Руслан Богатырев

Золотой треугольник

Источник: Мир ПК, #06-07/2001

Почти четыре столетия назад Блез Паскаль заметил: «Предмет математики настолько серьезен, что нужно не упускать случая сделать его немного занимательным». То же самое справедливо и по отношению к программированию.

Игра в камешки и лестница Кутафьей башни

Давайте рассмотрим простую игру [1]. Имеется кучка из N камешков. Каждый из двух играющих по очереди может извлекать их из этой кучки, при этом он должен брать не менее одного и не более удвоенного числа камешков, взятых противником на предыдущем ходу. Вначале он может взять любое их число (но не все сразу). Выигрывает тот, кто берет последний камешек. Вопрос: какой должна быть стратегия, ведущая к победе? Уточним задачу: сколько камешков надо взять первому игроку, если начальное их число равно 1000?

С чего начать? Как подступиться к решению? Можно ли решить задачу в уме, придется ли воспользоваться калькулятором или нам даже потребуется построить вспомогательные устройства и механизмы? А вообще имеет ли эта задача решение? Если на такой вопрос можно ответить однозначно, то считайте, что полдела сделано. Упростим нашу работу. Эта задача имеет решение, притом в уме, и с помощью калькулятора вы вряд ли его найдете (еще одна подсказка: оно единственное). Решение задачи строится в форме алгоритма. Значит, нам просто нужно описать задачу на каком-нибудь языке программирования и, нажав кнопку, получить ответ? Не спешите. Далеко не всегда «лобовой» способ приводит к успеху.

Попробуем сначала разобраться в природе задачи и отыскать ключ. То, что играющие могут удваивать число камешков по отношению друг к другу, наталкивает на мысль, что здесь не обойтись без методов, построенных на степени числа 2. Однако как учесть, что количество извлекаемых из кучки камешков может быть любым (в пределах удвоенного числа, взятого противником)? Пока не догадались? Тогда рассмотрим другую, более простую задачу, а к первой еще вернемся.

Из Александровского сада к Кутафьей башне, служащей входом на Троицкий мост Московского Кремля, ведет небольшая лестница. Ее высота 4,5 м, а длина основания 13,5 м (эти стороны образуют катеты прямоугольного треугольника). Для формирования ступенек используются плиты высотой 30 см и шириной 50 см. Вопрос: сколькими способами можно построить лестницу? [2]

Сначала обратим внимание на то, что плиты можно располагать встык (в том числе и поднимать ровно на их высоту), но нельзя их класть друг на друга. Поскольку высота лестницы составляет 4,5 м, то всего у нас будет ровно 15 ступенек по 30 см высотой, а количество плит равно 27 (это ясно из размеров основания лестницы и ширины плиты). Теперь до того, как понять способ решения задачи (но не точного определения числа вариантов — оно достаточно велико), нам остался всего один шаг. Девиз корпорации IBM «Думай!» (Think) — весьма популярная шутка в мире программистов, но в данной ситуации он весьма актуален.

Отложим на время обе задачи и зададимся вопросом: а что же такое алгоритм? Самое простое определение звучит так: это набор инструкций, который описывает, как некоторое задание может быть выполнено. Алгоритм характеризуют пять признаков:

- вход (входные данные, предусловия);
- выход (выходные данные, постусловия);
- конечность (алгоритм должен заканчиваться после выполнения конечного числа шагов);
- определенность шагов (каждый шаг алгоритма должен быть точно определен);
- выполнимость шагов (возможность ручной проверки).

Последний признак является самым туманным. Наиболее удачное неформальное определение выполнимости принадлежит Дональду Кнуту и приведено в его знаменитом труде «Искусство программирования» [1]. Он говорит, что инструкция выполнима, если включенные в нее операции достаточно элементарны, чтобы их за конечное время мог выполнить человек, вооруженный карандашом и бумагой.

Процесс интуитивного увязывания условий задачи с теми или иными алгоритмами, пожалуй, ключевой и наиболее трудный момент в ее решении. Но даже после того, как нам дают подсказки и приводят идею-«отмычку», встает вопрос технической реализации — ведь необходимо получить конкретный ответ. В программировании, как и в математике, поиск «отмычки» ведется на интуитивном уровне путем сопоставления задачи с уже знакомыми путями решения. Техническая же сторона здесь выглядит по-иному.

Комбинаторика и треугольник Паскаля

Вернемся к задаче о лестнице Кутафьей башни. Попробуем ее упростить. Если каждую ступеньку обозначить через 1, а плиту через 0, то любой вариант лестницы будет последовательностью нулей и единиц длиной m + k, где m — количество плит (в нашем случае 27), а k — количество ступенек (у нас их 15). Причем единицы не могут стоять рядом друг с другом (ступеньки не могут громоздиться). Сначала выписываем 27 нулей. Для единиц у нас есть 28 мест. Теперь нам осталось ответить на такой вопрос: сколькими способами можно расставить 27 нулей и 15 единиц так, чтобы никакие две единицы не стояли рядом? Другими словами, как выбрать 15 из 28 мест. Вспомним комбинаторику. Там есть такие понятия, как перестановки (n!), размещения без повторений ($A_n^k = n! / (n-k)!$) и сочетания ($C_n^k = n! / (k! \times (n-k)!)$). Нас интересуют сочетания. Это и есть путь к решению. Но попробуйте подсчитать это число (C_{28}^{15} = 37 442 160) в нашем случае. А если мы увеличим масштаб задачи? Итак, с технической точки зрения для задачи о лестнице Кутафьей башни нам нужно только построить специальную арифметическую машину, вычисляющую число сочетаний.

Как ее строить? Вычисление факториала — вроде бы довольно простая в реализации функция: последовательно уменьшай на единицу исходное число и все время перемножай с промежуточным результатом. Тут, правда, есть неприятная проблема: факториал растет так быстро, что уже 12! = 479 001 600, а поскольку представление целых чисел в современных ПК ограничено 32 разрядами, следующее значение выходит за разрядную сетку.

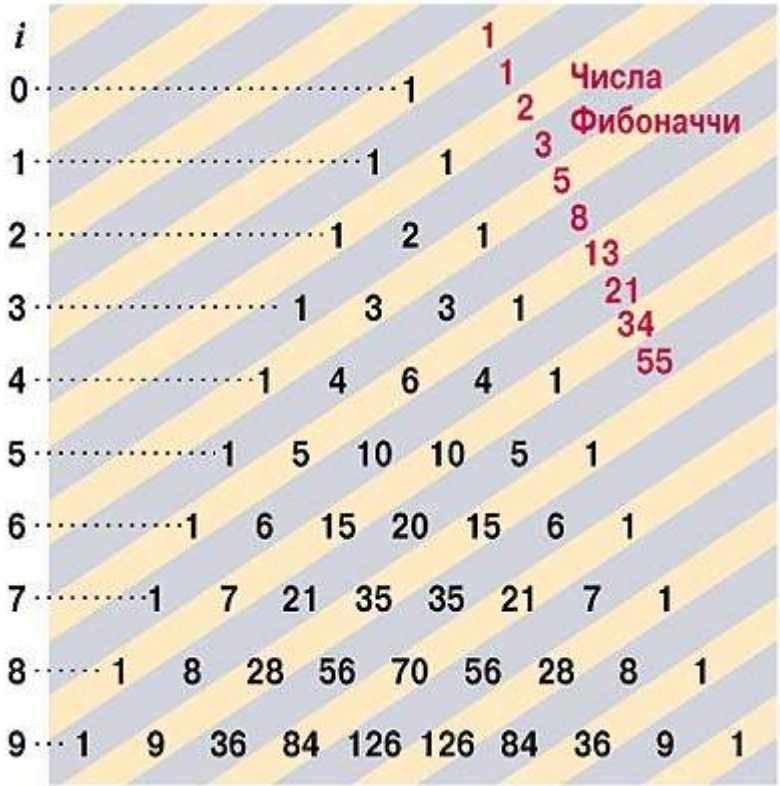


Рис.1. Треугольник Паскаля.

