Chapter 3. Fundamental Data Types

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National Univertisy

http://bi.snu.ac.kr

Variable Declaration

- Variables are the objects that a program manipulates.
- All variables must be declared before they can be used.

```
#include <stdio.h>

int main(void)
{
    int a, b, c;   /*declaration*/
    float x, y = 3.3, z = -7.7;  /*declaration with initialization*/
    printf("Input two integers: ");   /*function call*/
    scanf("%d%d",&b, &c);   /*function call*/
    a = b + c;   /*assignment*/
    x = y + z;   /*assignment*/
}
```

Variable Declaration

- Declarations
  - associate a **type** with each **variable** declared
  - This tells the cpu to set aside an appropriate amount of memory space to hold values associated with variables.
    - each type needs specified amount of space
  - The type and the memory location of the variable never changes.

Arithmetic Type

- Integer Type
  - 125, -100, 10245
- Floating-Point Type
  - 10.543, 1e+5, 1.0
- Representation
  - Integer types and floating-point types have different representation in the memory.
    - integer 1 (00000...0001) floating-point 1.0 (0 011...1 00....0)
  - They also use different circuits for the operations.
- Types have size in memory and range of values
  - The size is dependent to the compiler.

Integer Type

- **int** is a default type of integer constants (normally 4 bytes)
- Signed integer type values are stored by two's complement representation

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Integer Type

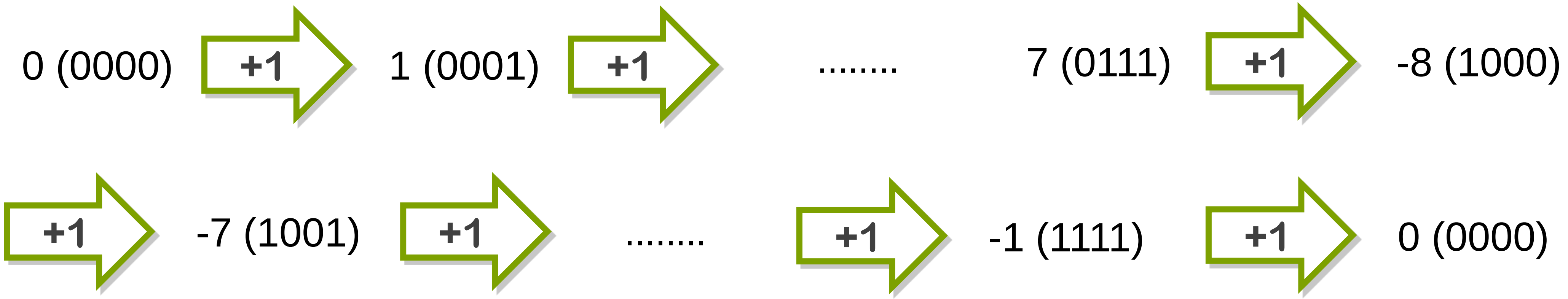
- Two’s Complement
  - It has great advantage in the computer system.
  - example of 4-bit value

Two's complement	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

Two's complement	Decimal
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Integer Type

- Two's Complement
  - example of 4-bit value



Floating-point Type

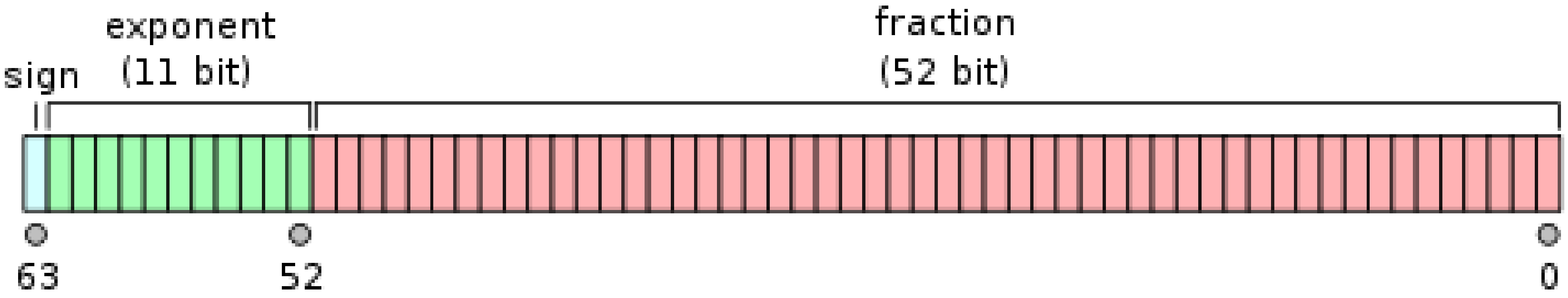
- **double** is a default type of floating-point type constants
- Scientific notation
  - 1.5E+14 means 1.5×10^{14}
- Representation
  - IEEE 754 standard
- Bigger size, more precise

