessors. So simplicity was sacrificed for performance—what else is new in computing? This compromise has, I believe, doomed computational science to remain an extremely difficult form of programming for experts only.

While we were experimenting with multiprocessing, the possibility of multicomputing had already been explored by a Caltech group headed by physicist Geoffrey Fox and computer scientist Charles Seitz. Together, they pioneered a new parallel architecture known as the *hypercube* (Seitz 1985).

Let me explain what a hypercube is: In each corner of a cube, you place a microcomputer with its own memory. Then you turn each edge of the cube into a communication link that connects two processor nodes. This gives you a cube architecture in which each of the eight nodes can exchange messages with its three nearest neighbors only.

If you link each node in a cube with the corresponding node in another cube, you obtain a hypercube architecture with sixteen processors. And, if you link two of these hypercubes in the same manner, you get a hypercube with 32 nodes, and so on. The key insight is that whenever you double the number of processors, the increase in the number of communication links is only proportional to the previous number of processors. So, as microprocessors become cheaper, a hypercube scales nicely without letting the number of links grow out of bounds.

By October 1983 Chuck Seitz had constructed a 64-processor hypercube at Caltech. The message communication was handled by a slow software kernel, known somewhat grandiosely as the Crystaline Operating System, "although," as Geoffrey Fox pointed out, "it was never really an operating system." This parallel machine was no academic toy. From the beginning, Fox (1988) used the Cosmic Cube (as it was called) to solve substantial computational problems in science and engineering.

In 1985, before leaving Denmark, I was instrumental in obtaining funding for DIKU's first parallel computer, an Intel iPSC hypercube with 32 microcomputers. At the time, this machine was only the third of its kind acquired by European research institutions.

Although I recognized the invention of inexpensive supercomputers as a major breakthrough, I was never enamored of hypercube architectures. I felt that hypercube algorithms would be dominated by the problem of mapping problem-oriented process configurations onto a hypercube. That prediction turned out to be true, I think.

Parallel programs were often written in traditional programming languages, such as Fortran or C, extended with subroutines for parallelism. To my taste these programs were difficult to read and lacked the beauty that scientists expect of their own research. I was convinced that *the most important task in computational science was to make the programming of parallel computers easier.* This was, in my opinion, even more important than increasing computational power, and I felt that we should be prepared to sacrifice some performance to solve the programming problem.

At a Supercomputing Conference in Boston in May 1988, I looked (in vain) for the ideal parallel architecture of the future. Such a machine should, in my opinion, (1) use general-purpose microcomputers, (2) be expandable from tens to thousands of processors, (3) support different processor configurations (pipelines, trees, matrices, and so on) in a transparent manner, (4) handle process creation, communication, and termination by machine instructions that are only an order of magnitude slower than memory references, and (5) automatically balance the computational load among and route messages between the processors.

The first requirement ruled out NPAC's Connection Machine, in which 64,000 synchronous processors executed identical processes in lock step (Hillis 1985). The second one excluded multiprocessors. The third condition made hypercubes unsuitable. The only architecture that satisfied the first four requirements was a multicomputer known as the *Meiko Computing Surface.* No parallel computer satisfied the fifth condition.

In the summer of 1988, I traveled to Bristol, England, to visit Inmos and Meiko. At the Inmos research center, I met David May, the architect of the T800 transputer chip, a 32 bit VLSI microprocessor with 64 bit floating-point arithmetic. Four on-chip links enabled the transputer to exchange messages with four other transputers.

All programming of the transputer was done in the parallel programming language *occam*, which David May had based on Hoare's Communicating Sequential Processes (CSP). This language made it possible to define parallel processes that communicate by messages. Direct communication between two connected transputers was very fast (a few microseconds). Process creation and termination were also hardware operation. The transputer could switch from one occam process to another in 1 microsecond. There was no other processor like it in the world!

A group of Inmos employees had formed a small company, named Meiko, to build a multicomputer with transputer nodes. On my last day in Bristol, July 8, 1988, I had dinner with the chairman of Meiko, Miles Chesney. My appointment letter at SU specified that the university "will further purchase computer equipment as needed for your work in an amount not greater than $100,000." For that amount of money, Miles was prepared to sell me a Computing Surface with 20 transputers and 40 Mbytes of distributed memory. I told him that 20 transputers would not add anything new to my research in computational science, since I had already used a multiprocessor at NPAC with 18 processors. I would need at least 40 transputers to make multicomputer programming interesting. On the other hand, I had no problem with reducing the memory of each transputer to 1 Mbyte only. I also offered to make a Computing Surface at SU available to Meiko for demonstrations to potential American customers.

Being a risk taker, I asked Miles Chesney to leave a message at my hotel the next morning informing me if he was be willing to offer me a 40-node system for my money. If that was unacceptable, I would fly home empty-handed and look for another machine (although I could not think of any worthy alternative). When I woke up the next morning, there was a message from Miles accepting my request.

Back in Syracuse the university hosted an inaugural symposium in March 1989 to celebrate the opening of its new Center for Science and Technology. The themes of the symposium were parallel computers, neural networks, and intelligent systems. It was organized by Alan Robinson and me from Syracuse University together with Michael Arbib from the University of Southern California. On that occasion, the three of us were awarded the Chancellor's Medal for Outstanding Achievement. The nine invited speakers included Ralph Gomery from IBM, who reviewed the evolution of computing, David May from Inmos, who discussed the possibility of designing general-purpose parallel computers, and Geoffrey Fox from Caltech, who described major applications of parallel supercomputers. As the first speaker, I described

"The nature of parallel programming" without going into technical details (Brinch Hansen 1990).

In July, a Computing Surface was installed at SU right next to my office. It had 48 transputers, each with 1 Mbyte of memory. The transputers were linked by a switching network that could be reconfigured before program execution. After two months of initial problems with hardware, software, and documentation, I was able to run a trivial occam program that sorted 65,536 integers on 31 transputers. I was now ready to experiment with parallel scientific programs (Brinch Hansen 1995).

Although I knew nothing about numerical analysis, I thought that parallel solution of linear equations would be a useful programming exercise for a beginner. I chose the problem for the following reason: When a pipeline with p processors solves n linear equations, the parallel computer time for the numerical computation is of the order of n^3/p. A computer scientist would say that the numeral computation requires $O(n^3/p)$ time, while the input/output of the equations takes $O(n^2)$ time. If the problem size n is large compared to the machine size p, the relative overhead of processor communication is negligible. The high ratio of computation to communication makes the problem ideal for efficient parallel computing.

A colleague recommended Householder reduction as an attractive method for solving linear equations on a parallel computer. The main strength of the method is its unconditional numerical stability (Householder 1958). The more familiar Gaussian elimination is faster but requires a dynamic rearrangement of the equations, known as pivoting, which complicates a parallel program somewhat.

Unfortunately, I could not find a well-written, understandable explanation of Householder's method. Most textbooks on numerical analysis produced the so-called "Householder matrix" like a rabbit from a magician's top hat without explaining why it is defined the way it is. At that point, I stopped writing parallel programs and concentrated on sequential Householder reduction. After several frustrating weeks I was able to write a tutorial on Householder reduction. Two pages were sufficient to explain the purpose and derive the equation for Householder's matrix. I then explained the computational rules for Householder reduction and illustrated the method by a numerical example and a Pascal program.

I was beginning to think that others might have the same difficulty understanding this fundamental computation. So I submitted the tutorial to a journal that published it (Brinch Hansen 1992). One reviewer wrote that he "found the presentation far superior to the several descriptions I have seen in numerical analysis books." I quote this review not just because I like it, but because it was my first lesson about computational science: In order to understand a computation, I must first explain it to myself by writing a tutorial that includes a complete sequential program.

After studying parallel programming for 25 years it was not difficult for me to program a Householder pipeline in occam for the Computing Surface. To achieve approximate load-balancing, the pipeline was folded three times across an array of transputers, so that each transputer executed four pipeline processes. The folded pipeline solved 1000 equations on 45 transputers in 87 sec. The Computing Surface made the computation 32 times faster than it would have been on a single transputer. I was able to derive an elegant formula that predicted the parallel run time accurately as a function of the number of equations solved, the number of transputers used, and the number of times the pipeline was folded.

My next exercise was to compute the trajectories of n particles that interact by gravitation only. I considered the n-body problem to be particularly challenging on a parallel computer since it involves interactions among all the particles in each computational step. This means that every processor must communicate, directly or indirectly, with every other processor. My description of an n-body pipeline included a brief summary of Newton's laws of gravitation and a Pascal program for sequential n-body simulation.

It was a complete surprise for me to discover that the sequential Pascal programs for Householder reduction and n-body simulation had practically identical control structures. I suddenly understood that both of them are instances of the same *programming paradigm*: Each algorithm solves an *all-pairs problem*—a computation on every possible subset consisting of two elements chosen from a set of n elements. I did not find this insight mentioned in any textbook on numerical analysis or computational physics.

I now discarded both parallel algorithms and started all over. This time I programmed a general pipeline algorithm for all-pairs computations. This program was a parallel implementation of the common control structure. It provided a mechanism for performing the same operation on every pair of elements chosen from an array of n elements without specifying what the elements represent and how they "interact" pairwise.

I then turned the all-pairs pipeline into a Householder pipeline by using a few data types and procedures from the sequential Householder program. This transformation of the parallel program was completely mechanical and required no understanding of Householder's method. A similar transformation turned the all-pairs pipeline into an n-body pipeline.

