Advanced Programming
Topic 1: C++ Basics

Nguyen, Thien Binh
Mechatronics and Sensor Systems Technology
Vietnamese-German University

21 January 2024

Vietnamese - German University

Outline

1. Why learning C++?
2. How to run a C++ program?
3. The first C++ program
4. C++ basics

Vietnamese - German University

1. Why learning C++?

Some Applications

Question: What are these software applications used for? What do they all have in common?

Some Applications

Question: What are these software applications used for? What do they all have in common?

- All these applications are written or partially written in C++.
- There are many more platforms, tools, open-source packages, projects, etc. employed in C++
  - https://www.mentofacturing.com/vincent/implementations.html
  - http://www.stroustrup.com/applications.html
  - https://en.cppreference.com/w/cpp/links/libs

What is C++?

C++

- is a *general-purpose*, *case-sensitive*, *free-form*, *high-level* programming language
- supports *procedural* (function-based), *object-oriented* (class-based), and *generic* (interface-based) programming approaches
- can be considered as a *superset* of the procedural C language since almost all C programs can be run as C++ programs

Why C++?

Question: Why learning C++ then?

- C++ has a vast ecosystem of applications, shared libraries, and tools
- C++ is consistently one of the most popular programming languages (ranked #4 (04/2019), https://www.tiobe.com/tiobe-index/)
- C++ is both powerful and dangerous since it grants programmers lots of freedom to get access to low-level hardware resources, e.g., memory allocation. This makes C++ one of the fastest programming languages
- C++ is object-oriented, which allows code modularity, re-usability, flexibility, debugging, and maintenance
- It is adaptable to other languages, e.g., Python, Java, once mastering C++

VGU Vietnamese - German University

2. How to run a C++ program?

How to run a C++ program?

- The input C++ source code: HelloWorld.cpp

```
/* Block comment:
* The first C++ code
* Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

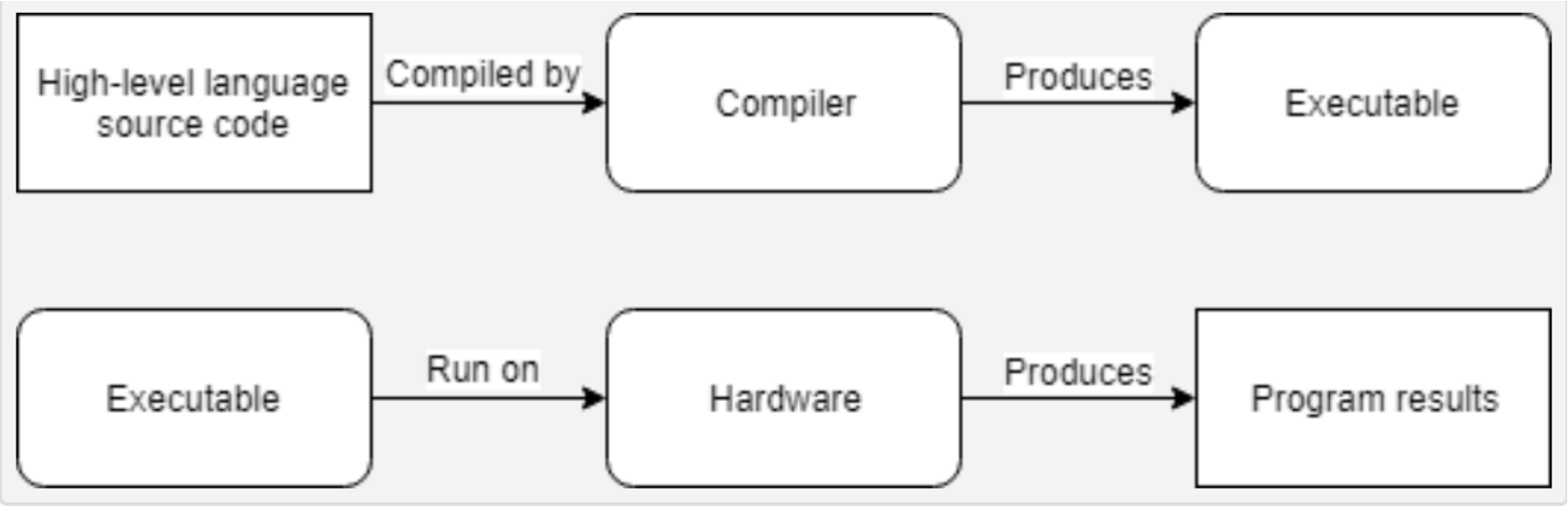
Listing 1: HelloWorld.cpp

- The output result: the world `Hello World!` is printed to the screen

