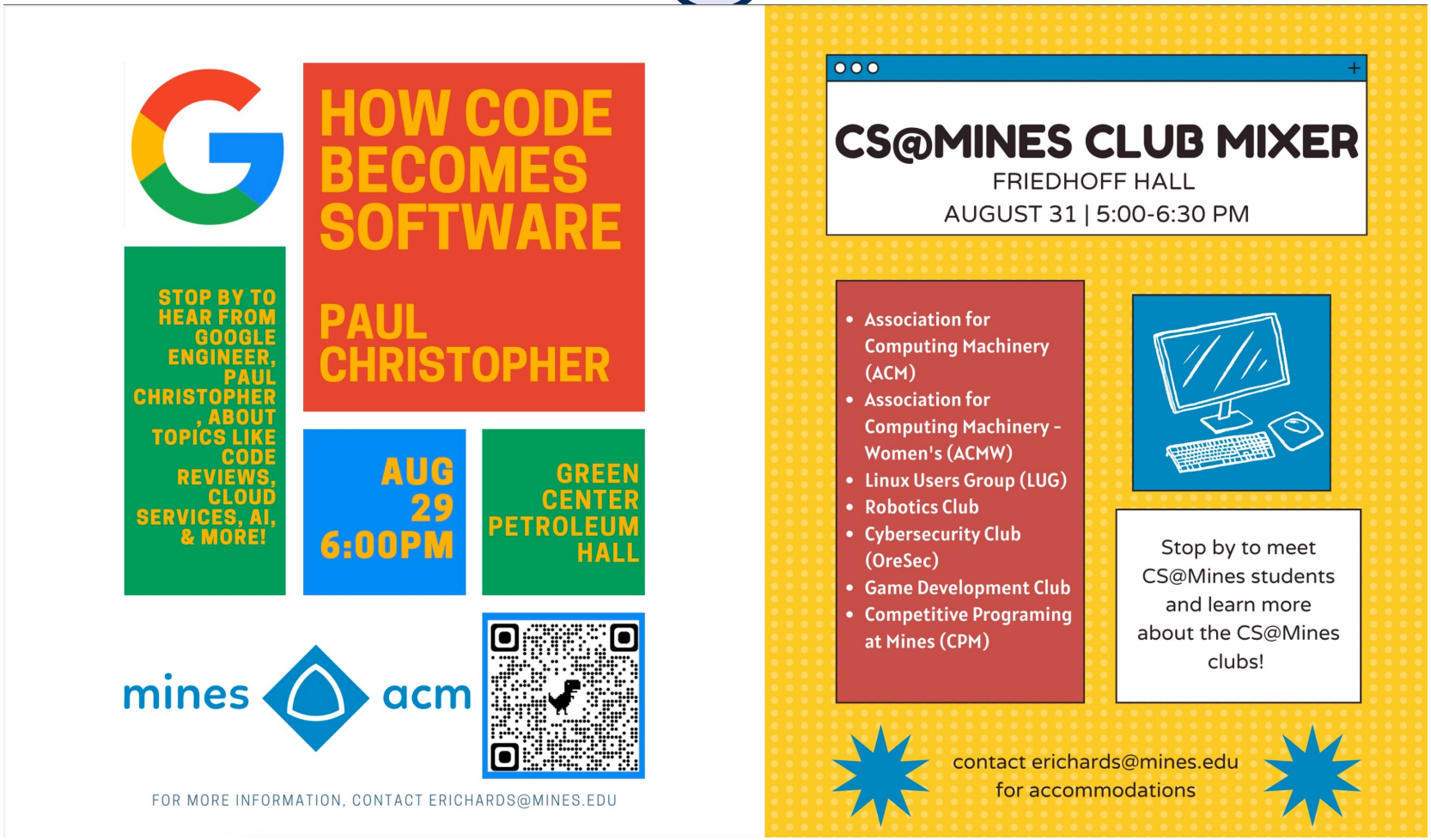
# CSCI 200: Foundational Programming Concepts & Design Lecture 01

Hello World!