On August 24, 1984, I made the following entry in the computer log book: "At midnight, I used 31 transputers to simulate 10,000 gravitational bodies in 47 sec/step!"

I had now found my research theme: I would explore the use of programming paradigms in parallel programming. In programming, the word "paradigm" is often used with a general (but vague) connotation, such as "the high level methodologies that we recognize as common to many of our effective algorithms." I used the term in a more narrow (but precise) sense: *A programming paradigm is a class of algorithms that solve different problems but have the same control structure.*

This was the beginning of my studies in computational science from the point of view of a computer scientist. I followed the advice of Geoffrey Fox to "use real hardware to solve real problems with real software." But, where the Caltech group concentrated on scientific applications for their own sake, I used them as realistic case studies to illustrate the use of structured programming in computational science.

In addition to all-pairs computations, I developed paradigms for tuple multiplication, divide-and-conquer, Monte Carlo trials and cellular automata. For each paradigm I wrote a general program that defined the common control structure. Such a program is sometimes called an algorithmic skeleton, a generic program, or a program template.

From a general parallel program I derived two or more *model programs* that illustrated the use of the paradigm to solve specific problems. A general program includes a few unspecified data types and procedures that vary from one application to another. A model program is obtained by replacing these data types and procedures with the corresponding data types and procedures from a sequential program that solves a specific problem. The essence of the programming methodology is that a model program has a parallel component that implements a paradigm and a sequential component for a specific application. The clear separation of the issues of parallelism and the details of application is essential for writing model programs that are easy to understand.

My own model programs solved typical problems in science and engineering: linear equations, n-body simulation, matrix multiplication, shortest paths in graphs, sorting, fast Fourier transforms, simulated annealing, primality testing, Laplace's equation, and forest fire simulation.

I ran these parallel programs on a Computing Surface configured as a pipeline, a tree, a cube, or a matrix of transputers.

⋆ ⋆ ⋆

I now turned my attention to the RSA cryptosystem,where large primes play an essential role in the encoding and decoding of messages (Rivest 1978). A user chooses two large random primes. These primes are used to compute a public encoding key and a secret decoding key. Both keys include the product of the primes. The user can receive encoded messages from anyone who knows the public key. But only the user (who knows the secret key) can decode the messages.

The crucial assumption is that it is feasible to generate large primes using a computer, but there is no known algorithm for finding the prime factors of large composite numbers in reasonable amounts of computer time. If that ever becomes possible, you will be able to break the code by factoring the public product of the secret primes.

At the time, the RSA cryptosystem was believed to be secure for keys of 150 decimal digits. The simplest way to find a 150-digit prime is to generate 150 random digits at a time, until you discover a prime. The probability that a 150-digit number is a prime is about 1 in 150 ln10. You must therefore expect to test about 350 numbers for primality before you find a prime. (Half of these tests can be skipped if you only examine odd numbers.)

So, the generation of primes is reduced to the problem of testing the primality of random numbers. Unfortunately, it is not feasible to determine whether or not a 150-digit integer is a prime by examining all the 10^{75} possible divisors (a truly astronomic number). The Miller–Rabin algorithm tests the same integer many times using different random numbers (Rabin 1980). If any one of the trials shows that a number is composite, then this is the correct answer. However, if all trials fail to prove that a number is composite, then it is almost certainly prime. The probability that the algorithm gives the wrong answer after, say, 40 trials is less than 10^{-24}.

This is far less than the probability of a computer error. A computer that performs one million operations per second, with the same probability of failure per operation, will fail once in thirty billion years. That is roughly the age of the universe since the Big Bang.

The advantage of using a multicomputer for primality testing is obvious. When the same random number has been broadcast to every processor, the trials can be performed simultaneously without any communication between the processors. Consequently, the processor efficiency is very close to 1 for non-trivial problems.

I programmed the Miller–Rabin algorithm in occam and used the Computing Surface to perform 40 tests of a 160-digit random number simultaneously on 40 transputers.

For the primality testing, I had to program multiple-length arithmetic. Most computers limit integer arithmetic to 32–64 bits, corresponding to 8–17 decimal digits. A larger integer must be represented by an array of digits, each occupying a single machine word. The arithmetic operations on multiple-length integers are serial operations that imitate paper-and-pencil operations.

I thought it would be easy to find a textbook that includes a simple algorithm for *multiple-length division* with a complete explanation. Much to my surprise, I was unable to find such a book. I ended up spending weeks on this "well-known" problem and finally wrote a tutorial that includes a complete Pascal algorithm (Brinch Hansen 1994). I mention this unexpected difficulty to illustrate what happens when a standard algorithm is not published as a well-structured program in an executable language.

Inspired by my use of a programming paradigm for primality testing, my student, Jonathan Greenfield, explored the development of distributed generic algorithms for RSA cryptography. He defined abstract algorithms in a variant of the parallel programming language Joyce. These algorithms were rewritten in the implementation language occam and tested on the Meiko Computing Surface. His PhD thesis was an appealing combination of the theory and practice of parallel computing. From the point of view of a computer scientist, it was an amazing feat to recognize five different aspects of the same application as instances of two simple paradigms for parallel computing. In addition, Jonathan's thesis was well-written and easy to understand. It was published as a volume in the Springer-Verlag Lecture Notes in Computer Science (Greenfield 1993)—a rare honor for a PhD student.

It had been fun to enter an interdisciplinary field, refresh my memory of mathematics and physics I learned as an undergraduate, study numerical analysis, and teach myself the art of multicomputer programming.

My one serious criticism of computational science was that it largely ignored the issue of precision and clarity in parallel programming that is essential for the education of future scientists. A written explanation is not an algorithm. A graph of computational steps is not an algorithm. A picture of a systolic array is not an algorithm. A mathematical formula is not an algorithm. A program outline written in non-executable "pseudocode" is not an algorithm. And, a complicated "code" that is difficult to understand will not do either.

Subtle algorithms must be presented in their entirety as well-structured programs written in readable, executable programming languages. This was my main reason for publishing model programs for computational science. I felt that the study of programming paradigms provides an architectural vision of parallel scientific computing.

My fifth book, *Studies in Computational Science: Parallel Programming Paradigms* was published in 1995. I wish I could say that this work influenced the way people program parallel computers, but—with the possible exception of my students—I don't think it did.

A graduate student, Anil Menon (1995), left this impression of my course on multicomputer programming:

> Over the last ten years, I've studied under many teachers and taken many courses. Dr. Brinch Hansen's course was unlike no other. He was interested in solving problems in parallel. I had no idea, even after five earlier courses, that it was so difficult. He took seven to eight different problems and showed by means of a series of beautiful and elegant programs, how one would go about writing parallel programs. His insights were often remarkable, for example, his deep idea that *process structures* were the correct way to reason and work with parallel processing, just as *data structures* are the key to sequential processing. Or the time he told us about the importance of constraints in the design process.
>
> Perhaps the conviction always evident in his presentation came from the fact that these programs were his own, and not copied off some standard book. Even now it mystifies me to some extent how he could reduce a really complex program to a series of subprograms each no more than a dozen lines, the whole piece elegantly connected.
>
> The course was especially enjoyable because Dr. Brinch Hansen is a character. He's passionate, outspoken, opinionated and intolerant of anything less than perfection. What a relief it was to find a professor who wasn't afraid to voice what he really felt about issues in computer science. None of that cowardly "on the one hand. . .on the other hand" balance with which the meek

> evade making choices. He was as opinionated about the state of NPAC, as he was about his language SuperPascal. I could go on and on: His rare sense of history, the remarkable perceptiveness with which he'd transform one problem into another etc. But perhaps the great physicist, Feynman put it best (though in a different context): "To do physics," he said, "you gotta have style". I believe it's true of computer science as well. Dr. Brinch Hansen does parallel programming in style, and for one great semester it was my privilege to learn by example.

⋆ ⋆ ⋆

I have always felt that professionals should study the history of their own field for the enjoyment and insight it gives. In 1978 and 1993, I attended two ACM conferences, which became milestones in the History of Programming Languages. The first conference (HOPL-I) covered the major languages of the 1960s (Wexelblat 1981). The program committee selected thirteen languages that had been in use for at least ten years, had significant influence, and were still in use. Each paper was presented by a pioneer who had played a key role in the development of the language. The following presentations were of special interest to me:

APL (Ken Iverson)
Algol (Alan Perlis and Peter Naur)
Basic (Tom Kurtz)
Cobol (Jean Sammet)
Fortran (John Backus)
LISP (John McCarthy)
PL/I (George Radin)
Simula (Ole-Johan Dahl and Kristen Nygaard)

On this occasion, Ole-Johan Dahl, Peter Naur, Alan Perlis and his wife Sydelle visited us in Altadena.

The second conference (HOPL-II) focussed on programming languages of the 1970s which "had significant influence on the theory or practice of computing" (Bergin 1996). This time the languages and speakers included:

Ada (Bill Whitaker)
C (Dennis Ritchie)
C++ (Bjarne Stroustrup)

CLU (Barbara Liskov)
Concurrent Pascal (Per Brinch Hansen)
Pascal (Niklaus Wirth)
Prolog (Alain Colmerauer and Philippe Roussel)
Smalltalk (Alan Kay)

The organizers set high technical and editorial standards. Historian Mike Mahony reviewed all the papers. Each author also worked with a technical expert who reviewed the various drafts. My own paper on "Monitors and Concurrent Pascal: A personal history" went through six drafts over a period of fifteen months.

