CS2310 Computer Programming

LT1: Introduction to Programming

Computer Science, City University of Hong Kong (Dongguan)

Semester A 2025-26

Which Programming Language?

ActionScript Ada ASP.NET Assembler Basic
C C++ C# Cobol Cobra CODE ColdFusion
Delphi Eiffel Fortran FoxPro GPSS HTML J#
J++ Java JavaScript JSP LISP Logo LUA
MEL Modula-2 Miranda Objective-C Perl
PHP Prolog Python SQL Visual Basic
Visual Basic.NET VBA Visual-FoxPro

About the Course

- Teaching pattern
  - Lectures
    - Explain the terminologies, concepts, methodologies, …
  - Labs
    - Hands-on programming practice
    - Analyzing example problems and implementing programs

- Canvas-based course website
  - Teaching materials are all in Canvas
  - It is your own responsibility to check Canvas and University emails regularly for updates

About the Course

- Assessment
  - **Coursework (50%)**
    - Assignments (15%)
    - Midterm quiz (20%)
      - Week 7
    - Lab Assessment (25%)
      - Deadline is 24 hours after your lab session
  - **Final examination (50%)**

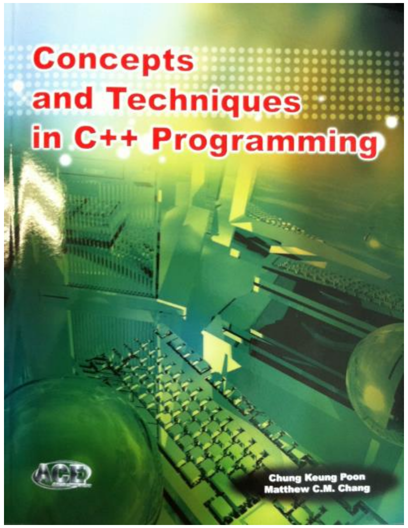
About the Course

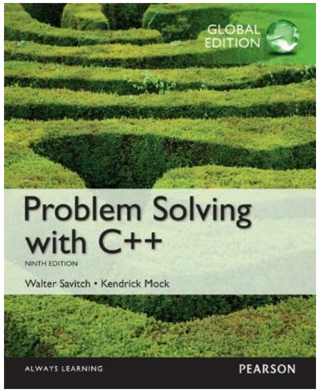
- Assessment
  - To **pass** the course you must obtain:
    - At least 40% of the max. mark of continuous assessment; **AND**
    - At least 30% of the max. mark of final exam

Student	Coursework	Exam	Final Mark	Grade
1	94.3	95.5	94.9	A+
2	43.8	34	38.9	D
3	37.2	65.8	51.5	F
4	86.8	26.5	56.7	F

About the Course

- Resources
  - Textbook (NIL)
  - Reference books
    - ***Concepts and Techniques in C++ Programming***, *by Chung Keung Poon, Matthew C.M. Chang*
    - ***Problem Solving with C++***, *by Walter Savitch, Kendrick Mock*





  - Microsoft Visual Studio (Windows)
    - Develop environment for compiling & debugging
  - E-Quiz
    - Program testing and submission

About the Course

- Key to success

Just Do It

But, do it yourself

About the Course

- “Do it yourself” means
  - Discuss the problems with any other people
  - Study materials on the internet
  - Refer to any books
- **But**, the details and write-ups must be entirely your own work

University requirement on academic honesty.

– Violations of academic honesty are regarded as serious offences in the University. Acts such as plagiarism (and fabrication of research findings) can lead to disciplinary action. Most commonly the penalty is **failure in a course**, but in the most serious cases expulsion from the University and debarment from re-admission may occur.

About the Course

- Things draw your attentions
  - Plagiarism
    - Punishment ranges from warning to course failure
    - May cause you be forced out of CityUDG
    - Can be automatically detected by Turnitin system
  - How to prevent
    - In plagiarism cases, both the **giver** and **copier** get punishments
    - Protect your code
  - As instructors
    - We have the responsibility to report academic dishonesty cases so as not to compromise the quality of education
    - We take suspected plagiarism cases very seriously.

ChatGPT Policy

- CityU CS Department-wide ChatGPT policy (31 August 2023)
  - 1) Students are not allowed to use GenAI for programming tasks
  - 2) For writing assignments and reports, students are allowed to use GenAI, but its use must be acknowledged through proper citation and referencing
- The above two rules apply to all CS courses by default

Outline for Today

- Programming languages

- Building a C++ program
- Simple program

- Variables and constants
  - Name
  - Type
  - Address
  - Scope

Programming Languages

- To write a program for a computer, we must use a computer language.

Machine Language

Language directly understood by the computer

binary code

PROGRAM 1-1 The Multiplication Program in Machine Language

```
         00000000 00000100 0000000000000000
01011110 00001100 11000010 0000000000000010
         11101111 00010110 0000000000000101
         11101111 10011110 0000000000001011
11111000 10101101 11011111 0000000000010010
         01100010 11011111 0000000000010101
11101111 00000010 11111011 0000000000010111
11110100 10101101 11011111 0000000000011110
00000011 10100010 11011111 0000000000100001
11101111 00000010 11111011 0000000000100100
01111110 11110100 10101101
11111000 10101110 11000101 0000000000101011
00000110 10100010 11111011 0000000000110001
11101111 00000010 11111011 0000000000110100
         01010000 11010100 0000000000111011
                  00000100 0000000000111101
```

The only language understood by computer hardware is machine language.

Programming Languages

- To write a program for a computer, we must use a computer language.

Machine Language

Language directly understood by the computer

Symbolic Language

English-like abbreviations representing elementary computer operations

binary code

assembly language

PROGRAM 1-2 The Multiplication Program in Symbolic Language

```
        entry   main,^m<r2>
        subl2   #12,sp
        jsb     C$MAIN_ARGS
        movab   $CHAR_STRING_CON

        pushal  -8(fp)
        pushal  (r2)
        calls   #2,SCANF
        pushal  -12(fp)
        pushal  3(r2)
        calls   #2,SCANF
        mull3   -8(fp),-12(fp),-
        pusha   6(r2)
        calls   #2,PRINTF
        clrl    r0
        ret
```

Symbolic language uses symbols, or mnemonics, to represent the various machine language instructions.

