PIC 10A: Week 2a

Section 1C, Spring 2016

Prof. Michael Lindstrom (TA: Eric Kim)

v1.0

Announcements

- Office Hours in effect this week

Reminders

- Lecture recordings (bruincast)
  - http://www2.oid.ucla.edu/webcasts/courses/2015-2016/2016spring/comptng10a-1
- My TA Page (where I post discussion slides/notes)
  - http://eric-kim.net/teaching/sp2016/pic10a_page/
- ccle.ucla.edu
  - You submit your homeworks here!

Today

- What is a programming language?
  - High Level vs Low Level
- Compilation Process
  - Preprocessor, Compiler, Assembler, Linker (P-CAL)
- Libraries
- Intro to C++

What is a Programming Language?

- (Wikipedia): "A formal constructed language designed to communicate instructions to a machine, particularly a computer."
- Typically are human-readable
  - as opposed to machine code, which is just a series of 0's and 1's.
- Popular Languages: C/C++, Python, Java, Ruby, Javascript, Matlab, …
- Each language has its pros and cons, but in principle, they can all accomplish any task

Pro tip: Once you learn ~2 languages well, then you can pick up a new language in a few weekends!
Lots of shared concepts between languages.

High vs Low level programming languages

- Most programming languages can be grouped into two categories: High level vs Low level
- To generalize: high vs low is a **tradeoff** between speed/efficiency of your program and convenience for writing programs.

High Level Programming Language

- Examples: Python, Java, Matlab, Javascript
- Designed to make programming ***easier***
- **Pros**: Easy to *quickly* prototype things in these languages
- **Cons**: Programs tend to run slower than low-level languages.
    - Ex: A program written in Matlab can be **~10-100x slower** than the equivalent program written in C++.
- In practice: Many people/companies first program their product in a high-level language.
    - Then, rewrite the code causing performance bottlenecks in a lower-level language, eg C/C++.

Low Level Programming Language

- Examples: C, C++.
- Sacrifices programmer convenience for speed
  - Forces you to manually keep track of things that higher-level languages manage for you
- **Pros**: Can write extremely efficient programs.
- **Cons**: Programming is a slower, more laborious task. Many more opportunities to make mistakes.

High vs Low: Hello World

```cpp
#include <iostream>
using namespace std;
int main() {
  // Displays "Hello world!" to user
  cout << "Hello world!" << endl;
  return 0;
}
```

C++

```python
print("Hello world!")
```

Python

Compilation Process

Me: Writing in C++

Input: My code (eg C++)
Output: "Transformed", expanded code. Still C++.

Input: (Expanded) Code (eg C++)
Output: Assembly language (eg x86_64)

Input: Assembly code
Output: Machine code (1's and 0's)

Input: Machine code
Output: Executable

(1/4) Preprocessor

```
#include <iostream>
/* My Program */
int main() {
  ...
}
```

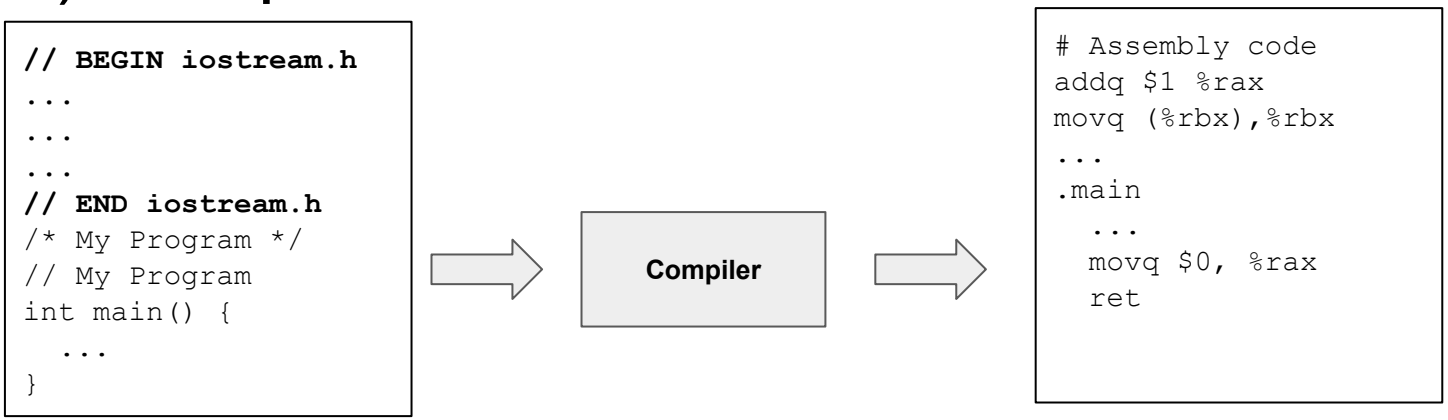

```
// BEGIN iostream.h
...
...
...
// END iostream.h
/* My Program */
// My Program
int main() {
  ...
}
```