In a "no holds barred" panel discussion, the following exchange took place (Bergin 1997):

> *Per Brinch Hansen:* I'm going to sit down, since you have already answered my question which is, "Is there a future for insecure, low-level languages like C, and huge, incomprehensible languages like Ada?" But, I wish to make a less loaded comment, which is that there may be differences of style between programming languages, but there ought to be some common idea of the minimal requirements, so we can all agree that we are looking at a programming language. And I think that's part of the problem.
>
> If you look at physics, for example, I would say that a theory ought to satisfy at least three requirements and so should a programming language. First, a notation, which is what a programming language is, is supposed to enable you to express a theory of computation, not necessarily a mathematical theory (although that would be ideal), but theories can also be helpful if they are informal, as in geology. In any programming language, you will recognize a set of abstractions that are machine-independent, but at a certain point those concepts break down. If you have overflowing arithmetic, your results become meaningless; and that goes for every one of them, that they only apply under certain conditions, which should be stated in the language manual. The requirement that a language should be secure is the simple requirement that a compiler and a computer should tell you, when the programming concepts break down. If we can't agree on that being a minimum requirement for all programming languages, then I think we are just using the same word for con-

venience to denote things that have very little in common. By that definition, C is not a programming language.

The second requirement is that a theory in physics must be simple. If a Niels Bohr can't comprehend it, or a [Richard] Feynman, then a committee of physicists won't be able to master it either. That boils down to the simple requirement that language manuals must be short, concise, and so must their compilers. By that definition, Ada is not a programming language. [laughter]

The third requirement was illustrated by the German physicist, [Wolfgang] Pauli, who once said to Bohr, "I have a crazy theory, you are going to like it!" To which Bohr responded: "It is not crazy enough!"

When I look at this conference, I do see a certain sameness in what we have done. Apart from these obvious violations of what programming languages should be, there is precious little difference between Fortran and Concurrent Pascal. They are mostly the same thing: x becomes $x+1$. To me, it is not terribly interesting which languages will win, because that appears to be a study for sociologists, rather than computer scientists. What I like are the crazy paradigms we have seen, and there are two of those: Prolog and Smalltalk.

So I leave you with this question: Can we agree that a programming language must represent a theory of computation, that compilers and computers must check if the assumptions behind the abstractions apply when we run our programs, that the manuals must be short, and that the ideas must be crazy?

Dennis Ritchie: Could you please repeat the question? [laughter] I know what the question is. Is there a place for, in particular, C? Well, my guess is that there will not be any more significant low level languages—in other words, the niche is occupied—maybe that's just hoping. I guess the other response is that you have even stricter criteria than Jean [Sammet], whose criteria for considering what a language is, I think, are already too strict.

Niklaus Wirth: Is there any agreement among the four of you on the minimum requirements for us to call something a programming language other than the fact that it can change bits in a computer?

> *Dennis Ritchie:* Are you kidding? [laughter] No, of course, there is no agreement. That is the point. [laughter]
>
> *Bill Whitaker:* In particular, we didn't agree with YOU! [laughter]
>
> *Alan Kay:* I don't know, I think he hit it right on the head—I like the crazy part.

To understand what was going on here, I will quote what the biologist Francis Crick (1988) wrote about another "soft" science:

> [The work] tended to fall into a number of somewhat separate schools, each of which was rather reluctant to quote the work of the others. This is usually characteristic of a subject that is not producing any definite conclusions. (Philosophy and theology might be good examples.)

⋆ ⋆ ⋆

On May 8, 2002, I was awarded the IEEE Computer Pioneer medal "For pioneering development in operating systems and concurrent programming exemplified by work on the RC 4000 multiprogramming system, monitors, and Concurrent Pascal." In my acceptance speech (borrowing liberally from my own writing), I said (Brinch Hansen 2002):

> It is an unexpected pleasure for me to receive the first major award for the work I did from 1965 to 1975. I must confess, I was beginning to feel like Duke Ellington, who once said, "Fate doesn't want me to be famous too young." So, I thank the IEEE Computer Society for honoring me and making this speech necessary. And, I thank my friend, Jonathan Greenfield, for his tireless efforts in nominating me for the Computer Pioneer Award.
>
> Now, you should not for a minute imagine that I knew what I was doing as a young programmer. On two occasions, the work, you are honoring me for, almost came to nothing.
>
> In 1963, I graduated from the Technical University of Denmark without any programming experience (it was not yet being taught). There were (as far as I remember) no textbooks available on programming languages, compilers or operating systems.

With this background, I began my career as a systems programmer with Regnecentralen in Copenhagen. At age 29, I became head of software development for the *RC 4000 computer*. The senior manager of Regnecentralen, Niels Ivar Bech, gave me only one directive: "I need something new in multiprogramming!"

After a while, Jørn Jensen, Søren Lauesen, and I realized that we had no original ideas about multiprogramming. So, I told Bech: "We aren't getting anywhere. Is it all right with you if Jørn, Søren, and I spend a weekend at a country inn?" I wanted to give us one last chance. We had already agreed that we would either return with new ideas or give up and copy the best ideas we could find elsewhere. Bech immediately agreed (he had done the same thing when Regnecentralen's Cobol compiler project had come to a standstill).

It worked! The thought of returning to Regnecentralen without new ideas was simply unacceptable to us. Out of that weekend came the first ideas for the *RC 4000 multiprogramming system*, which introduced the now-standard concept of an operating system kernel.

Since 1970, I have been a computer scientist in the United States. While writing my textbook on operating system principles, I invented the *monitor notation*, which combines process synchronization with object-oriented programming.

At California Institute of Technology my goal was to develop a concurrent programming language with monitors. You would think it would be easy for me to extend Pascal with monitors. But I had no idea of how to do this. I remember sitting in my garden in Altadena, day after day, staring at a blank piece of paper and feeling like a complete failure. It took me two years to find reasonable solutions to most of the problems and make compromises which enabled me to ignore the most thorny issues.

In 1974, I distributed a description of the programming language *Concurrent Pascal*. I now understood what I was doing. One day the Caltech president, Harold Brown, came to my office and asked me to explain my research. After listening for half an hour, he said, "That sounds easy." I agreed because that was how I felt at the time. So, in the end, things turned out all right.

Let me conclude by quoting the biologist Francis Crick: "It's true that by blundering about we stumbled on gold, but the fact remains that we were looking for gold."

Thank you for your attention.

⋆ ⋆ ⋆

I am now sixty-six years old and close to retirement. My adult children left home many years ago after graduating from Syracuse University. My wife, Milena, received her second master's degree from SU and started a new career as a public librarian in Onondaga County.

I have been fortunate to live the creative life I dreamt of as a young man. It would have been easier for my family and colleagues, if I had been a more patient man, but you don't get to chose your temperament (or gifts for that matter).

I will end this programmer's story on a philosophical note by quoting Albert Einstein:

> In the light of knowledge attained, the happy achievement seems almost a matter of course, and any intelligent student can grasp it without too much trouble. But the years of anxious searching in the dark, with their intense longing, their alternations of confidence and exhaustion, and the final emergence into the light—only those who have experienced it can understand it.

Life has been good.

SOURCES

Asimov, I. 1976. *Science, Numbers, and I.* Ace Books, New York.

Backus, J. W. 1981. Question and answer session on Algol 60. In Wexelblat (1981), 162.

Bal, H. E., Steiner, J. G., and Tanenbaum, A. S. 1989. Programming languages for distributed computing systems. *ACM Computing Surveys 21*, (September), 261–322.

Bashe, C. J., Johnson, L. R., Palmer, J. H., and Pugh, E. W. 1986. *IBM's Early Computers.* The MIT Press, Cambridge, MA.

Bell, J .R. 1973. Threaded code. *Communications of the ACM 16*, 6 (June), 370–372.

Bergin, T. J., and Gibson, R. G. (eds.) 1996. *History of Programming Languages II.* ACM Press, NY.

Bergin, T. J. (ed.) 1997. HOPL II—Closing panel. *ACM SIGPLAN Notices 32*, 9 (September), 15–37.

Bishop, J. 1986. *Data Abstraction in Programming Languages.* Addison-Wesley, Reading, MA.

Bjerrum, L. B. 1969. *Jørgen Brinch Hansen, 1909–1969.* Géotechnique 19, 3 (September).

Bohm, D. 1951. *Quantum Theory.* Prentice-Hall, Englewood Cliffs, NJ.

Brinch Hansen, P. 1961–68. *Letters to my parents and wife.*

Brinch Hansen, P. 1962. Maser—et nyt forstærkerelement der muliggør kommunikation ved optiske frekvenser. (“Maser—a new amplifier element that makes communication at optical frequencies possible.”) *Ingeniøren*, (May).

Brinch Hansen, P. 1963. Rubinens røde straale. (“The red ray of the ruby”) *Vor Viden*, (May–June), 545–555 and 577–583.

Brinch Hansen, P. 1964. *An optimal compilation of Boolean Expressions in Cobol 61.* NordSAM 64, Stockholm, Sweden (August).

Brinch Hansen, P., and House, R. 1966. The Cobol compiler for the Siemens 3003. *BIT 6*, 1, 1–23.