Programming Languages

- To write a program for a computer, we must use a computer language.

Machine Language

Language directly understood by the computer

binary code

Symbolic Language

English-like abbreviations representing elementary computer operations

assembly language

High-level Language

Close to human language.

Example: $a = a + b$

[add values of *a* and *b*, and store the result in *a*, replacing the previous value]

C, C++, Java, Basic

PROGRAM 1-3 The Multiplication Program in C

```c
/* This program reads two integers from the keyboard
   and prints their product.
      Written by:
      Date:
*/
#include <stdio.h>

int main (void)
{
// Local Definitions
   int number1;
   int number2;
   int result;

// Statements
   scanf  ("%d", &number1);
   scanf  ("%d", &number2);
   result = number1 * number2;
   printf ("%d", result);
   return 0;
}  // main
```

high-level languages are easier for us to understand.

Different Programming Languages

ActionScript Ada ASP.NET Assembler Basic
C C++ C# Cobol Cobra CODE ColdFusion
Delphi Eiffel Fortran FoxPro GPSS HTML J#
J++ Java JavaScript JSP LISP Logo LUA
MEL Modula-2 Miranda Objective-C Perl
PHP Prolog Python SQL Visual Basic
Visual Basic.NET VBA Visual-FoxPro

Programming Languages

- Programming languages usually differ in two aspects
  - Language Syntax
  - Standard libraries/SDKs/functions
- Java

```
if (a>b){
     System.out.println("a is larger than b");
}else{
    System.out.println("a is smaller than or equal to b");
}
```

- Pascal

```
if a>b then
    writeln('a is larger than b');
else
   writeln('a is smaller than or equal to b');
```

Programming Languages

- Syntax is well-defined, **NO** exceptions
  - if (…){…}else{…};
  - for (;;;){…}
  - while (){…}

- Basic Components:
  - Variable / structure /function declaration
  - Variable / structure /function access
  - Conditional statement
  - Iteration statement
  - SDK/built-in functions

