Introduction

James Daly
CSE 232
Michigan State University
Course Description

About Me

- Dr. James Daly
- Michigan State University 2017
- Research Focus: Network Security and Algorithm Design
- Email: dalyjame@msu.edu
- Office: 3110 EB
TAs

- Atra Akandeh
  - akandeha4@msu.edu
- Doug Kirkpatrick
  - kirkpa48@msu.edu
Course Expectations

- Previous programming experience is *expected*
  - Will go much faster than 231 because you know the basics
  - Extra fast because this is a summer course

- You will learn
  - C++ / STL
  - Working in a Linux environment
  - Terminal / Command Line
  - Debugging
Time and Location

- Lecture
  - Mon / Wed / Thurs 3:00 – 4:50 pm
  - 1145 Engineering Building

- Lab
  - Tues / Fri
  - Section 1: 12:40 – 2:30 pm
  - Section 3: 3:00 – 4:50 pm
  - 3345 Engineering Building (Dog Lab)
Textbook

- C++ Primer, 5th ed
  - by Lippman, Lajoie, and Moo
  - Optional book
  - Useful reference
Course Resources

- Course Website: cse.msu.edu/~cse232
  - Course materials here
- D2L: d2l.msu.edu
  - Grades posted here
- Piazza: piazza.com/msu/summer2018/cse232/home
  - Ask questions here
- Mimir: class.mimir.io
  - Submit projects here
Grades

- Grade Breakdown
  - 3 Examinations (50% total)
  - 6-7 Projects (45% total)
  - Pre-Lab exercise (5% total)

- Mastery Requirement
  - 50% of exam points
  - 50% of project points
  - Miss no more than 2 labs

<table>
<thead>
<tr>
<th>Grade</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>90%</td>
</tr>
<tr>
<td>3.5</td>
<td>85%</td>
</tr>
<tr>
<td>3.0</td>
<td>80%</td>
</tr>
<tr>
<td>2.5</td>
<td>75%</td>
</tr>
<tr>
<td>2.0</td>
<td>70%</td>
</tr>
<tr>
<td>1.5</td>
<td>65%</td>
</tr>
<tr>
<td>1.0</td>
<td>60%</td>
</tr>
</tbody>
</table>
Exams

- 3 Exams
  - Equally weighted
  - Taken during class on Thursdays
    - Midterm 1: May 31
    - Midterm 2: June 14
    - Final: June 28
  - Contain a mix of multiple-choice, short answer, and free response questions
Pre-Labs

- Pre-labs will before the labs
- Must be completed before the lab begins
- **No make-up for pre-labs**
- Can attempt as many times as you want
Labs

- In 3345 EB (Dog Lab)
- Taught by Teaching Assistants
- Collaborative exercises
- Help prepare you for the projects
- Physical attendance is **required**
  - Can miss **two** labs for **any reason**
- You must do the work **during the lab**
Projects

- 6-7 Projects
- Will be written in C++
- Submitted electronically via Mimir
- You may submit many times
  - Only the most recent submission is graded
- Your code must compile on Mimir
  - Otherwise you may receive a 0
  - No, “but it works on my machine”
Auto-Grading

- We will use the website Mimir (class.mimir.io) for automatic grading
  - Invitation is on the front page of the CSE 232 page
- Mimir provides tests for your code
- Your project grade will be based on the Mimir tests that you pass
Plagiarism

- You are expected to do your own projects
- Don’t copy code from anywhere
- Submissions are compared against other submissions
  - Do not share your project with others
  - If they copy from you, you will also be penalized
- Penalties
  - 1st offense: 0 for project and a half-mark penalty to final grade
  - 2nd offense: Failure for the course and dismissal from the program
  - Egregious offenses may receive additional penalties
On-Campus Computing

- All students taking CSE courses get an account on EGR computers which they can access remotely or in the EGR labs
- The EGR computing labs are open 24x7 and your account is active on all the machines in all the labs
- Lab00 gets you set up for this
Working from home