Brinch Hansen, P. 1967a. The logical structure of the RC 4000 computer. *BIT 7*, 3, 191–199.

Brinch Hansen, P. 1967b. The RC 4000 real-time control system at Pulawy. *BIT 7*, 4, 279–288.

Brinch Hansen, P. 1968. The structure of the RC 4000 monitor. Regnecentralen, Copenhagen, Denmark (February).

Brinch Hansen, P. 1969a. *RC 4000 Computer Software: Multiprogramming System.* Regnecentralen, Copenhagen, Denmark, (April).

Brinch Hansen, P. 1969b. *RC 4000 Computer Reference Manual.* Regnecentralen, Copenhagen, Denmark, (June).

Brinch Hansen, P. 1970. The nucleus of a multiprogramming system, *Communications of the ACM 13*, 4 (April), 238-241, 250.

Brinch Hansen, P. 1971. An outline of a course on operating system principles. In C. A. R. Hoare and R. H. Perrott (eds.) 1972, *Operating Systems Techniques*, Proceedings of a Seminar at Queen's University, Belfast, Northern Ireland, August–September 1971, Academic Press, New York, 29–36.

Brinch Hansen, P. 1972. Structured multiprogramming. *Communications of the ACM 15*, 7 (July), 574–578.

Brinch Hansen, P. 1973a. *Operating System Principles.* Prentice-Hall, Englewood Cliffs, NJ, (July).

Brinch Hansen, P. 1973b. Concurrent programming concepts. *ACM Computing Surveys 5*, 4 (December), 223–245.

Brinch Hansen, P. 1974a. Deamy—a structured operating system. Information Science, California Institute of Technology, Pasadena, CA, (May).

Brinch Hansen, P. 1974b. The programming language Concurrent Pascal, Information Science, California Institute of Technology, Pasadena, CA, (November). Revised version in *IEEE Transactions on Software Engineering 1*, 2 (June 1975), 199–207.

Brinch Hansen, P. 1975a. Concurrent Pascal report. Information Science, California Institute of Technology, Pasadena, CA, (June).

Brinch Hansen, P., and Hartmann, A. C. 1975b. Sequential Pascal report. Information Science, California Institute of Technology, Pasadena, CA, (July).

Brinch Hansen, P. 1975c. The Solo operating system. Information Science, California Institute of Technology, Pasadena, CA, (June–July). Also in *Software—Practice and Experience 6*, 2 (April–June 1976), 141–200.

Brinch Hansen, P. 1975d. Concurrent Pascal machine. Information Science, California Institute of Technology, Pasadena, CA, (October).

Brinch Hansen, P. 1975e. A real-time scheduler. Information Science, California Institute of Technology, Pasadena, CA, (November).

Brinch Hansen, P. 1976a. The job stream system. Information Science, California Institute of Technology, Pasadena, CA, (January).

Brinch Hansen, P. 1976b. Concurrent Pascal implementation notes. Information Science, California Institute of Technology, Pasadena, CA.

Brinch Hansen, P. 1976c. Innovation and trivia in program engineering. Guest Editorial, *Software—Practice and Experience 6*, 2 (April-June), 139–140.

Brinch Hansen, P. 1976d. The programmer as a young dog. In Sveistrup (1976), 65–68 (In Danish). Also in P. Brinch Hansen 1996, *The Search for Simplicity: Essays in Parallel Programming.* IEEE Computer Society Press, Los Alamitos, CA, 142–156 (In English).

Brinch Hansen, P. 1977. *The Architecture of Concurrent Programs.* Prentice-Hall, Englewood Cliffs, NJ, (July).

Brinch Hansen, P. 1978a. Distributed processes: a concurrent programming concept. *Communications of the ACM 21*, 11 (November), 934–941.

Brinch Hansen, P. 1978b. Multiprocessor architectures for concurrent programs. *ACM 78 Conference*, Washington, DC, December, 317–323.

Brinch Hansen, P. 1981. Three papers on the programming language Edison. *Software—Practice and Experience 11*, 4 (April), 325–414.

Brinch Hansen, P. 1983. *Programming a Personal Computer.* Prentice Hall, Englewood Cliffs, NJ, (April).

Brinch Hansen, P. 1985. *Brinch Hansen on Pascal Compilers.* Prentice Hall, Englewood Cliffs, NJ, (August).

Brinch Hansen, P. 1987a. Joyce—a programming language for distributed systems. *Software—Practice and Experience 17*, 1 (January), 29–50.

Brinch Hansen, P. 1987b. A Joyce implementation. *Software—Practice and Experience 17*, 4 (April), 267–276.

Brinch Hansen, P. 1987c. The Joyce language report. *Software—Practice and Experience 19*, 6 (June), 553–579.

Brinch Hansen, P., and Rangachari, A. 1988. Joyce performance on a multiprocessor. School of Computer and Information Science, Syracuse University, Syracuse, NY, (September).

Brinch Hansen, P. 1989. A multiprocessor implementation of Joyce. *Software—Practice and Experience 19*, 6 (June), 579–592.

Brinch Hansen, P. 1990. The nature of parallel programming. In M. A. Arbib and J. A. Robinson (eds.), *Natural and Artificial Parallel Computation*, The MIT Press, Cambridge, MA, 31–46.

Brinch Hansen, P. 1992. Householder reduction of linear equations. *ACM Computing Surveys 24*, 2 (June), 185–194.

Brinch Hansen, P. 1993. Monitors and Concurrent Pascal: a personal history. *2nd ACM Conference on the History of Programming Languages*, (April), Cambridge, MA.

Brinch Hansen, P. 1994. Multiple-length division revisited: A tour of the minefield. *Software—Practice and Experience 24*, 6 (June), 579–601.

Brinch Hansen, P. 1995. *Studies in Computational Science: Parallel Programming Paradigms.* Prentice Hall, Englewood Cliffs.

Brinch Hansen, P. 1999a. *Programming for Everyone in Java.* Springer-Verlag, New York.

Brinch Hansen, P. 1999b. Java's insecure parallelism. *SIGPLAN Notices 34*, 4 (April), 38–45.

Brinch Hansen, P. 2002. IEEE Computer Pioneer Award: Acceptance speech. *IEEE Annals of the History of Computing 24*, 4 (October–December), 56.

Bronowski, J. 1973. *The Ascent of Man.* Little, Brown and Company, Boston, MA.

Buchholz, W. (ed.) 1962. *Planning a Computer System: Project Stretch.* Mc-Graw Hill, New York.

Campbell, R. H., and Habermann, A. N. 1974. The specification of process synchronization by path expressions. *Lecture Notes in Computer Science 16*, Springer-Verlag, Heidelberg, Germany, 89–102.

Ceruzzi, P. E. 2003. *A Modern History of Computing*, 2nd edition. The MIT Press, Cambridge, MA.

Cobol Discussion 1981. Question and answer session on Cobol. In Wexelblat (1981), 263–276.

Computerworld 1983. Stanford gets top grade in grad school poll. (January 24), 22.

Corbató, F. J., Merwin-Daggett, M., and Daley, R. C. 1962. An experimental time-sharing system. *Spring Joint Computer Conference 21*, 1962, 335–344.

Cosine Committee 1971. *An Undergraduate Course on Operating Systems Principles.* Commission on Education, National Academy of Engineering, Washington, DC, (June).

Crick, F. 1988. *What Mad Pursuit: A Personal View of Scientific Discovery.* Basic Books.

Dijkstra, E. W. 1965. Cooperating sequential processes. Technological University, Eindhoven, The Netherlands, (September).

Dijkstra, E. W. 1968a. Go to statements considered harmful.*Communications of the ACM 11*, 3 (March), 147–148.

Dijkstra, E. W. 1968b. The structure of the THE multiprogramming system. *Communications of the ACM 11*, 5 (May), 341–346.

Dijkstra, E. W. 1971a. Hierarchical ordering of sequential processes. In C. A. R. Hoare and R. H. Perrott (eds.) 1972, *Operating Systems Techniques*, Proceedings of a Seminar at Queen's University, Belfast, Northern Ireland, August–September 1971. Academic Press, New York, 72–93.

Dijkstra, E. W. 1971b. Reisverslag van Edsger W. Dijkstra aan Summer School, Marktoberdorf, juli 1971 (in Dutch).

Dijkstra, E. W. 1972. The humble programmer. *Communications of the ACM 15*, 10 (October), 859–866.

Dijkstra, E. W. 1973. Summer School Munich, July 25 to August 4. In E. W. Dijkstra 1982. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, New York.

Dijkstra, E. W. 1975. Summer School Marktoberdorf, August.

Dijkstra, E. W. 1999. Computing science: achievements and challenges. *ACM Symposium on Applied Computing*, (March), San Antonio, TX.

Discussions 1971. Discussions of conditional critical regions and monitors. In C. A. R. Hoare and R. H. Perrott (eds.) 1972, *Operating Systems Techniques*, Proceedings of a Seminar at Queen's University, Belfast, Northern Ireland, August–September 1971. Academic Press, New York, 100–113.

Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K. and Walker, D. W. 1988. *Solving Problems on Concurrent Processors*, Vol. I, Prentice Hall, Englewoods, NJ.

Gleick, J. 1992. *Genius—The Life and Science of Richard Feynman.* Pantheon Books, New York.

