Oberon Data Types

Matteo Corti
corti@inf.ethz.ch

December 5, 2001

1 Introduction

This document is aimed at students without any previous programming experience. We briefly describe some data types of the Oberon language and how they are internally represented.

This document is distributed in the hope that it will be useful, but without any warranty; Without even the implied warranty of merchantability or fitness for a particular purpose.

2 Definitions

Bit A bit (binary digit) is the smallest unit of information the computer uses. It can assume only two values 1 or 0.

Byte A byte is a group of 8 bits, strung together.

3 Numbering systems

The decimal numbering system represents numbers using ten different symbols (digits from 0 to 9) in a positional system: The meaning of a symbol is also determined its position. A digit d at the n^{th} position from right to left starting at 0, has a value of $d \cdot 10^n$.

Similarly other numbering systems with different sets of digits can be used.

The binary system uses only two digits 0 and 1, which are called bits. The value of a bit at the n^{th} position is $b \cdot 2^n$.

The hexadecimal system uses 16 symbols (digits from 0 to 9 and letters from A to F[1]). The value of an hexadecimal digit at the n^{th} position is $h \cdot 16^n$.

The following examples show some convertions between different number systems:

$$\begin{aligned} 2001 &= 2 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^2 + 1 \cdot 10^0 \\ &= 2000 + 0 + 0 + 1 \\ &= 2001 \end{aligned}$$

[1]Letters from A to F have a value of 10 to 15 in the decimal system, e.g. D has a value of 13.

$$
\begin{aligned}
01010110 &= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
&= 0 + 64 + 0 + 16 + 0 + 4 + 2 + 0 \\
&= 88
\end{aligned}
$$

$$
\begin{aligned}
3F4C &= 3 \cdot 16^3 + F \cdot 16^2 + 4 \cdot 16^1 + C \cdot 16^0 \\
&= 3 \cdot 16^3 + 15 \cdot 16^2 + 4 \cdot 16^1 + 12 \cdot 16^0 \\
&= 12288 + 3840 + 64 + 12 \\
&= 16204
\end{aligned}
$$

The hexadecimal numbers are widely used in computer science, because they can be easily converted from and to binary once: Each hexadecimal digit corresponds exactly to four bits. An example is depicted in the following table:

binary	0011	1111	0100	1100
decimal	3	15	4	12
hexadecimal	3	F	4	C

3.1 Negative numbers

With n bits you can represent 2^n different numbers, for example integers between 0 and $2^n - 1$. The problem arises when we want to express negative numbers, as we have to choose an appropriate representation.

A trivial method uses the first bit to express the minus sign, but we have the problem that two different representations for 0 are possible (with 8 bits 00000000 will be 0 and 10000000 will be -0 which is nonsensical).

Negative numbers are usually represented by the so called 2's complement notation. To obtain the 2's complement of a number, first take the complement (invert each bit) and then add 1. All the negative numbers will have a 1 in the most significant bit[2] position (MSB), and the numbers will now range from -2^{n-1} to $2^{n-1} - 1$.

For example, if we want to express -42 in a 8 bit system we first convert 42 to binary obtaining 00101010 ($0 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$). We then compute the complement 11010101 and we finally add 1 obtaining 11010110.

3.2 Floating-point

Floating-points numbers approximate real ($\mathbb{R}$) numbers. A floating-point number r is represented by a signed mantissa m and a signed exponent n with respect to a base b:

$$r = \pm m \cdot b^{\pm n}$$

The IEEE standard for single-precision floating point format allocates 1 bit for the sign of the number, 8 bits for the signed exponent and 23 bits for the mantissa.

[2]The MSB or most significant bit is the bit with the most significat position, in our case the leftmost bit

3.3 Numbers in Oberon

Integer numbers in the decimal notation are expressed as usual with a sequence of digits that can be preceded by a minus sign:

```
integer := digit {digit}.
```

Hexadecimal numbers are expressed with a capital H at the end:

```
integer := digit {hexDigit} "H".
```

Note that the first digit of an hexadecimal number cannot be a letter, numbers beginning with a letter must be preceded by a 0 (e.g. 0AFH).