Input: C++ source code

Output: Expanded out C++ code

- Transforms an input C++ source file into an equivalent source file with all shorthand expanded out
- Example: Preprocessor replaces "#include <iostream>" with the contents of the iostream header file

(2/4) Compiler



Input: Expanded C++ source code (output of preprocessor)

Output: Assembly Code, eg x86-64

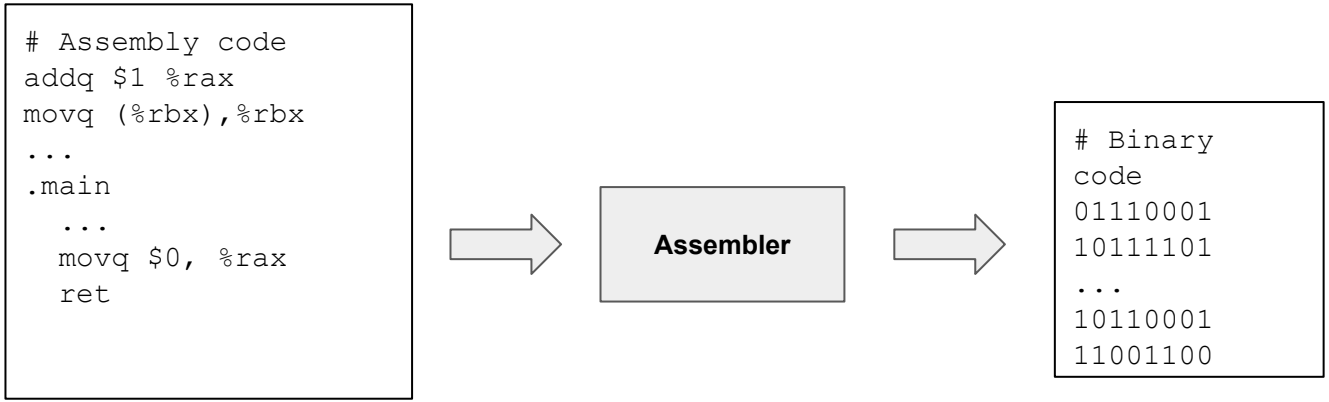
- Converts the C++ source code into assembly language
- Assembly is still human-readable, but is much more convenient to program in C++ than in assembly.

Aside: Back in the day, programmers used to program only in assembly! Then, people realized this was insane, and developed higher-level programming languages, eg C++.

Compilers

- Additional Features
  - Optimize code to make faster
    - "Free" speed improvements! No action from programmer
  - Detect syntactical errors, output meaningful error messages to programmer
    - Ex: "x" is an undefined identifier.
  - Register allocation
    - Figures out which registers to use for computations
  - Placeholders
    - Inserts placeholders for functions/variables that are declared by other libraries