Greenfield, J. S. 1993. Distributed programming with cryptography applications. PhD thesis, Computer and Information Science, Syracuse University, Syracuse, NY, (December). Also in *Lecture Notes in Computer Science 870*, (1994), Springer-Verlag, New York.

Gregorian, V. 2003. *The Road to Home: My Life and Times.* Simon & Schuster, New York.

Grimberg, C. 1959. *Verdenshistorien.* ("The World History"). Politikens Forlag, Copenhagen, Denmark (in Danish).

Habermann, A. N. 1973. Critical comments on the programming language Pascal. *Acta Informatica 3*, 47–57.

Hartmann, A. .C. 1975. A Concurrent Pascal compiler for minicomputers. PhD thesis, Information Science, California Institute of Technology, Pasadena, CA, (September). Also in *Lecture Notes in Computer Science 50*, (1977), Springer-Verlag. New York.

Hillis, W. D. 1985. *The Connection Machine.* MIT Press, Cambridge, MA.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Communication of the ACM 12*, 10 (October), 576–580, 583.

Hoare, C. A. R. 1971. Towards a theory of parallel programming. In C. A. R. Hoare and R. H. Perrott (eds.) 1972, *Operating Systems Techniques*, Proceedings of a Seminar at Queen's University, Belfast, Northern Ireland, August–September 1971, Academic Press, New York, 61–71.

Hoare, C. A. R. 1973. A structured paging system. *Computer Journal 16*, (August), 209–214.

Hoare, C. A. R. 1974a. Monitors: an operating system structuring concept. *Communications of the ACM 17*, 10 (October), 549–557.

Hoare, C. A. R. 1974b. Hints on programming language design. In *Computer Systems Reliability*, C. Bunyan (ed.), Infotech International, Berkshire, England, 505–534.

Hoare, C. A. R. 1976a. Hints on the design of a programming language for real-time command and control. In *Real-time Software: International State of the Art Report*, J.P. Spencer (ed.), Infotech International, Berkshire, England, 685–699.

Hoare, C. A. R. 1976b. The structure of an operating system. *Lecture Notes in Computer Science 46*, F. L. Bauer and K. Samelson (eds.), 242–265. Springer-Verlag, New York.

Hoare, C. A. R. 1985. *Communicating Sequential Processes.* Prentice Hall, Englewood Cliffs, NJ.

Hoare, C. A. R., and C. B. Jones (eds.) 1989. *Essays in Computing Science.* Prentice Hall, New York.

Householder, A. S. 1958. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM 5*, 339–342.

Huskey, H. D., and Korn, G. A. (eds.) 1962. *Computer Handbook.* McGraw-Hill, New York.

IEEE Computer Society 2002. Brochure describing Per Brinch Hansen's Computer Pioneer Award.

Isaksson, H. 1976. *Fra Gier til RC 4000* ("From Gier to RC 4000"). In Sveistrup (1976), 57–64 (in Danish).

Joyce, J. 1937. *Ulysses.* The Bodley Head, London, England.

Kilburn, T., Payne, R. B., and Howarth, D. J. 1961. The Atlas supervisor. *National Computer Conference 20*, 279–294.

Kittel, C. 1956. *Introduction to Solid State Physics.* John Wiley, New York.

Klasseavisen ("The class newspaper") 1945–49. Konrad Jahn's class, Niels Ebbesensvej School, Frederiksberg, Denmark (in Danish).

Knuth, D. E. 1968. *The Art of Computer Programming. Vol. 1. Fundamental Algorithms.* Addison-Wesley, Reading, MA.

Kraks Blå Bog ("Who's Who in Denmark") 1959. Kraks Legat, Nytorv 17, Copenhagen, Denmark (in Danish).

Lauesen, S. 1975. A large semaphore based operating system. *Communications of the ACM 18*, 7 (July), 377–389.

McKeag, R. M. 1972–73. A survey of system structures & synchronization techniques. Department of Computer Science, The Queen's University of Belfast, Belfast, Northern Ireland (October 1972), with Supplement (January 1973).

Maddux, R. A., and Mills, H. D. 1979. Review of Per Brinch Hansen: The Architecture of Concurrent Programs. *IEEE Computer 12*, (May), 102–103.

Mason, S. J., and Zimmermann, H. J. 1960. *Electronic Circuits, Signals, and Systems.* John Wiley, New York.

Matelan, N. 1985. The Flex/32 multicomputer. *IEEE/ACM Symposium on Computer Architecture.* Boston, MA, (June), 209–213.

Medawar, P. B. 1979. *Advice to a Young Scientist.* Harper & Row, New York.

Menon, A. 1995. Unpublished student evaluation of professor Brinch Hansen's course on multicomputer programming at Syracuse University, (March).

Moore, G. 1979. VLSI: some fundamental challenges. *IEEE Spectrum*, (April), 30–37.

Naur, P. (ed.), Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauer, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., and Woodger, M. 1960. Report on the algorithmic language Algol 60. *Communications of the ACM 3*, 5 (May), 299–314.

Naur, P. 1963a. The design of the Gier Algol compiler. *BIT 3*, 2–3, 124–140 & 145–66.

Naur, P. 1963b. Go to statements and good Algol style. *BIT 3*, 3, 204–208.

Naur, P. 1966a. Program translation viewed as a general data processing problem. *Communications of the ACM 9*, 3 (March), 176–179.

Naur, P. 1966b. The science of datalogy: Letter to the editor. *Communications of the ACM 9*, 7 July, 485.

Naur, P. 1968. Datalogy, the science of data and data processes and its place in education. *IFIP Congress 68*, vol. II, 1383–1387.

Naur, P., and Randell, B. (eds.) 1969. *Software Engineering.* Nato Science Committee, (October), Brussels, Belgium.

Naur, P. 1974. *Concise Survey of Computer Methods.* Petrocelli/Charter, New York.

Naur, P. 1975. Review of Per Brinch Hansen: Operating System Principles. *BIT 15*, 455–457.

Naur, P., Johansen, P., and Koster, C. H. A. Koster 1984. Recommendation to University of Copenhagen regarding the appointment of a new professor in datalogy.

Nori, K. V., Ammann, U., Jensen, K., and Naegeli, H. H. 1974. The Pascal P compiler: implementation notes. Institut für Informatik, ETH, Zurich, Switzerland, (December).

Ostenfeld, C. (ed.) 1976. *Christiani & Nielsen: The Danish Pioneers of Reinforced Concrete.* Polyteknisk Forlag, Lyngby, Denmark.

Perlis, A. J. 1962. The computer in the university. In M. Greenberger (ed.), *Computers and the World of the Future.* The MIT Press, Cambridge, MA, 180–217.

Perlis, A. J. 1981. Talk on "Computing in the fifties." *ACM National Conference.* Nashville, TN. Transcript in J. A. N. Lee (ed.), *Computer Pioneers*, IEEE Computer Society Press, Los Alamito, CA, 1995, 545–556.

Perlis, A. J. 1982. Epigrams on programming. *ACM SIGPLAN Notices 9*, (September), 7–13.

Rabin, M. O. 1980. Probabilistic algorithms for testing primality. *Journal of Number Theory 12*, 128–138.

Randell, B., and Russell, L. J. 1964. *Algol 60 Implementation.* Academic Press, New York.

Rivest, R. L., Shamir, A. and Adleman, L. M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 120–126.

Rosovsky, H. 1990. *The University: An Owner's Manual.* W. W. Norton & Company, New York.

Scherfig, H. 1940. *Det Forsømte Forår.* ("The Neglected Spring"). Gyldendal, Copenhagen, Denmark (in Danish).

Seitz, C. L. 1985. The Cosmic Cube. *Communications of the ACM 28*, (January), 22–23.

Sellar, W. C., and Yeatman, R. J. 1964. *1066 and All That.* Penguin Books, Hammondsworth, Middlesex, England.

Shute, N. 1954. *Slide Rule: The Autobiography of an Engineer.* Ballantine Books, New York.

Siegel, M., and Smith, A. E. 1962. Interim report on Bureau of Ships Cobol evaluation program. *Communications of the ACM 5*, (May).

Simonton, D. K. 1984. *Genius, Creativity, and Leadership.* Harvard University Press, Cambridge, MA.

Snow, C. P. 1970. *Last Things.* Charles Scribner's Sons, New York.

Speiser, A. P. 1961. *Digitale Rechenanlagen.* Springer-Verlag, Berlin, Germany.

Spenke, E. 1958. *Electronic Semiconductors.* McGraw-Hill, New York.

Sveistrup, P., Naur, P., Hansen, H. B., and Gram, C. (eds.) 1976. *Niels Ivar Bech—en epoke i edb-udviklingen i Danmark.* ("Niels Ivar Bech—An Era in the Development of Electronic Data Processing in Denmark"). Data, Copenhagen, Denmark (mostly in Danish).

Terman, F. E. 1955. *Electronic and Radio Engineering.* McGraw-Hill, New York, 1955.

U. S. Department of Defense 1961. *Cobol-1961, Report to Conference on Data Systems Languages.* Washington, DC.

Welsh, J., and Bustard, D. W. 1979. Pascal-Plus—another language for modular multiprogramming. *Software—Practice and Experience 9*, 11 (November), 947–957.

Wexelblat, R. L. (ed.) 1981. *History of Programming Languages.* Academic Press, New York.

Wulf, W. A., and Bell C. G. 1972. C.mmp—a multiprocessor. *Fall Joint Computer Conference*, 765–777.

