

MH1402 Algorithms & Computing II

Lecture 1 Data Types

Wu Hongjun

Overview

- **Identifier, Keyword**
- **Variables and constants**
- **Data Types**

Identifier and Keyword

- Identifiers are names for variables, functions, and other objects
- An identifier is a sequence of letters (a to z, A to Z), numbers (0 to 9), and underscore “_”, except that an identifier should not start with a number
- Examples:

Correct: x x32p8 n3_6K

Wrong: x!31 2r xy# ja.ck

Not recommended: _s

it is **not recommended** to use identifier starting with _

Reason: possible conflict with reserved identifier

Identifier and Keyword

- C++ identifiers are *case sensitive*
 - for example,
 `int x23y;` is different from `int x23Y;` `int X23y;`
 `cout << x;` is correct;
 `Cout << x;` is incorrect
- Try to use meaningful identifier
 - so that writing and reading programs become easier

Identifier and Keyword

- Keywords are predefined *reserved identifiers* that have special meanings. They *cannot* be used as the identifiers of your variables and functions in your program
 - In Code::Blocks, the keywords are in blue color
 - Some keywords in C++:
for if else switch while do break continue false true
bool char short long int float double signed unsigned void sizeof
return using namespace private public new
.....

Identifier and Keyword

```
#include <iostream>
```

```
using namespace std;
```

```
int main()  
{  
    int y = 3;  
    cout << y << endl;  
    return 0;  
}
```

Keywords: using namespace int return

Identifiers: std main y cout endl

How about *#include* ? You can simply consider *#include* as a special command. [[details: In C++, the lines preceded by a hash sign (#) are not program statements, but directives for the preprocessor. The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements.]]

Data Types

Data on Computer

- **Some elementary memory concepts**
 - **Bit**
 - Bit is the basic unit of memory/information in computing and digital communications.
A bit can have only one of two possible values (0 and 1).
 - **Byte**
 - Most of the time, byte is the smallest unit of data storage on computer
You need one byte or multiple bytes to store data
 - A byte consists of eight bits: b7 b6 b5 b4 b3 b2 b1 b0
 - Two possible values for each bit, so there are **$2^8=256$ possible values for each byte**

Data on Computer

- **All the information on computer are stored as a sequence of bits/bytes (0's and 1's)**
 - Including numerical data, text, picture, video, software,
- **So base-2 (binary) number system is used in computer**
 - But we are familiar with the base-10 number system
 - such as: 17 dollars, 18 years old
 - Based-10 number system is used in computer mainly for human-computer interface, i.e., we input base-10 number to the computer, and the computer prints the base-10 numbers to the screen
 - When we input a base-10 number into the computer, it is stored as base-2 format inside the computer

Data on Computer

- **Base-2 (binary) number system**

- **Base-10 number system example:**

The value of 326 is: $3 \times 10^2 + 2 \times 10^1 + 6 \times 10^0$

- **Base-2 number example:**

For a 4-bit binary number $b_3 b_2 b_1 b_0$, its decimal value is:

$$b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

For binary number 1101, its value is: $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

For decimal number 19, it is stored as binary number 10011 in computer

- **A non-integer binary number, $b_2 b_1 b_0 . a_1 a_0$, its value is**

$$b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0 + a_1 \times 2^{-1} + a_0 \times 2^{-2}$$

For binary number 11.01, its value is: $1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

Data Type in C++

- **Every data in C++ must have a type**
 - Every data must be stored in computer memory when the program gets executed.
 - Data of different types take different amounts of memory to store
- **In MATLAB, a default data type is used: double, so normally you do not need to specify a data type in MATLAB**
 - But there is no such default data type in C++

