Chapter 3. Fundamental Data Types

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University

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.

```c
#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

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage size</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1 byte</td>
<td>-128 to 127 or 0 to 255</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
<td>0 to 255</td>
</tr>
<tr>
<td>signed char</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>int</td>
<td>2 or 4 bytes</td>
<td>-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2 or 4 bytes</td>
<td>0 to 65,535 or 0 to 4,294,967,295</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 bytes</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>
Integer Type

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

<table>
<thead>
<tr>
<th>Two’s complement</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two’s complement</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>-1</td>
</tr>
<tr>
<td>1110</td>
<td>-2</td>
</tr>
<tr>
<td>1101</td>
<td>-3</td>
</tr>
<tr>
<td>1100</td>
<td>-4</td>
</tr>
<tr>
<td>1011</td>
<td>-5</td>
</tr>
<tr>
<td>1010</td>
<td>-6</td>
</tr>
<tr>
<td>1001</td>
<td>-7</td>
</tr>
<tr>
<td>1000</td>
<td>-8</td>
</tr>
</tbody>
</table>
Integer Type

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

0 (0000) → +1 → 1 (0001) → +1 → ... → 7 (0111) → +1 → -8 (1000)

+1 → -7 (1001) → +1 → ... → +1 → -1 (1111) → +1 → 0 (0000)
Floating-point Type

- **double** is a default type of floating-point type constants
- **Scientific notation**
  - 1.5E+14 means $1.5 \times 10^{14}$
- **Representation**
  - [IEEE 754](https://en.wikipedia.org/wiki/IEEE_754) standard
- **Bigger size, more precise**

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage size</th>
<th>Value range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4 byte</td>
<td>1.2E-38 to 3.4E+38</td>
<td>6 decimal places</td>
</tr>
<tr>
<td>double</td>
<td>8 byte</td>
<td>2.3E-308 to 1.7E+308</td>
<td>15 decimal places</td>
</tr>
<tr>
<td>long double</td>
<td>10 byte</td>
<td>3.4E-4932 to 1.1E+4932</td>
<td>19 decimal places</td>
</tr>
</tbody>
</table>
Floating-point Type

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

\[(\text{sign: } - \text{ or } +) (\text{fraction}) \times 2^{(\text{exponent})}\]
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.

```c
float a = 1.1;
printf("%.32f\n", a); // 1.1000000238418579101562500000000000
```
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
Characters and the Data Type char

- ASCII code table

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
</table>
| 0   | 00  | Null | 32  | 20  | Space| 61  | 31  | '!'
| 1   | 01  | Start of heading | 33  | 21  | '1'  | 62  | 32  | '
| 2   | 02  | Start of text | 34  | 22  | ''  | 63  | 33  | '?'
| 3   | 03  | End of text | 35  | 23  | 8  | 64  | 34  | 'B'
| 4   | 04  | End of transit | 36  | 24  | 9  | 65  | 35  | 'C'
| 5   | 05  | Enquiry | 37  | 25  | @  | 66  | 36  | 'D'
| 6   | 06  | Acknowledge | 38  | 26  | S  | 67  | 37  | 'E'
| 7   | 07  | Audible alert | 39  | 27  | '  | 68  | 38  | 'F'
| 8   | 08  | Backspace | 40  | 28  | (  | 69  | 39  | 'G'
| 9   | 09  | Horizontal tab | 41  | 29  | )  | 70  | 40  | 'H'
| 10  | 0A  | Line feed | 42  | 2A  | *  | 71  | 41  | 'I'
| 11  | 0B  | Vertical tab | 43  | 2B  | +  | 72  | 42  | 'J'
| 12  | 0C  | Form feed | 44  | 2C  | ,  | 73  | 43  | 'K'
| 13  | 0D  | Carriage return | 45  | 2D  | -  | 74  | 44  | 'L'
| 14  | 0E  | Shift out | 46  | 2E  | .  | 75  | 45  | 'M'
| 15  | 0F  | Shift in | 47  | 2F  | /  | 76  | 46  | 'N'
| 16  | 10  | Data link escape | 48  | 30  | 0  | 77  | 47  | 'O'
| 17  | 11  | Device control 1 | 49  | 31  | 1  | 78  | 48  | 'P'
| 18  | 12  | Device control 2 | 50  | 32  | 2  | 79  | 49  | 'Q'
| 19  | 13  | Device control 3 | 51  | 33  | 3  | 80  | 4A  | 'R'
| 20  | 14  | Device control 4 | 52  | 34  | 4  | 81  | 4B  | 'S'
| 21  | 15  | Negative acknowledge | 53  | 35  | 5  | 82  | 4C  | 'T'
| 22  | 16  | Start of text | 54  | 36  | 6  | 83  | 4D  | 'U'
| 23  | 17  | End of text | 55  | 37  | 7  | 84  | 4E  | 'V'
| 24  | 18  | Cancel | 56  | 38  | 8  | 85  | 4F  | 'W'
| 25  | 19  | End of medium | 57  | 39  | 9  | 86  | 50  | 'X'
| 26  | 1A  | Substitution | 58  | 3A  | :  | 87  | 51  | 'Y'
| 27  | 1B  | Escape | 59  | 3B  | ;  | 88  | 52  | 'Z'
| 28  | 1C  | File separator | 5A  | 3C  | <  | 89  | 53  | 'a'
| 29  | 1D  | Group separator | 5B  | 3D  | =  | 90  | 54  | 'b'
| 30  | 1E  | Record separator | 5C  | 3E  | >  | 91  | 55  | 'c'
| 31  | 1F  | Unit separator | 5D  | 3F  | ?  | 92  | 56  | 'd'

ASCII code table
Characters and the Data Type char

Example

```c
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

- ```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.

<table>
<thead>
<tr>
<th>Special Characters</th>
<th>Written in C</th>
<th>Integer value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert</td>
<td>\a</td>
<td>7</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
<td>92</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
<td>34</td>
</tr>
<tr>
<td>newline</td>
<td>\n</td>
<td>10</td>
</tr>
<tr>
<td>null character</td>
<td>\0</td>
<td>0</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
<td>39</td>
</tr>
</tbody>
</table>
typedef

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

```c
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`. 
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.
  - \(100(\text{int}) + 5.12(\text{double}) \rightarrow 100(\text{double}) + 5.12(\text{double}) \rightarrow 105.12(\text{double})\)

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

```c
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`.

```c
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: \((\text{type}) \text{ variable or constant}\)
  - Type cast is a unary operation that converts the type of value of variables or constants.
  - Type of the variable doesn’t change.

```c
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