4 ASCII

Numbers are also used to represent characters. The standard way to do this is the American Standard Code for Information Interchange or ASCII (pronounced "Ask-ee"). ASCII is a 7-bit code, thus allowing to code 128 (2^7) different symbols.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH	101	65	41	A
002	2	02	STX	102	66	42	B
003	3	03	ETX	103	67	43	C
004	4	04	EOT	104	68	44	D
005	5	05	ENQ	105	69	45	E
006	6	06	ACK	106	70	46	F
007	7	07	BEL '\a'	107	71	47	G
010	8	08	BS '\b'	110	72	48	H
011	9	09	HT '\t'	111	73	49	I
012	10	0A	LF '\n'	112	74	4A	J
013	11	0B	VT '\v'	113	75	4B	K
014	12	0C	FF '\f'	114	76	4C	L
015	13	0D	CR '\r'	115	77	4D	M
016	14	0E	SO	116	78	4E	N
017	15	0F	SI	117	79	4F	O
020	16	10	DLE	120	80	50	P
021	17	11	DC1	121	81	51	Q
022	18	12	DC2	122	82	52	R
023	19	13	DC3	123	83	53	S
024	20	14	DC4	124	84	54	T
025	21	15	NAK	125	85	55	U
026	22	16	SYN	126	86	56	V
027	23	17	ETB	127	87	57	W
030	24	18	CAN	130	88	58	X
031	25	19	EM	131	89	59	Y
032	26	1A	SUB	132	90	5A	Z
033	27	1B	ESC	133	91	5B	[
034	28	1C	FS	134	92	5C	\ '\\'

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
035	29	1D	GS	135	93	5D	]
036	30	1E	RS	136	94	5E	ˆ
037	31	1F	US	137	95	5F	_
040	32	20	SPACE	140	96	60	‘
041	33	21	!	141	97	61	a
042	34	22	”	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	’	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	—
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	˜
077	63	3F	?	177	127	7F	DEL

The extended ASCII code (8 bits) can represent an additional set 128 characters which are not part of the standard and are platform and and configuration specific (this means that german letters as ä, ü and ö are not part of the standard ASCII code).

5 Oberon data types

Oberon offers eight different basic data types: BOOLEAN, CHAR, SHORTINT, INTEGER, LONGINT, REAL, LONGREAL and SET.

5.1 BOOLEAN

BOOLEAN can express only two different values: TRUE or FALSE. Although a boolean can be expressed with one bit only, for practical reasons one or more bytes are commonly used.

5.2 CHAR

Character constants are expressed with the corresponding symbol or with the index in the ASCII table in hexadecimal form followed by a capital X (e.g. “o” or 06FX).

```
CharConstant  = """ character """ | digit {hexDigit} "X".
```

CHAR can express the extended ASCII set (from 000X to 0FFX).

5.3 SHORTINT, INTEGER and LONGINT

Integer types represent signed integer values between MIN(type) and MAX(type). On 32 bits CPUs this normally means:

Type	Size	MIN(type)	MAX(type)
SHORTINT	8 bit	-128	127
INTEGER	16 bit	-32678	32677
LONGINT	32 bit	2147483648	-2147483647

5.4 REAL and LONGREAL

REAL and LONGREAL express real numbers between MIN(type) and MAX(type).

On 32 bits CPUs the IEEE single and double precision floating point format is usually used (32 and 64 bits).

5.5 SET

A SET represents the set of integers between 0 and MAX(SET). On 32 bits CPUs MAX(SET) is normally 31.

5.6 Order

Numeric types form a hierarchy; the larger type can represent (the values of) the smaller type:

$$\text{LONGREAL} \subseteq \text{REAL} \subseteq \text{LONGINT} \subseteq \text{INTEGER} \subseteq \text{SHORTINT}$$

5.7 Conversions

A value of a smaller type is converted in a larger type implicitly, as shown in the following example:

```
PROCEDURE Foo();
VAR
  s: SHORTINT;
```

```
  i: INTEGER;
  l: LONGINT;
BEGIN
  i := s;
  l := s;
  l := i
END Foo;
```

Other explicit conversions are possible as shown in the following table

Name	Argument type	Result type	Function
ORD(x)	CHAR	INTEGER	ordinal number of x
CHR(x)	integer type	CHAR	character with ordinal number x
SHORT(x)	LONGINT INTEGER LONGREAL	INTEGER SHORTINT REAL	identity (truncation possible)
LONG(x)	INTEGER SHORTINT REAL	LONGINT INTEGER LONGREAL	identity
ENTIER(x)	real type	LONGINT	largest integer not greater than x Note that ENTIER(i/j) = i DIV j

Particular attention must be paid to explicit convertions since the smaller type is not always able to hold the converted value as in the following example:

```
PROCEDURE Foo();
VAR
  s: SHORT;
  i: INTEGER;
BEGIN
  i := 300;      (* i = 000000100101100 or 300 *)
  s := SHORT(i)  (* s =         0101100 or 44  *)
END Foo;
```