Programming with C++

COMP2011: C++ Basics

Cecia Chan
Brian Mak
Dimitris Papadopoulos
Pedro Sander
Charles Zhang

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

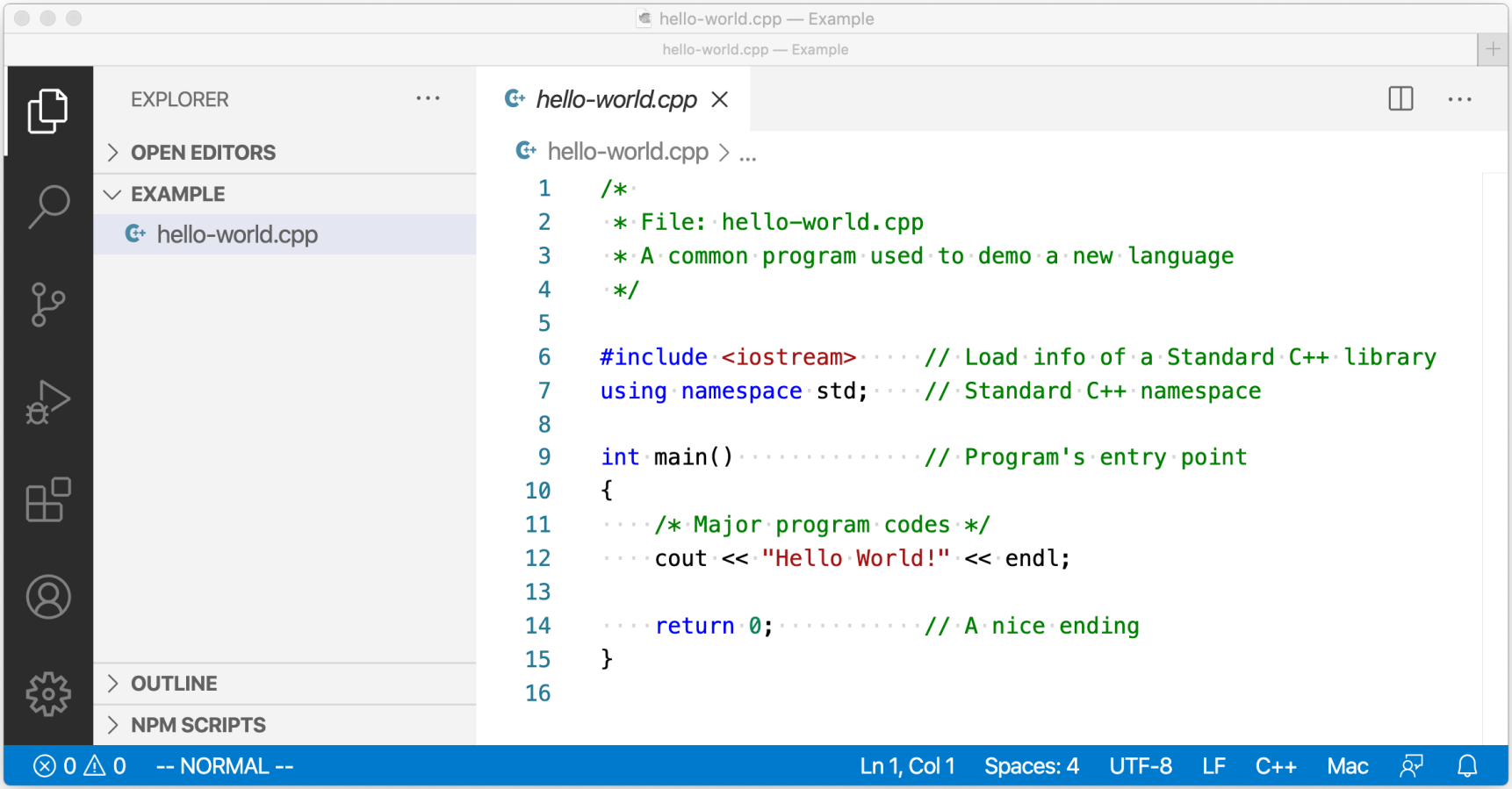
Programming Languages

- Just like different human languages (e.g. Chinese, English, Japanese, French, etc.), each programming language (e.g. Pascal, C++, Java, etc.) has its own
  - vocabulary = the set of legal "words"
  - grammar or syntax: how "words" are put together to make a legal "sentence"
- A program consists of a sequence of statements (c.f. sentence in human language)
- Some parts of a statement are called expressions (c.f. phrase in human language). e.g.
  - logical expression: $x > y$
  - arithmetic expression: $5 + 4$

Part I

A Simple C++ Program

Visual Studio Code: hello-world.cpp



Write, Compile, and Run a Program in a Terminal

STEP 1 : Write the program using an editor.
e.g., Visual Studio Code (Microsoft), Eclipse, vi (Unix/Linux), or even MS Word.