Data Type in C++

- There are two main types of data we need to deal with:
 - Numerical data: real numbers
 - In C++, the data types are:
 - int** for integers (when the absolute value of integer is not too large),
float/**double** for general rational numbers
 - Character: for example, the letters A - Z, a-z, 0-9, _, *, %, ^, @, #,
....
 - In C++, the data type is **char**
 - The widely used character encoding scheme is ASCII: it encodes a character into an integer in the range of 0 to 127.
 - So **each character is stored as a small integer** in computer

Data Type: char

- char is the type of text character (stored as small integer on computer) with memory size of exactly one byte (8 bits)

b7 b6 b5 b4 b3 b2 b1 b0

- The range of *char* type is -128 to 127
- The range of *unsigned char* type is 0 to 255
- The difference between char and unsigned char:
 - If one byte of memory is declared as *char*, then the left-most bit is considered as sign bit. If the sign bit is 1, it is negative; otherwise, positive.
 - If one byte of memory is declared as *unsigned char*, then there is no sign bit

Data Type: char

b7	b6	b5	b4	b3	b2	b1	b0	value (char)	value (unsigned char)
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	0	2	2
0	0	0	0	0	0	1	1	3	3
0	0	0	0	0	1	0	0	4	4
0	0	0	0	0	1	0	1	5	5
.....									
0	1	1	1	1	1	1	0	126	126
0	1	1	1	1	1	1	1	127	127
1	0	0	0	0	0	0	0	-128	128
1	0	0	0	0	0	0	1	-127	129
1	0	0	0	0	0	1	0	-126	130
1	0	0	0	0	0	1	1	-125	131
1	0	0	0	0	1	0	0	-124	132
1	0	0	0	0	1	0	1	-123	133
.....									
1	1	1	1	1	1	0	1	-3	253
1	1	1	1	1	1	1	0	-2	254
1	1	1	1	1	1	1	1	-1	255

In memory

Data Type: char

ASCII printable characters and their integer values: (some characters are not printable)
(all the 128 ASCII characters are given at: <http://en.wikipedia.org/wiki/ASCII>)

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	\$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(56	8	72	H	88	X	104	h	120	x
41)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	
45	-	61	=	77	M	93]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

Data Type: char

- How to assign a value to a character:

```
char x1 = 'A'; // We use single quotes (apostrophe ' ) to indicate a character
               // an integer with value 65 ('A' is encoded as integer 65) stored into x1
char x2 = 65;  // Another method. The values of x1 and x2 are the same
```

```
cout << "x1 is " << x1 << " its integer value is " << int(x1) << endl ;
cout << "x2 is " << x2 << " its integer value is " << int(x2) << endl ;
```

```
x1 is A its integer value is 65
x2 is A its integer value is 65
```

() is a type cast operator here.
int(x1) is used to reveal the integer value of x1

Data Type: char

```
char y1 = '9';
```

```
char y2 = 9;    // the values of y1 and y2 are different  
               // character with integer value 9 is not printable
```

```
cout << "y1 is " << y1 << " its integer value is " << int(y1) << endl ;
```

```
cout << "y2 is " << y2 << " its integer value is " << int(y2) << endl ;
```

```
y1 is 9 its integer value is 57  
y2 is   its integer value is 9
```

Data Type: char

```
char z1 = 35;
```

```
char z2 = '35'; // we will get the warning of overflow
```

```
// two characters can not be stored into one byte , only '5' is stored
```

```
cout << "z1 is " << z1 << " its integer value is " << int(z1) << endl ;
```

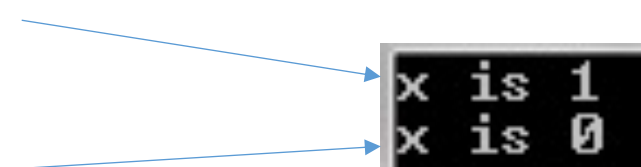
```
cout << "z2 is " << z2 << " its integer value is " << int(z2) << endl ;
```

```
z1 is # its integer value is 35
z2 is 5 its integer value is 53
```