INDEX

Abstract machine, 67, 84
Ada language, 175–177
Adleman, Leonard, 134
Albertsen, Jørgen, 12–13
Algol 58 Anniversary, 74
Algol 60 language, 38, 151, 175
Algol compilers, 39–41
Algol W language, 74
Alto personal computer, 62
Andersen, Birger, 156–157
Andersen, Christian, 80
Andersen, Frode, 9
Andersen, Hanne, 12
Andersen, Kurt Henning, 76
APL language, 175
Arbib, Michael, 168
Architecture of Concurrent Programming, The, 4, 137–140
ARPA, 82
Assembly language, 39, 49
Atlas computer and supervisor, 41, 74
Await statement, 112

Backus, John, 23
  BNF notation of, 38–39
  HOPL conference, at, 175
  Naur's contribution to Algol 60, on, 38
Bagger, Bent, 42
Bak, Ole (cousin), 12
Balling, Knud, 148
Basic language, 175
Basic monitors, 96
Batch processing, Caltech, at, 108–109
Bauer, Friedrich (Fritz), 89
Bayer, Rudolph, 89
Bech, Niels Ivar, birth of, 76
  childhood and youth of, 76
  death of 79
  fired as director, 79
  first impression of, 36
  horse trading of, 78
  IFIP, founding member of, 78
  IFIP Silver Core Award of, 78
  middle name, invents, 76
  NordSAM conferences, organizes, 78
  Pulawy project, on, 60
  portrait of, 75–80
  stuttering of, 76
  unorthodox management style of, 76–77
Belfast Symposium, 89–96
Bell, Gordon, 85, 115, 142
Bishop, Judy, 152
BIT (Scandinavian journal of computing), 78
Bjerge, Gunnar, 7
Bohr, Harald, 19
Bok, Derek, 135–136
Borup, Morten, 13
Boss 2 system, 73
Bravo editor, 62
Brinch Hansen, Elsebeth (mother, née Ring), 1–4
Brinch Hansen, Eva (sister), 9
Brinch Hansen, Jørgen (father), 1–4, 9
  Doctor technices degree of, 138
  inspiration of, 22
  making gunpowder, 12
Brinch Hansen, Mette (daughter), 85, 163, 180
Brinch Hansen, Milena (wife, née Hrastar), 23, 131, 180
  Caltech, visits, 105
  Denmark, in, 153
  India, in, 118–119
  Pittsburgh, in, 85–85
  Syracuse, in, 163–164
Brinch Hansen on Pascal Compilers, 157
Brinch Hansen, Per,
  Ada language, on, 176–177
  Algol 58 Anniversary, at, 74
  alienation of foreigners, on, 161
  all-pairs computation of, 170–171
  America, in, 81–152, 163–180
  American citizenship of, 164
  American football, on, 133–133
  American hospitality, on, 161
  Architecture of Concurrent Programs, The, 137–140

Bech, Niels Ivar, meets 36
Belfast symposium, at, 92–96
birth of, 1
Brinch Hansen on Pascal compilers by, 157
brushfire and mudslides, facing, 145–149
California Institute of Technology, at 103–130
Caribean vacations of, 164
Carnegie-Mellon University, at, 85–89
Chancellor's medal, awarded, 168
chemical experiments of, 11–12
Chesney, Miles, negotiating with, 168
childhood friends of, 6–8
C language, on, 176–177
Cobol file system, programs, 53–55
computational science of, 165–175
  criticism of, 173–174
computer architecture, on, 32–33
computer science chair at USC, 134–137
Computing Surface, programs, 167–175
Concurrent Pascal, invents, 113–115
conditional critical regions, on, 95
consulting for GNT, 158–160
consulting for Mostek, 141–145
crazy ideas, on, 177
Danish students, on, 156–157
datamaton, invents the term, 42
Denmark, in, 1–80, 153–162
Dijkstra, Edsger, and, 91, 99, 124
Distinguished Professor, named, 163
division algorithm of, 173
Doctor technices degree of, 138–141
drinking Easter brew, 21, 42
driving across America, 106–108
Edison language, invents, 143–144
Edison multiprocessor architecture, designs, 142–144
education of, early, 6–15
England, in, 23–28
fired for being late, 22
first computer program of, 43
first major report by, 7
first technical paper by, 29
future wife, meets, 31–32
Ginsburg, Seymour, on, 137
graduation of, 35
Haarder, Bertel, meets, 154
Habermann, Nico, on, 85, 87–88
Hartmann, Al, on, 127–128
Harvard tenure committee, advisor to, 135–136
Head of RC 4000 software development, as, 68–75
Henry Salvatori Professor, named, 141
high-school education of, 9–15
history of computing, on, 175–78
Hoare, Tony, and, 74, 123
HOPL II conference, at, 176–178
IBM Hursley Laboratory, at, 26–28
IEEE Computer Pioneer medal, awarded, 178
IEEE Fellow, elected as, 161
India, in, 118–119
industrial student practice of, 22
Ingargiola, Giorgio, on, 127
international exchange student, as, 22–28
Joyce, James, *Ulysses*, on, 14
Joyce language, invents, 159–160
Kaprielian, Zohrab, on, 132
kindergarten, in, 5
language reports, on, 37–38. 177
languages as theories of computation, on, 177
Lasers and Masers, essays and radio talk by, 29–30
late-night studying of, 22
London, in, 24–26
love of jazz and literature, 13
McCann, Gilbert, on, 108
magic of naming, the, on 60–61
Marktoberdorf Summer Schools, at, 89–92, 99, 123–124
Menon, Anil, on, 174–175
monitor notation, invents, 111–115
Munich, in, 50–57
Naur, Peter, and Jensen, Jørn, meets, 39
negotiates salary, 49
Neuschwanstein Castle, at, 92
nominates Tony Hoare for honorary doctorate, 141
NordSAM conferences, at, 53, 85
operating system courses, on, 138
Operating System Principles by, 87–101
parallel scientific computation of, 165–175
parents of, 1–4
Pascal language, on, 87
Perlis, Alan, on, 83–84
PhD students, on, 105–106
professional discipline, on, 21
program descriptions, on, 56
Programming a Personal Computer by, 150–151
programming language requirements, on, 176–177
program optimization, on, 53
queuing variables of, 114
rapid prototyping, on, 160
RC 4000 computer architecture, designs, 62–64

RC 4000 multiprogramming system, describes, 72–73
RC 4000 real time system, designs, 61, 67
Regnecentralen, at, 35–80
research, on, 77, 113
research proposals, on, 124–126
“secretary” concept of Edsger Dijkstra, on, 96
secure programming languages, on, 176–177
Siemens Cobol, working on, 46–57
simplicity, on, 177
Slovenia, in, 30–31, 56–57
Solo operating system of, 120
Structured multiprogramming by, 104–105
student democracy, on, 155–156
student travel guide, as 23–24
Studies in Computational Science by, 174
Syracuse University, at, 163–180
teaching, at Caltech, 110–111
teaching mathematics, on, 19–20
Technical University of Denmark, at, 17–33
temperament of, 180
tenure standards of, 135, 161
The nature of parallel programming by, 169
tunnel diode project of, 21
University of Copenhagen, at, 153–162
University of Southern California, at, 131–152
wedding of, 56–57
Winchester, in, 25–26
writing, on, 1–4
Yugoslavia, in, 30–32
Zepko, Tom, on, 129–130

Brinch Hansen, Thomas (son), 85, 133–134, 163, 180
British Museum, 24–25
Brøndum, Johannes Arboe, 30
Brown, Harold, 103–104, 114–115
Bruun, Georg, 21, 30–31
Burstall, Rod, 118
Business data processing, 44

California Institute of Technology (Caltech) 103–130
batch processing at, 108–109
computer science at, 104, 108, 110, 126–127
Honor Code of, 111
student pranks at, 104

Campbell, Roy, 140
Cannon, Robert, 126–128
Canute the Great (King of Denmark and England), 25
Carnegie-Mellon University, 81–89
CDC 1604 computer, 41–42, 60
Ceruzzi, Paul, 44–45
Christensen, Helge, 20–21
Christensen, Leif, 12
C language, 175–178
C++ language, 175
Class newspaper, 6–8
Clauser, Francis, 105, 111
CLU language, 176
C.mmp multiprocessor, 142
Cobol compiler project, 41–56
Bureau of Ships, evaluated by, 55
compilation speed of, 55
compiler passes of, 47–50
file system of, 53–55
paper on, 56
parser of, 48
program documentation of, 49
programming effort of, 55
restart feature of, 55
size of, 55
system updates, handling of, 55-56
testing of, 49–50

Cobol language, 37, 44–45
Colmerauer, Alain, 176
Communications of the ACM, 82
Compilation checks
critical regions, of, 94–95
Joyce language, in, 159–160
monitors, of, 111–112, 114

Compilers, 39
multipass structure of, 47–50

Compiler testing, 41
Computer architecture, 32–33
Computing Surface, 167–175
Concurrent Pascal language, 113–115
abstract machine of, 117
compiler of, 116–117
compromises in, 117
contributions of, 151–152
Danish industry uses, 157–158
Hoare, Tony, on, 123
Job-stream system in, 121
kernel of, 117–118
history of, 176
McDonnell Douglas, uses, 158
microcomputer subset mCP, 158
model operating systems in, 119–124
modular programming in, 114, 119–121
PDP 11 implementation of, 116–118
portable implementation of, 117
processes in, 114–115
program testing in, 121–122