STEP 2 : Save the program into a file called hello-world.cpp.

STEP 3 : Compile the program using g++ compiler.

```
g++ -o hello-world hello-world.cpp
```

If you don't specify the output filename using the "-o" option, the default is a.out.

```
g++ hello-world.cpp
```

STEP 4 : Run the program in a terminal (command window):

```
linux:: hello-world
Hello World!
```

Example: Addition of 2 Numbers

```
/* File: add.cpp */
#include <iostream>      // Load info of a Standard C++ library
using namespace std;     // Standard C++ namespace

int main()               // Program's entry point
{
    /* Major program codes */
    cout << "123 + 456 = " << 123 + 456 << endl;

    return 0;            // A nice ending
}
```

Main: the Entry Point

- Every program must have exactly one and only one main( ) function.

Simple Form of the main Function

```
int main ( ) { ··· }
```

General Form of the main Function

```
int main (int argc, char** argv) { ··· }
```

(We'll talk about argc and argv later.)

- Between the braces "{" and "}" are the program codes consisting of zero or more program statements.
- Each simple C++ statement ends in a semicolon ";".

C++ Comments

- Use `/* ··· */` for multiple-line comments.

```
/*
 * A common program used to demo a new language
 */
```

- Single-line comments start with `//`.

```
// Program's entry point
```

- Comments are just for human to read.
- They will not be translated by the compiler into machine codes.

#include and Standard C++ Libraries

- #include will include information of a library — a collection of sub-programs. e.g. `#include <iostream>` gets the information of the standard C++ library called iostream that deals with I/O:
  - cin: an object to read, e.g., from the keyboard or file
  - cout: an object to print to, e.g., to the screen or file
  - cerr: an object to print error message, e.g., to the screen or file

Examples

```
// endl means "end of a line"
cout << "Einstein: God does not play dice." << endl;

// You may also break down the message in several lines
cerr << "Error: "
     << "There is no stress and tension in HKUST!"
     << endl;
```

- These library information files are called header files.

#include and User-defined Header Files

- #include causes the contents of the specified header file to be inserted into the original file.
- You may create your *own* header files, which contain the definition or declaration of some objects and data structures.
- Again you need to use #include to include them into your sub-programs.
- Example: `#include "drawing.h"` gets the information of a user-defined C++ library about drawing.
- By convention, the name of a header file ends in ".h" or ".hpp", while Standard C++ library header files have no file suffix.
- Also by convention, a header file created by users is delimited by double-quotes "..." , while Standard C++ library header files by < ... >.

C++ is a Free Format Language

- Extra blanks, tabs, lines are ignored.
- Thus, codes may be indented in any way to enhance readability.
- More than one statement can be on one line.
- Here is the same Hello World program:

```
#include <iostream> /* File: hello-world-too.cpp */
using       namespace std; int main   (int argc,
   char** argv) { cout<<"Hello World!"<<endl;return   0;}
```

- On the other hand, a single statement may be spread over several lines.

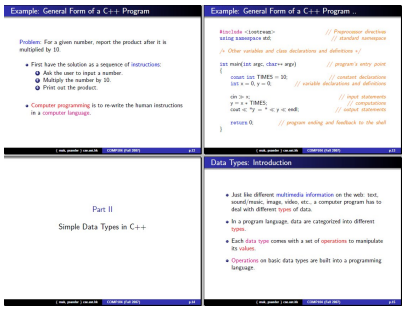
```
cout << "Hello World!"
     << endl;
```

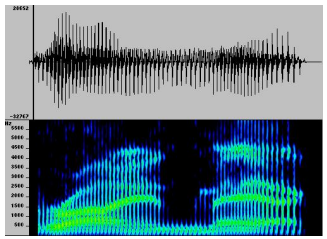
Good Programming Style

- Place each statement on a line by itself.
- For long statements
  - if possible, break it down into several shorter statements.
  - wrap it around with proper indentation (since extra space doesn't matter!)
- Use blank lines to separate sections of related codes that together perform some action.
- Indent consistently. Use the same indentation for the same block of codes.

Part II

Simple C++ Data Types: Integers, Characters, and Strings





Data Types: Introduction

- The Web has to deal with different multimedia data: text, sound/music, image, video, etc., and they can only be read/viewed with different softwares such as MS Notepad, Acrobat Reader, RealPlayer, etc.
- Similarly, a computer program has to deal with different types of data. In a programming language, data are categorized into different types.
- Each data type comes with a set of operations for manipulating its values. Operations on basic data types are built into a programming language.

Integers, Characters, Character Strings

- Integers
  - Examples: ..., -2, -1, 0, 1, 2, ...
  - C++ type name: int
- Characters
  - Examples: 'a', 'b', '4'
  - Represent a single character by delimiting it in single quotes.
  - For special characters, use the escape character \. e.g.