Outline for Today

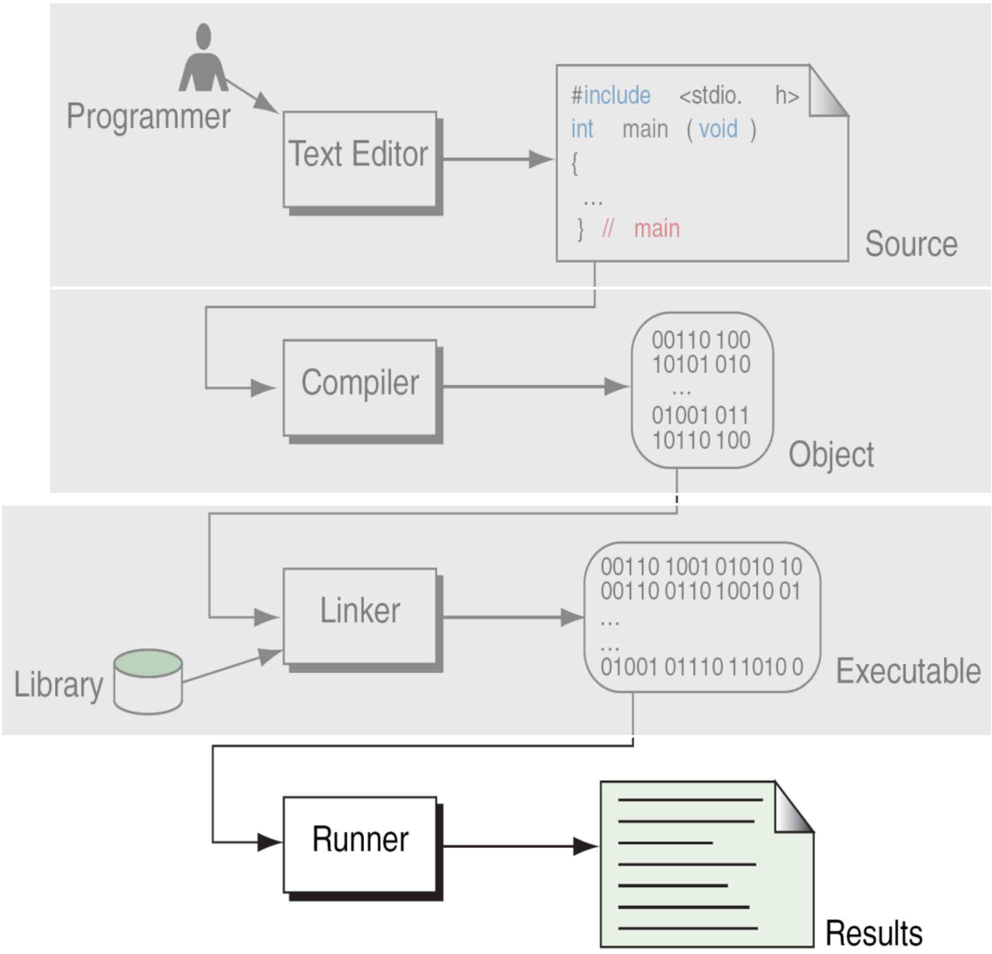
- Programming languages

- Building a C++ program
- Simple program

- Variables and constants
  - Name
  - Type
  - Address
  - Scope

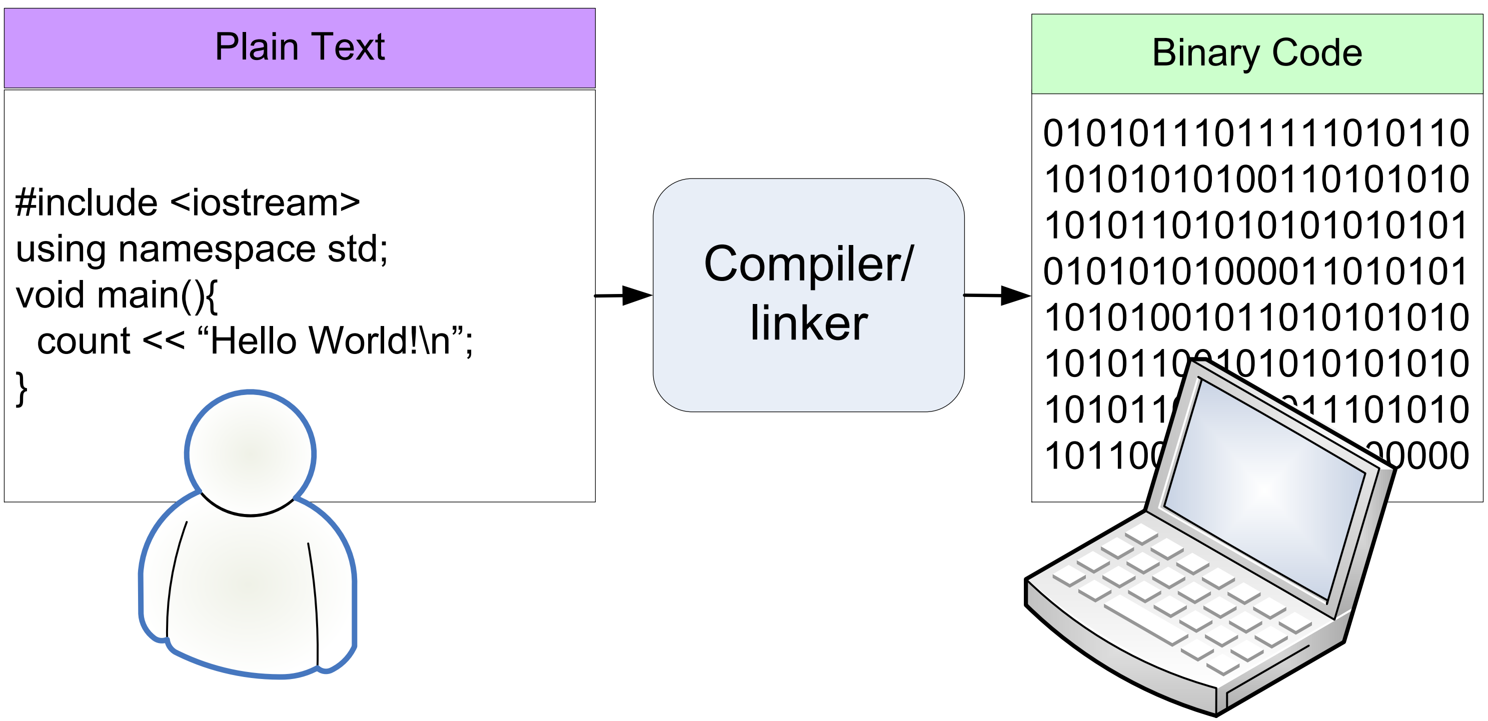
Building a C++ program

- **Writing** source code as a C++ file.
  - e.g., "***hello.cpp***" file
- **Preprocessing**
  - Processes the source code for compilation.
- **Compilation**
  - Checks the grammatical rules (syntax).
  - Source code is converted to object code in machine language (e.g. "***hello.obj***" file)
- **Linking**
  - Combines object code and libraries to create an executable (e.g. "***hello.exe***" file).
  - <u>Library</u>: common functions (input, output, math, etc).



Code: lec01-01-build.cpp

Building a C++ program



Simple Program

```
/* The traditional first program in honor of
   Dennis Ritchie who invented C at Bell Labs
   in 1972 */

#include <iostream>
using namespace std;

void main()
{
   cout << "Hello, world!\n";
}
```

Function - main

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

void main()
{
   cout << "Hello, world!\n";
}
```

- `void main()`
  - `void` means there is **NO return** value
  - `main` is the **name** of the function
  - **No** semi-colon after main()
  - C++ is case sensitive:
  - E.g., `void Main()`, `VOID main()` are incorrect

- `{ }`
  - Braces: left brace begins the body of a function. The corresponding right brace must end the function

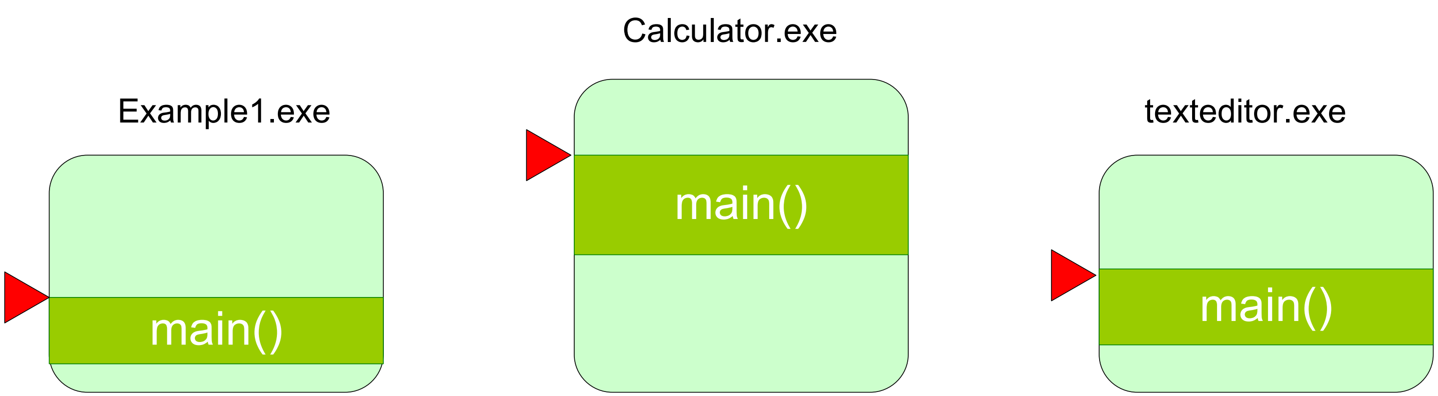
Function - main

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

void main()
{
  cout << "Hello, world!\n";
}
```

```
void main()
{

}
```