Real-time scheduler in, 120–121
report on, 114
system distribution of, 125–126
Wirth, Niklaus, on, 124
Concurrent programming, 64
Conditional critical regions, 94–96, 104–105
Brinch Hansen, Per, on, 95
Edison language, in, 143
Connection machine, 167
Control Data Corporation (CDC), 41
"Conversational process," 96
Cooperating Sequential Processes, 66–67
Corbató, Fernando, 108
Coroutines, in Boss 2 system, 73
Cosmic Cube, 166
Critical regions, 94–96
monitors, in, 112
semaphores, using, 94
Crystaline operating system, 166

Dahl, Ole-Johan, 111, 139, 175
Danish Servo Technology, 22
Dask Algol compiler, 40
Dask computer, 35–36
Datalogy, datamaton, and datamatics, 42
David May, 167–168
Deadlock prevention, in Boss 2 system, 73
in THE multiprogramming system, 85
Demand paging, 41, 74
Deverill, Robert, 116–117
Digital Equipment Corporation (DEC), 115
Dijkstra, Edsger, 131
Algol 58 Anniversary, at, 74
Algol 60 compiler, first, 40
Brinch Hansen, Per, and, 91, 123–124
Cooperating Sequential Processes by, 66–67
goto statments, on, 78
Hoare, Tony, on, 90
IBM OS/360, on, 141
Naur's Algol 60 report, on, 39
Perlis, Alan, on, 91–92
program correctness, on, 84–85
RC 4000 multiprogramming system, on, 73
"secretary" concept of, 95–96
semaphores, introduces, 66
software crisis, on, 75
speed independence, on, 66
structured programming, invents, 75
THE multiprogramming system of, 84–85
Wirth, Niklaus, on, 90–91
Wirth's Pascal compiler, on, 90–91
Wulf, Bill, on, 123–124
DIKU *see Institute of Datalogy*

Early school democracy, 8–9
Easter Brew ("Påske bryg"), 21, 42
Edison compiler, 144
Edison language, 143–144
Edison multiprocessor, architecture defined in Edison language, 143–144
United Technologies cancels project, 144
Edison system, for microcomputers, 150–151
Einstein, Albert, 29, 180
Elisabeth II (Queen of England), 17
Encore Multimax computer, 115, 165
Eriksen, Sven, 48, 55–56
Euler language, 74
Event queues, insecure, 66

Fellows, Jonathan, 149–150
Feynman, Richard, PhD students, on, 105
Flex/32 multiprocessor, 144
Følner, Erling, 19
Fortran language, 23, 38, 151
Fox, Geoffrey, 166, 168, 171
Francez, Nissim, 134
Franzen, Wolfgang, 120
Fraser, Alexander (Sandy), 92

Gargarin, Juri, 26
Generic programs, 171
Gier computer, 36
Algol compiler of, 40
hand coding of, 62
open shop operation of, 43
Giese, Allan, 63
Ginsburg, Seymour, 131–132, 134–135
Brinch Hansen, Per, on, 137
Fletcher Jones Professor, named, 140
Goings, Stephen, 141–143, 145
Golomb, Solomon, 132
Gomory, Ralph, 168
Gram, Christian, 59, 63–64, 80, 139
Great Northern Telegraph Company (GNT), 158–160
Greenfield, Jonathan, 173, 178
Griffith, Michael, 74
Gundel, Leif, 13
Gundel, Sven, 12–13

Haarder, Bertel, 154
Haarder, Bob and Eileen, 161
Habermann, Arie Nicolas (Nico), 84–85, 87–88, 140
Pascal language, on, 87
Hald, Jens, 59
Hansen, Henning Bernhard, 42, 59, 80
Harsen, Ann, 12

Hartley, David, 92
Hartmann, Alfred, 116–117
  Brinch Hansen, Per, on, 127–128
Harvard University, tenure appointments at, 135–136
Havsteen, Nils, 20
Hayden, Charles, 150
Hede, Ernst, 159
Heidam, Niels Zeuthen, 18
Hintz, Edith, 10
History of Programming Languages Conferences, *see* HOPL and HOPL II
Hoare, Charles Anthony Richard (Tony), 105, 134
  Belfast Symposium, at, 89–90
  Brinch Hansen, Per, on, 111–112, 123
  Caltech, visits, 125–126
  Concurrent Pascal, on, 123
  conditional critical regions of, 94–96
  CSP concept of, 159
  Dijkstra, Edsger, on, 90
  honorary doctorate at USC, receives, 141
  IFIP 68 Conference, at, 74
  Markoberdorf Summer Schools, at, 89–90
  monitor tutorials by, 112–113
  Naur's Algol 60 report, on, 140
Holt, Rick, 88
Hopkins, Martin, 141
HOPL conference, 44–46, 175
HOPL II conference, 175–178
Horning, James, 88
  Operating System Principles, on, 98–99
Horowitz, Ellis, 132, 134–135
Hotel Marina, meeting at, 69
House, Roger, 48, 56, 162
Howarth, David, 65, 74
Hrastar, Milena (future wife), 31–32
  student travel guide, as, 30–32
Husum, Sven, 12–13
Hydra operating system, 124
Hypercube computers, 166–167

IBM/360 computers, 28
IBM 704 computer, 23
IBM Hursley Laboratory, 23, 26–28
IBM OS/360, 141
IBM Personal Computer, 150, 154, 159
IBM Project Stretch, 32
IBM SCAMP computer, 28
IFIP 68 Conference, 74
Ilsøe, Peter, 9–10
Information Sciences Institute (ISI), 133
Ingargiola, Giorgio, 110
  Brinch Hansen, Per, on, 127
Inmos, 167–168
Institute of Datalogy (DIKU), 153–161
Interpreted code, defined, 67
Isaksson, Henning, 36, 60–61, 63, 80
Iverson, Ken, 175

Jahn, Konrad, 6–9
Java language, 117
Jensen, Birgit, 13
Jensen, Henning Højgaard, 20
Jensen, Jens Rasmus, 21
Jensen, Jørn, 63
  clever handcoding of, 62
  first impression of, 37
  Perlis, Alan, on, 39–40
Jensen, Per Gert, 21
Jensen, Toke, 40
Johansen, Peter, 153
Jones, Anita, 85
Joyce, James, 14
Joyce language, 159–160

Kampmann, Viggo, 79
Kampp, Aage, 9
Kaprielian, Zohrab, 132–133
  death of, 141
  Golomb, Sol, on, 132
  negotiating with, 132–133
Karlstrom, Karl, 98
Kay, Alan, 176
Keller, Herbert, 126
Kernel, of operating system, 71–72
Kilburn, Tom, 65
Kiær, Berta, 48
Knudsen, Hans Lottrup, 21
Knuth, Donald, 1, 105, 111, 131
  Art of Computer Programming, The, by, 59
Koster, C. H. A. (Kees), 153
Kraft, Peter, 48, 72, 80
  RC 4000 computer architecture, designs, 62–64
  RC 4000 prototype, installs, 68
  RC 4000 real time system, designs, 61
  Toft, Villy, on, 61
Krutar, Rudy, 85
Kurtz, Thomas, 175

Lampson, Butler, 62, 75, 105, 131
Lasers, 29
Lauesen, Søren, 68–69, 73–74, 80
Licklider, Joseph Carl Robnett, 82
Lindgreen, Paul, 48, 74, 80
  datamatics, invents the term, 42
Liskov, Barbara, 176
LISP language, 175

Load balancing, 165
Locanthi, Bart, 130
Ludwig II (King of Bavaria), 92
Lundgren, Helge, 17
Lynch, William, 131
Lyngsøe, Søren T., 22

McCann, Gilbert, 105–106, 125
  background and personality of, 108
  Head of computing center, as, 109–109
  invites Per Brinch Hansen to Caltech, 103–106
  loses influence, 109–110, 126
  PDP 11 computer of, 116
McCarthy, John, 126, 175
Machine code, *see* Assembly language
McKeag, Michael, 92, 96, 114
Maddux, Roy, 137–138
Magnetic tapes, error recovery of, 54–55
Mahoney, Michael, 176
Mainframe computers, 109, 116
Manna, Zohar, 134
Margrethe II (Queen of Denmark), 154–155
Marktoberdorf, Summer Schools in, 89–92, 99, 123–124
Masers, 29
Matelan, Nicholas, 143–145
Mattson, Harold (Skip), 76
Mead, Carver, 126–127
Meiko, 168
Meiko Computing Surface, 167–175
Melbye, Aage, 36, 80
Memory protection, 51–52, 117
Menon, Anil, 174–175
Message passing, 70–72
Michaelsen, Erik, 6
Microcomputers, 150–151
Miller-Rabin algorithm, 172–173
Mills, Harlan, 131
  Architecture of Concurrent Programs, The, reviews, 137–138
Minicomputers, 115
Model programs, 171
Modula and Modula-2 languages, 74
Møller, Ole, 80
Møller, Preben, 7
Modular programming, 114, 119–120, 151
Mondrup, Per, 40
Monitor program, in RC 4000 multiprogramming system, 65
Monitors,
  Await statements in, 112
  Concurrent Pascal, in, 113–115
  Edison multiprocessor, in, 142–143
  object-oriented programming, as, 151–152
  queuing variables in, 114
  Shared classes as, 111–112
Monitors and Concurrent Pascal: A personal history, 176
Monte Carlo method, 44
Moore, Gordon, 165
Mossin, Einar, 72
Multicomputers, 159, 166–175
Multipass compilation, 47–50
Multiple-lenth division, 173
Multiprocessors, 115, 142–145, 165–166
Multiprogramming, 65, 74
Munushian, Jack, 132
Mutual exclusion problem, 93