'\t'	=	tab	'\n'	=	newline
'\b'	=	backspace	'\0'	=	null character

  - C++ type name: char
- Character Strings
  - Examples: "hkust", "How are you?", "500 dollars"
  - Character strings are *not* a basic data type in C++.
  - They are sequences of basic char data.

Note: There is a string library that defines string objects which are more than a character string. (More about it later.)

How Numbers are Represented in Computers: Binary Numbers

- Computer uses binary numbers (base 2) to represent data.
- In the decimal system: $423_{10} = 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$.
- In the binary system:
  - A digit has only 2 possibilities: $\{0,1\}$.
  - Example: $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
  - Thus, the maximum N-digit number in base 2 =
  - A binary digit is aka bit.
  - 8 bits = 1 byte.
    (smallest amount of data that a computer can "*bite*" at once.)

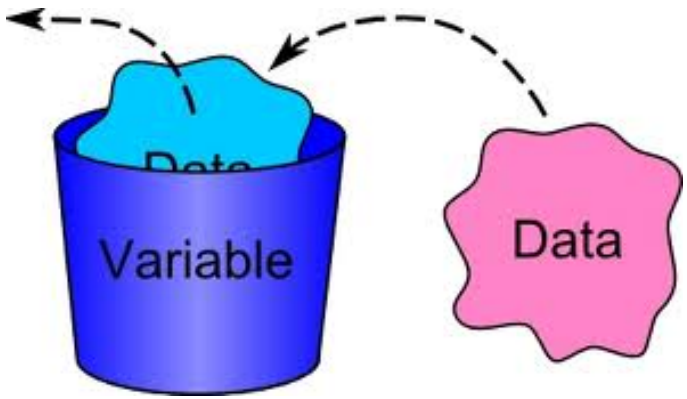
Relation between Characters and Integers

- In C++, a char datum is represented by 1 byte (8 bits).
- Question: How many different characters can 8 bits represent?
- Put it in another way, a char datum is encoded by one of the possible 8-bit patterns.
- The most common encoding scheme is called ASCII (American Standard Code for Information Interchange).
- Since a computer only recognizes bits, a char datum may also be interpreted as an integer!

Character	ASCII Code	Integral Value
'0'	00110000	48
'1'	00110001	49
'9'	00111001	57
'?'	00111111	63
'A'	01000001	65
'B'	01000010	66
'Z'	01011010	90
'a'	01100000	97
'b'	01100001	98
'z'	01111010	122

Part III

C++ Variables



Motivation Example: Addition of 2 Numbers Again

Example: Add 2 Numbers

```
/* File: add.cpp */
#include <iostream>     // Load info of a Standard C++ library
using namespace std;    // Standard C++ namespace

int main()              // Program's entry point
{
    /* Major program codes */
    cout << "123 + 456 = " << 123 + 456 << endl;

    return 0;           // A nice ending
}
```

- In this old example, the 2 numbers to be added are hard-coded into the program file.
- Can we write a program that takes 2 arbitrarily numbers to add?

Identifiers

$$f(x) = x^2 + c$$

where

f : name of a function
x : name of a variable
c : name of a constant

In programming languages, these "names" are called identifiers.

Rules for Making up Identifier Names

- Only the following characters may appear in an identifier:

  0–9, a–z, A–Z, _

- The first character cannot be a digit (0–9).
- C++ keyword — reserved words — are not allowed.
- Examples: amount, COMP2011, _myname_
- C++ identifiers are case-sensitive: lowercase and uppercase letters are considered different.

  ⇒ hkust, HKUST, HkUst, HKust are different identifiers.
- Examples of illegal C++ identifiers:
- Guidelines:
  - use meaningful names. e.g. amount instead of a
  - for long names consisting of several words, use '_' to separate them or capitalize them. e.g. num_of_students or numOfStudents instead of numofstudents.
  - usually identifiers starting with '_' are used for system variables.

Reserved Words in C++

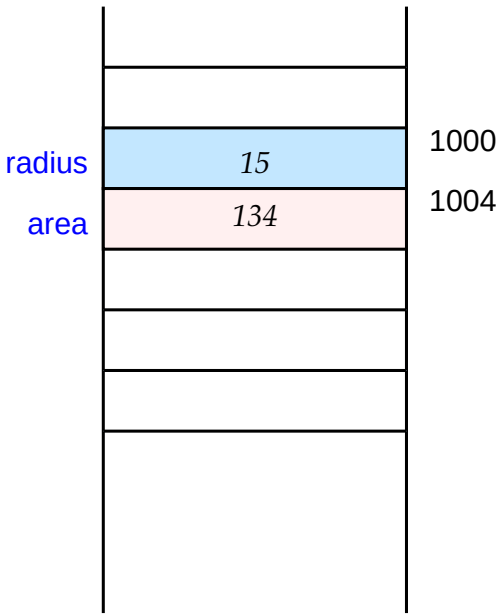
asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	extern
false	float	for	friend	~~goto~~
if	inline	int	long	mutable
namespace	new	operator	private	public
protected	register	reinterpret	return	short
signed	sizeof	static	static_cast	struct
switch	template	this	throw	true
try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile
wchar_t	while			

Introduction to Variables

- Name: Chan Tai Man's home
- Address: Unit X, Kam Wah House, Choi Hung Estate, Kowloon
- Content: Chan's family

Variables

A variable is a named memory location for a value that we can write to, retrieve from, and manipulate.