- FAQ (232 web page provides details):
  - x2go: provides a full desktop as a window in your laptop
    - Needs a decent network connection
    - You connect to your CSE directory
  - Mimir (also your handin)
  - Work on your own on your machine
    - Can get Visual Studio from the Microsoft Developer Network Academic Alliance (MSDN AA)
    - [www.cse.msu.edu/Facility/Services/MSDN.php](http://www.cse.msu.edu/Facility/Services/MSDN.php)
    - Can also use other IDEs
    - Beware of grading
C++11

- In 2011 the latest standard for C++ came out, named C++11
  - Extended by C++14 and C++17
- You will learn the latest standard
  - Easier to work with
  - Makes C++ “better”
  - Amaze you friends, professors, and future employers (they may not know it)
The STL

- The Standard Template Library (STL) is your best friend.
  - Does things so you don’t have to
  - We will *focus* on using the STL
  - Use it when you can!
Getting Started

James Daly
CSE 232
Michigan State University
What is C++?

- General-purpose programming language
  - Imperative
  - Object-oriented
- Developed by Bjarne Stroustrup between 1979-83
  - Enhanced C with Simula-like features
  - C++ 2.0 released in 1989
  - C++11 released in 2011
Quote

- “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.”

-Bjarne Stroustrup
Why C++?

- Every programmer needs to know two classes of language
  - Script-y language for everyday / simple kinds of things
    - Ex: Python, javascript
  - System-y kind of language that provides speed, efficiency, power to do harder, more computational stuff
    - Ex: C++, C
Differences between Python and C++

- Code is compiled
  - No interpreter
  - Have to do most of the bookkeeping ourselves
    - Memory creation / destruction
    - No garbage collector
  - Executable created from your code
Types of error

- **Compile-Time Error**
  - Broke the rules of C++
  - Compiler doesn’t know what you want to do
  - Compiler flags the error and quits
  - No executable created!

- **Runtime Error**
  - Followed the rules of C++ but still messed up
  - Executable was created but it did something wrong
Standard Template Library (STL)

- Provides libraries that handle particular tasks
  - Containers
  - Algorithms
  - Memory management
Differences between Python and C++

- Python
  - Dynamic typing (determined at run-time)
  - Duck typing (if it quacks like a duck)

- C++
  - Static typing (determined at compile-time)
  - Lots of types
  - Modifiers to those types
  - The compiler must know the types of everything before it will compile
Example

```cpp
const map<string, vector<long>> const * m = &some_var;
```

- **Green** stuff is all part of a single type declaration
  - A constant pointer to a constant map of strings to a vector of longs
- You have to declare your variable’s type
- It only holds that type (or does something weird to what you try to stuff into that type)
Some context required

- Syntax requires context
  - C++ tries to stay backwards-compatible with C (mostly)
  - C++ continues to add features to the language that really help
  - There are only so many characters on a standard keyboard

- Results in symbols that get reused to mean different things
  - The symbol is not enough to sort out what it means
  - Ex: `aa bb(cc);`
    - Declaration of object `bb` of type `aa` with variable `cc` passed to its constructor
    - Declaration of function `bb` that takes an object of type `cc` and return type `aa`
Another Example

```c
long my_int = 123;
long &ref_int = my_int;
long *another_int = &my_int;
my_int = *another_int;
my_int = my_int * my_int;
```

- & means “a reference type” in line 2 and “the address of” in line 3
- * means “pointer type” in line 3, “value pointer points to” in line 3 and multiplication in line 5
Error Messages

- C++ has a well-earned reputation for cryptic error-messages
- A single syntax error can generate 10s of lines of error.
  - Most important info is the line number
cout << "Size of short:" << sizeof(short) << endl;
Compile early, compile often

- You should *compile all the time*
  - Write a line
  - Compile
  - Repeat
- If you don’t you will suffer
  - Especially early when you don’t understand what the error message mean
Simple Programs
First Program: Hello, World

```cpp
#include <iostream>

/*
 * hello world program
 */

int main() {
    std::cout << "hello, world"; // output
}
```
includes

