Programming Languages, C++ Basics

CSE100

Sections 1.3, 1.4, 2.1-2.3, 2.16

From Last Time

- Program - a set of instructions that tell the computer how to solve a particular problem or perform a specific task
- We saw the basic parts of a computer system
- We saw how a computer runs a program at a low level (loaded in RAM, Fetch/Decode/Execute by CPU)

Today we will talk about how we write programs and convert those to binary instructions

PROGRAMS IN MAIN MEMORY

- All programs must be **loaded** into main memory before they can be executed
- All data must be **loaded** into main memory before it can be manipulated
- Example of a program in main memory

```
100100 010001    //Load rates
100100 010011    //Load hours
100110 010010    //Multiply
100010 010011    //Store the results in wages
```

PROGRAMMING LANGUAGES

- The language a computer program is written in
- Programming Language Evolution
- Machine Languages
- Low level languages
- High-level programming languages

Machine Language

- Machine language: language of a computer. The only instructions a computer understands.
- Digital signals are sequences of 0s and 1s, stored in bits and bytes
- Every different type of CPU has its own machine language
- Difficult to read and write

PROGRAMMING LANGUAGE EVOLUTION

- Early computers were programmed in machine language (Punch card)

- The program to calculate `wages = rates * hours` in an example machine language

```
100100 010001    //Load rates
100100 010011    //Load hours
100110 010010    //Multiply
100010 010011    //Store the results in wages
```

Low-level Languages

- *Low-level languages* are those close to machine language but use words instead of bits for each instruction
- Still have to worry about memory locations
- Still tied to a specific CPU
- Examples include various types of Assembly

```
lw $s1.100($s3)   // load rates
lw $s2.160($s3)  // load hours
mult $s2, $s3      // multiply numbers
mfhi $s4           // store result in wages
```

Another example: http://en.wikipedia.org/wiki/Low-level_programming_language

HIGH-LEVEL LANGUAGES

- *High-level languages* are those that resemble natural languages more than machine
- They abstract away machine concepts like having to deal with memory
- Multiple steps can be written in one instruction.
- Include Basic, FORTRAN, COBOL, Pascal, C++, C, Java, Python, Ruby, JavaScript, etc.
- The equation `wages = rate • hours` can be written in C++ as:

```
wages = rate * hours;
```

C++ History

- Development started by Bjarne Stroustrup in 1979
- An expansion of the older language C
- Retains features of C
- A standardized language
- Continues to evolve adding new features
- Latest standard 2014

Why C++

- One of the most widely used languages
  - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
  - http://www.stroustrup.com/applications.html
- One of the fastest high level languages
  - http://shootout.alioth.debian.org/u64q/which-programming-languages-are-fastest.php
- Has access to low level features
  - Makes it powerful, but less safe
- Multiple Paradigm: Procedural, OO, functional, generic
- *Portability*- Can write a program once and compile it on almost any machine (except for some OS graphic specific programs)

First C++ Program

```
//Sample C++ program
#include <iostream>
using namespace std;

int main()
{
   cout << "Hello, there!";
   return 0;
}
```

From a High-level Program to an Executable File

To run a high level program we have to translate it back down to machine language.

a. Create file containing the program with a text editor.
   - Called the *source code*. Saved in *source file* (.cpp)

b. Run preprocessor - Looks for symbols starting with #. Amends or processes the source code
   - example: #include <iostream>
     - Copies the file needed for cout to the source code

c. Run compiler - Converts source program statements into machine instructions.
   - Creates *object code* (in machine language), in an *object* (.obj) file