Опять вспомним математику (куда же без нее?). Число сочетаний можно вычислять не через факториал, а с помощью арифметического треугольника. Взгляните на рис.1. Строится треугольник довольно просто: его бедра и вершина состоят из единиц, а в основании каждый элемент строки получается суммированием двух стоящих непосредственно над ним элементов.

Арифметический треугольник мы неизменно связываем с именем Паскаля. В 1653 г. Блез Паскаль опубликовал одну из самых своих известных работ — «Трактат об арифметическом треугольнике» (в нем треугольник был повернут: его ребра выполняли роль катетов). Этот треугольник был известен еще в Древней Индии (X в.), а в XVI в. вновь открыт Штифелем.

Несложно заметить особенности треугольника: он симметричен относительно вертикальной оси, проходящей через его вершину. Первые внутренние диагонали, параллельные бедрам треугольника, образуют ряд натуральных чисел. Суммы чисел, стоящих вдоль не круто падающих диагоналей (получаемых ходом коня), образуют последовательность Фибоначчи. Связь между треугольником Паскаля и числами Фибоначчи была обнаружена лишь в XIX в. Обратите внимание на следующие свойства чисел треугольника Паскаля:

- Сумма всех чисел каждого ряда равна соответствующей степени числа 2 (известное свойство биномиальных коэффициентов).
- Сумма чисел на любой диагонали равна числу, расположенному снизу и слева от последнего слагаемого, соответственно $\Sigma\, n_{i=1}\, i = C_{n+1}{}^{2}$.
- $C_n^k = (n / k) \times C_{n-1}^{k-1}$ (внесение/вынесение).
- $C_n^k = C_n^{n-k}$ (симметрия).
- $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ (сложение).
- $C_n^k = (-1)^k \times C_{k-n-1}^k$ (обращение индекса).
- $C_n^m \times C_m^k = C_n^k \times C_{n-k}^{m-k}$ (упрощение произведений).

Известный популяризатор математики Мартин Гарднер [3] справедливо заметил, что «всякий раз, когда целые числа располагаются по какой-нибудь красивой единственно возможной схеме, у этой схемы оказывается масса самых неожиданных свойств». Свойства треугольника Паскаля поистине неисчерпаемы. Но пока нас все же будет интересовать его использование для нашей конкретной задачи.

Будем считать, что адресация элементов нашего треугольника записывается в виде: c(i,j), где i — номер строки (основания), а j — номер элемента на строке. Оба индекса нумеруются от 0. Самым важным для нас свойством треугольника Паскаля является то, что эти элементы и определяют число искомых сочетаний, т. е. $c(i,j) = C_i^{\,j}$, они же являются биномиальными коэффициентами, используемыми при разложении $(x+y)^n$ в бином Ньютона. (Биномиальные коэффициенты были подробно описаны индийским математиком Бхаскара Ачарья еще в 1150 г.)

Теперь у нас есть два варианта:

- реализовывать треугольник в виде какого-нибудь алгоритма и затем (для возможности практического применения) пытаться построить к нему «подъездные» пути;
- сначала построить каркас, обеспечивающий «подъездные» пути, а затем наращивать его соответствующими алгоритмами.

Для программирования в малом обычно используют первый вариант, как более легкий, но мы выберем второй — он позволит лучше понять процесс конструирования. Процесс превращения алгоритма в программу Ахо, Хопкрофт и Ульман [4] называют его пошаговой кристаллизацией. Ею мы займемся сразу после построения каркаса.

Проект Triangle

Начнем с архитектуры. Основными строительными блоками для нас будут модули и процедуры, а не классы и методы (их мы также будем применять, но лишь там, где это действительно требуется). Причина прозаична: эйфория объектно-ориентированного программирования (ООП) потихоньку уходит в небытие. Попытки решить все проблемы с помощью ООП уступают место прозрению: с наступлением компонентного программирования (component-based programming) меняется отношение к средствам инкапсуляции (предложенным еще Д.Парнасом в начале 1970-х годов), которые за последнее десятилетие были оттеснены классами на задний план. Речь идет о модулях.

С точки зрения области видимости и области существования между модулем и процедурой имеется существенная разница. Процедура прозрачна для импорта (все, что описано снаружи, автоматически видно и внутри нее) и закрыта для экспорта (за исключением списка формальных параметров). Модуль играет роль контейнера и вбирает в себя процедуры (а также классы, как специальные структуры данных со связанными процедурами) и устанавливает жесткие границы экспорта и импорта. Локальные элементы в процедурах существуют ровно то время, пока вызвана процедура, и уничтожаются после ее завершения. Локальные элементы в модулях существуют ровно то время, пока загружен модуль (как правило, до конца работы программы). Модули могут динамически подгружаться и выгружаться.

В качестве главной архитектурной схемы мы будем опираться на ставшую популярной благодаря языку Smalltalk концепцию MVC (Model/View/Controller , модель—вид—контроллер). Идея крайне проста: подобно тому как любая программа стоит на трех «китах» — на данных, логике и внешнем интерфейсе, в MVC они отображаются соответственно на модель, контроллер и вид. Предполагается, что модель обычно выступает в одном экземпляре, видов может быть много, а контроллер играет роль связующего слоя между первым и вторым.

Дополним и разовьем эту идею. По аналогии с концепцией MVC разобьем все наши модули на три вида:

- машины («системный» слой, модель);
- преобразователи (связующий слой, контроллер);
- сценарии («прикладной» слой, вид).

Преобразователи могут быть универсальными и специализированными. Универсальные имеют общий характер (привычные библиотечные модули); специализированные соединяют машины и сценарии. Деление на три вида совсем не обязательно должно предусматривать соответствие машин, скажем, уровню модели. Машина может использоваться и на уровне контроллера, если это удобно для данной задачи. Для иллюстрации наших решений мы будем использовать язык Оберон-2 [5].

В качестве примера программирования на этом языке приведем использование новой базовой идеи Оберона (http://www.oberon.ethz.ch) и Оберона-2 [5] — концепции расширения типа. Она раскрывается в реализации вспомогательного библиотечного модуля Time (листинг 1). При тестировании и выявлении эффективности реализации алгоритмов нам потребуется его секундомер.