- It can be thought of as a container/box for a value.
- A variable must be declared and/or defined before it can be used.

Syntax: Variable Definition

<data-type> <identifier> ;

Examples

```
int radius = 10, sum = 0;
```

Variable Declaration/Definitions

Syntax: Defining Several Variables of the Same Type at Once

<data-type> <identifier1>, <identifier2>, ··· ;

Examples

```
int radius, num_of_words;
char choice, gender, pass_or_fail;
```

- When a variable is defined, the compiler allocates memory for it.
- The amount of memory is equal to the size of its data type.

** *Some books will call this variable declaration. Actually there is a big difference between variable declaration and variable definition. We'll talk about that later. When a variable is defined, it is also declared. The other way is not true.*

Variable Initialization

Syntax: Initialize Variables While they Are Defined

<data-type> <identifier> = <value> ;

- Several variables of the same type may also be initialized at the same time. e.g.

```
int radius = 10, sum = 0;
```

- A variable may also be initialized by a separate assignment statement after it is defined: e.g.

```
int radius;     // Variable definition
radius = 5;     // Initialization by assignment
```

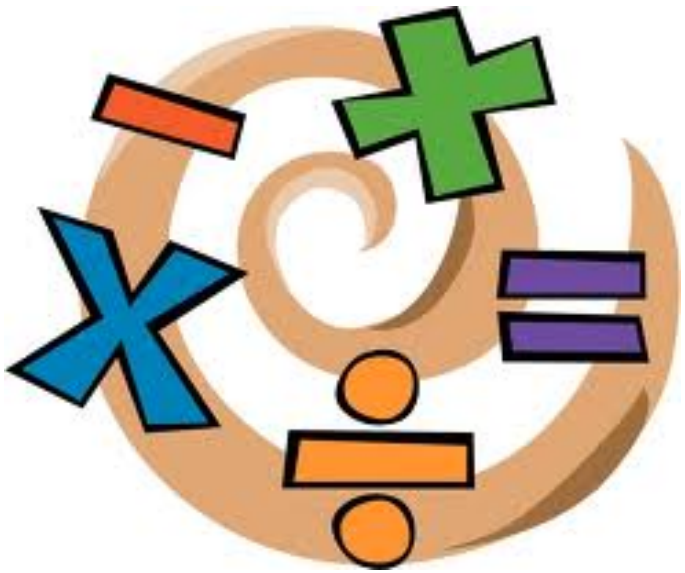
- ANSI C++ does not require compilers to initialize variables.
- Thus, in general, if you do not explicitly initialize variables while you are defining them, their initial contents may be garbage. (Global variables are an exception.)

Example: Addition of 2 Numbers Using Variables

```
#include <iostream>      /* File: add-var.cpp */
using namespace std;

int main()               // Program's entry point
{
    int x, y;            // Define 2 variables to hold the int values to add

    cin >> x;            // You may also shorten the 2 statements into one:
    cin >> y;            // cin >> x >> y;

    cout << x << " + " << y << " = " << x+y << endl;

    return 0;            // A nice ending
}
```

Part IV

Operators



Assignment Operator

Syntax: Assignment

<variable> = <value> ;

- In C++, the "=" sign is used to assign a value to a variable; it is the assignment operator.

Examples

```
int a, b, x = 2, y = 3, z = 4;

a = 10*x;
b = a - (100*y - 1000*z);
a = a + b;
```

- Don't try to understand the assignment statement: `a = a + b;` using normal math notation, otherwise, it doesn't make sense.
- Nor should you treat it as a boolean relational "equality" sign.