(3/4) Assembler

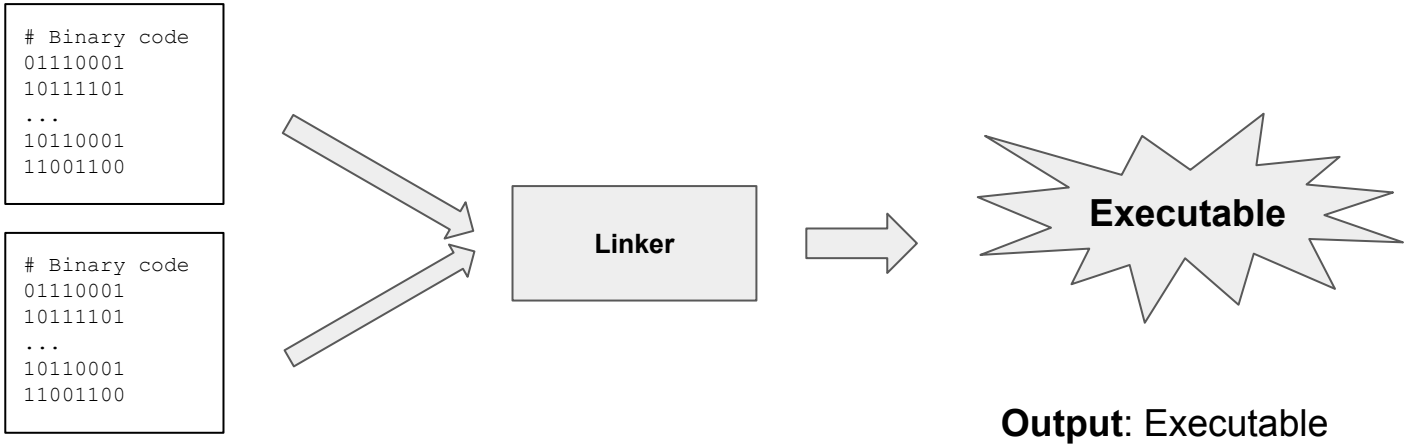


Input: Assembly Code, eg x86-64

Output: Machine Code, "object file"

- Converts the assembly code into machine code, a binary-representation of the instructions
- Machine code is not (easily) human readable

(4/4) Linker



Input: Object files

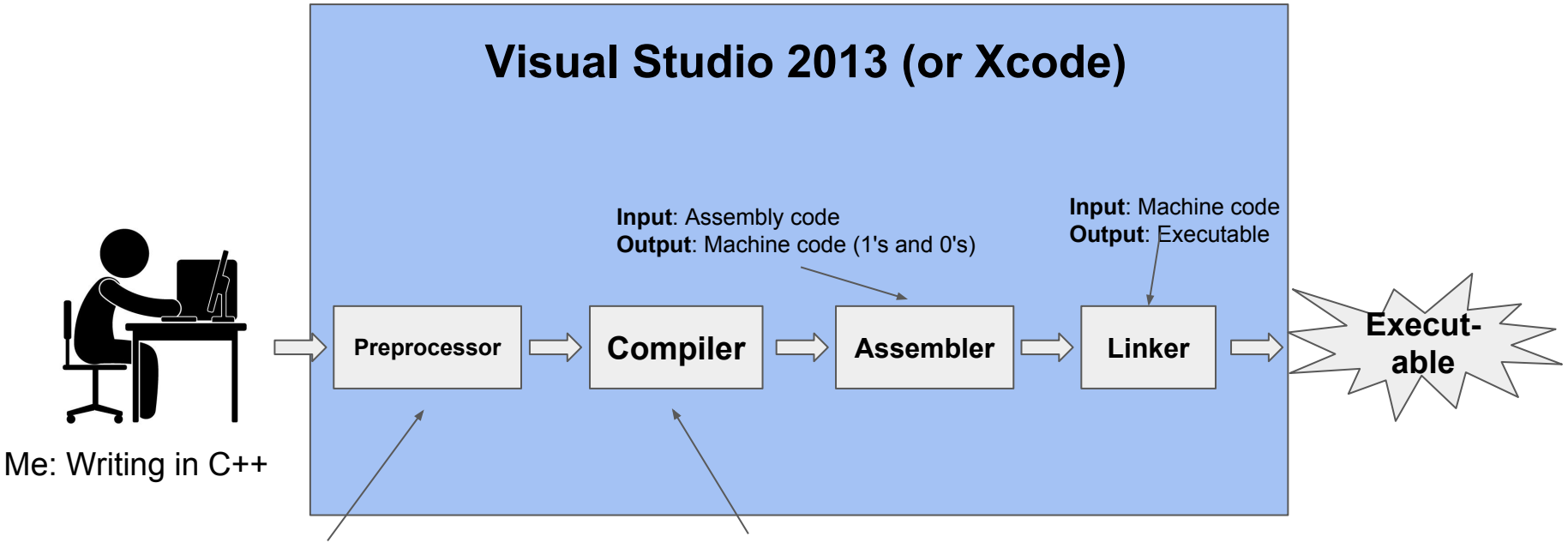
Output: Executable

- Stitches together multiple object files to generate final executable.
- Example: If your code uses the iostream library, then the iostream code will be linked to your own code by the linker.

