Introduction of C++ Programming

Qi Wang
wangqi03@bu.edu

Overview

Syllabus

Introduction of C++

The Compilation Process

Editor

Installation

hello_world.cpp

Requirements

- Attendance
- Participation
- Final Project

Final Project

- Present on May 11th
- Design and implement your own choice of project in C++
- Work in a team of two or three
- Project ideas: Hang Man Game, Tic-Tac-Toe Game, helper apps like to-do lists

Introduction of C++

C is a programming language developed in the 1970's alongside the UNIX operating system.

C provides a comprehensive set of features for handling a wide variety of applications, such as systems development and scientific computation.

C++ is an "extension" of the C language, in that most C programs are also c++ programs.

C++, as opposed to C, supports "object-oriented programming". Easier large scale projects, reusability.

Important Definition

Algorithm: Ordered set of actions to accomplish a certain task

Program: Implementation of algorithms.

Compiler, function, library, bug

Variables, constants

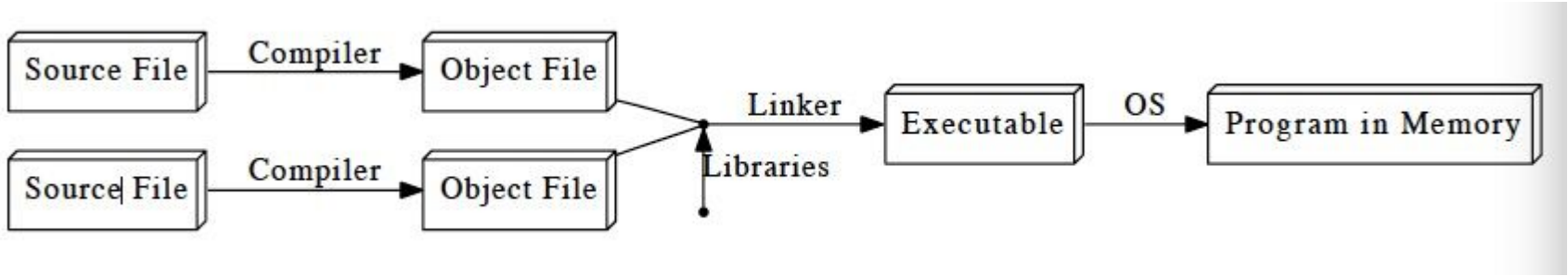
Keywords (if, while, for)

Data Types (long, int)

The Compilation Process

A compiler is a system software that converts source code written in a programming language (source language) into another computer language (target language).

Some other languages compilation process:

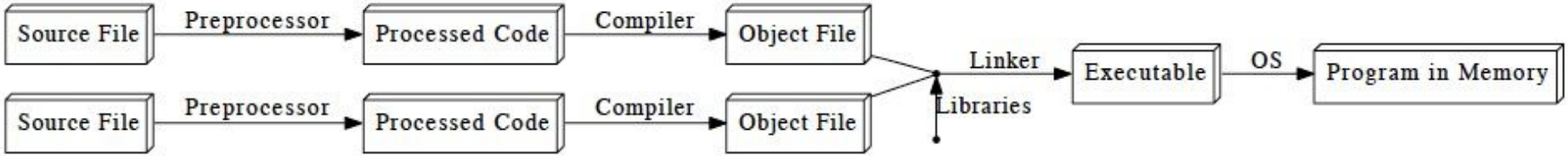


The Compilation Process

Preprocessor applies some modifications to the source code, before being fed to the compiler.

Far faster.

C++ compilation process:



Editor

- Extension: .cpp(default), .cp or .c

- Editors: text editor, xemacs, emacs, pico, vim, sublime

Installation

- Linux or UNIX: check if have gcc installed `g++ -v`

  Install: https://gcc.gnu.org/install

- Mac OS X: check if have gcc installed `g++ -v`

  Xcode Install: developer.apple.com/technologies/tools/

- Windows: check if have gcc installed `gcc`

  MinGW Install: www.mingw.org you must install gcc-core, gcc-g++, binutils, and the MinGW runtime

  https://www.youtube.com/watch?v=DHekr3EtDOA

hello_world.cpp

```
// A Hello World program

#include <iostream>

int main() {

    Std :: cout << “Hello, world!\n”;

    return 0;

}
```

Compiling with C++: g++

- To compile a file to an executable called hello:

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

- To run the executable output:

```
./hello
```

- To compile a file with warnings:

```
g++ hello_world.cpp -Wall -o hello
```

Explanation

1. `//`: indicates that everything following it until the end of the line is a comment: it is ignored by the compiler.

   Another way to comment may span multiple lines: `/* ..... */`

2. `#`: preprocessor commands, change what code is actually being compiled, `#include` tells the preprocessor to dump in the contents of another file, here the `iostream` file, which defines the procedures for input/output.
3. `int main() { …}` defines the code that should execute when the program starts up.
4. a. `cout <<`: syntax for outputting some piece of text to the screen.