Have VS Code Open

(If your machine isn't verified working, use the lab machines)

# ACM Events


HOW CODE BECOMES SOFTWARE
PAUL CHRISTOPHER
STOP BY TO HEAR FROM GOOGLE ENGINEER, PAUL CHRISTOPHER, ABOUT TOPICS LIKE CODE REVIEWS, CLOUD SERVICES, AI, & MORE!
AUG 29 6:00PM
GREEN CENTER PETROLEUM HALL
mines acm
FOR MORE INFORMATION, CONTACT ERICHARDS@MINES.EDU
CS@MINES CLUB MIXER
FRIEDHOFF HALL
AUGUST 31 | 5:00-6:30 PM
Association for Computing Machinery (ACM)
Association for Computing Machinery - Women's (ACMW)
Linux Users Group (LUG)
Robotics Club
Cybersecurity Club (OreSec)
Game Development Club
Competitive Programing at Mines (CPM)
Stop by to meet CS@Mines students and learn more about the CS@Mines clubs!
contact erichards@mines.edu for accommodations

# Previously on CSCI 200

- C++ is statically typed
- C++ programs are compiled
- C++ whitespace doesn't matter

- *(we'll see examples of these today)*

# Common Fears

- Not good at programming / lack of knowledge
- Learning a new language
- Won’t be able to keep up
- A lot of work
- Everyone else will get it but I won’t
- Won’t be able to find what’s wrong

# LOTS of Help Available

1. In Person Help
   - Office Hours: Instructors
     - M, T, W, R, F days
   - Tutors (begins Tuesday)
     - U, M, T, W, R nights
2. Ed Discussion
3. Email last resort

- https://cs-courses.mines.edu/csci200/resources/people.html
- https://cs-courses.mines.edu/csci200/resources/help.html

# Help Schedule

- Sunday
  - 6p-8p: Tutoring
- Monday
  - 10a-12p: Office Hours
  - 1p-3p: Office Hours
  - 5p-9p: Tutoring
- Tuesday
  - 1p-3p: Office Hours
  - 5p-9p: Tutoring
  - Assignment Sets due at midnight
- Wednesday
  - 10a-2p: Office Hours
  - 5p-9p: Tutoring
- Thursday
  - 10a-12p: Office Hours
  - 5p-7p: Tutoring
- Friday
  - 10a-12p: Office Hours
  - 1p-3p: Office Hours

- See website for locations

# Learning Outcomes For Today

- Create a Hello World program, construct a simple interactive application, and build the program via the terminal.
- List common Linux terminal commands and choose the correct commands to work with a file system via the command line.
- Describe how a computer generates a program from code.
- Implement various techniques to trace & debug a program.

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# Follow Along On Canvas

- Open Quiz > 8/23 Post Class Survey
  - Needs to be submitted before next class

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# First, think about context

- Brown bear
- Bear the weight
- Bear Bryant
- Bearing a tray

- Human language is highly “context dependent”

# Programming languages

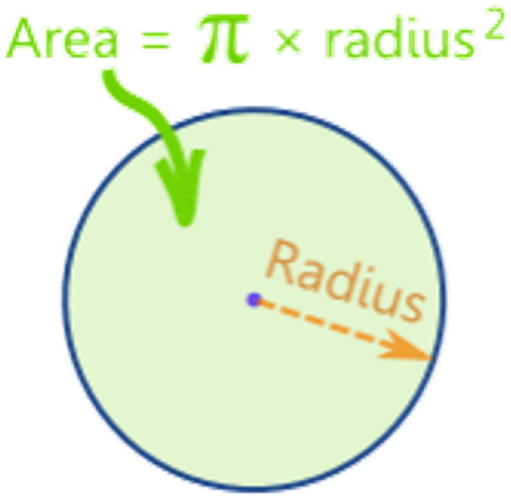
- Are context free

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

# Syntax Examples

- C++
  - area = 3.14*(diameter/2)*(diameter/2)
- Matlab
  - area = pi*((diameter/2)^2)
- Python
  - area = 3.14*((diameter/2)**2)
- Basic
  - let a=3.14*(d/2)*(d/2)
- Structure + rules