Data Type: bool

- bool is the Boolean type with **only two possible values**, and the memory size is **one byte**
 - The value of a bool variable is true or false
 - The Boolean false is stored as an integer with value 0; the Boolean true is stored as an integer with value 1.
- Example:

```
bool x;  
x = true;  
cout << "x is " << x << endl;  
x = false;  
cout << "x is " << x << endl;
```



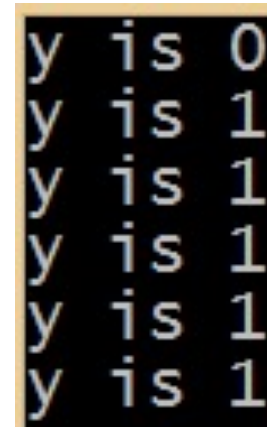
```
x is 1  
x is 0
```

Data Type: bool

Note that when a non-zero value is assigned to a bool variable, it is the same as assigning 1 to that variable, and the value of that bool variable is true.

- Another example:

```
bool y;  
y = 0;    cout << "y is " << y << endl;  
y = 1;    cout << "y is " << y << endl;  
y = 3;    cout << "y is " << y << endl;  
y = -7;   cout << "y is " << y << endl;  
y = 0.1;  cout << "y is " << y << endl;  
y = -3.5; cout << "y is " << y << endl;
```



```
y is 0  
y is 1  
y is 1  
y is 1  
y is 1  
y is 1
```

- Note that bool is very different from bit
 - Bit is a **memory/information unit**, and it takes two possible values: 0 or 1.
 - bool is a **data type**, it occupies one byte of memory (8 bits), and it takes two possible values: true (1) or false (0).

Data Type: int

- If the numerical data is an integer, and its absolute value is not too large, we may use the data type *int* (normally 4 bytes) to represent it *precisely*
- In the current version of Code::Blocks (MinGW-GCC compiler), the memory size of *int* type is 4 bytes (32 bits)
 - The range of *int* is -2^{31} to $2^{31}-1$
 - The range of *unsigned int* is 0 to $2^{32}-1$
- The memory size of *long long int* type is 8 bytes (64 bits)
 - The range of *long long int* is -2^{63} to $2^{63}-1$
 - The range of *unsigned long long int* is 0 to $2^{64}-1$

Data Type: int

- There are four integer data types:

short (or short int)	unsigned short
int	unsigned int
long (or long int)	unsigned long
long long (or long long int)	unsigned long long

- *The size/range for each data type is not fully standardized*
- In our Code::Blocks (MinGW-GCC), the sizes are given below

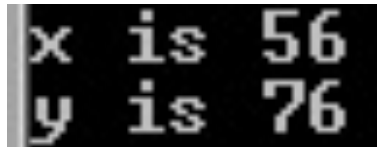
short	(2 bytes)
int	(4 bytes)
long	(4 bytes)
long long	(8 bytes)

Warning: The size of **long int** may be 4 or 8 bytes on different computers/compilers (avoid using **long int** in your program. Use either int or long long)

Data Type: int

- **Example:**

```
int x, y;  
x = 56;  
cout << "x is " << x << endl;  
y = 76.89; // only the integer part gets stored into y  
cout << "y is " << y << endl;
```



```
x is 56  
y is 76
```

Data Type: floating-point

- Floating-point is commonly used for real number on computer
 - if the significant figure of a real number is too large (the absolute value is too large, or the fractional part is too long), approximation is needed
- Two floating-point types:
 - float (single-precision floating-point) occupies 4 bytes (32 bits)
 - double (double-precision floating-point) occupies 8 bytes (**64** bits)
 - About **15 significant precision digits**
- Float is less precise than double, but with similar efficiency as double on today's computers
 - It is better to use double instead of float
- In MATLAB, the data type is double by default

Data Type: Double

This slide will not be tested.