Arithmetic Operators

Assuming $x = 100$, $y = 67$:

OPERATION	OPERATOR	int
unary minus	$-$	
addition	$+$	
subtraction	$-$	
multiplication	$*$	
division	$/$	
modulus	%	
increment	$++$	
decrement	$--$	

Pre- and Post- Increment, Decrement

- The unary increment operator $++$ add 1 to its operand.
- The unary decrement operator $--$ subtract 1 from its operand.
- However, there are 2 ways to call them: pre-increment or post-increment. e.g.

$$++x \quad x++ \quad --x \quad x--$$

- If used alone, they are equivalent to: $x = x + 1$ and $x = x - 1$.
- But if used with other operands, then there is a big difference:
  - $++x \Rightarrow$ add 1 to x, and use the result for further operation.
  - $x++ \Rightarrow$ use the current value of x for some operation, and then add 1 to x.

```
cout << ++x;
/* same as */
x = x + 1;
cout << x;
```

```
cout << x++;
/* same as */
cout << x;
x = x + 1;
```

Example: %, ++, −−

```
#include <iostream>     /* File: inc-mod.cpp */
using namespace std;

int main()
{
    int x = 100, y = 100; // Variable definitions and initialization
    int a = 10, b = 10, c = 10, d = 10;

    b = a + (x++) + ++y;

    cout << ++x << "\t"; cout << "x = " << x << endl; // Pre-increment
    cout << y++ << "\t"; cout << "y = " << y << endl; // Post-increment

    a = ++b; cout << "a = " << a << "\t" << "b = " << b << endl;
    c = d++; cout << "c = " << c << "\t" << "d = " << d << endl;

    cout << 17%5 << endl; // Trickiness of the mod function
    cout << (-17)%5 << endl;
    cout << 17%(-5) << endl;
    cout << (-17)%(-5) << endl;

    return 0;
}
```

Shorthand Assignment Operators

SHORTHAND NOTATION	NORMAL NOTATION
$n \mathrel{+}= 2$	$n = n + 2$
$n \mathrel{-}= 2$	$n = n - 2$
$n \mathrel{*}= 2$	$n = n * 2$
$n \mathrel{/}= 2$	$n = n / 2$
$n \mathrel{\%}= 2$	$n = n \% 2$

Precedence and Associativity

Operator	Description	Associativity
$-$ $++$ $--$	minus increment decrement	Right-to-Left
$*$ $/$ $\%$	multiply divide mod	Left-to-Right
$+$ $-$	add subtract	Left-to-Right
$=$	assignment	Right-to-Left

Precedence

Example: $1/2 + 3 * 4 = (1/2) + (3 * 4)$
because $*$, $/$ has a higher precedence over $+$, $-$.

- Precedence rules decide which operators run first.
- In general,

$$x\ P\ y\ Q\ z \quad = \quad x\ P\ (\ y\ Q\ z\ )$$

if operator Q is at a higher precedence level than operator P.

Associativity: Binary Operators

Example: $1 - 2 + 3 - 4 = ((1 - 2) + 3) - 4$
because $+$, $-$ are left associative.

- Associativity decides the grouping of operands with operators of the *same* level of precedence.
- If binary operator P, Q are of the same precedence level
  - if operator P, Q are both right associative, then

$$x\ P\ y\ Q\ z \quad = \quad x\ P\ (\ y\ Q\ z\ )$$

  - if operator P, Q are both left associative, then

$$x\ P\ y\ Q\ z \quad = \quad (\ x\ P\ \ y\ )\ Q\ z$$

Cascading Assignments

- C++ allows assigning the same value to multiple variables at once.

Examples

```
int w, x, y, z;

y = z = 5;          // Same as y = (z = 5);
w = x = y + z;      // Same as w = (x = (y+z));
```

Expression and Statement

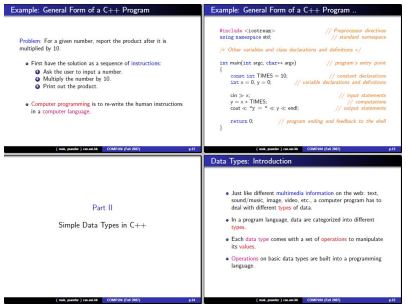
- An expression has a value which is the result of some operation(s) on its(theirs) operands.
- Expression examples:

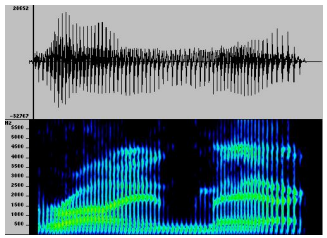
  4 x - y 2 - a - (b * c)

- A statement is a sentence that acts as a command.
  - It does not have a value.
  - It always ends in a ';'.
- Statement examples:
  - Input statement: cin $\gg$ x;
  - Output statement: cout $\ll$ x;
  - Assignment statement: x = 5;
  - Variable definition: int x;
- For the first 3 statement examples above, if we take out the ending ';', they become input/output/assignment expressions! (More about this later.)

Part V

More Basic Data Types in C++





C++ Basic Types

Types	Common Size(#bytes on a 32-bit machine)	Value Range
bool	1	{ true, false }
char	1	[-128, 127]
short	2	[-32768, 32767]
int	4	$[-2^{31}, 2^{31}-1]$
long	4	$[-2^{31}, 2^{31}-1]$
float	4	$\pm$ [1.17549E-38, 3.40282E+38]
double	8	$\pm$ [2.22507E-308, 1.79769E+308]

- Not all numbers of a type can be represented by a computer.
- It depends on how many bytes you use to represent it: with more bytes, more numbers can be represented.