Final Thoughts: Linking

- Loader: program that loads the program that you want to run into RAM
  - Ex: When you open Firefox, the loader will load the Firefox executable into memory
- Dynamic Linking
  - Many executables perform additional linking during a program's runtime.
  - Example: Typically, the C++ standard library is dynamically linked to your C++ program. Thus, the final executable does *not* contain the standard library (which is quite large, several megabytes).
- Static Linking
  - A library is statically linked when you provide the library's object file during link time (see previous slide).

Where does Visual Studio fit in?



Input: My code (ie C++)
Output: "Transformed", expanded code. Still C++.

Input: (Expanded) Code (eg C++)
Output: Assembly language (eg x86_64)

C++: Dissecting a simple program

C++, line by line

Question: What happens when I try to compile+run this program?

```
#include <iostream>
using namespace std;
// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

Answer: Simply outputs "Hi!" to the user, then exits immediately.

C++, line by line: include

Include statement.
Purpose: Unlocks additional functionality for the program.

Syntax: #include LIBRARYNAME

```
#include <iostream>

using namespace std;
// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

What is a Library?

- A library is collection of code that has functionality that will likely be useful to other programs.
- Example: If you want your program to have a user interface (ie windows, buttons), then you'll need to find a **graphical user interface library** (GUI).
- Example: If you want your program to recognize faces in a picture, you'll want to use a **face detection library**
- Lots of people release libraries online that are free to use!
    - Open source code: code that is free for use by anyone

C++ Standard Libraries

- Most languages (including C++) offer standard, "built-in" libraries
  - Common: File reading/writing, text manipulation, core data structures
- Popular C++ standard libraries include:
  - iostream, string, random
- List of standard libraries here:
  - http://en.cppreference.com/w/cpp/header

iostream

- Purpose: "...defines the standard **input/output** stream objects."
- The documentation about iostream says it defines: cin, **cout**, cerr, clog
  - http://www.cplusplus.com/reference/iostream/
- So, including iostream tells our program that **cout** *exists*.

What if we removed the include?

Question: What happens if I try to compile this program?

Answer: The program doesn't compile!
Error: "cout" is an undeclared identifier.

```
#include <iostream>

using namespace std;
// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

Aside: cout vs cin vs cerr vs clog

- cout: "Console Out", aka "standard out"
  - Writing to cout -> output text to user
- cin: "Console In", aka "standard in"
  - Reading from cin -> get text/number input from user
- cerr: "Console Error", aka "standard error"
  - Writing to cerr -> output warnings/error-messages
- clog: "Console Log"
  - Writing to clog -> output text relating to logging/debugging/whatever-you-like

In this class: focus on cout and cin.

Note: cerr, clog are meant for programmer, not for the user.

C++, line by line: namespaces

Purpose: Introduces variables/functions from a *namespace* into your program.

Syntax: using namespace ID;

```
#include <iostream>

using namespace std;

// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

using namespace std;

- Tells compiler we are using the "standard namespace"
  - std: "standard"
- Imports all of the functions/variables that a namespace **defines**
- **Example**: the `std` namespace defines `cout` and `cin`
  - More generally: all C++ standard library identifiers live in the `std` namespace

In this class, we will only use the standard namespace: std.

What if we remove "using namespace std;"?

Question: What happens when I try to compile+run this code?

```
#include <iostream>

using namespace std;

// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

Answer: Program doesn't compile!
Error message: cout is an undeclared identifier.

With/Without using namespace std

```cpp
#include <iostream>
using namespace std;
// Will print "Hi!" to the screen.
int main() {
  cout << "Hi!\n";
  return 0;
}
```

With

```cpp
#include <iostream>
// Will print "Hi!" to the screen.
int main() {
  std::cout << "Hi!\n";
  return 0;
}
```

Without

Verdict: "using namespace std;" simply lets us not have to type "std::" a bunch of times.

std::cout means to access the identifier "cout" from the namespace "std".
Anything that the C++ **standard library** defines lives in the **std namespace**.

C++ line by line: Comments

Comment
Purpose: Provide information or explanation useful for a programmer/reader.

Computer **ignores** everything you put in a comment.

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

// Will print "Hi!" to the screen.

int main() {
  cout << "Hi!\n";
  return 0;
}
```

C++ line by line: Comments

Question: What happens when I try to compile+run this program?