Critical Thinking

- The starting point of program (the first function called by the computer)



Library / SDK /Package

- Normally, we won't write a program all by ourselves. Instead, we will reuse the code written by ourselves / other developers. Especially for the repeating tasks or low-level operations like disk I/O
- The reusing code is well designed and pack a library / SDK / Package
- Standard C++ program comes with a set of package to make programmer task easier
- ***cout*** is one of the example

cout

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

void main()
{
  cout << "Hello, world!\n";
}
```

cout << "Hello, world!\n“ ;

- **cout**: “Console OUTput” allows our program to output values to the standard output stream (the screen)
- **cout**: object provided by **iostream** library (package) for screen (console) output (we will elaborate this concept in future classes)
- **<<**: output (also called insertion) operator that output values to an output device. In this case, the output device is **cout** (the screen)
- The value on the right hand side of the operator ("Hello, world!\n“) is the string you want to output
  - Any *literal (character string)* that is to be output must be in between a pair of double quotes

Object - cout

- `\n`
  - **escape sequence**: the character following \ is not interpreted in the normal way
  - represents a newline character: the effect is to advance the cursor on the screen to the beginning of the next line
  - newline: position the character to the beginning of next line
- `\\`
  - backslash: Insert the backslash character \ in a string
- `\"`
  - double quote: Insert the double quote character " in a string
- `endl`
  - Same as the string "\n".
  - No \ before `endl`

Syntax errors

```
/* The traditional first program in honor of
   Dennis Ritchie who invented C at Bell Labs
   in 1972 */

#include <iostream>
using namespace std;

void main()
{
   cout < Hello, world! < endl
   cout < Hello, world Again! < endl
}
```

The texts to output should be placed in a pair of double quotes " texts".

< is not an operator of `cout`. We need to use <<.

We need ; at the end of each statement

Preprocessor directive

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

void main()
{
  cout << "Hello, world!\n";
}
```

using namespace std;

- Standard (std) *namespace* is used such that we can use a shorthand name for the element *cout*
  - std::cout <-> cout

#include <iostream>

- Include library `iostream` into the program as it contains the definition of `cout`, which is used to print something to the screen.
- Load contents of a certain file / library
- **NO** semi-colon at the end of the include directive

Code: lec01-02-world.cpp

Simple Program

/* The traditional first program in honor of
Dennis Ritchie who invented C at Bell Labs
in 1972 */

- Enclosed by "/*" and "*/" or begin with "//"
  - // single line comments

// this is a single line comment
// each line must begin with the "//" sign

A general C++ program

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

void main()
{

    /* Place your code here! */



}
```

Syntax

- Like any language, C++ has an alphabet and rules for putting together words and punctuations to make a legal program. This is called *syntax* of the language

- C++ compilers detect any violation of the syntax rules in a program

- C++ compiler collects the characters of the program into ***tokens***, which form the basic vocabulary of the language

- **Tokens** are separated by **space**

Syntax - Tokens

- Tokens can be categorized into:
  - *keywords*, e.g., `return, namespace, int`
  - *identifiers*, e.g., user-defined variables, **objects**, functions, etc.
  - *string constants*, e.g., `“Hello”`
  - *numeric constants*, e.g., `7, 11, 3.14`
  - *operators*, e.g., `+`
  - *punctuators*, e.g., `;` and `,`