```
Hello World!
```

How to run a C++ program?

- Recall: Source code → Compilation → Linking → Executable → Running on a console



(*source: learncpp* https://www.learncpp.com/cpp-tutorial/introduction-to-programming-languages/)

How to run a C++ program?

- Procedure steps:
  1. Step 1 (*Input*): C++ source file `HelloWorld.cpp`
  2. Step 2: Compilation: object file `HelloWorld.o`
  3. Step 3: Linking: executable, e.g., `aaa.exe` (for Windows), `aaa.out` (for Linux/Unix)
  4. Step 4: (*Output*) Running the exe on a console: `Hello World!` is printed out

How to run a C++ program?

Question: What ingredients are needed for each aforementioned step?

How to run a C++ program?

Question: What ingredients are needed for each aforementioned step?

- An text editor to write C++ source files.
  - Any text editor is possible, e.g., *notepad*, *Codelite* (https://codelite.org/), *Notepad++* (https://notepad-plus-plus.org/) (Windows), *emacs*, *nano*, *vim* (Linux/Unix)
- A C++ compiler to compile source files into object files, and link them to make an executable.
  - Example: *g++* (for Linux/Unix), *minGW* (for Windows)
- Debuggers for tracing and correcting compile-time and run-time errors. This process is called *debugging*.
  - Example: *gdb* (for debugging mostly compile-time errors), *Valgrind* (for debugging memory-related run-time errors)
- A console for inputting parameters / outputting results
  - Example: *Windows PowerShell* (for Windows), *terminal* (for Linux/Unix)

VGU Vietnamese - German University

How to run a C++ program?

Question: What ingredients are needed for each aforementioned step?

- Using an IDE (Integrated Developer Environment): contains all the ingredients discussed above
  - Visual Studio Community 2019: for Windows and MacOS https://visualstudio.microsoft.com/vs/ (recommended in this course)
  - Code::Blocks: cross-platform (Windows, MacOS, Linux) http://www.codeblocks.org/downloads
- Using a Makefile: a manual way to compile source files and link object files following step 1 to 4
  - More freedom and compiling options
  - No needed time to load heavy IDEs
  - Favorable for Linux-based programming
  - Make for makefile: (http://gnuwin32.sourceforge.net/packages/make.htm)

How to run a C++ program?

Question: How to compile a C++ source file manually?

How to run a C++ program?

Question: How to compile a C++ source file manually?

- 2 steps from C++ source files to an executable:
  - Compilation to get object files
    `g++ -Wall -g -c HelloWorld.cpp`
  - Linking to get the executable
    `g++ -Wall -g -o aaa.exe HelloWorld.o`

How to run a C++ program?

How to compile a C++ source file?

- `g++` is the GNU C++ compiler. To check if g++ is successfully installed, type the following command line in Windows Power Shell
  `g++ --version`

```
PS D:\work\teaching_vgu\MSST_Cpp_Mar18to30\Lectures_sides\Beamer\Lec1\codes> g++ --version
g++.exe (tdm64-1) 5.1.0
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- `-Wall -c -o` are the compiler flags. Common compiler flags include
  - `-Wall`: to turn on all compilation warning messages
  - `-c`: to compile a `.cpp` source file to a `.o` object file
  - `-o`: to link the object files to make an executable, e.g., `aaa.exe`
  - `-g`: to turn on the debugger
  - `-Werror`: to convert all warnings into error messages (strict!)
  - `-O`: to compile with optimization, i.e., compilation takes longer time, but the execution time is much reduced.

VGU Vietnamese - German University

How to run a C++ program?

How to compile a C++ source file?

