Program components and compilers

A deeper understanding of how things work
Goals

• By the end of this lesson you will be able to:
  – Understand the basic components of a computer program
  – Understand how to write comments in a program
  – Understand how the compiler works in detail
Review

• From “Hello world!” some terms you saw were:
  – Linux
  – Server
  – Terminal emulator
  – Command line / shell
  – Secure shell
  – Directory
  – Text editor
Review

- From “Hello world!” some terms you saw were:
  - Compiler
  - Syntax
  - Style
  - Whitespace
  - Blank line
  - Indenting
  - Preprocessor directive
Review

• From “Hello world!” some terms you saw were:
  – Namespace
  – Statement
  – Main function / main body
  – Block
  – Standard output stream
  – Insertion operator
  – String
Review

- From “Hello world!” some terms you saw were:
  - Buffer flushing
  - Return
  - File extensions
  - Compiler error message
Comments

• Comments can be written in a program for a variety of reasons:
  – To help others understand your program
  – To help you understand your program (especially after a hiatus)
  – To leave yourself a note about things to modify later
  – To “comment out” portions of your program for testing purposes

• Comments were not mentioned as part of the “Hello world!” program because they are not processed by the compiler and thus are never required

• You should, and are required to, write comments for this course because they are a component of good style!
Comments

• Comments can be written in one of two ways:
  – C-style commenting: One or more lines between /* and */
  – C++-style commenting: One line after //

• Text editors featuring syntax highlighting will alter the color of comments and commented out portions of the program
New vocabulary

- **Source code / code**: A program or part of a written program
- **Tokens**: Individual program elements (including *reserved words*)
  - namespace
  - `<<`
  - `;
  - `cout`
- **Semantics**: While syntax refers to the organization of the tokens you use to write a program that will compile, semantics refers to the meaning behind the tokens
What does the compiler do?

- We will compile C++ programs using the `g++` compiler that we write in a text editor in order to be able to execute them.
- Behind the scenes, compilation is a multi-step process.
- The goal of compilation is to turn a higher level language into machine code.
Steps in compiling

- C++ preprocessor (any directives starting with #)
  - Result: Expanded source code
- Compiler
  - Result: Assembler file (platform specific)
- Assembler
  - Result: Object code (a portion of machine code)
- Linker
  - Result: Executable program (machine code including libraries)
The most common preprocessor directive we will use is `#include` to gain access to additional functionality from the library of the language.

To stop the compilation process after the preprocessor, use the `-E` flag.

The entire program will output to the screen.
C++ compiler

- To stop the compilation process after the compiler, use the `-S` command
- The output will go to a `.s` file
- View the file using a text editor
C++ assembler

- To stop the compilation process after the assembler, use the `-c` command
- The output will go to a `.o` file
- View the file using a hex editor
  - `cat filename.o | xxd | less`
C++ linker

- Simply using `g++` with no options will result in the entire compilation process occurring
- The resulting file after the linker step is `a.out`
- View the file using a hex editor
  - `cat a.out | xxd | less`
Compiling vs interpreting

- Most languages can be compiled or interpreted
- Compilation produces an executable file
- Interpreting runs the original source code of the program by going line-by-line, converting one line to machine code, then executing it
- Interpreting is very common for scripting languages such as HTML
- Syntax errors in interpreted languages are not caught until they are encountered, whereas a compiled language must be all syntactically correct before the program can run
- It is becoming more common to combine compiling and interpreting (e.g. Java, JavaScript, Python)
Exploration

- When viewing code from other sources (textbook, Internet) examine various aspects of style including spacing, indenting, and comments
- For other languages that you’ve heard of before, are they interpreted or compiled?
- For other languages that you’ve heard of before, what steps do you take in order to “run” a program?