  => programming syntax

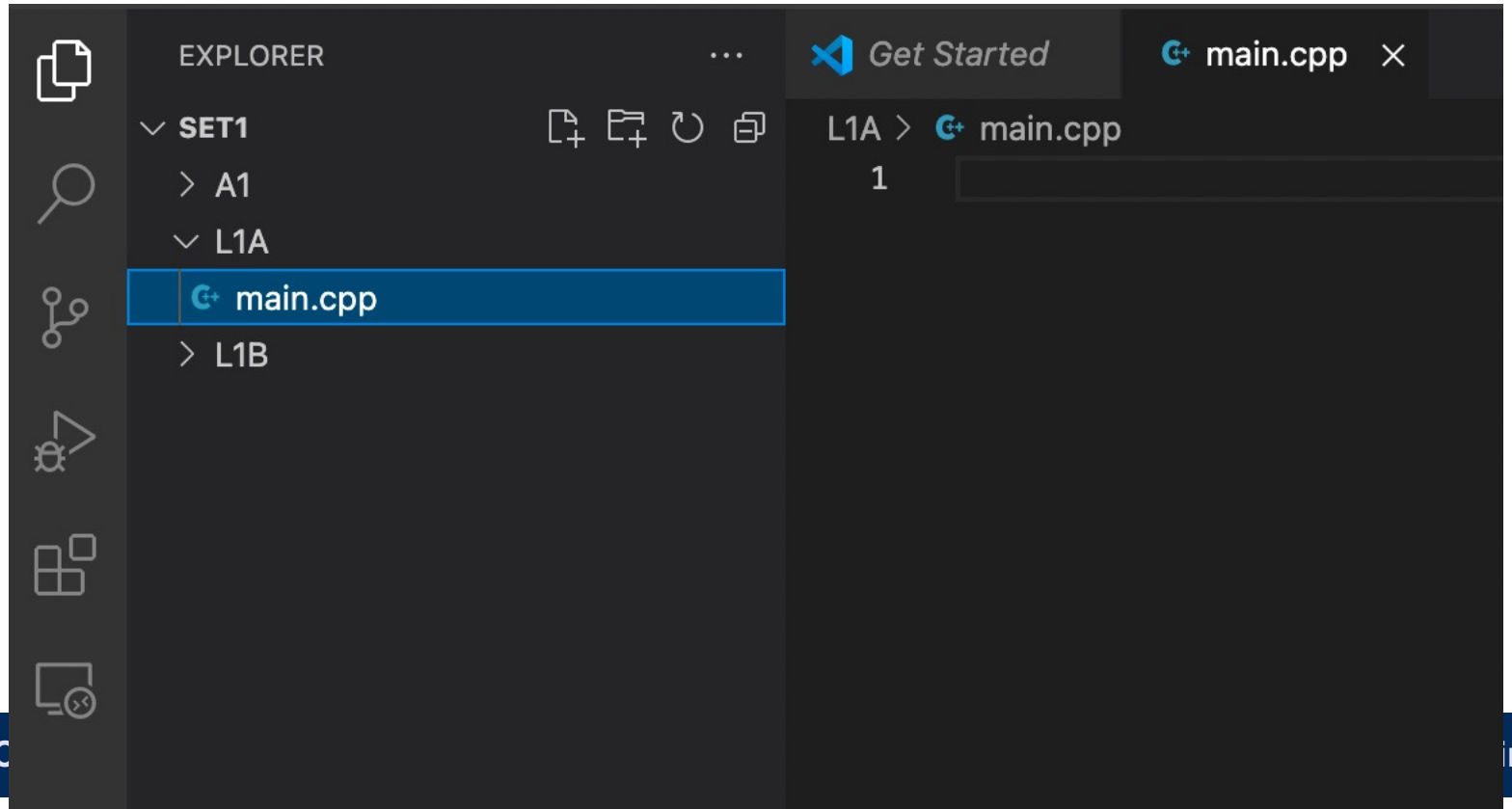


# A “Generic” C++ Program

```
// comments

#preprocessing directives

int main() {

    variable declarations;

    statements;

     input & output;

    return result;

}
```

# First Bit of Help

- Get VS Code open!  (Lab0)
- We'll set up our folder/file structure



# int main()

- All C++ programs start with and must include main()

- Why?
  - What does your program do?
    - Whatever is in the code block following main()