- Details of double

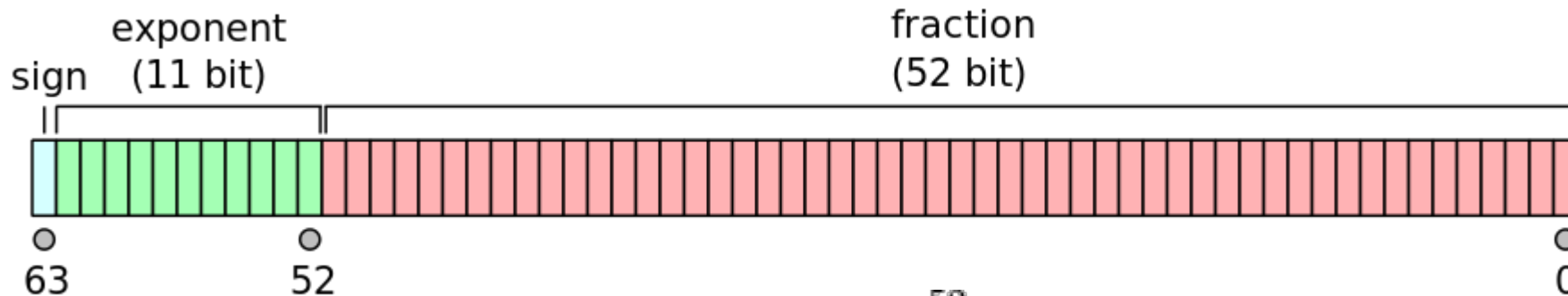
- http://en.wikipedia.org/wiki/Double-precision_floating-point_format

- **One bit** (the left-most bit) is the sign bit
 - **11** exponent bit
 - 53 significant precision bits (the left most bit is always 1, so not stored. Only **52** significant precision bits are stored)
 - About **15 significant precision digits**

This slide will not be tested.

Data Type: double (cont.)

- double in memory (64 bits: b63 b62 b61 b2 b1 b0)



- The value of double:
$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{e-1023}$$
- The range of double: $\pm [2.2\text{E-}308, 1.79\text{E}308]$ (scientific format)
- Zero is not directly representable in the straight format, due to the assumption of a leading 1.
 - For a floating-point with an exponent field of zero and a fraction field of zero, the computer treats its value as 0.

Floating-point format in C++

- For input or output, floating-point (float or double) values may be specified in either of two formats:

- **Fixed-point format:** 34.567 -4.2
- **Scientific format:** more efficient for very small or very large numbers.

-5.71e103 // means -5.71×10^{103}

4.67e-15 // means -4.67×10^{-15}

6.7e78

Either e or E can be used in the scientific format.

- Input examples:


double z = 3456.788678;

double x = -4.8E45;

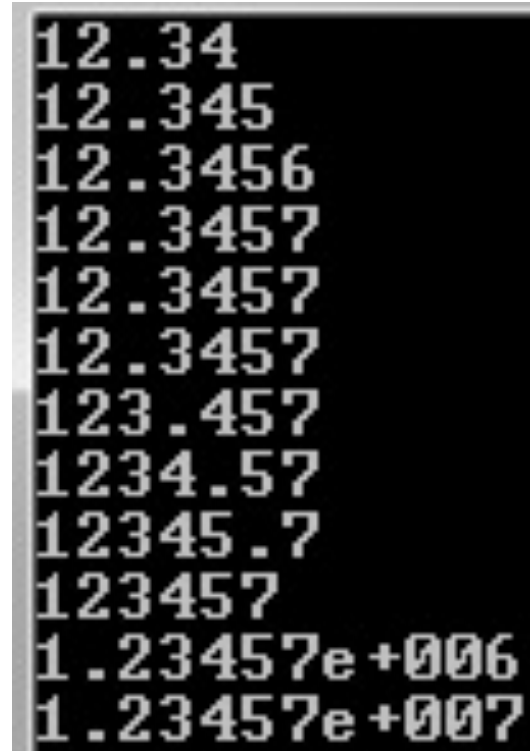

double y = 5.4e-65;