Integers

- Type names: short (int), int, long (int), long long (int)
- Their sizes depend on the CPU and the compiler.
- ANSI C++ requires:
  size of short $\leq$ size of int $\leq$ size of long $\leq$ size of long long
- e.g., What are the numbers that can be represented by a 2-byte short int?
- Each integral data type has 2 versions:
  - signed version: represents both +ve and -ve integers.
    e.g. signed short, signed int, signed long
  - unsigned version: represents only +ve integers.
    e.g. unsigned short, unsigned int, unsigned long
- signed versions are the default.
- Obviously unsigned int can represent 2 times more +ve integers than signed int.

Floating-Point Data Types

- Floating-point numbers are used to represent real numbers and very large integers (which cannot be held in long long).
- Type names:
  - float for single-precision numbers.
  - double for double-precision numbers.
- Precision: For decimal numbers, if you are given more decimal places, you may represent a number to higher precision.
  - for 1 decimal place: 1.1, 1.2, 1.3, ... etc.; can't get 1.03.
  - for 2 decimal places: 1.01, 1.02, 1.03, ... etc.; can't get 1.024.
- In scientific notation, a number has 2 components. e.g., 5.16E-02
  - mantissa: 5.16
  - exponent: -2
- More mantissa bits $\Rightarrow$ higher precision.
- More exponent bits $\Rightarrow$ larger real number.

Integer Arithmetic and Floating-Point Arithmetic

- Arithmetic expressions involving *only* integers use integer arithmetic.
- Arithmetic expressions involving *only* floating-point numbers use floating-point arithmetic.
- For $+$, -, $\times$ operations, results should be what you expect.
- However, integer division and floating-point division may give different results. e.g.,
  - $10/2 = 5$ and $10.0/2.0 = 5.0$
  - $9/2 = 4$ and $9.0/2.0 = 4.5$
  - $4/8 = 0$ and $4.0/8.0 = 0.5$

Boolean Data Type

- Type name: bool.
- Used to represent the truth value, true or false of logical (boolean) expressions like:

$$a > b \qquad x + y == 0 \qquad \text{true \&\& false}$$

- Since C++ evolves from C, C++ follows C's convention:
  - zero may be interpreted as false.
  - non-zero values may be interpreted as true.
- However, since internally everything is represented by 0's and 1's,
  - false is represented as 0.
  - true is represented as 1.
- Even if you put other values to a bool variable, its internal value always is changed back to either 1 or 0.

Example: Output Boolean Values

```
#include <iostream>     /* File: boolalpha.cpp */
using namespace std;

int main()
{
    bool x = true;
    bool y = false;

    // Default output format of booleans
    cout << x << " && " << y << " = " << (x && y) << endl << endl;

    cout << boolalpha;   // To print booleans in English
    cout << x << " && " << y << " = " << (x && y) << endl << endl;

    cout << noboolalpha; // To print booleans in 1 or 0
    cout << x << " && " << y << " = " << (x && y) << endl;

    return 0;
}
```

Example: Use of bool Variables

```
#include <iostream>     /* File: bool-blood-donation.cpp */
using namespace std;

int main()
{
    char donor_blood_type, recipient_blood_type;
    bool exact_match, match_all;

    cout << "Enter donor's bloodtype: A, B, C (for AB), and O. ";
    cin >> donor_blood_type;
    cout << "Enter recipient's bloodtype: A, B, C (for AB), and O. ";
    cin >> recipient_blood_type;

    exact_match = (donor_blood_type == recipient_blood_type);
    match_all = (donor_blood_type == 'O');

    if (exact_match || match_all)
        cout << "Great! A donor is found!" << endl;
    else
        cout << "Keep searching for the right donor." << endl;

    return 0;
}
```

Part VI

Type Checking and Type Conversion

Type Checking and Coercion

Analogy:

Blood Types	
Receiver	Donor
A	A, O
B	B, O
AB	A, B, AB, O
O	O

- For most languages, data types have to be matched during an operation $\Rightarrow$ type checking.
- However, sometimes, a type is made compatible with a different type $\Rightarrow$ coercion.

Operand Coercion

Coercion is the automatic conversion of the data type of operands during an operation.

- Example: $3 + 2.5 \Rightarrow$ int + double.
- The C++ compiler will automatically change it to
$$3.0 + 2.5 \Rightarrow \text{double} + \text{double}$$
- Thus, the integer 3 is coerced to the double 3.0.

Example: Convert a Small Character to Capital Letter

```
char small_y, big_y;
cin >> small_y;                  // Character in small case
big_y = small_y + 'A' - 'a'; // Character in big case
```

Here big_y, small_y, 'A', and 'a' are "coerced" by "promoting" it to int before addition. The result is converted back (or coerced) to char.