From a High-level Program to an Executable File

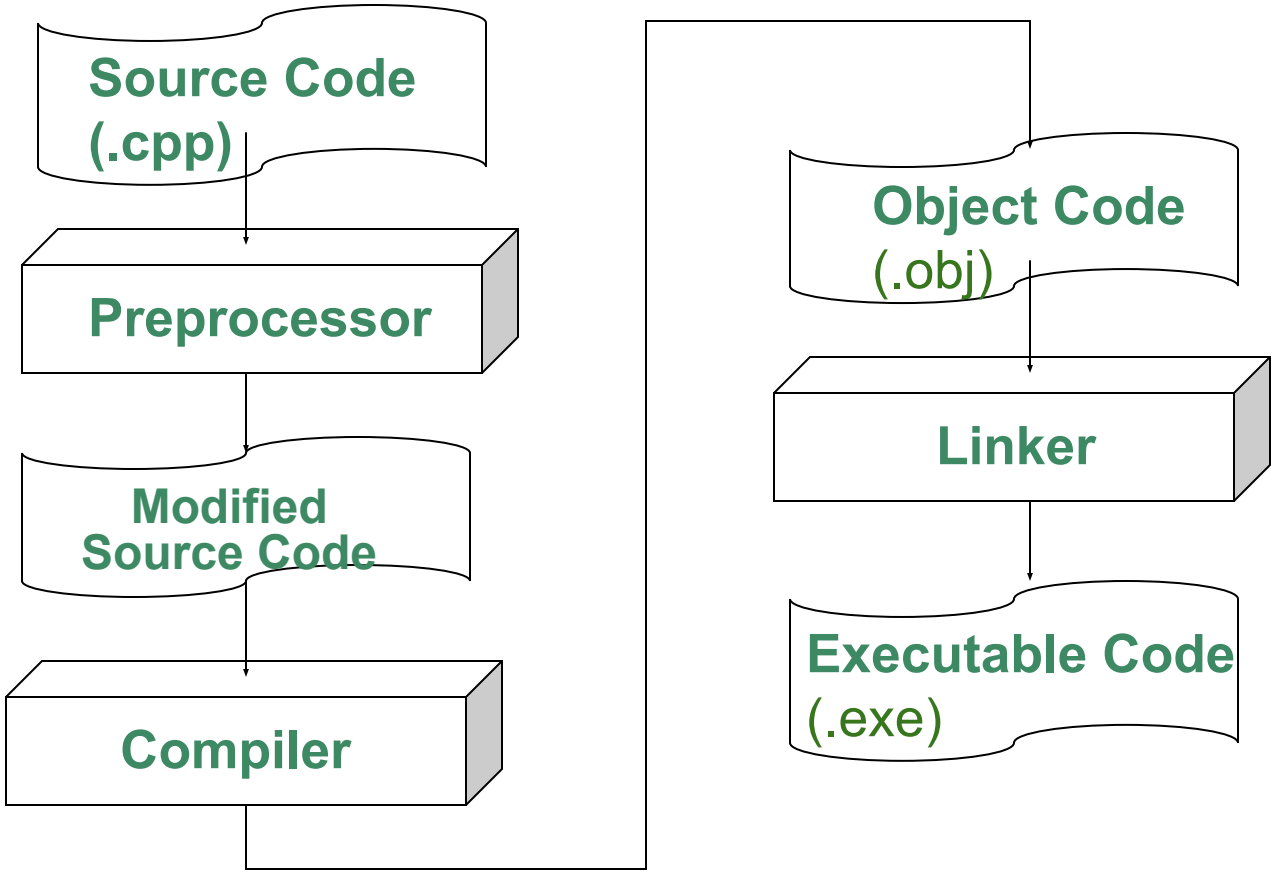
d.Run linker - Connects hardware-specific library code (called *run-time library*) to machine instructions, producing an *executable file* (.exe).

Steps b)–d) are often performed by a single command or button click.

Errors detected at any step will prevent execution of the following steps.

Every change to the source file requires going through all the steps again.

From a High-level Program to an Executable File



Our First C++ program

```
// sample C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, there!";
    return 0;
}
```

What Is a Program Made Of?

Common elements in programming languages:

- Key Words
- Programmer-Defined Symbols
- Constants
- Data Types
- Operators
- Special Characters
- Syntax

Key Words

- Also known as reserved words
- Make up the core of the C++ language
- Have a special meaning in C++
- Cannot be used for another purpose
- Written using lowercase letters
- Examples in program:

```
using, namespace, int, return
```

- All keywords:
  - http://en.cppreference.com/w/cpp/keyword

Programmer Defined Identifiers

- Made up by programmers
- Not part of the core C++ language
- Used for variable names and function names
- Examples from the program:
  - main, std, cout

Constants

- Also called literals
- These are values that cannot be changed during the programs execution
- A number or string occurring in a program
- Examples from the program:
  - `"Hello, there!"`
  - 0

Special Characters, Punctuation

Character	Name	Description
`//`	Double Slash	Begins a comment
`#`	Pound Sign	Begins preprocessor directive
`< >`	Open, Close Brackets	Encloses filename used in `#include` directive
`( )`	Open, Close Parentheses	Used when naming function
`{ }`	Open, Close Braces	Encloses a group of statements
`" "`	Open, Close Quote Marks	Encloses string of characters
`;`	Semicolon	Ends most programming statements

Syntax

- Rules that must be followed when constructing a program.
- Syntax dictates how keywords and operators may be used, and where punctuation symbols must appear.
  - Most statements end in a semicolon.
  - Ex. cout must be followed by << and a variable or constant and end in a semicolon.
- Syntax errors are found by the compiler.
- As opposed to semantics which is the meaning of the language.

Lines and Statements

- A *line* is any line that appears in a program.
  - Most lines contain meaningful code
  - But lines can be empty
  - Why is it a good idea to include empty lines?

- A *statement* is a complete instruction that causes the computer to perform some action
  - cout << "Hello, there!";
  - usually written on one line in a program
  - BUT statements can run more than one line.