```
#include <iostream>
using namespace std;
void main()
{
    cout << "Hello, world!\n";
}
```

Keywords (reserved words)
- covered in this course -

Data type	char	double	float	int	bool
	long	short	signed	unsigned	void
Flow control	if	else	switch	case	
	break	default	for	do	
	while	continue			
Others	using	namespace	true	false	sizeof
	return	const	class	new	delete
	operator	public	protected	private	friend
	this	try	catch	throw	struct
	typedef	enum	union		

Keywords (cont'd)

- Each keyword has a reserved meaning and cannot be used as identifiers
  - Can we have a variable called "main"?

Code: lec01-03-var.cpp

Identifiers

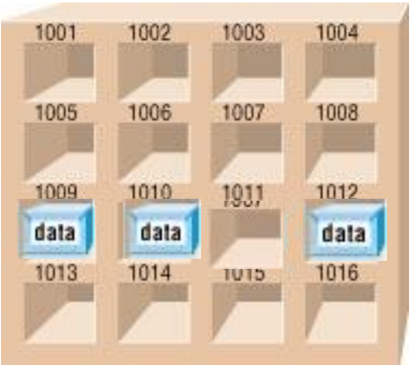
- Identifiers give unique **names** to objects, variables, functions, etc.
- Keywords cannot be used as identifiers
- An identifier is composed of a sequence of letters, digits and underscores
  - No hyphen (-)
- An identifier **must** begin with either an **underscore** (not recommended) or a **letter**
  - valid identifier: _income, record1, my_income , My_income
  - Invalid identifier: 3D_Point, my-income
- Always use meaningful names for identifiers
  - Bad examples: x, xx, xxx, xxxx …

Outline for Today

- Programming languages
- Building a C++ program
- Simple program
- Variables and constants
  - Name
  - Type
  - Address
  - Scope

Variables and Constants

- Data stored in memory, in binary format.
  - They do **not** exist after the program execution.



- A variable: its value may be changed during program execution.

- A constant: its value will **NOT** be changed during program execution.

Variables and Constants

- Every variable/constant have **four** attributes: *name, type, address* *and* *scope*
  - *Name:* identifier of the variable
  - *Type:* variables/constants must belong to a data type, either ***predefined*** or ***user-defined***
  - *Address*: the memory location of the variable
  - *Scope:* it defines **where** the variable can be accessed, and also the **conflict domain** for identifiers

Variable Declaration Format

- Format

  data_type variable/constant_identifier**;**

- Variables and constants **must** be *declared* before use
  - `int age;`

- Variable names (identifiers)
  - Variable names are composed of the characters:

    a,b,c,..,z,A,B,C,...,Z,0,1,2,...,9 and _

  - Variables names must begin with:

    a,b,c,..,z,A,B,C,...,Z or _

Variable Names

- Capitalized and lower case letters are different
- Examples:
  - `int age;`
  - `int age1, age2, Age;`

Their values are **undefined** at this point

- Optionally, the initial value of a variable can be set with declaration.
  - `int age=18;`
  - `int age1=18, age2=23;`
  - `int age1 = 18,age2 = 23; //Space is okay`

C++ predefined data types

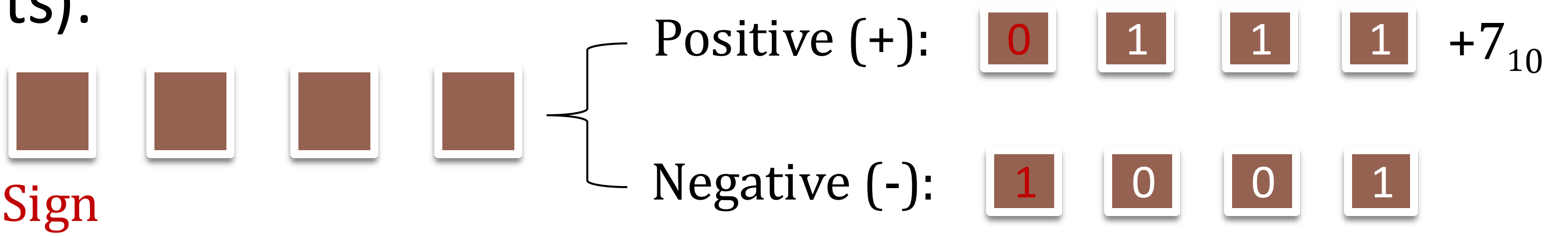
- Numerical
  - int — integer (1, 3, 8 , 3222, 421, 0, -45)
  - float, double — real number (0.25, 6.45, 3.01e-5)

- Character
  - char — single character (‘a’, ‘e’, ‘o’, ‘\n’, ‘\\’, ‘\”’)

- Logical
  - bool — boolean (true, false)

int

- Typically, an int variable is stored in four bytes (1 byte = 8 bits).



- A 32-bit int can store any integer in the range of -2^{31} and $2^{31}-1$, i.e. -2147483648 to 2147483647

$-(2^{31}-1)$ 11111111 11111111 11111111 11111111 - 01111111 11111111 11111111 11111111 $2^{31}-1$

00000000 00000000 00000000 00000000 0

10000000 00000000 00000000 00000000 -2^{31}

- When an int is assigned a value greater than its maximum value, overflow occurs; similarly underflow occurs when a value smaller than the minimum value is assigned. However, C++ does not inform you the errors.

short, long and unsigned

- short, long and unsigned are special data types for integers.
  - `short x;`
  - `long x;`

- short is used for small integers to conserve space (2 bytes).

- long is used for large integers (4/8 bytes).

- unsigned int is of the same size as int (4 bytes) except it assumes the value to be stored is **positive** or **zero**. The **sign bit** can thus be conserved to store a **positive integer** larger than the maximum value of int (which is $2^{31}-1$).

  00000000 00000000 00000000 00000000 - 11111111 11111111 11111111 11111111

- The range of an unsigned int is from 0 to $2^{32}-1$

Two's complement [Optional]

- The way that computers represent integers
  - **For a positive integer, two's complement is the same as the integer**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

Decimal number 11 in memory

  - **For a negative integer, e.g., -11**