- Output using cout

- floating-point values with magnitude in the range 0.1 to 999,999 will normally be printed in the format with **6-digit precision**;
- all others will be printed in scientific format with **6-digit precision**
- Note that *the value being printed to the screen is not necessarily the same as (normally less precise than) the actual value being stored in the computer.*
- Example:



```
double x1 = 12.34;      cout << x1 << endl;
double x2 = 12.345;     cout << x2 << endl;
double x3 = 12.3456;    cout << x3 << endl;
double x4 = 12.34567;    cout << x4 << endl;
double x5 = 12.345678;   cout << x5 << endl;
double x6 = 12.3456789;  cout << x6 << endl;
double x7 = 123.456789;  cout << x7 << endl;
double x8 = 1234.56789;  cout << x8 << endl;
double x9 = 12345.6789;  cout << x9 << endl;
double x10 = 123456.789; cout << x10 << endl;
double x11 = 1234567.89; cout << x11 << endl;
double x12 = 12345678.9; cout << x12 << endl;
```



```
12.34
12.345
12.3456
12.3457
12.3457
12.3457
123.457
1234.57
12345.7
123457
1.23457e+006
1.23457e+007
```

Floating-point format in C++

- By default printing a floating-point using cout in C++ gives only 6-digit precision
- How to print out a floating-point in high precision? Use **setprecision**
- Example: use 15 significant digits to output floating-point:

```
#include <iostream>
```

→ `#include <iomanip>` // in order to use `std::setprecision`, we **must include iomanip**
`using namespace std;`

```
int main()
```

```
{
```

→ `cout << setprecision(15);` // use 15-digit precision for printing floating-point

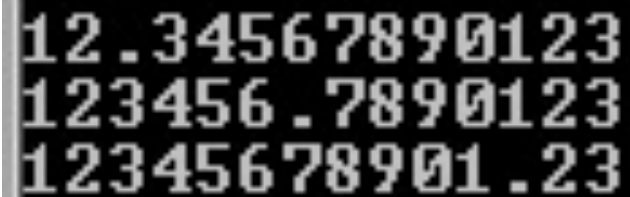
```
double y1 = 12.34567890123; cout << y1 << endl;
```

```
double y2 = 123456.7890123; cout << y2 << endl;
```

```
double y3 = 12345678901.23; cout << y3 << endl;
```

```
return 0;
```

```
}
```



```
12.34567890123
123456.7890123
12345678901.23
```

Round-off Error

- We use floating point to approximate a real number, so round-off error is unavoidable when the significant figure (in binary format) of a real number is large

- In some cases, the round-off errors can cause serious problems

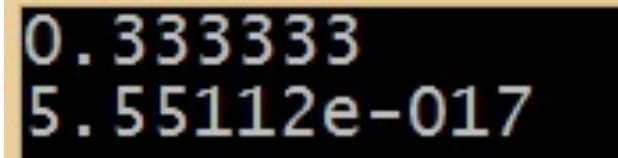
- Example of round-off: $1.0/3.0 = 0.3333333333333333\ldots$

When storing it on computer as double, only about 15 significant precision digits (53 bits) are stored

```
double x, y;
```

```
x = 1.0/3.0;    cout << x << endl;
```

```
y = 1.0-x*3.0;  cout << y << endl;
```



```
0.333333  
5.55112e-017
```

Most Commonly Used Data Types

Type Names	Description	Size	Range
<code>char</code>	Single text character or small integer. Indicated with single quotes (<code>'a'</code> , <code>'3'</code>).	1 byte	signed: -128 to 127 unsigned: 0 to 255
<code>int</code>	Larger integer.	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
<code>bool</code>	Boolean (true/false). Indicated with the keywords <code>true</code> and <code>false</code> .	1 byte	Just <code>true</code> (1) or <code>false</code> (0).
<code>double</code>	“Doubly” precise floating point number.	8 bytes	$\pm [2.2\text{E-}308, 1.79\text{E}308]$ (15 significant precision digits)