Nato Conferences on Software Engineering, 75
Naur, Peter, 63, 131, 153
  Algol 60 report of, 38–39
  Architecture of Concurrent Programs, The, reviews, 139–140
  BIT, contributions to, 78
  BNF notation of, 38–39
  compilation, on, 59
  compiler testing method of, 41
  datalogy, invents the term, 42
    vision of, 59
  Edison language report, on, 143
  first impression of, 37
  goto statements, on, 78
  HOPL conference, at, 175
  Operating System Principles, reviews, 99–100
  Programming a Personal Computer, reviews, 150–151
  University of Copenhagen, at, 80, 155
Nedergaard, Niels, 72
Needham, Roger, 92, 105
Neiiendam, Jan, 9–10
Newell, Alan, 82
  Perlis, Alan, on, 84
Northeast Parallel Architectures Center (NPAC), 165
Nucleus of a multiprogramming system, The, 72
Numeric computation, 44
Nygaard, Kristen, 111, 175

Oberon language, 74
Object-oriented concurrent programming, 151
occam language, 168
Olsen, Bent Vang, 12
Olsen, Willy, 79
On Pascal Compilers, 9
Open shop operation, 43, 116

Operating System Principles, 87–101
Operating system terminology, 98
Ørsted, Hans Christian, 17–18

Parallel programming, *see* Concurrent programming,
Parallel recursion, 159
Pascal compiler, Dijkstra, Edsger, on, 90–91
Pascal language, 44, 75, 87, 176,
Pascal Plus language, 123
Path expressions, 140
PDP 11 computers, 115–116
Pedersen, Niels Holm, 157–158
Perlis, Alan,
  America and the Soviet Union, on, 82
  invites Per Brinch Hansen to Carnegie-Mellon, 85–86
  computing, on, 81
  death of, 84
  Dijkstra, Edsger, on, 91–92
  epigrams on programming of, 83, 92
  HOPL conference, at, 175
  Jensen, Jørn, on, 39–40
  Marktoberdorf Summer School, at, 91
  programming course, purpose of, 81–82
  story telling of, 91
  university administrators, on, 83
Perlis, Sydelle, 175
Petersen, Bent Scharøe, 36
Petersen, Richard ("Little P"), 20
Petersen, Risto, 157
Piece, John, 126
PL/I language, 66, 175
PL 360 language, 74
Portable code, 67
  Concurrent Pascal implementation, in, 117
  Joyce implementation, in, 159–160
Primality testing, 172
Processes,
  Concurrent Pascal, in, 114–115
  load balancing of, 165
  RC 4000 multiprogramming system, in, 70–71
  trace model of, 140
Programming a Personal Computer, 80, 150–151
Programming for Everyone in Java, 44
Programming paradigms, 170–172
Program testing,
  Concurrent Pascal programs, of, 121–122
  Cobol compiler, of, 49–50
  THE multiprogramming system, of, 121–122
Prolog language, 176
Pulawy project, 60–68

Raasted, Anders, 159–160
Radin, George, 175
Rahbek, Just, 9, 11
Randell, Brian, 40, 74
Rangachari, Anand, 165
RC 2000 paper tape reader, 76, 78
RC 3000 data converter, 61
RC 4000 computer, 60–68
  architecture of defined in Algol 60, 63
  floating-point arithmetic of, 64
  instruction format of, 62–63
  prototype of, 64, 68
  RC 4000 Computer: Reference Manual, 64
RC 4000 multiprogramming system, 68–74
  "conversational processes" in, 96
  design of, 68–72
  Dijkstra, Edsger, on, 73
  kernel of, 71–72
  paper and manual about, 72–73
  process concept of, 70–71
  process swapping in, 72–73
  RC 4000 Computer Software: Multiprogramming System, 72–73
  reliability of, 73
  remote procedure calls in, 70–71
  separation of policy and mechanism in, origin of, 71–72
  server processes in, 96
  system kernel of, 71
  Wirth, Niklaus on, 74
RC 4000 real time systems,
  fertilizer plant, for, 64–68
  power plants, for, 72
  weather bureau, for, 72–73
Regnecentralen (RC), Denmark, 35–80
  ceases to exist, 79
  compiler group of, 37
  Danish universities, and, 79–80
  Pulawy project, at, 60–68
  RC 2000 paper tape reader of, 76, 78
  RC 3000 data converter of, 61
  RC 4000 computer of, 60–74
Remote procedure calls, 71–72
Resource managers, 96
Reynolds, John, 131, 163
Riis, Ole, 48
Ring, Børge (uncle), 13
Ring, Oluf (grandfather), 2
Ritchie, Dennis, 175–178
Robinson, Alan, 168
Rosovsky, Henry, 135–136, 156
Roulette simulation, 43–44

Roussel, Philippe, 176
Rovsing, Christian, 23
RSA cryptosystem, 134, 172–173
Russell, Lawford John, 40
Rybner, Jørgen, 21

Saitz, John, 64
Salvatori, Henry, 133
Sammet, Jean, 175, 177
Schai, Alfred, 74
Schoubye, Peter, 12
"Secretary" concept, 95–96
Seitz, Charles (Chuck), 166
Semaphores, 66–67
  error prone nature of, 70, 94
  mutual exclusion using, 94
Separation of policy and mechanism, 71–72
Sequential Pascal language, 116–117
Sevin, L. J., 141, 143–145
Shared classes, 111–112
Siemens Cobol compiler, 41–57
Siemens 3003 computer, 41
  magnetic tape files, 54
  memory protection of, 51
Simon, Herbert, 82
Simonyi, Charles,
  Microsoft, at, 62
  Microword Word, designs, 62
  RC 4000 real time system, programs, 61–62
  Regnecentralen, at, 61
  Xerox Parc, at, 62
Simula 67 language, 111
Simula language, 175
Smalltalk language, 176
Software crisis, 75
Solo operating system, 120
  Maddux, Roy, and Mills, Harlan, on, 138
  Naur, Peter, on, 139–140
  Wirth, Niklaus, on, 124
Sørensen, Eskild, 6
Sørensen, Knud Steenberg, 10–11
Sørensen, Per Just, 6
Soviet Exhibition in London, 26
Speed independence, 66
Sputnik, 20
Stankovič, Renata, 30
Stimulated light emission, 29
Stockholm, Per, 8
Stonehenge, 26
Stroustrup, Bjarne, 175
Structured multiprogramming, 98, 104–105
Structured programming, 75
Student democracy in Denmark, 155–157
Studies in Computational Science, 4, 174

Sutherland, Ivan, 126–127
Svalgaard, Leif, 72–73
Sveinsdottir, Edda, 155–156
Sveistrup, Poul, 80
Svejgaard, Bjarner, 42, 63
Sylvis, Edward, 148–149
Synchronization, 65–66
  Await statements, using, 112
  compilation checks of, 94–95
  conditional critical regions, using, 94–96
  message passing, using, 70–72
  monitors, using, 111–115
  queuing variables, using, 104–105, 114
  semaphores, using, 66–67, 94
  server processes, using, 95–96
  speed independence of, 66–67
  synchronous channels, using, 159
  time-dependent errors of, 66
  time-independence of, 94
Syracuse University (SU), 163–180
System kernel, extensible, 71–72

Tang Jespersen, René, 158–159
Tata Institute of Fundamental Research, 118
Telstar, 29
THE multiprogramming system, 84–85
  programming style in, 95–96
  testing of, 121–122
Thompson, Frederick, 108
Toft, Villy, 72, 68
Topsøe, Haldor, 60
Transputers, 167–168
Traveling Salesperson problem, 44
Tsichritzis, Dennis, 88

Udupa, Sriram, 118
Ulysses, 14
Uncapher, Keith, 133
United Technologies, 144–145
University of Southern California (USC), 131–152

VAX computer, 115
Villemoes, Peter, 48, 54
Vincow, Gershon, 163
Vinter, Otto, 72
VLSI microprocessors, 167–168
VLSI technology, 126–127

Waltenburg, Carl, 30
Waltenburg, Paul, 30
Wegstein, Joe, 44
Wessel, Alan, 59
Whiffen, Richard, 158
Whitaker, William, 175, 178

Wiehle, Hans Rudolf, 74
Winchester, England, 25–26
Wirth, Niklaus, 44, 105, 131
  Algol 58 Anniversary, at, 74
  Concurrent Pascal and Solo, on, 124
  Dijkstra, Edsger, on, 90–91
  HOPL II conference, at, 176
  innovative languages of, 74
  Markoberdorf Summer School, at, 89–90
  Pascal language, invents, 75
  Pascal report of, 87
  programming language requirements, on, 177
  RC 4000 multiprogramming system, on, 74
World War II, 3–5
Wulf, William, 85, 118
  C.mmp multiprocessor of, 142
  Hydra operating system of, 124

Zepko, Tom, 117
  Brinch Hansen, Per, on, 129–130
Zonneveld, Jaap, Algol 60 compiler, first, 40