    - **Remove the sign: 11**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

    - **Invert the bits (0 goes to 1, and 1 to 0)**

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0

    - **Add 1 to the resulting number**

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

Data type char

- Used to store a single character, enclosed by the single quotation mark
  - `char c = 'a';`
  - `char c = '\n';`
- ASCII codes
  - A character takes **one byte (that is 8 bits, 0 or 1)**
    - 'a' is stored as the following bit pattern **0 1 1 0 0 0 0 1**
    - It is equivalent to an integer 97 (= $\mathbf{2^6 + 2^5 + 2^0}$)
- Characters are (almost the same as) Integers
  - Characters are treated as small integers, and conversely, small integers can be treated as characters.
  - $2^8 = 256$, it can represent up to 256 integers

ASCII Code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	00 0000 0000 NUL	01 0000 0001 SOH	02 0000 0010 STX	03 0000 0011 ETX	04 0000 0100 EOT	05 0000 0101 ENQ	06 0000 0110 ACK	07 0000 0111 BEL	08 0000 1000 BS	09 0000 1001 HT	10 0000 1010 LF	11 0000 1011 VT	12 0000 1100 FF	13 0000 1101 CR	14 0000 1110 SO	15 0000 1111 SI	8
1	16 0001 0000 DLE	17 0001 0001 DC1	18 0001 0010 DC2	19 0001 0011 DC3	20 0001 0100 DC4	21 0001 0101 NAK	22 0001 0110 SYN	23 0001 0111 ETB	24 0001 1000 CAN	25 0001 1001 EM	26 0001 1010 SUB	27 0001 1011 ESC	28 0001 1100 FS	29 0001 1101 GS	30 0001 1110 RS	31 0001 1111 US	9
2	32 0010 0000 SP	33 0010 0001 !	34 0010 0010 "	35 0010 0011 #	36 0010 0100 $	37 0010 0101 %	38 0010 0110 &	39 0010 0111 '	40 0010 1000 (	41 0010 1001)	42 0010 1010 *	43 0010 1011 +	44 0010 1100 ,	45 0010 1101 -	46 0010 1110 .	47 0010 1111 /	A
3	48 0011 0000 0	49 0011 0001 1	50 0011 0010 2	51 0011 0011 3	52 0011 0100 4	53 0011 0101 5	54 0011 0110 6	55 0011 0111 7	56 0011 1000 8	57 0011 1001 9	58 0011 1010 :	59 0011 1011 ;	60 0011 1100 <	61 0011 1101 =	62 0011 1110 >	63 0011 1111 ?	B
4	64 0100 0000 @	65 0100 0001 A	66 0100 0010 B	67 0100 0011 C	68 0100 0100 D	69 0100 0101 E	70 0100 0110 F	71 0100 0111 G	72 0100 1000 H	73 0100 1001 I	74 0100 1010 J	75 0100 1011 K	76 0100 1100 L	77 0100 1101 M	78 0100 1110 N	79 0100 1111 O	C
5	80 0101 0000 P	81 0101 0001 Q	82 0101 0010 R	83 0101 0011 S	84 0101 0100 T	85 0101 0101 U	86 0101 0110 V	87 0101 0111 W	88 0101 1000 X	89 0101 1001 Y	90 0101 1010 Z	91 0101 1011 [	92 0101 1100 \	93 0101 1101]	94 0101 1110 ^	95 0101 1111 _	D
6	96 0110 0000 `	97 0110 0001 a	98 0110 0010 b	99 0110 0011 c	100 0110 0100 d	101 0110 0101 e	102 0110 0110 f	103 0110 0111 g	104 0110 1000 h	105 0110 1001 i	106 0110 1010 j	107 0110 1011 k	108 0110 1100 l	109 0110 1101 m	110 0110 1110 n	111 0110 1111 o	E
7	112 0111 0000 p	113 0111 0001 q	114 0111 0010 r	115 0111 0011 s	116 0111 0100 t	117 0111 0101 u	118 0111 0110 v	119 0111 0111 w	120 0111 1000 x	121 0111 1001 y	122 0111 1010 z	123 0111 1011 {	124 0111 1100 |	125 0111 1101 }	126 0111 1110 ~	127 0111 1111 DEL	F

char as integers

- Any integer expression can be applied to char type variables
  - `char c = 'a'; //c is 97`
  - `c = c + 1; // c becomes to 98`
  - `cout << "variable c+1 is the character " << c;`
  - The output will be: "`variable c+1 is the character b`"

Code: lec01-04-char-string-sizeof.cpp

String

- A string is a sequence of characters.
  - A string is treated as an *array* of characters. We call it cstring

- Strings are delimited by double quotation marks "", and the identifier must be followed with [] or begin with *
  - `char lecture[] = "CS2310 Lecture";` or
  - `char * lecture = "CS2310 Lecture";`

  - `char lecture[] = "C";` vs. `char lecture = 'C';`

- How to display: "hello"
  - Remember escape sequences?
  - `char name[] = "\"hello\"";`

Floating types

- Represent real numbers using the floating-point representation.
  - `float height;`
  - `double weight = 120.82;`

- float uses less memory (4 bytes), but is less accurate (7 digits after decimal point); double uses more memory (8 bytes) but more accurate (15 digits after decimal point)

- We use double most of the time. It's also the default type for floating numbers in C++.

- Exponent representation is also acceptable, e.g., **1.23e2** (which is **1.23 x 10**^{2}) and **3.367e-4** (which is **3.367 x 10**$^{-4}$)
  - `double weight = 1.23e2;//double weight = 123.0`

The sizeof operator

- sizeof can be used to find the *number of bytes needed to store an object* (which can be a **variable** or a **data type**);

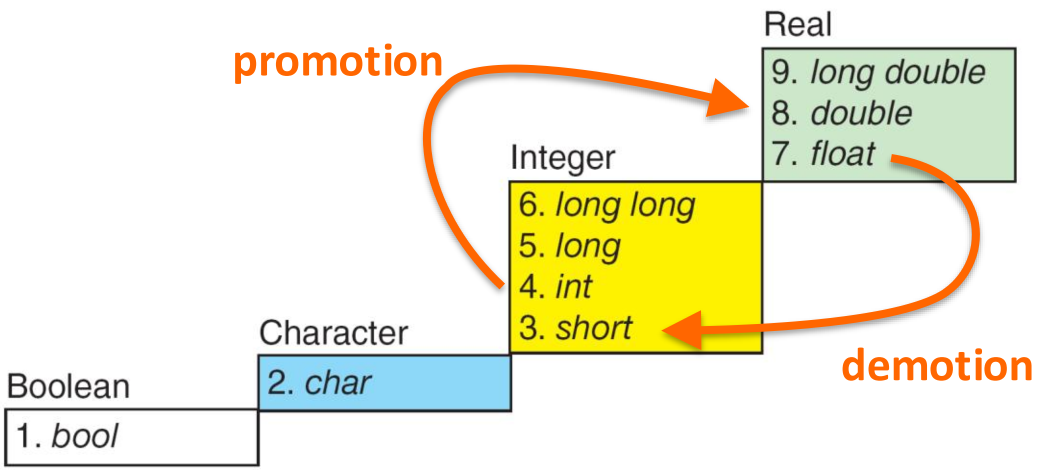
- Its result is typically returned as an unsigned integer, e.g.,