- Compiling with a `makefile`:
  - For `makefile`, indentation is critical
  - To compile, simply type `make -f [makefile_name]`

```
CC       = g++
CFLAGS   = -g -Wall
LDFLAGS  =
OBJS     = HelloWorld.o
TARGET   = aaa

all: $(TARGET)

$(TARGET): $(OBJS)
  $(CC) $(CFLAGS) -o $(TARGET) $(OBJS) $(LDFLAGS)

HelloWorld.o: HelloWorld.cpp
  $(CC) $(CFLAGS) -c -o HelloWorld.o HelloWorld.cpp $(LDFLAGS)

clean:
  rm -f $(OBJS)
```

Listing 2: makefile for HelloWorld.cpp

Outline

4. The first C++ program

The First C++ Program

```
/* Block comment:
* The first C++ code
* Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

Listing 3: HelloWorld.cpp

The First C++ Program

```
/* Block comment:
 * The first C++ code
 * Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

Block comment

Line comment

- Multiple-line comments (in between /*...*/), single-line comments (after //)
- Comment lines are ignored by the compiler, served for programmers' personal uses, e.g., explanations of code segments, notifications.

The First C++ Program

```
/* Block comment:
 * The first C++ code
 * Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

Preprocessor directive

- Including contents of header file `iostream`, which is a C++ standard library for input/output streaming, e.g., `cout <<` (print the data to a standard console) and `cin >>` (read data from a keyboard).

The First C++ Program

Common C++ header files:

- `iostream` standard stream objects, e.g., `cout` cout and `cin`
- `fstream` if you want to read and write files
- `cstdlib` for general operations, e.g., dynamical memory allocation, random numbers, etc.
- `cassert` for assertion commands
- `cmath` common mathematics functions
- Others: `https://en.cppreference.com/w/cpp/header`

Vietnamese - German University

The First C++ Program

```
/* Block comment:
 * The first C++ code
 * Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

main program

- `main` function in C++, which always runs first. Every C++ program must have a `main` function, otherwise it will fail to compile. The content in-between the curly bracket { } is called the function *body*.

The First C++ Program

```
1  /* Block comment:
2   * The first C++ code
3   * Coded by Thien Binh Nguyen, 28 jan 2019 */
4  #include <iostream>
5  using namespace std;
6  int main()
7  {
8    // line comment: print "Hello World" to the screen
9    cout << "Hello World!" << endl;
10   return 0;
11 }
```

statements

statements

- Statements contain instructions of the program

The First C++ Program

- All statements must be ended with a semi-colon (`;`), otherwise C++ will complain with a syntax error message
- `cout` is to print the string `Hello World` to the screen, `endl` is to break and enter a new line
- `return 0` is a returned type of function `main`, which indicates that the program is run successfully
- Using namespace `std` where function `cout` and `endl` belongs to. If `std` is not specified, `std::cout` and `std::endl` which indicates their scope must be used

The First C++ Program

- Little debugging: What is wrong with the following program?

```
/* Block comment:
* The first C++ code
* Coded by Thien Binh Nguyen, 28 jan 2019 */
using namespace std;
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

Listing 4: HelloWorld_1.cpp

The First C++ Program

- Error message:

```
g++ -g -Wall -c -o ExampleHelloWorld.o ExampleHelloWorld.cpp
ExampleHelloWorld.cpp: In function 'int main()':
ExampleHelloWorld.cpp:10:3: error: 'cout' is not a member of 'std'
   std::cout << std::endl;
   ^
ExampleHelloWorld.cpp:10:16: error: 'endl' is not a member of 'std'
   std::cout << std::endl;
                ^
```

- Debugging: missing `#include ⟨iostream⟩`
  - Library `iostream` defines streaming `cin` and `cout`

The First C++ Program

- Little debugging: What is wrong with the following program?

```
/* Block comment:
* The first C++ code
* Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl;
  return 0;
}
```

Listing 5: HelloWorld_2.cpp

The First C++ Program

- Error message:

```
g++ -g -Wall -c -o ExampleHelloWorld.o ExampleHelloWorld.cpp
ExampleHelloWorld.cpp: In function 'int main()':
ExampleHelloWorld.cpp:11:3: error: 'cout' was not declared in this scope
   cout << endl;
   ^
ExampleHelloWorld.cpp:11:3: note: suggested alternative:
In file included from ExampleHelloWorld.cpp:5:0:
C:/TDM-GCC-64/lib/gcc/x86_64-w64-mingw32/5.1.0/include/c++/iostream:61:18: note:   'std::cout'
   extern ostream cout;  /// Linked to standard output
                  ^
```

- Debugging: missing namespace `std` where `cin`, `cout`, and `endl` are grouped within.

The First C++ Program

- Little debugging: What is wrong with the following program?