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

// Will print "Hi!" to the screen.

int main() {
  // cout << "meow" << endl;
  cout << "Hi!\n";
  return 0;
}
```

Answer: Compiles correctly, and outputs "Hi".
The "meow" isn't output because it's part of a **comment**.

Multiple ways to comment

```
// (1) Single line comments must always start
//     with two forward slashes.

/* (2) Anything in here is
       considered to be
a comment.
*/
```

(1) Single-line comments
(2) Multi-line comments

C++ line by line: the main() function

main
Purpose: Contains code that actually runs when you run the executable.

```
#include <iostream>
using namespace std;
// Will print "Hi!" to the screen.

int main() {
  cout << "Hi!\n";
  return 0;
}
```

The main() function

- The *return value* of the main function is known as the **status code**
- As convention, 0 means that the program terminated normally.
- non-zero return values (ie -1) mean that the program exited abnormally
    - Examples: File wasn't found, invalid input, etc.

C++ line by line: cout

Purpose: Output text to the user.

`cout`: Console output
Defined by: `<iostream>`

```
#include <iostream>
using namespace std;
// Will print "Hi!" to the screen.
int main() {

  cout << "Hi!\n";

  return 0;
}
```

cout: Chaining

- Can chain "<<" together to output multiple things
- Example: `cout << "I am taking " << 3 << " classes this quarter.\n";`
- Outputs: `I am taking 3 classes this quarter.`

cout: numbers

- cout understands numbers as well!
- Examples:

```
cout << "I am " << 26 << " years old.";
```
Outputs:
```
I am 26 years old.
```

```
cout << "There are " << 42+57 << " red balloons.";
```
Outputs:
```
There are 99 red balloons.
```

"Special" characters, ie \n, \t,

- We've seen that "\n" is special: it creates a new line. Known as the new-line **escape sequence**.
- Other escape sequences:
  - \t Tab
  - \" Double-quote
  - \' Single-quote
  - \\ Back-slash
  - \a Creates an audible beep
  - \b Moves the cursor backwards, ie to the left.

Exercises: cout

Question: What do the following output? If it errors, explain the error.

```
cout << "For" << "No\n";
cout << "One";
```

Answer:

```
ForNo
One
```

```
cout << "Toe\n";
cout << "\n" << "To " << "Toe";
```

Answer:

```
Toe

To Toe
```

Exercises: cout

Question: What do the following output? If it errors, explain the error.

```
cout << ""Hello"" << "Goodbye";
```

Answer:

```
Compile error!
The word Hello is not
contained within double-
quotation marks, so it
doesn't make sense.
```

```
cout << "Revolution " << "3+6";
```

Answer:

```
Revolution 3+6
```

Exercises: cout

Question: Write some code that will exactly generate the following output:

```
I "love" waking up at 6 AM!
```

Answer:

```
cout << "I \"love\" waking up at 6 AM!";
```

Exercises: cout

Question: Write some code that will exactly generate the following output:

```
I "love" waking up at 6 AM!
```

Question: Is the following answer correct?

```
cout << "I " << " << "love" << " << " waking up at 6 AM!";
```

Answer: Nope! This will actually error.

```
cout << "I " << " << "love" << " << " waking up at 6 AM!";
```

String 1 String 2

Uhoh, what's that?
Error!

cout: endl

- Alternative to typing "\n" a bunch of times: endl
  - Stands for: "end line"

```
cout << "Hi there\n" << "Face here";
```

outputs the same thing as:

```
cout << "Hi there" << endl << "Face here";
```

Output:

```
Hi there
Face here
```

String Literal

- To create a string literal, wrap some text with double quotation marks
- Examples: "Hi there", "3+4", "bye\n" are all **string literals**
    - We've been creating string literals all along!

String Literals

- Important: Computer will not "execute" contents of string literals. Leaves the contents as-is.
- Example: `cout << "3+4";`
  - Outputs: 3+4, not 7
- **Exception**: Escape sequences. `\n, \t, \\, \", \'`
  - Example: `cout << "hi\nthere";`
  - The `\n` is **expanded** out to a new-line.

Exercise

Question: Write some code that outputs the following:

```
I put
a newline \n there!
```

Answer:

```
cout << "I put\n" << "a newline \\n there!";
```

or:

```
cout << "I put" << endl << "a newline \\n there!";
```