```
int length1, length2;
double x;
length1 = sizeof(int);
cout<<length1<<endl;
//same as length1 = sizeof(length1);
//or      length1 = sizeof(length2);
length2 = sizeof(x);//same as sizeof(double)
cout<<length2<<endl;
```

Data type conversion

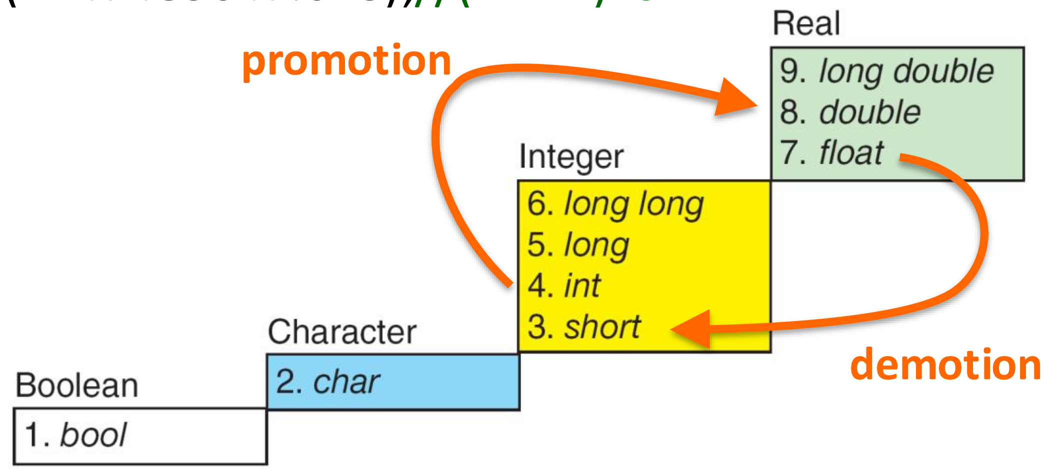
- Arithmetic conversions occur if necessary for the operands of a **binary operator**

- A char can be used in any expression where an int may be used, e.g., ‘a’ + 1 is equal to 97 + 1



Data type conversion

- Implicit type conversion
  - Binary expressions (e.g. **`x + y`**): lower-ranked operand is promoted to higher-ranked operand. //int x = 1; double y = 2.2;
  - Assignment (e.g. **`x = y`**): right operand is promoted/demoted to match the variable type on the left, e.g., int **x**=1.8; //x will be integer 1.
- Explicit type conversion (type-casting)
  - Example: **`int i = 10; double j=(double) i;`**
  - Demoted values might change or become invalid
  - E.g., int b=(int)(2147483647.0*3);//(2^{31} -1)*3



Constants

- Everything we covered before for variables can be applied to constants
  - type, name, scope
- Declaration format:
  - data_type variable/constant_identifier = value;
  - const data_type variable/constant_identifier = value;
- Examples:
  - `const float PI = 3.14159;`
  - `const int MAXVALUE = 255;`
  - `const char INITIAL = ‘D’;`
  - `const char STUDENT_NAME[] = ”Andy Lau”;`

Outline for Today

- Programming languages

- Building a C++ program
- Simple program

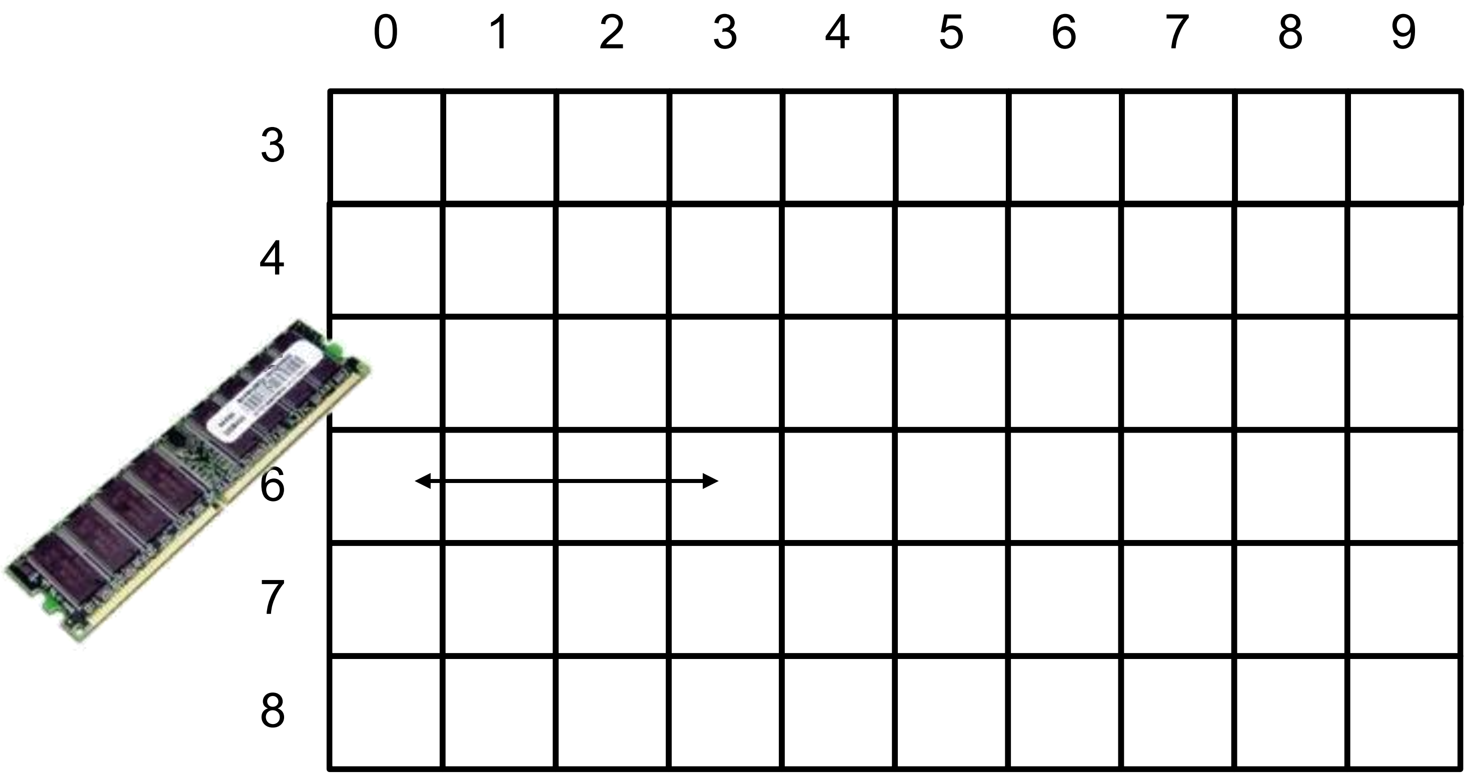
- Variables and constants
  - Name
  - Type
  - Address
  - Scope

Memory and Variable

- Variable is used to store data that will be accessed by a program on execution

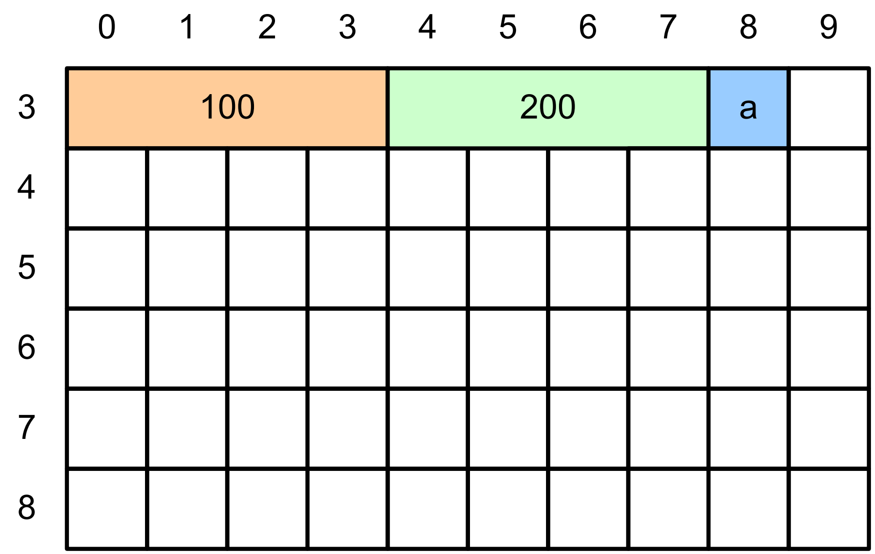
- Normally, variable will be stored in the **main memory**

Main Memory



Variable and Memory