Automatic Type Conversion During Assignment

Examples

```
float x = 3.2;          // Initialize x with 3.2 by assignment
double y = 5.7;         // Initialize y with 5.7 by assignment

short k = x;            // k = ?
int n;
n = y;                  // n = ?
```

- Since float|double can hold numbers bigger than short | int, the assignment of k and n in the above program will cause the compiler to issue a warning — not an error.

Compiler Warnings

a.cpp:9: warning: converting to 'short int' from 'float'
a.cpp:11: warning: converting to 'int' from 'double'

Automatic Type Conversion During Assignment ..

- A narrowing conversion changes a value to a data type that might not be able to hold some of the possible values.
- A widening conversion changes a value to a data type that can accommodate any possible value of the original data.
- C++ uses truncation rather than rounding in converting a float|double to short | int | long.

Manual Type Conversion (Casting)

```
int k = 5;
int n = 2;
float x = n/k;          // What is the value of x?
```

- In the above example, one can get x = 0.4 by manually converting *n* and/or *k* from int to float|double.

Syntax: static_cast for manual type casting

static_cast<data-type> (value)

- No more warning messages on narrowing conversion.

```
int k = 5, n = 2;
float x = static_cast<double>(n)/k;
float y = n/static_cast<double>(k);
float z = static_cast<double>(n)/static_cast<double>(k);
```

Part VII

Constants

Literal Constants

- Constants represent fixed values, or permanent values that cannot be modified (in a program).
- Examples of literal constants:
  - char constants: 'a', '5', '\n'
  - string constants: "hello world", "don't worry, be happy"
  - int constants: 123, 456, -89
  - double constants: 123.456, -2.90E+11

Symbolic Constants

- A symbolic constant is a named constant with an identifier name.
- The rule for identifier names for constants is the same as that for variables. However, by convention, constant identifiers are written in capital letters.
- A symbolic constant must be defined and/or declared before it can be used. (Just like variables or functions.)
- Once defined, symbolic constants cannot be changed!

Syntax: Constant Definition

const <data-type> <identifier> = <value> ;

Example

```
const char BACKSPACE = '\b';
const float US2HK = 7.80;
const float HK2RMB = 0.86;
const float US2RMB = US2HK * HK2RMB;
```

Why Symbolic Constants?

Compared with literal constants, symbolic constants are preferred because they are

- more readable. A literal constant does not carry a meaning. e.g. the number 320 cannot tell you that it is the enrollment quota of COMP2011 in 2015.

```
const int COMP2011_QUOTA = 320;
```

- more maintainable. In case we want to increase the quota to 400, we only need to make the change in one place: the initial value in the definition of the constant COMP2011_QUOTA.

```
const int COMP2011_QUOTA = 400;
```

- type-checked during compilation.

Remark: Unlike variable definitions, memory is not allocated for constant definitions with only few exceptions.

Example: Use of Symbolic Constants

```
#include <iostream>     /* File: symbolic-constant.cpp */
#include <cmath>        // For calling the ceil() function
using namespace std;

int main()
{
    const int COMP2011_QUOTA = 320;
    const float STUDENT_2_PROF_RATIO = 100.0;
    const float STUDENT_2_TA_RATIO = 40.0;
    const float STUDENT_2_ROOM_RATIO = 100.0;

    cout << "COMP2011 requires "
         << ceil(COMP2011_QUOTA/STUDENT_2_PROF_RATIO)
         << " instructors, "
         << ceil(COMP2011_QUOTA/STUDENT_2_TA_RATIO)
         << " TAs, and "
         << ceil(COMP2011_QUOTA/STUDENT_2_ROOM_RATIO)
         << " classrooms" << endl;

    return 0;
}
```

Part VIII

Further Readings and Examples

Find Out the Size of a Data Type using sizeof

```
#include <iostream>      /* File: value.cpp */
using namespace std;

int main()
{
    cout << "sizeof(bool) = " << sizeof(bool) << endl;
    cout << "sizeof(char) = " << sizeof(char) << endl;
    cout << "sizeof(short) = " << sizeof(short) << endl;
    cout << "sizeof(int) = " << sizeof(int) << endl;
    cout << "sizeof(long) = " << sizeof(long) << endl;
    cout << "sizeof(long long) = " << sizeof(long long) << endl;
    cout << "sizeof(float) = " << sizeof(float) << endl;
    cout << "sizeof(double) = " << sizeof(double) << endl;
    cout << "sizeof(long double) = " << sizeof(long double) << endl;

    return 0;
}
```