```
/* Block comment:
* The first C++ code
* Coded by Thien Binh Nguyen, 28 jan 2019 */
#include <iostream>
using namespace std
int main()
{
  // line comment: print "Hello World" to the screen
  cout << "Hello World!" << endl
  return 0;
}
```

Listing 6: HelloWorld_3.cpp

The First C++ Program

- Error message:

```
g++ -g -Wall -c -o ExampleHelloWorld.o ExampleHelloWorld.cpp
ExampleHelloWorld.cpp:8:1: error: expected ';' before 'int'
 int main()
 ^
ExampleHelloWorld.cpp: In function 'int main()':
ExampleHelloWorld.cpp:13:3: error: expected ';' before 'return'
   return 0;
   ^
```

- Debugging: missing semi-colons at the end of some statements.

The First C++ Program

Compilation note:

- C++ compilers are very strict on code syntax. A program must be absolutely correct in order to be compiled successfully.

Outline

Data Types

```
#include <iostream>
using namespace std;
int main()
{
  int      i, j;          // variables of type integer
  double     x, y, z;         // variables of type double
  i = 10; j = 20;
  x = 5.3; y = 1.0e+2; z = x * y;

  cout.precision(16);        // double precision
  cout << "i = " << i << ", j = " << j << endl;
  cout << "fixed:" << endl;
  cout << fixed;
  cout << "x = " << x << ", y = " << y
      << ", z = " << z << endl;
  cout << "scientific:" << endl;
  cout << scientific;
  cout << "x = " << x << ", y = " << y
      << ", z = " << z << endl;

  return 0;
```

Data Types

- `i`, `i`, `j`, `y`, `z`: variables (to be discussed later)
- `int`, `double`: data types $\Rightarrow$ to indicate how much memory is allocated to store a variable of a data type
- Generally, a memory allocation of size `n` bits can store to 2^n values. For example, a memory of 1 byte (8 bits) can store up to $2^8 = 256$ values.

Data Types

- Data types:
  - `char`: stores characters
  - `int`: stores integer numbers
  - `float`: stores single precision (i.e., 7 decimal digits of precision) floating-point numbers
  - `double`: stores double precision (i.e., 15 decimal digits of precision) floating-point numbers
  - `bool`: stores boolen values, requires 1 byte of memory. In C++, `(bool) TRUE = (int) 1, (bool) FALSE = (int) 0`
- Data modifiers: used before basic data types to modify the their range
  - `signed`: signed values
  - `unsigned`: non-negative values
  - `short`
  - `long`

Vietnamese - German University

Data Types I

Question: How to check the range of a data type??? ⇒ sizeof(data type)

```
#include <iostream>
#include <climits>          // for int, char macros
#include <cfloat>           // for float, double macros
#include <cstddef>          // for size_t type
using namespace std;
int main()
{
   cout << "int range min = " << INT_MIN
        << ", max = " << INT_MAX << endl;
   cout << "size(int) = " << sizeof(int)
        << " byte(s)" << endl;
   cout << "short int range: min = " << SHRT_MIN
        << ", max = " << SHRT_MAX << endl;
   cout << "size(short int) = " << sizeof(short int)
        << " byte(s)" << endl;
   cout << "long int range: min = " << LONG_MIN
```

Vietnamese - German University

Data Types II

```
17          << ", max = " << LONG_MAX << endl;
18      cout << "size(long int) = " << sizeof(long int)
19          << " byte(s)" << endl;
20      cout << "unsigned int range: min = " << 0
21          << ", max = " << UINT_MAX << endl;
22      cout << "size(unsigned int) = " << sizeof(unsigned int)
23          << " byte(s)" << endl;
24
25      return 0;
26  }
```

Listing 8: DataTypes.cpp

Data Types

Exercise: Check the range and data size of the following types.

1. `bool`
2. `signed char`, `unsigned char`
3. `float`, `short float`, `long float`
4. `double`, `short double`, `long double`

Refs:

- https://docs.microsoft.com/en-us/cpp/c-language/cpp-integer-limits?view=vs-2017
- https://docs.microsoft.com/en-us/cpp/cpp/floating-limits?view=vs-2017

typedef

- typedef: to define aliases for a type.
  typedef type ALIAS 1, ALIAS 2;