- To use aspects of the library system, you must include the definitions into the system
  - Like import in python
  - Table A.1 (pg 866-870) lists many of the elements and their associated include file
#include<iostream>
is a pre-processor statement
It does *not* end in a semicolon
  - Part of the pre-processor, not C++
What you include goes between the chevrons < and >
  - No `.h` or `.hpp` at the end of the `include`
Comments, two kinds

- /* ... */
  - Anything between those symbols are ignored
  - Multiline
- // ... 
  - Everything after this is ignored
  - One line only
main

- A runnable program requires a specific function named **main**
- This function is what will start the program when the compile code is loaded
#include <iostream>

/**
 * hello world program
 */

int main() {
    std::cout << "hello, world"; // output
}

Begin and end of a block
Curly braces have multiple meanings
- A **block** in the context of a function or a control statement (most common)
- A **sequence** of statements that fill in for a single statement

Indentation means nothing
- Only helps the reader
- Like comments, it is helpful for making your code more readable
- Can put many statements on a single line
- The following is valid (but sucks)

```cpp
#include <iostream> /* hello world program*/ int main(){ std::cout << "hello, world" << std::endl; }
```
First Program: Hello, World

```cpp
#include <iostream>

/*
 * hello world program
 */

int main() {
    std::cout << "hello, world"; // output
}
```
std

- When you use `include` you insert definitions into your program.
- Those elements are in namespaces.
  - We’ll talk more about those later.
  - Need to reference them by their namespace.
  - The common libraries are in the `std` namespace.
    - Short for standard.
    - Yes, it is an unfortunate acronym.
The lazy typist

- No group of human beings are lazier typists than programmers
  - Short acronyms instead of full names
  - Short variable names
- Need to find a balance between readability and typing
- Different communities have different customs
  - Java tends to favor more descriptive names
  - Unix names: remove all of the vowels, randomly drop some letters, and maybe add a vowel back
Scope resolution operator

- To differentiate namespaces, you include the namespace name followed by ::
- :: is the scope resolution operator
- `std::cout` means `cout` in the standard namespace
Streams

- When you include `<iostream>` you get three streams to use:
  - `std::cout` (short for console output)
  - `std::cin` (short for console input)
  - `std::cerr` (short for console error)
First Program: Hello, World

#include <iostream>
/*
 * hello world program
 */
int main() {
    std::cout << "hello, world"; // output
}
Output

- To move information from your program to output you use the `<<` (insertion) operator
  - Take info and place it in the output
  - Is a binary operator
    - returns the stream being used
  - Outputs either strings (between " ") or the value in a C++ variable
“” vs ‘’

- “abc” is a string type (sequence of characters)
- ‘a’ is a character type, a *single* character
- Different than Python!
Semicolons

- C++ uses the ; (semicolon) to terminate a line
- Not all lines require them
  - Expressions require them
  - Declarations require them
  - Blocks **do not** require them
- We’ll get the hang of where they go
- A good editor will help with missing ; and similar issues
First Program: Hello, World

#include <iostream>
/*
 * hello world program
 */

int main() {  
    std::cout << "hello, world"; // output
}

Return type

No semicolon

semicolon
Style Guide

- We select a style for writing our programs so:
  - They are more readable (very important)
  - We have a system amongst ourselves so we can agree on format
  - Be religiously picky about how we do things ;-)}
Google

- Why not use Google’s Style guide?
- [https://google.github.io/styleguide/cppguide.html](https://google.github.io/styleguide/cppguide.html)
- They have a very complete style guide
  - When in doubt, consult
- Not a problem for the first project, will be later
Variable names: rules and style

- Variable name rules
  - Only digits, letters, and underscore are allowed
  - Can’t start with a digit
  - Case matters
  - Namespace matters

- Variable name style
  - Lowercase with underscores between words (like in Python)
  - Meaningful / readable
  - Avoid unfamiliar abbreviations
Comments

- Comment at the top of the file
  - Name, date, what it is about
- Variable names shouldn’t require comments (descriptive names)
- Functions should have comments
  - Input, output, what it does
- “If it was hard to write, it will be hard to read”
  - Comment the hard parts
First Program: Hello, World

#include <iostream>
/*
 * hello world program
 */
int main() {
    std::cout << "hello, world"; // output
}