```
void main(){
        int x;
        int y;
        char c;
        x=100;
        y=200;
        c=`a`;

}
```



Identifier	Value	Address
x	100	30
y	200	34
c	‘a’	38

Variable and Memory

- ***Most of the time***, the computer allocates adjacent memory locations for variables declared one after the other
- A variable's address is the first byte occupied by the variable
- Address of a variable depends on the computer, and is usually in hexadecimal (base 16 with values 0-9 and A-F).
  - e.g. 0x00023AF0, 00023AF0

Pointers

- In C++, a pointer is a **variable** which designs to store the address of another **variable**. When a pointer store the address of a variable, we said the pointer is pointing to the variable

- Pointer, like normal variable has a type, its type is determined by the type of variable it points to

Variable type	int x;	float x;	double x;	char x;
Pointer type	int*Pointx;	float*Pointx;	double*Pointx;	char*Pointx;

```
int *Pointx;
int* Pointx;
```

* and & operator

- To declare a pointer variable, place a "*" sign before an identifier:
  - `char *cPtr;      //a character pointer`
  - `int *nPtr;       //a integer pointer`
  - `float *fp;       //a floating point pointer`

- To retrieve the address of a variable, use the "**&**" operator:
  - `int x;`
  - `nPtr=&x;`

Address of variable x

- To access the variable a pointer pointing to, use "*" operator (dereference)
  - `*nPtr=10;//x=10`

  - `int y;`
  - `y=*nPtr; //y=x`

reference vs. **dereference**

& *****

Code: lec01-05-pointer-declare.cpp

Example

```
int x,y;                //x and y are integer variables
void main(){
  int *p1,*p2;          /*p1 and p2 are pointers of
                               integer typed      */
  x=10;
  y=12;
  p1=&x;                /* p1 stores the address of
                               variable x    */
  p2=&y;                /* p2 stores the address of
                               variable y    */
  *p1=5;                /* p1 value unchanged but x is
                               updated to 5 */
  *p2=*p1+10;           /*what are the values of p2 and
                               y?     */
}
```

Common operations

- Set a pointer *p1* point to a variable *x*
  - *p1* = &x;
- Set a pointer *p2* point to the variable pointed by another pointer *p1*
  - *p2* = *p1*
- Update the value of variable pointed by a pointer
  - **p2* = 10;
- Retrieve the value of variable pointed by a pointer
  - int y = **p2*;

Summary

- * operator will give the **value** of pointing variable (so that you can ***indirectly*** update/modify the pointing variable)
  - E.g., int x; int*p=&x; then using "*p" is equal to "x";

- & operator will give the **address** of a variable

Exercise: what are the errors?

```
int x=3;
char c='a';
char *ptr;
ptr=&x;

ptr=c;

ptr=&c;
```

Exercise: What is the output?

```
int num=100;
int *ptr1;
ptr1=&num;
*ptr1=40;
cout << num;
```

Outline for Today

- Programming languages
- Building a C++ program
- Simple program
- Variables and constants
  - Name
  - Type
  - Address
  - Scope

Variable Scope – Local vs. Global

- Scope of a variable refers to the **accessibility/visibility** boundary of a variable
  - We need to be able to "see" a variable in order to access it

- Local variables
  - Declared in a block {} and can be only accessed within the block
  - Try to access a local variable outside the block will produce unpredictable result

- Global variable
  - Defined in the global declaration sections of a program, e.g., defined outside a function block.
  - Can be seen and accessed by all functions after declaration

Global and local variables

- The local variable makes the global variable with the same name out-of-scope inside the function – **it “hides” the global variable**
- The same applies to local variables with **different scopes**

```
void main(){
        int x = 11;
        cout << x << ".\n";
        {
                int x = 10;
                cout << x << ".\n";
        }
        cout << x << ".\n";
}
```


Code: lec01-06-local-gloabal.cpp

Scope and namespace

- A scope can be defined in many ways: by {}, functions, classes, and namespaces
- Namespace is used to explicitly define the scope. A namespace can only be defined in global or namespace scope.
- The **scope operator ::** is used to resolve scope for variables of the same name.

Scope and namespace

```
int a = 90; // this a is defined in global namespace
namespace level1 {
    int a = 0;
   namespace level2 {
        int a = 1;
   }
}
```

- Inside the main function, we can then resolve the variable's scope

```
// :: resolves to global namespace
cout << ::a << "\n";
cout << level1::a << "\n";
cout << level1::level2::a << "\n";
```

Simple Program

```
#include <iostream>
using namespace std;
void main()
{
  cout << "Hello, world!\n”;
}
```

```
namespace std {
    ostream cout;
    istream cin;
    //…
}
```

```
#include <iostream>

void main()
{
  std::cout << "Hello, world!\n”;
  // :: resolves to std namespace
}
```