Explanation

b. Namespaces: identifiers defined within a context. When we want to access an identifier defined in a namespace, we tell the compiler to look for it in that namespace using the *scope resolution operator (::)*. Here, we're telling the compiler to look for cout in the std namespace, in which many standard C++ identifiers are defined. A cleaner alternative is to add the following line below line 2:

```
using namespace std;
```

If we do this, we can omit the std:: prefix when writing cout. Recommend

Explanation

c. Strings: A sequence of characters.

d. Escape sequences: `\n` indicates a new line character.

`\a`: System bell (beep sound)

`\b`: Backspace

`\f`: Formfeed (page break)

`\n`: Newline (line break)

`\r`: “Carriage return” (returns cursor to start of line)

`\t`: Tab

`\\`: Backslash

`\'`: Single quote character

`\”`: Double quote character

`\int x`: x represented character

Explanation

7. `return 0`: the program should tell the operating system it has completed successfully. Include it as the last line in the `main` block.

Note: every statement ends with a semicolon (except preprocessor commands and blocks using `{}`).

Values and Statements

Statement: a unit of code that does something - a basic building block of a program.

Expression: a statement that has a value - for instance, a number, a string, the sum of two numbers, etc. `4 + 2`, `x - 1`, and "`Hello, world!\n`" are all expressions.

Not every statement is an expression.

Operators

Operators act on expressions to form a new expression. Replace "`Hello, world!\n`" with `(4 + 2)/3`, the program would print the number `2`. The `+` operator acts on the expressions `4` and `2`.

Types:

- Mathematical: +, -, *, /, % and parentheses.
- Logical: and, or and so on.
- Bitwise: used to manipulate the binary representations of numbers.

Data Types

Every expression has a type – a formal description of what kind of data its value is.

Type Names	Description	Size	Range
`char`	Single text character or small integer. Indicated with single quotes (`'a'`, `'3'`).	1 byte	signed: -128 to 127 unsigned: 0 to 255
`int`	Larger integer.	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean (true/false). Indicated with the keywords `true` and `false`.	1 byte	Just `true` (1) or `false` (0).
`double`	"Doubly" precise floating point number.	8 bytes	+/- 1.7e +/- 308 (15 digits)

Variables

We give a value a name so we can refer to it later using *variables*.

A variable is a named location in memory.

The name of a variable is an identifier token. It may contain numbers, letters, and underscores(_), and may not start with a number.

Variables Example

```
#include <iostream>

using namespace std;

int main() {
    int x;                  // the declaration of the variable x

    X = 4 + 2;    // the initialization of x, initial value

    cout << x / 3 << ' ' << x * 2;

    return 0;

}
```

Variables

A single statement that does both declaration and initialization

```
int X = 4 + 2;
```

Data Types

A data type is a template for

- how a particular set of values is represented in memory, and
- what operations can be performed on those values.

In C++ a *type* is the same as a *class*.

There are

- predefined data types
- system-defined types
- user-defined types

Data Types

Predefined data types are part of the C++ language definition.

- Examples: float, double - real. int - integer. char
- We denote char literals with single quotes, for example: ‘A’ ‘*’ ‘2’
- A string literal is a sequence of characters in double quotes:
- “ABCDE”
- “127” (not the same as int 127)
- “true” (not the same as bool true)

Data Types

System-defined types - part of the C++ class libraries. Not part of the original C++ language definition but added when the compiler is written.

- The standard I/O stream objects cin and cout are defined in iostream library
- Also there is a string class (type) and classes for input and output files.
- To declare an output file: ofstream cprint (“file.txt”);

User-defined types - e.g., enum type, classes

Declarations

Declarations inform the compiler that it will need to set aside space in memory to hold an object of a particular type (class) with a particular name.

Constant declarations

- Used to associate meaningful names with constants -- items that will never change throughout the execution of the program.
- One convention is to use all uppercase letters for constant identifiers.

```
const float PI=3.14159;
const float METERS_TO_YARDS=1.196;
```

Declarations

Variable declarations:

- Used to associate identifiers of a given type with memory cells used to store values of this type. - the values stored in the data cells are changeable.

```
char letter;
char letter1, letter2;
float x, y;
```

Data Types

Primitive Data Types:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

Input

Outputting values: `cout <<`

Inputting values: `cin >>`

Input

```
#include <iostream>

using namespace std;

int main() {
    int x;

    cin >> x;

    cout << x / 3 << ' ' << x * 2;

    Return 0;

}
```

Debugging

Compilation errors: problems raised by the compiler, generally resulting from violations of the syntax rules or misuse of types.

Runtime errors: problems that you only spot when you run the program: you did specify a legal program, but it doesn’t do what you wanted it to. More tricky to catch.

Questions?