Comments

- C++ has two types of comments
- Single Line Comments
  - Begin with // and run until the end of the line
  - Can come after code on a line
    - cout << "hi"; // This prints out hi
- Multi-Line Comments
  - Begin with /* and ends with */
  - Everything inside the two is a comment no matter how many lines it covers.
- Ignored by the compiler
- Very important to properly document code so that you and others understand the source code

Comments Example

```
/*
   PROGRAM: PAYROLL.CPP
   Written by Herbert Dorfmann
   This program calculates company payroll
   Last modified: 8/20/2006
*/
#include <iostream>
using namespace std;

int main()
{
   int employeeID;   // Employee ID number
   double payRate;   // Employees hourly pay rate
   double hours;     // Hours employee worked this week

   (The remainder of this program is left out.)
```

Our First C++ program

```
// sample C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, there!";
    return 0;
}
```

comment

preprocessor directive

which namespace to use

beginning of function named `main`

beginning of block for `main`

output statement

send `0` back to operating system

end of block for `main`

Preprocessor Directives

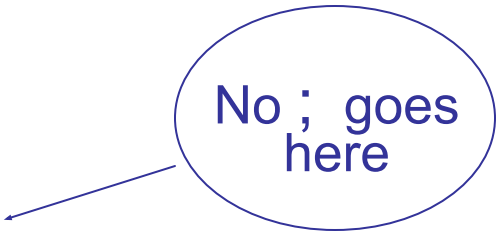
- All preprocessor directives begin with the # character
- These are not C++ statements and do not end with the semicolon
- Are not seen by the compiler. The preprocessor uses them to modify the source code in some way

The `#include` Directive

- Inserts the contents of another file into the program

- Example:

```
#include <iostream>
```



  - Needed to use cout
  - The <> mean that iostream is part of the standard library of files C++ can use
  - iostream stands for *input-output-stream*

- Used with *header files*, so goes at the top of source file

using namespace std;

- Needed for cout and many other standard library functions
- Every object in the iostream file is defined in the namespace std
- *Namespaces* are special sections of memory where the variables in the namespace are defined
  - Used to avoid overlapping variable names
- Used as a convenience
  - otherwise we would have to write std::cout everywhere we want to use cout

int main(){...return 0;}

- Every program must have a main function
- Tells the computer where to start executing the program
- The () define main as a function
- It returns a type int (hence the return 0; at the end)
  - The 0 is used to tell the operating system the program exited successfully
- The body of the function goes between the {}
- The computer starts running the program at the first statement inside the {
- The definition of the main function is complete at the }

The **`cout`** Object (Basics)

- `cout << "Hello, there!";`
- Displays information on computer screen (stands for console output)
- Defined in the **<iostream>** library.
  - And in particular in the std namespace of iostream
- **<<**
  - sends information to cout
  - Called the *stream-insertion operator*
  - Must be followed by a constant or variable

cout Example

```
cout << "Hello, there!";
```

- Can use << to send multiple items to cout

```
cout << "Hello, " << "there!";
```

Or

```
cout << "Hello, ";
cout << "there!";
```

Or even

```
cout << "Hello, "
     << "there!";
```

Starting a New Line

- cout will pass the data in a continuous stream with no line breaks or spaces unless specifically told to do so
- To get multiple lines of output on screen
  - Use **`endl`**

```
cout << "Hello, there!" << endl;
```

    - endl is a *stream manipulator*
    - defined in iostream in the std namespace
    - Can be inserted anywhere in the stream
  - Use **`\n`** in an output string

```
cout << "Hello, there!\n";
```

Escape Sequences

- \n is an example of an *escape sequence*
  - Unlike endl (which is inserted into a stream with <<), \n must be in a string
- Escape sequences are used to control the way output is displayed
  - Always start with \ (backslash above enter on most keyboards)
  - Considered one character
- Other useful escape sequences
  - \t - moves the cursor to the next tab stop
  - \\ - prints \
  - \' - prints single quote '
  - \" - prints double quote "

cout Tips

- Look closely at your output and be sure to include whitespace to make your output more readable to the user.
- If a string is long, break it up onto several lines in the source code. This is easier for other people to read.
- Escape sequences use the backslash (\) not the forward slash (/)
- No space between the \ and character in an escape sequence, ‘\t’ and NOT ‘\ t’

cout Problems

1. What is the output of the following code segment:

```
cout << "I am the \"incredible\"";
cout << "computing\n\tmachine";
cout << "\nand I will \n\namaze\n";
cout << "\tyou.\n";
```

cout Problems

2. Find and fix the errors:

```
Cout << "red /n" << "blue \ n" << "yellow"  "endl" <<
green  << \n;
```

cout Problems

3. Write a complete program that displays your name on the first line, your street address on the second, and city, state, ZIP on the third.

Book Problems

The following problems are for extra practice, but are not due.

Checkpoint questions on page 13 and 37 (cout). (answers in Appendix C)

Review Questions page 23-24: 7-13, 16

Review Questions page 69-72 : 1-6, cout -> 11,12, 19, 20, 25C, 26A

Programming Challenges page 74: 10,11 (answers to odd numbers in Appendix D)

Next Time

- Variables and valid identifiers (Sections 2.6-2.7)
- int Data Type (2.7)
- Arithmetic expressions (3.2)
- input with cin (3.1)