Листинг 1. Модуль Time. Секундомер. Расширение типа. Связанные процедуры.

```
TYPE
  T* = RECORD
         hour* : INTEGER; (* [0..23] *)
         min*  : INTEGER; (* [0..59] *)
         sec*  : INTEGER; (* [0..59] *)
         msec* : INTEGER; (* [0..TactSEC-1] *)
       END;

  Stopwatch* = RECORD (T)
                 start : LONGINT;  (* начальное время в упакованном виде  *)
                 on-   : BOOLEAN;  (* контроль состояния: включен ли?     *)
               END;
```

```
PROCEDURE (VAR t: T) Pack* (): LONGINT;
  VAR  p: LONGINT; (* процедура связана с типом T; *)
                   (* это реализация метода класса *)
BEGIN
  ASSERT((t.hour >= 0) & (t.hour <= 23)); (* контрольные вставки *)
  ASSERT((t.min  >= 0) & (t.min  <= 59));
  ASSERT((t.sec  >= 0) & (t.sec  <= 59));
  ASSERT((t.msec >= 0) & (t.msec <= 999));
  p := t.hour * 60; (* упаковываем время, переводим в бинарный вид *)
  p := p + t.min; p := p * 60;
  p := p + t.sec; p := p * TactSEC;
  p := p + t.msec;
  RETURN p
END  Pack;

PROCEDURE (VAR sw: Stopwatch) Start* ;
  VAR t: T;
BEGIN
  ASSERT(~sw.on);
  sw.on := TRUE;  t.Get; sw.start := t.Pack();
END Start;

PROCEDURE (VAR sw: Stopwatch) Stop* ;
  VAR t,t2: T; dlt: LONGINT;
BEGIN
  ASSERT(sw.on);
  sw.on := FALSE; t.Unpack(sw.start); t2.Get; dlt := t.Delta(t2);
  sw.Inc(dlt); sw.start := 0;
END Stop;
```

Арифметическая машина реализуется в виде модуля, у которого есть команды (процедуры без параметров, берущие данные из регистров и сохраняющие там результаты) и вспомогательные процедуры. Она имеет индикаторы состояния (булевы переменные, характеризующие текущее состояние), а также обязательные команды On (включение, инициализация) и Off (выключение, финализация). Взгляните на «печатную машинку» — простейшую машину TypeWriter для вывода на экран (листинг 2).

Листинг 2. Модуль TypeWriter. Простейшая «печатная машинка».

```
MODULE TypeWriter;

  IMPORT Out;

  VAR sOn- : BOOLEAN;

  TYPE Register* = RECORD
                     ch*  : CHAR;
                     str* : ARRAY 128 OF CHAR;
                   END;
  VAR r* : Register;

(* -- основные процедуры *)

  PROCEDURE Char*;
  BEGIN Out.Char(r.ch);
  END Char;

  PROCEDURE NewLine*;
  BEGIN Out.Ln;
  END NewLine;

  PROCEDURE String*;
  BEGIN Out.String(r.str);
  END String;
```

```
  (* -- процедуры изменения состояний и регистров *)

    PROCEDURE StateOn;
    BEGIN sOn := TRUE;
    END StateOn;

    PROCEDURE StateOff;
    BEGIN sOn := FALSE;
    END StateOff;

    PROCEDURE ResetRegister;
    BEGIN r.ch := 0X;  r.str[0] := 0X;
    END ResetRegister;

  (* -- процедуры включения и выключения машины *)

    PROCEDURE On*;
    BEGIN StateOn; ResetRegister;
    END On;

    PROCEDURE Off*;
    BEGIN StateOff; ResetRegister;
    END Off;

  BEGIN
    On;
  END TypeWriter.
```

Сценарии подразумевают обработку информации на уровне текстовых строк. Посмотрим, как для нашей задачи реализуется сценарий (листинг 3). Он выполнен в виде компактного модуля Triangle (это будет наш головной модуль) и опирается на специальный преобразователь с именем TriangleTF, которому для удобства работы мы присвоим синоним TT.

Листинг 3. Уровень внешнего представления (View). Модуль Triangle.

```
  MODULE Triangle;   (* головной модуль проекта *)

    IMPORT
      In, (* RTS *)
      TT := TriangleTF; (* спец.преобразователь *)

    PROCEDURE Main;
      VAR s: ARRAY 256 OF CHAR;
    BEGIN
      TT.Print("-- Золотой треугольник");
      In.Open; In.Echo(FALSE);
      REPEAT
        In.String(s);
        TT.Print(s);
        TT.ExecCommand(s);
      UNTIL TT.Stop();
      TT.Print("-- Работа окончена");
   END Main;

  BEGIN
    Main;
  END Triangle.
```

При работе на каждом слое будем придерживаться следующего деления по видам представления данных:

- «регистровое» представление (поля записи);
- бинарное представление (бинарный вид);
- внешнее представление (строка).

Теперь на каждом из уровней определим поддерживаемые виды представления данных:

- машины — вход: «регистры»; выход: «регистры»;
- преобразователи — вход: бинарный вид, «регистры», строки; выход: бинарный вид, «регистры», строки;
- сценарии — вход: строки; выход: строки.

Проект Triangle. Структура модулей

Уровень	Модуль	Назначение
RTS		
	In	Ввод
	Out	Вывод
Library		
	Str	Обработка строк
	CharSet	Множества литер
	Number	Форматирование чисел
	Time	Часы и секундомер
	Math	Математические функции
	Parsing	Разбор строк
	TypeWriter	Машина вывода на экран
	TriangleTst	Тестирование
Model		
	Pascaline	Абстрактная машина
Controller		
	TriangleTF	Преобразователь
View		
	Triangle	Сценарий, головной модуль

Посмотрим, какой в итоге будет структура модулей нашего проекта (см. таблицу). Наибольший интерес в реализации для нас представляет модуль Pascaline, содержащий основные алгоритмы исходной задачи и созданный по аналогии с «паскалиной». При проектировании этого модуля будем исходить из того, что нам потребуется реализовывать несколько вариантов алгоритма (для их сравнения). При этом понадобится переключатель, который обеспечивал бы настройку команды машины на соответствующую процедуру, реализующую данный алгоритм (в нашем случае это семейство процедур PascalTriangle и Fibonacci, реализованные через процедурные типы). Интерфейс модуля Pascaline представлен на листинге 4. Основные структуры данных модуля Pascaline можно видеть на листинге 5.