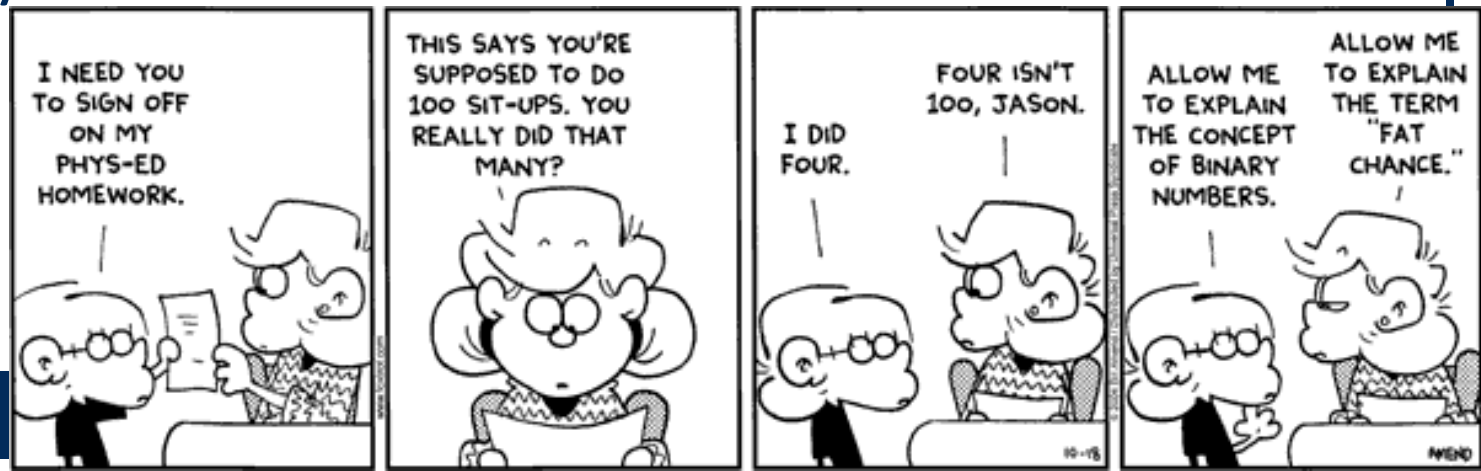
# My first program!

- { all the code in between the curly braces is a code block }

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

# Programming Languages Review

- High Level Languages
  - C++
- Assembly Languages
  - x86
- Machine Language
  - Binary



# Making a Program

- Three steps (in order)
  1. Compile
  2. Link
  3. Execute (run)

# Step 1: Compiling

- The process of converting your code from C++ (a high level language) to binary (machine language)
  - Produces an “object file”
    - main.cpp → main.o
- Use a compiler (g++) to translate your code

# Object File

- Contains machine instructions (binary)
- Not executed
- Combined with other object files to make an executable or program

# Step 2: Linking

- Your program relies on other libraries
  - iostream
  - cmath
- The linker combines all the necessary object files
  - g++ is also our linker
- Produces an executable program
  - main.o + libiostream.a → a.exe (on Win)
    a.out (on OS X / *nix)

# Step 3: Execution

- Executable programs are a file on disk
- To “execute” a program means to run it
  - Load the executable file into memory
  - Tell the computer where the first instruction is
  - Run the program!

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# Command Line Interface (CLI)

- Textual representation to move through file system and directory structure
  - Move through folder hierarchy
  - Manipulate files
  - Run programs

- (*more details next week*)

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# How Do I Know It's Running?

- We need to communicate with the machine while our program is running.

- We want the machine to generate a response and tell us.

# Preprocessing Directives

- Other things (aka "libraries" or files) your program will use
  - e.g. math functions, input/output, graphics

```
#include <iostream>
#include <cmath>
```

- "Computer, my program is using functions from the iostream and math libraries"

# C++ Compiler Flow

```
// preprocessing directive

#include <someLibrary>


int main() {

    variable declarations;

     statements;

     return 0;

}
```

Compiler starts at the top and goes down line by line

# C++ Program Flow

```
// preprocessing directive

#include <someLibrary>


int main() {

    variable declarations;

     statements;

     return 0;

}
```

Computer starts at `main()` and goes down line by line

# Input/Output (I/O)

- Include the Input Output Library

```
#include <iostream>

using namespace std;
```

- Gives us access to
  - **`cout`** : output
  - **`cin`** : input

# cout

- Character OUTput
  - or standard output to the screen

```
cout << "Do you have the time ";
cout << "to listen to me whine?" << endl;
cout << "No" << endl;
```

```
Do you have the time to listen to me whine?
No
```

# Output in a Program

- { all the code in between braces }

```
int main() {
    // what is the meaning of life?
    cout << "42" << endl;
    cout << 42 << endl;
    return 0;
}
```

Hey look! This line is a comment! It starts with // so we know the computer will ignore it

# Semicolons;

- Like a period at the end of a sentence.

- EXCEPT for (most) preprocessing directives (they're special)

No semicolon

```
#include <iostream>
```

Semicolon!

```
cout << "Hi" << endl;
```

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# Syntax Errors == Code Errors

- Reported by compiler (a.k.a. compiler errors)
  - Gives:
    - File error is in
    - Line error is on
    - “Descriptive” message what error is

# On Tap For Today

- A C++ Program - Hello World!

- Command Line Interface (CLI)
- Input / Output (I/O)
- Syntax Errors

- Practice

# To Do For Next Time

- Complete Post-Class Survey in Canvas

- Friday
  - Input
  - Memory: Storing & Manipulating Data
  - Generating Random Values