Size of Basic Types on 2 Computers

on a 32-bit machine

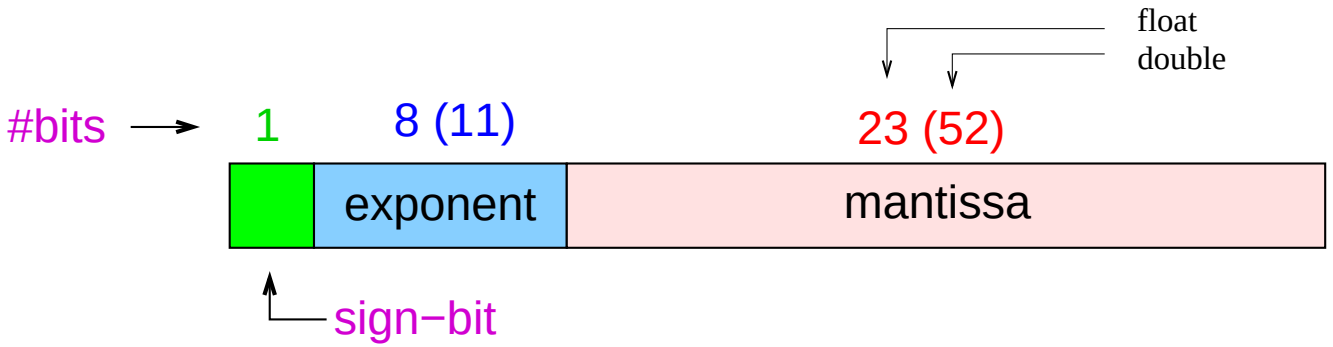
```
sizeof(bool) = 1
sizeof(char) = 1
sizeof(short) = 2
sizeof(int) = 4
sizeof(long) = 4
sizeof(long long) = 8
sizeof(float) = 4
sizeof(double) = 8
sizeof(long double) = 12
```

on a 64-bit machine

```
sizeof(bool) = 1
sizeof(char) = 1
sizeof(short) = 2
sizeof(int) = 4
sizeof(long) = 8
sizeof(long long) = 8
sizeof(float) = 4
sizeof(double) = 8
sizeof(long double) = 16
```

- Note that the figures may be different on your computer.
- A 32(64)-bit machine uses CPUs of which the data bus width and memory address width are 32 (64) bits.

Representation of Floating-Point Data



- Many programming language uses the IEEE 754 floating-point standard.
- Binary Representation of mantissa: e.g.

$$1.011_2 = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

- Binary Representation of exponent: signed integer
- All floating-point data types in C++ are signed.
- ANSI C++ requires: size of float $\leq$ size of double

Question: Can every real number be represented by float in C++?

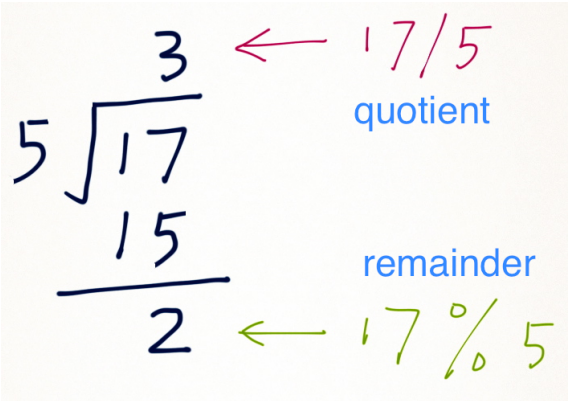
Underflow and Overflow in Integral Data Types

- Overflow: occurs when a data type is used to represent a number larger than what it can hold. e.g.
  - if you use a short int to store HK's population.
  - when a short int has its max value of 32767, and you want to add 1 to it.
- Underflow: occurs when a data type is used to represent a number smaller than what it can hold. e.g.
  - use an unsigned int to store a -ve number.

Underflow and Overflow in Floating-Point Data Types

- Underflow: when the -ve exponent becomes too large to fit in the exponent field of the floating-point number.
- Overflow: when the +ve exponent becomes too large to fit in the exponent field of the floating-point number.
- To prevent these from happening, use double if memory space allows.
- In fact, all floating literals (e.g., 1.23) is treated as double unless explicitly specified by a suffix (e.g., 1.23f).

Modulo Arithmetic



- mod is used to get the remainder in an integer division.

$$\text{mod}(17, 5) = 17 \bmod 5 = 17\%5 = 2$$

- Strictly speaking, $m \bmod n$ is defined only if n is +ve.
- Most programming languages support -ve divisor and different languages may give you different results!
- In C++, the modulo arithmetic is supported by the remainder operator % which allows -ve divisor.
- Question: What are the results of (-17)%5, 17%(-5), or (-17)%(-5)?

Priority Rules for the Usual Arithmetic Conversions for Binary Operations

- If either operand is of type long double, convert the other operand also to long double.
- If either operand is of type double, convert the other operand also to double.
- If either operand is of type float, convert the other operand also to float.
- Otherwise, the integral promotions shall be performed on both operands.
  - Similar rules are used for integral promotion of the operands.
  - Compute using integer arithmetic.

Question: What is the result of 3/4?