Листинг 4. Интерфейс модуля Pascaline.

```
DEFINITION Pascaline;

  TYPE
    Register = RECORD (* регистры машины *)
                 ind1*   : SHORTINT;
                 ind2*   : SHORTINT;
                 int*    : LONGINT;
                 result* : LONGINT;
               END;
  VAR
    r*             : Register;
    sOn-           : BOOLEAN;
    sBadIndex-     : BOOLEAN;
    sPrimeIsEmpty- : BOOLEAN;
```

```
  (* -- команды машины *)

    PROCEDURE On;
    PROCEDURE Off;

    PROCEDURE C;  (* C(i,j) - вычисление бин. коэфф. *)
    PROCEDURE F;  (* F(i)   - i-е число Фибоначчи    *)

    PROCEDURE InitPrime;
    PROCEDURE FirstPrime;
    PROCEDURE NextPrime;
    PROCEDURE CountPrime;

  (* -- настройка машины *)

    TYPE
      ProcType1 = PROCEDURE (i: SHORTINT; j: SHORTINT): LONGINT;
      ProcType2 = PROCEDURE (i: SHORTINT): LONGINT;

    VAR
      PascalTriangle* : ProcType1;
      Fibonacci*      : ProcType2;

    PROCEDURE PascalTriangle0 (i: SHORTINT; j: SHORTINT): LONGINT;
    PROCEDURE PascalTriangle1 (i: SHORTINT; j: SHORTINT): LONGINT;
    PROCEDURE PascalTriangle2 (i: SHORTINT; j: SHORTINT): LONGINT;

    PROCEDURE Fibonacci0 (i: SHORTINT): LONGINT;
    PROCEDURE Fibonacci1 (i: SHORTINT): LONGINT;
    PROCEDURE Fibonacci2 (i: SHORTINT): LONGINT;
    PROCEDURE Fibonacci3 (i: SHORTINT): LONGINT;
    PROCEDURE Fibonacci4 (i: SHORTINT): LONGINT;

  END Pascaline.
```

Листинг 5. Уровень модели (Model). Модуль Pascaline. Структуры данных.

```
    TYPE (* регистры машины *)
      Register* = RECORD
                    ind1*   : SHORTINT;
                    ind2*   : SHORTINT;
                    int*    : LONGINT;
                    result* : LONGINT;
                  END;

    TYPE (* процедурные типы для переключателей процедур *)
      ProcType1* = PROCEDURE (i: SHORTINT; j: SHORTINT): LONGINT;
      ProcType2* = PROCEDURE (i: SHORTINT): LONGINT;

    VAR (* абстрактные процедуры для разных вариантов *)
      PascalTriangle* : ProcType1;
      Fibonacci*      : ProcType2;

    CONST
      MaxPasINDEX  = 31; (* 0..31 *)
      MaxFibINDEX  = 45; (* 0..45 *)
      MaxFactINDEX = 12; (* 0..12 *)
      PasCOUNT = ((MaxPasINDEX DIV 2)-1) * ((MaxPasINDEX-1) DIV 2); (* 210 *)
      FibCOUNT = 32;
```

```
VAR
  r*       : Register;
  primeDesc: PrimeDesc;
  isPrime  : POINTER TO ARRAY OF SET;
  pas      : ARRAY PasCOUNT OF LONGINT;
  fib      : ARRAY FibCOUNT OF LONGINT;
```

Итак, каркас у нас практически готов, осталось только реализовать алгоритмы и написать связующий модуль TriangleTF.

(Продолжение следует)

Литература

1. Кнут Д. Искусство программирования, т.1. М.: Вильямс, 2000.
2. Виленкин Н.Я. Популярная комбинаторика. М.: Наука, 1975.
3. Гарднер М. Математические новеллы. М.: Мир, 2000.
4. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. М.: Вильямс, 2000.
5. Moessenboeck H. Object-Oriented Programming in Oberon-2. Springer, 1993.
6. Гиндикин С.Г. Рассказы о физиках и математиках. М.: МЦНМО, 2001.

Алгоритм

Как это ни странно, слово «алгоритм» возникло от имени автора знаменитого персидского учебника по математике аль-Хорезми. Именно в его изложении десятичная система счисления индусов после перевода на латынь и выпуска книги Леонардо Пизано (Фибоначчи) стала доступна европейцам. Интересно, что гордостью коллекции Дональда Кнута является гашеная почтовая марка с изображением аль-Хорезми, выпущенная в 1983 г. в СССР к 1200-летию ученого и просветителя Востока.

Блез Паскаль и «паскалина»

Блез Паскаль (1623–1662) — один из самых знаменитых людей в истории человечества [6]. В 1640 г., чтобы помочь своему отцу, Этьену Паскалю, Блез придумывает машину, позднее названную Pascaline — «паскалина». Основная идея ее работы — каждое колесо (всего их восемь), совершая движение на десять цифр, заставляет двигаться соседнее колесо ровно на одну. Машина производила четыре действия над пятизначными цифрами. По чертежам Паскаля было изготовлено около 50 экземпляров. Относительно недавно стало известно, что Шиккард, друг Кеплера, в год рождения Паскаля построил другую арифметическую машину, однако машина Паскаля была куда совершеннее. «Это приспособление, — вспоминает сестра Блеза, Жильберта Перье, — рассматривали как новшество в природе, сводившее к совокупности машинных действий то знание, которое всегда было закреплено за умом».

Языки Оберон и Оберон-2

Язык Оберон (Oberon) был создан Никлаусом Виртом в 1988 г. при активном участии Юрга Гуткнехта. Он стал дальнейшим развитием линии языков Паскаль (1970) — Модула-2 (1979). Оберон — это король эльфов и муж Титании (Titania), ставший персонажем известного произведения Вильяма Шекспира «Сон в летнюю ночь». Именем Оберон был назван самый дальний и второй по величине спутник планеты Уран, открытый Уильямом Гершелем в 1787 г. Никлаус Вирт решил дать название «Оберон» языку, системе и специальному проекту под впечатлением успешного полета американского корабля Voyager 2 (из серии NASA Mariner), который, стартовав осенью 1977 г. (в год проектирования языка Модула-2), к январю 1986 г. достиг Урана. Язык Оберон-2 (1991) явился небольшой ревизией Оберона (добавлены открытые массивы, связанные процедуры, ограничение экспорта, оператор FOR). Два языка развиваются независимо и подобно ОС UNIX формируют целое дерево разных диалектов (http://www.oberon.ethz.ch/genealsys.html).

Реализация треугольника Паскаля

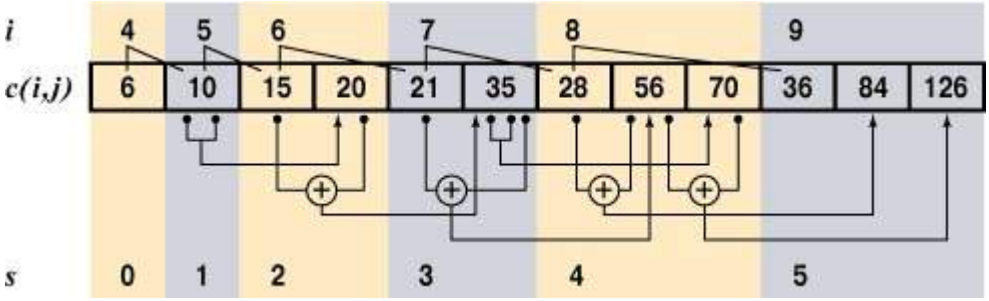


Рис. 1. Треугольник Паскаля в сегментированном массиве.