```
typedef double DISTANCE, VELOCITY, ACCELERATION;
DISTANCE x;          // = double x;
VELOCITY v;            // = double v;
ACCELERATION a;      // = double a;
```

Listing 9: DataTypes.cpp

Operators

Commonly used operators

- Unary: `++a (a=a+1)`, `a++ (a=a+1)`, `--a (a=a-1)`, `a-- (a=a-1)`, `-a`, `&a (address of a)`
- Binary: `a+b`, `a-b`, `a*b`, `a/b`, `i%j (modulus of i/j where i,j are integers)`
- Assignment: `a=b`, `a+=b (a=a+b)`, `a-=b (a=a-b)`, `a*=b (a=a*b)`, `a/=b (a=a/b)`, `i%=j (i=i%j)`
- Relational: `a>=b a<=b`, `a>b`, `a<b`
- Logical: `a==b`, `a!=b`, `a||b (a OR b)`, `a&&b (a AND b)`

Operators

Exercise: What are the results of the following operations?

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

int main()
{
  int      i(4), j(2);
  double     x(6.0), y(2.0);

  cout << "++x = " << ++x << endl;     // prefix operator
  cout << "x = " << x << endl;
  cout << "x++ = " << x++ << endl;     // postfix operator
  cout << "x = " << x << endl;

  cout << "i % j = " << i%j << ", i/j = " << i/j << endl;
  cout << "x/y = " << x/y << ", x/i = "


      << x/i << ", i/x = " << i/x << endl;
  cout << "&x = " << &x << endl;
```

Type Conversion

- Type demotion:

```
int i, j, k;
i = 3.142;          // double truncated to into
j = -3.142;         // double truncated to into
k =  2.997e40;      // Undefined.
cout << "i = " << i << ", j = " << j << endl;
cout << "k = " << k << endl;
```

Type Conversion

- Casts: explicit type conversion

```
double x;
int i = 4;
x = static_cast<double>(i);
```

- Exercise: $x = y$?

```
double x, y;
x = (1.0 + 1) / 2;
y = 0.5 + 1 / 2;
cout << "x = " << x << ", y = " << y << endl;
```

Maths functions

Common maths functions with `<cmath>`:

- Power: `pow, sqrt`, ...
- Exponential: `exp, exp2, log, log10`, ...
- Trigonometric: `sin, cos, tan, asin, acos, atan`, ...
- Rounding: `ceil, floor, round`, ...
- Error functions: `erf, erfc, tgamma, lgamma`, ...

Ref: `http://www.cplusplus.com/reference/cmath/`

Control Structures

- `if` statement: bi-branching

```
if (condition)
  {statements;}
else
  {statements;}
```

- Condition expression operator

```
(condition) ?  result 1 :  result 2;
```

- `switch` statement: multiple branching

```
switch (condition) {
  case 1:  statements; break;
  case 2:  statements; break;
      ⋮
  case n:  statements; break;
  default:  statements; break;
}
```

Control Structures

- `for` loop: finite iterations

```
for (initializing; condition; stepping)
  {statements;}
```

- `while` loop: infinite iterations, `condition` is checked before `statements` are executed

```
while (condition)
  {statements;}
```

- `do` loop: infinite iterations, `statements` are executed before `condition` is checked

```
do
  {statements;}
while (condition);
```

Control Structures I

- Example:

$$I = \sum_{k=0}^{100} k^2 = 338350$$

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  int i, sum;

  // for loop
  sum = 0;
  for (i = 0; i < 101; ++i)
    sum += pow(i, 2);

  // while loop
```

Control Structures II

```
14        i = 0; sum = 0;
15        while (i < 101) {
16          sum += pow(i, 2); ++i;
17        }
18
19        // do loop
20        i = 0; sum = 0;
21        do {
22          sum += pow(i, 2); ++i;
23        } while (i<101);
24
25        cout << "sum␣=␣" << sum << endl;
26        return 0;
27      }
```

Listing 11: Sum.cpp

- What is the result of `sum`?

Control Structures III

```
1     sum = 0;
2     for (int i = 0; i < 10; ++i);
3       sum += 100;
4     cout << "sum = " << sum << endl;
```

- What is the result of `x`?

```
1     x = 10;
2     if (x = 5)
3       x = 50;
4     cout << "x = " << x << endl;
```

Reading

1. Capper, *Introducing C++ for Scientists, Engineers, and Mathematicians*, **Chapters 1 - 4**
2. Pitt-Francis, and Whiteley, *Guide to Scientific Computing in C++*, **Chapters 1 - 2**