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Floating-point Type

- IEEE 754 standard
- Floating-point values are stored as combination of sign, exponent, fraction(or mantisa, significand, 기수)

$$(sign: -or+)\,(fraction)\ \times 2^{(exponent)}$$



Floating-point Type

- Bigger size of fraction can store more precise values
  - 3.14
  - 3.1415926535
  - 3.141592653589793238462643383279502884
- Precision Loss (or Loss of significance)
  - It is impossible to save precisely exact value in memory due to finite size of the storage size.
  - Decimal values can suffer precision loss when they are converted to the binary values.

```
float a = 1.1;
printf("%.32f\n", a);  // 1.10000002384185791015625000000000
```

Characters and the Data Type char

- type **char**
  - A variable of type **char** can be used to hold small integer values.
- 1 byte (8 bits) in memory space
  - 28, or 256, distinct values
- Characters are treated as small integer with type **char**
  - ‘a’ (97)   ‘b’ (98)   ‘c’ (99) …
  - encoded by ASCII code
    - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

Characters and the Data Type char

- ASCII code table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Characters and the Data Type char

- Example

```
char c = 'a';
printf("%c", c); /* a is printed */
printf("%d", c); /* 97 is printed */
printf("%c%c%c", c, c+1, c+2); /* abc is printed */
```

Characters and the Data Type char

- **char c = ‘a’;**
  - **c** is stored in memory in 1 byte as **01100001** **(97)**
- The type **char** holds 256 distinct values
  - **signed char** : -128 ~ 127
  - **unsigned char** : 0 ~ 255

Characters and the Data Type char

- Nonprinting and hard-to-print characters require an escape sequence.
- \ (backslash character)
  - an escape character
  - It is used to escape the usual meaning of the character that follows it.

Special Characters		
Name of character	Written in C	Integer value
alert	\a	7
backslash	\\	92
double quote	\"	34
newline	\n	10
null character	\0	0
single quote	\'	39

typedef

- **typedef**
  - allows the programmer to explicitly associate a type with an identifier

```
typedef char                    uppercase;
typedef int                 INCHES, FEET;
typedef unsigned long            size_t;

int main(void)
{
        uppercase u;
        INCHES length, width;
        …
}
```

(1) abbreviating long declarations
(2) having type names that reflect the intended use

sizeof

- **sizeof**
  - a unary operator to find the number of bytes needed to store an object
- **sizeof(object)**
  - **object** can be a type such as **int** or **float**, or an expression such as **a+b.**

sizeof

- Example

```
/* Compute the size of some fundamental types. */
#include <stdio.h>

int main(void)
{
    printf(“The size of some fundamental types is computed.\n\n”);
    printf(“ char:%3u byte \n”, sizeof(char));
    printf(“ short:%3u bytes\n”, sizeof(short));
    printf(“ int:%3u bytes\n”, sizeof(int));
    printf(“ float:%3u bytes\n”, sizeof(float));
    printf(“ double:%3u bytes\n”, sizeof(double));
}
```

Type Conversion

- For binary operations with different types of operands, the "**lower**" type is **promoted** to the "**higher**" type before operation proceeds.
  - **100**(int) + **5.12**(double) → **100**(double) + **5.12**(double) → **105.12**(double)

- For assignment operations, the value of the **right side** is converted to the type of the **left**, which is the type of the result.

```
int a = 10.5; // 10.5 is initially double type
printf("%d", a);   // 10
```

Informal Conversion Rule

- If there is no unsigned operands
  - If either operand is long double, convert the other to long double
  - Otherwise, if either operand is double, convert the other to double
  - Otherwise, if either operand is float, convert the other to float
  - Otherwise, convert char and short to int
  - Then, if either operand is long, convert the other to long
- Unsigned types are not recommended.
  - They confuse programmers with complicated conversion rules.

Integral Promotions

- A char, a short integer, or an integer bit-field, all either signed or not, or an object of enumeration type, may be used in an expression whenever an integer may be used.
- If all the values of the original type in an expression can be represented by an **int**, then the value is converted to an **int**; otherwise the value is converted to **unsigned int**.

```
short x, y;
x + y (the type int, not short)
```

Informal Conversion Rule

- **floating-point type** to **integer type** causes truncation of any fractional part

```
float f = 10.752;
int i = f;
printf("%d", i );  // 10
```

- Longer integers are converted to shorted ones or chars by **dropping** the excess **high-order** bits

```
int a = 1023;          // 00000000 00000000 00000011 11111111
char c = a;            // 11111111
printf("%d\n", c);     // -1 (11111111)
```

Explicit Type Cast

- Explicit conversions
  - syntax: **(type) variable or constant**
  - Type cast is a unary operation that converts the type of value of variables or constants.
  - Type of the variable doesn’t change.

```
double a = 10 / 3;    // integer division
double b = 10 / (double) 3;  // floating number division
printf(“%f”, a );  // 3.000000
printf(“%f”, b );  // 3.333333
```

Types and Operations

- Modulus operation(%) do not allow floating-point type values.
  - `10.5 % 2.16 ;   // error !!`

- Integer division calculates the quotient
  - `int a = 7/2 ;           // 3`
  - `float f = 7.0/2.0 ;   // 3.5`