Простейшая (и наименее эффективная) реализация треугольника Паскаля выполняется в виде рекурсивной процедуры PascalTriangle0. В ней нужно предусмотреть возврат 1 при условии (i = 0) OR (j = i), а также возврат i при (i = 1) OR (j = i-1). Эти условия определяют бедра треугольника и первые внутренние диагонали. В остальных случаях должно вычисляться выражение x:= PascalTriangle0(i-1,j-1) + PascalTriangle0(i-1,j). Оно определяет суммирование элементов, стоящих выше текущего. Неэффективность алгоритма связана с тем, что чем глубже мы будем спускаться внутрь треугольника, тем больше будет повторяться вычислений, сделанных уже на предыдущих шагах.

Другим вариантом реализации алгоритма может служить использование факториала для значений, не превышающих 12 (как мы уже отмечали выше, это связано с ограничением разрядной сетки). Эту реализацию разместим в процедуре с именем PascalTriangle1. Здесь при реализации факториала (но уже не в виде рекурсивной процедуры, а через импровизированный стек) мы добьемся куда лучших результатов: stack[0] := x; stack[1] := k; x := stack[0] * stack[1]; INC(k).

Наконец мы вплотную подошли к реализации основного решения, за которое будет отвечать процедура PascalTriangle2. Обратим внимание (рис.1) на ту особенность треугольника Паскаля, что он симметричный, следовательно, вычисления необходимо делать только для левой его половины. Далее заметим, что вычислять верхние две диагонали не нужно — одна состоит из единиц, а лежащая под ней из последовательности натуральных чисел. Чтобы сократить время вычислений, имеет смысл вырезать в треугольнике Паскаля значимый внутренний треугольник и построить процесс определения его элементов. Для этого нам понадобится ввести понятие сегмента (это половина строки исходного треугольника Паскаля без первых двух элементов). Воспользуемся идеей динамического программирования (этот термин активно используется в области оптимизации). Данный подход позволяет решать задачу, разбивая ее на подзадачи и объединяя их решения. Нам необходимо выявить набор значений (для строк от 0 до 31), которые мы вычислим заранее (при загрузке модуля), а недостающие при запросе будем вычислять отдельно. Этот «зарезервированный» набор будем хранить в одномерном массиве pas. Его размер (за это отвечает константа PasCOUNT) при общем количестве 32 строки треугольника вы можете посчитать самостоятельно.

Для адресации элементов треугольника будем использовать индексы i (номер строки) и j (номер элемента в строке), а для адресации элементов в сегментах — индексы s (номер сегмента) и t (номер элемента в сегменте). Первым элементом одномерного массива pas будет c(4,2). Соответственно связь между индексами выглядит так: i = s+3; j = t+2.

Обратим внимание на зависимости при формировании сегментов: они показаны на рис.2. Реализуем их через процедуру LoadPas (для преобразования индексов она использует процедуру Segment). Теперь осталось включить вызов процедуры LoadPas в инициализацию модуля (процедура On) и для вычисления элементов треугольника Паскаля написать простейшую процедуру PascalTriangle2, обеспечивающую обращение к элементам готового массива. Нам еще необходимо доделать каркас, и задача будет решена (листинг 6).

Листинг 6. Модуль Pascaline. Реализация треугольника Паскаля через динамические сегменты.

```
PROCEDURE Segment (s,t: SHORTINT): LONGINT;
  VAR k: INTEGER;
BEGIN (* обращение к одномерному массиву через динамические сегменты *)
  k := (((s+3) DIV 2)-1) * ((s+2) DIV 2) + t;  (* вычисление индекса *)
  ASSERT(pas[k] > 0);          (* проверка на инициализацию элемента *)
  RETURN pas[k]
END Segment;

PROCEDURE LoadPas;
  VAR s,t,size: SHORTINT; k: INTEGER;
BEGIN (* заполнение массива готовыми значениями *)
  FOR k := 0 TO MaxPasINDEX-1 DO
    pas[k] := -1;(* для контроля за заполнением *)
  END;
  pas[0] := 6; (* первый элемент знаем *)
  k := 1; (* индекс массива pas *) s := 1; (* номер сегмента *)
  REPEAT (* цикл по сегментам *)
    size := ((s+4) DIV 2) - 1; (* кол-во элементов в сегменте *)
    t := 0; (* индекс внутри сегмента *)
    WHILE (k < PasCOUNT) & (t < size) DO (* цикл внутри сегмента *)
      IF (t = 0) THEN pas[k] := Segment(s-1,0) + (s+3);
      (* ODD — признак нечетности; смотрим посл. эл-ты четных сегментов *)
      ELSIF ~ODD(s) & (t = size-1) THEN
        pas[k] := 2*Segment(s-1,size-2);
      ELSE
        pas[k] := Segment(s-1,t-1) + Segment(s-1,t);
      END;
      INC(t); INC(k)
    END;
    INC(s);
  UNTIL (s > MaxPasINDEX+4);
END LoadPas;

PROCEDURE PascalTriangle2* (i,j: SHORTINT): LONGINT;
  VAR k: LONGINT;
BEGIN (* PRE: i >= 0; j >= 0; j <= i; i <= MaxPasINDEX *)
  IF    (j = 0) OR (j = i)   THEN RETURN 1 (* стороны треугольника    *)
  ELSIF (j = 1) OR (j = i-1) THEN RETURN i (* диагонали под сторонами *)
  ELSE (* j <= i; i > 3; j > 1 *)
    IF (j-2 >= ((i DIV 2)-1)) THEN j := i-j END; (* за счет симметрии *)
    k := (((i-1) DIV 2)-1) * ((i-2) DIV 2) + (j-2);   (* нужен индекс *)
    RETURN pas[k]
  END; (* POST: sBadIndex = FALSE *)
END PascalTriangle2;
```

Реализация архитектуры

Теперь, когда у нас каркас с базовыми алгоритмами решения нашей задачи почти готов, можно дополнить модуль Pascaline полезными командами вычисления чисел Фибоначчи (листинг 7) и простых чисел (листинг 8). Поскольку особенность алгоритма нахождения простых чисел связана с тем, что он вычисляет сразу целый набор чисел в заданном диапазоне, то для доступа к результатам мы можем вопользоваться таким полезным приемом, как реализация механизма итератора с помощью классов. Основная его идея (впервые, пожалуй, реализованная в языке CLU) состоит в том, чтобы объединить список и процедуры доступа к нему в специальный объект.

В нашем случае класс итератора обладает тремя методами: Init, First и Next. Подобная реализация (листинг 9) удобна на уровне контроллера и внесена в модуль TriangleTF.