Variables and Constants

Variables

- A variable in C++ must be declared before being used (*Different from MATLAB*)
- Variable declaration: specify the type and the name of the variable
(the variable name is an identifier)

Examples:

```
int x, y;           // declare two integer variables x and y
double y2, r;       // declare two double-precision floating-point y2 and r
                    // more details on double in this lecture
char t;             // declare a character variable t ; more details soon
bool v;             // declare a Boolean variable v; more details soon
int 3sd, x$y;       // wrong
```

Variables

- **Variable initialization**
 - The first assignment of a value to a variable is called an initialization
 - In C++, it is **risky to use a variable without initialization**
`int x, y;`
`y = x + 8; // what is the value of y?`

Variables

- **Variable initialization (cont.)**

- **Two ways to initialize a variable**

- 1) **At the time of declaration.**

Example: `int x = 4, y = 5, z;`
 `double r1 = 4.5, r2 = 5.6, r3, r4;`

- 2) **After the declaration**

Example: `int x, y;`
 `// some codes`
 `x = 4;`
 `y = x + 8;`

Constants

- **Constants are treated just like regular variables except that their values cannot be modified after their definition**
- **It is recommended to define constants in CAPITALs**
- **Two ways to define a constant**
 - **Use preprocessor “#define”**
 - **Use keyword “const”**

Constants Example 1 (define constant using #define):

```
#include <iostream>
```

```
using namespace std;
```

```
#define RADIUS 10 // data type should not be used, semicolon should not be used
```

```
#define PI 3.1415926
```

```
int main( )
```

```
{
```

```
    double circumference, area;
```

```
    circumference = 2*PI*RADIUS;
```

```
    area = PI*RADIUS*RADIUS;
```

```
    cout << "Radius is: " << RADIUS << endl;
```

```
    cout << "Circumference is: " << circumference << endl;
```

```
    cout << "Area is: " << area;
```

```
    return 0;
```

```
}
```

Constants Example 2 (define constant using the key word const)

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{
```

```
    const int RADIUS=10;    // data type should be used, semicolon should be used
```

```
    const double PI = 3.1415926;
```

```
    double circumference, area;
```

```
    circumference = 2*PI*RADIUS;
```

```
    area = PI*RADIUS*RADIUS;
```

```
    cout << "Radius is: " << RADIUS << endl;
```

```
    cout << "Circumference is: " << circumference << endl;
```

```
    cout << "Area is: " << area;
```

```
    return 0;
```

```
}
```

Constants Example 3 (a constant must be initialized when it is defined):

```
#include <iostream>
using namespace std;

int main( )
{
    const int RADIUS;    // error: uninitialized const 'RADIUS'
    const double PI = 3.1415926;

    double circumference, area;

    circumference = 2*PI*RADIUS;
    area = PI*RADIUS*RADIUS;
    cout << "Radius is: " << RADIUS << endl;
    cout << "Circumference is: " << circumference << endl;
    cout << "Area is: " << area;
    return 0;
}
```

Constants Example 4 (the value of a constant should not be modified after being defined):

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{
```

```
    const int RADIUS=10;    // data type should be used, semicolon should be used
```

```
    const double PI = 3.1415926;
```

```
    double circumference, area;
```

```
    circumference = 2*PI*RADIUS;
```

```
    area = PI*RADIUS*RADIUS;
```

```
    RADIUS = 20;    // compilation error: assignment of read-only variable 'RADIUS'
```

```
    cout << "Radius is: " << RADIUS << endl;
```

```
    cout << "Circumference is: " << circumference << endl;
```

```
    cout << "Area is: " << area;
```

```
    return 0;
```

```
}
```