Листинг 7. Модуль Pascaline. Реализация чисел Фибоначчи через стек.

```
PROCEDURE Fibonacci3* (i: SHORTINT): LONGINT;
  VAR x: LONGINT; k: SHORTINT; stack: ARRAY 2 OF LONGINT;
BEGIN (* PRE: i >= 0; i <= MaxFibINDEX *)
  IF (i = 0) OR (i = 1) THEN RETURN 1
  ELSE (* i > 1 *) stack[0] := 1; stack[1] := 1; k := 2;
    REPEAT
      x := stack[0] + stack[1]; stack[0] := stack[1]; stack[1] := x;
      INC(k);
    UNTIL (k > i);
    RETURN x
  END; (* POST: sBadIndex = FALSE *)
END Fibonacci3;
```

Листинг 8. Модуль Pascaline. Реализация простых чисел через «решето Эратосфена».

```
PROCEDURE InitPrime*;
  VAR k,j,p,q: LONGINT; step: SHORTINT; init: BOOLEAN; set: LONGINT;
BEGIN (* PRE: r.int - верхняя граница диапазона чисел *)
  ASSERT (r.int > 3);
  primeDesc.this := 1; primeDesc.count := 0;
  primeDesc.max := (r.int DIV 2) - 1; (* только среди нечетных *)
  set := (primeDesc.max DIV MAX(SET)) + 1;   (* кол-во наборов *)
  NEW(isPrime,set);  (* формируем динамический массив множеств *)
  FOR k := 1 TO primeDesc.max DO
    (* выставляем бит; isPrime<0> не используется *)
    INCL(isPrime[k DIV MAX(SET)], k MOD MAX(SET));
  END;
  k := 1; j := 1; p := 3; q := 4; (* p = 2*j + 1; q = 2*j + 2*j^2 *)
  step := 1; (* шаг алгоритма *)
  LOOP (* цикл фиктивный: это переключатель шагов *)
    CASE step OF
      1: IF ~((j MOD MAX(SET)) IN isPrime[j DIV MAX(SET)]) THEN
           step := 3;
         ELSE k := q; step := 2;
         END;
    | 2: IF (k <= primeDesc.max) THEN
           EXCL(isPrime[k DIV MAX(SET)],k MOD MAX(SET));
           INC(k,p); (* еще раз *)
         ELSE step := 3;
         END;
    | 3: INC(j); INC(p,2); q := q + 2*p - 2;
         IF (j <= primeDesc.max) THEN step := 1;
         ELSE EXIT (* выходим из цикла LOOP *)
         END;
    ELSE (* других вариантов нет *)
    END;
  END;
  init := FALSE; sPrimeIsEmpty := TRUE;
  FOR k := 1 TO primeDesc.max DO
    IF ((k MOD MAX(SET)) IN isPrime[k DIV MAX(SET)]) THEN
      IF ~init THEN
        primeDesc.this := k; r.result := 2*k + 1;
        init := TRUE; sPrimeIsEmpty := FALSE;
      END;
      INC(primeDesc.count);
    END;
  END; (* POST: r.result содержит первое простое число *)
END InitPrime;
```

Листинг 9. Уровень контроллера (Controller). Модуль TriangleTF. Реализация итератора.

```
TYPE
  PrimeIterator = RECORD
    (* вырожденная запись, с ней связываем методы *)
  END;

  PROCEDURE (VAR t: PrimeIterator) Init (high: LONGINT);
  BEGIN
    Pascaline.r.int := high; Pascaline.InitPrime;
  END Init;

  PROCEDURE (VAR t: PrimeIterator) First (VAR elem: LONGINT): BOOLEAN;
  BEGIN
    Pascaline.FirstPrime; elem := 0;
    IF Pascaline.sPrimeIsEmpty THEN RETURN FALSE
    ELSE elem := Pascaline.r.result; RETURN TRUE
    END;
  END First;

  PROCEDURE (VAR t: PrimeIterator) Next (VAR elem: LONGINT): BOOLEAN;
  BEGIN
    Pascaline.NextPrime; elem := 0;
    IF Pascaline.sPrimeIsEmpty THEN RETURN FALSE
    ELSE elem := Pascaline.r.result; RETURN TRUE
    END;
  END Next;

  PROCEDURE P (high: LONGINT);
    VAR num: Number.Integer; iter: PrimeIterator;
        ok: BOOLEAN; s: ARRAY 256 OF CHAR;
  BEGIN
    iter.Init(high); num.picture := "@N_12";
    ok := iter.First(num.body);
    WHILE ok DO
      num.Format(s); Print(s); ok := iter.Next(num.body);
    END;
  END P;
```

Для проверки эффективности наших альтернативных реализаций создадим модуль TriangleTst, процедуры которого будут запускаться после обработки командной строки процедурой TriangleTF.ExecCommand. В модуле TriangleTst сформируем неоднородный список процедур (листинг 10) с альтернативными реализациями PascalTriangle и Fibonacci и, последовательно его обрабатывая, будем запускать процедуры с параметрами, полученными с помощью генератора случайных чисел. Попутно мы реализовали процедуры вычисления чисел Фибоначчи.

А теперь вернемся к нашей первой задаче об игре в камешки. Она носит название «фибоначчиев ним» и была предложена Р.Гаскелом и М.Виниганом (подробное доказательство см.[1], с.552). В одном ее названии уже таится ключ к разгадке. Да, это числа Фибоначчи.

Листинг 10. Модуль TriangleTst. Динамические типы, классы, неоднородные списки.

```
TYPE
  Proc = POINTER TO ProcDesc; (* класс процедур *)

  ProcDesc = RECORD
               id  : INTEGER; (* номер реализации *)
               next: Proc; (* ссылка на следующий *)
             END;

  Pas = POINTER TO PasDesc; (* класс процедур для треугольника Паскаля *)
  Fib = POINTER TO FibDesc; (* класс процедур для чисел Фибоначчи      *)

  PasDesc = RECORD (ProcDesc) proc: Pascaline.ProcType1
            END;

  FibDesc = RECORD (ProcDesc) proc: Pascaline.ProcType2
            END;

  List = RECORD     (* неоднородный список *)
           first: Proc; (* первый элемент  *)
           this : Proc; (* текущий элемент *)
         END;

PROCEDURE ExecThisProc* (VAR isPas: BOOLEAN;
                         VAR id: INTEGER; VAR p1,p2,f1: SHORTINT;
                         VAR result: LONGINT);
BEGIN
  IF (list.this # NIL) THEN
    IF (list.this IS Pas) (* проверяем динамический тип *) THEN
      isPas := TRUE; id := list.this^.id;
      result := list.this(Pas)^.proc(pas1,pas2) (* приводим к типу *)
    ELSIF (list.this IS Fib) (* проверяем динамический тип *) THEN
      isPas := FALSE; id := list.this^.id;
      result := list.this(Fib)^.proc(fib1)  (* приводим к типу *)
    END;
    p1 := pas1; p2 := pas2; f1 := fib1;
  END;
END ExecThisProc;
```

Упростим условия задачи. Будем считать, что число камешков в кучке не 1000, а всего 20. Как мы помним, любое натуральное число можно представить в виде суммы несоседних чисел Фибоначчи единственным способом: $20 = 13 + 5 + 2 = F_7 + F_5 + F_3$. Последнее слагаемое в таком разложении и показывает, сколько камешков для победы должен взять игрок, делающий первый ход. Если второй игрок, скажем, возьмет в ответ максимально возможное на этом ходу число камешков — 4, то первому играющему останется вновь разложить оставшиеся 14 камешков в сумму ряда Фибоначчи ($14 = F_7 + F_2 = 13 + 1$) и взять один камешек. Такая стратегия всегда будет приводить его к успеху. А вот если число камешков в кучке изначально совпадало бы с одним из чисел Фибоначчи, то победу одержал бы второй игрок (проверьте это сами). Теперь нам осталось выяснить, сколько же надо взять камешков первому игроку, когда общее их число составляет 1000. Разложим его в сумму ряда Фибоначчи: $1000 = F_{16} + F_7 = 987 + 13$. Итак, первому игроку для победы нужно взять 13 камешков. Попробуйте сами расширить модуль Pascaline, включив в него процедуру разложения любого натурального числа в ряд Фибоначчи.

Красота — критерий правильности

Подведем итоги. Построение каркаса отняло у нас значительное время, однако позволило создать такую архитектуру, которая может легко наращиваться и при этом оставаться достаточно простой и прозрачной для понимания и поиска возможных ошибок. Реализация альтернативных вариантов алгоритмов сняла многие вопросы за счет проверки на практике их эффективности. К тому же они служили дополнительным контролем для более сложных в реализации, но зато и более эффективных алгоритмов. Не менее важно и то, что мы на конкретном проекте познакомились с механизмами языков Оберон и Оберон-2, поддерживающих как концепцию ООП, так и концепцию модульного (компонентного) программирования.

Модули стали главными строительными блоками, а классы ввиду их гибкости и расширяемости больше всего подошли для уровня контроллера (специального преобразователя), где требуется соединять разные виды представления данных и обеспечивать исполнение команд модельного слоя.

Профессор Роберт Флойд, автор одного из первых компиляторов языка Алгол-60, раскрывая секреты своей творческой лаборатории, заметил: «После того как поставленная задача решена, я повторно решаю ее с самого начала, прослеживая и повторяя только суть предыдущего решения. Я проделываю эту процедуру до тех пор, пока решение не становится настолько четким и ясным, насколько это для меня возможно. Затем я ищу общее правило решения аналогичных задач, которое заставило бы меня подходить к решению поставленной задачи наиболее эффективным способом с первого раза».

Вы, наверное, уже обратили внимание, что ряд Фибоначчи и треугольник Паскаля могут реализовываться через рекурсивные процедуры. Иными словами, их элементы подобны общему. В этом смысле они вполне подпадают под определение фрактала, предложенное Бенуа Мандельбротом (1987), который называл этим именем структуру, состоящую из частей, которые в каком-то смысле подобны целому. Кстати, относительно недавно в контуре канонического фрактала Мандельброта обнаружили «лепестки», в точности подчиняющиеся числам Фибоначчи.

Формируя каркас проекта и решая поставленные задачи, мы познакомились с одним из возможных стилей программирования. Разумеется, выбор автора был субъективен, и на практике вам придется самостоятельно выбирать стиль. Главное, чтобы он помогал строить понятные и эстетичные программы. Красота не только в математике служит критерием правильности. Если реализация алгоритмов на языке программирования выглядит понятно, органично и не вызывает отторжения, значит, у нее есть все шансы быть правильной.

Литература

1. Кнут Д. Искусство программирования, т.1. М.: Вильямс, 2000.
2. Виленкин Н.Я. Популярная комбинаторика. М.: Наука, 1975.
3. Гарднер М. Математические новеллы. М.: Мир, 2000.
4. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. М.: Вильямс, 2000.
5. Moessenboeck H. Object-Oriented Programming in Oberon-2. Springer, 1993.
6. Гиндикин С.Г. Рассказы о физиках и математиках. М.: МЦНМО, 2001.

Оберон/Оберон-2 и Object Pascal. Десять важных отличий

1. **Имена идентификаторов.** Ключевые и зарезервированные идентификаторы записываются заглавными буквами (при просмотре текста это сразу бросается в глаза и как бы задает каркас программы). Идентификаторы чувствительны к регистру. Поэтому очень удобно переменную записывать со строчной буквы, а тип с заглавной, например, map: Map.

2. **Составные операторы.** Отсутствие лишних скобок begin-end. Для составных операторов требуется только завершающая скобка END: начало определяется самим оператором. Связка BEGIN-END используется лишь для задания границ процедур и модулей (в теле инициализации). При этом после END обязательно указывается имя процедуры/модуля. В операторе IF применяется ступенчатая схема IF-ELSIF-ELSE, сокращающая избыточную вложенность.

3. **Операторы безусловного перехода.** Отсутствие оператора goto и его меток. В случае необходимости оператор goto можно легко моделировать с помощью универсального цикла LOOP (с выходом из него по оператору EXIT). Кроме того, имеется оператор HALT, принудительно останавливающий выполнение программы.

4. **Типы данных.** Числовые типы задают иерархию, учитываемую при совместимости типов: LONGREAL > REAL > LONGINT > INTEGER > SHORTINT. Больший тип поглощает меньший. Для работы со строками используются массивы литер: ARRAY OF CHAR. Признаком завершения строки является литера с кодом 0X (до X указывается шестнадцатизначное значение кода). Все операции над строками, кроме стандартной процедуры COPY, вынесены в библиотечные модули.

5. **Конструкторы типов.** Вся индексация массивов ведется с 0: при его описании задается размер, а не диапазон индексов. В качестве типа для формальных параметров процедур и

базового типа для указателей могут использоваться открытые массивы. Отсутствие типа «перечисление». Отсутствие вариантных записей. Тип «множество» (SET) задает только набор целых чисел 0..MAX(SET), зависящий от платформы.

6. **Экспорт-импорт идентификаторов.** Области видимости идентификаторов определяются не блоками, а границами модулей и процедур. Вместо конструкций unit, interface, uses, отвечающих в Object Pascal за экспорт-импорт идентификаторов в Обероне/Обероне-2 используется единая конструкция модуля (MODULE с оператором IMPORT, регламентирующим список импортируемых модулей). Экспорт в Обероне-2 избирательный: если после идентификатора ставится признак «*», то экспорт полный (поддерживается чтение и запись), если признак «-», то экспорт частичный, или симплексный (только чтение). Интерфейс модуля (DEFINITION) генерируется автоматически по заданным признакам экспорта и обеспечивает раздельную компиляцию. Имеется механизм локальной синонимизации имен модулей.

7. **Обработка исключений.** Отсутствие операторов обработки исключений. Используются только контрольные вставки ASSERT с указанием булевого выражения. Если оно ложно, то выполнение программы в данной точке будет остановлено.

8. **Системное программирование.** Средства низкоуровневого программирования (работа с байтами и битами, регистрами, адресами, обощенными указателями) заключены в псевдомодуле SYSTEM (в исполняющей системе языка) и его импорт сигнализирует о возможном появлении проблем переносимости при последующем использовании импортирующего модуля на других платформах.

9. **Наследование.** В Обероне/Обероне-2 используется понятие расширения типа. Расширению подвергается только тип «запись» (как универсальная конструкция). Это означает, что новый тип может создаваться с расширением полей своего предшественника (механизм наследования): новый тип «проецируется» на предшествующий.

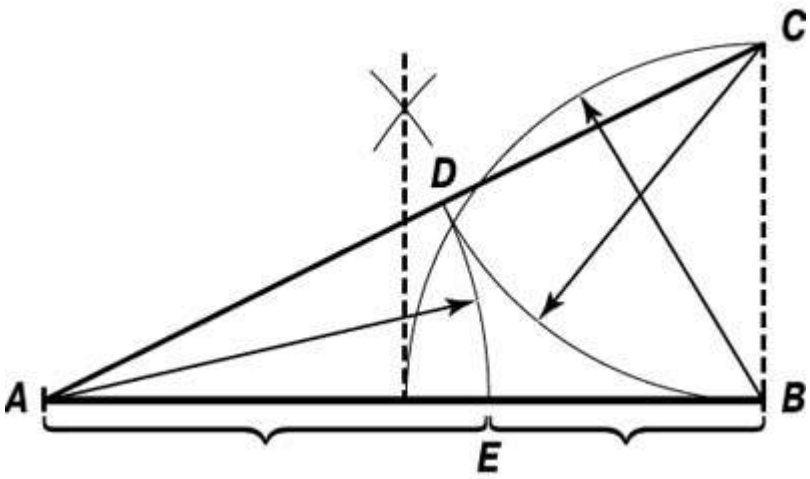
10. **Классы и объекты.** Для объектно-ориентированного программирования в дополнение к расширению типа добавляются связанные процедуры (type-bound procedures), играющие роль методов классов/объектов. С их помощью и благодаря операторам IS и WITH (в Обероне/Обероне-2 это аналог оператора CASE для разных классов) обеспечивается полиморфизм. Классы в Обероне/Обероне-2 присутствуют не в явном виде, а через расширение типов и специальные процедуры—методы. Динамическое порождение объектов (с помощью NEW, с размещением в куче и последующей сборкой мусора) и понятие изменяемого динамического типа (а не вариантного типа, как в Object Pascal) обеспечиваются указателями в комбинации с расширением типа. Инкапсуляция реализуется механизмом экспорта-импорта модулей.

Числа Фибоначчи и «золотое сечение»

0, 1, 1, 2, 3, 5, 8, 13, 21 ... — это знаменитая последовательность чисел Фибоначчи ($F_{n+2} = F_n + F_{n+1}$). Она хорошо известна в связи с так называемой задачей о кроликах, которую в 1202 г. в работе «Liber Abaci» («Книга абака») представил великий европейский математик средневековья, итальянский купец Леонардо Фибоначчи (Пизанский). В соответствии с ее условиями каждая пара кроликов ежемесячно дает одну пару приплода, при этом она становится способной к размножению спустя месяц после появления на свет. Числа Фибоначчи дают ответ на вопрос, сколько пар кроликов будет всего через 1, 2, 3 и т.д. месяцев.

Леонардо Фибоначчи (1170—1250), будучи купцом, неоднократно путешествовал по странам Востока и в своей книге использовал труды арабских математиков (аль-Хорезми, Абу Камила и др.). Числа Фибоначчи были известны еще индийским ученым, которые анализировали ритмику стихосложения. Имя итальянца было увековечено в названии последовательности этих чисел с легкой руки французского математика Эдуарда Люка, доказавшего благодаря их удивительным свойствам, что число $2^{127}-1$ является простым.

Если посмотреть на отношение соседних чисел Фибоначчи, то можно заметить, что оно стремится к числу 6=(1+$5^{1/2}$)/ 2 = 1,61803398874989484820... Наиболее замечательное свойство ряда Фибоначчи состоит в том, что отношение двух последовательных членов ряда попеременно то больше, то меньше «золотого сечения». Соответственно, 6 $^{n-2}$Fn <= 6 $^{n-1}$. Евклид называл число 6 отношением крайнего и среднего (пропорции при делении отрезка). И по сей день многие архитекторы, скульпторы, художники, писатели, поэты считают «золотое сечение» наиболее эстетичным.



«Золотое сечение» было известно архитекторам эпохи Возрождения, но они применяли его сравнительно редко. Лука Пачоли называл «золотое сечение» божественной пропорцией. Термин «золотое сечение» возник в Германии в первой половине XIX в. В XX в. известный архитектор Ле Корбюзье изобрел модулер — систему двух шкал, обеспечивающую соблюдение архитектурных пропорций и повторение однотипных форм. Деления голубой шкалы вдвое крупнее делений красной шкалы, которая основана на числах Фибоначчи (т. е. на «золотом сечении»).

Одно из интересных свойств чисел Фибоначчи было открыто в начале XVII в. Й.Кеплером: F_{n+1}x F_{n-1} – F^2_n = $(-1)^n$. Числа Фибоначчи могут вычисляться не только рекурсивно. В начале XVIII в. Бине вывел следующую формулу: Fn = (($1+5^{1/2}$) $^{n+1}$ – ($1-5^{1/2}$)^n+1) / (2^{n+1}x$5^{1/2}$). Как мы уже отмечали, числа Фибоначчи имеют прямую связь с треугольником Паскаля: Fn = Σ $^n_{k=0}$ C_k^{n-k}. Хорошо известна теорема Люка, в соответствии с которой некоторое целое число делит Fm и Fn тогда и только тогда, когда оно является делителем Fd, где d = gcd(m,n), т. е. d — наибольший общий делитель m и n. В частности, gcd(Fm,Fn) = $F_{gcd(m,n)}$.

Другие интересные свойства чисел Фибоначчи:

1. Если n не является простым числом, то и Fn не является простым.
2. $\Sigma^n_{k=0}$ F_k = F_{n+2} – 1.
3. F^2_n + F^2_{n+1} = F^2_{2n+1}.
4. $F2_n$ — $F2_{n-1}$ = F_{n-2} x F_{n+1}.
5. Последние цифры чисел Фибоначчи образуют периодическую последовательность с периодом 60. Две последние цифры образуют периодическую последовательность с периодом, равным 300. Для трех последних цифр период равен 1500, для четырех — 15 000, для пяти — 150 000.
6. Делители чисел Фибоначчи сами образуют ряд Фибоначчи (каждое третье число делится на 2, каждое четвертое на 3, каждое пятое на 5, каждое шестое на 8 и т.д.).
7. Каждое число Фибоначчи, которое является простым (кроме F_4 = 3), имеет простой индекс. Обратное утверждение неверно.

Интересен и тот факт, что с помощью суммы чисел Фибоначчи можно представлять любое натуральное число. Так что подобно двоичной системе счисления может использоваться и система счисления Фибоначчи. Она также записывается в виде последовательности 0 и 1 (стоящих на местах соответствующих чисел Фибоначчи). Таких представлений может быть несколько, но если мы примем правило, согласно которому рядом не могут находиться две единицы (их можно обнулить и перенести в старший разряд), то представление будет единственным.

Ряд Фибоначчи привлекал внимание математиков своей загадочной особенностью возникать в самых неожиданных местах. Числа Фибоначчи эффективно применяются при распределении памяти, при сортировке и обработке информации, генерировании случайных чисел, в методах оптимизации, позволяющих находить приближенные значения максимумов и минимумов сложных функций. #