183 Discussion

Week 2 – Diana Gage

www-personal.umich.edu/~drgage

About Me

- Junior in CS-LSA with minor in Applied Stats
- I took EECS 280 & 203 last semester, currently in 281 & 376
- Looking for an internship this summer
- U of M Women's Glee Club, Graham Sustainability Scholars, Michigan Backpacking Club

Introduction

- What’s your name?
- What’s your year and major?
- What do you want out of EECS 183?
- Something memorable about you

Discussion Format

- Review lecture material
- Practice problems/code similar to what you'll see on projects and exams
- Come with questions!
- I'll do my best to leave time for questions at the end, and I'll stay after class to answer more personal questions
- Correct my mistakes! ☺
- Challenge/Extended Problems
- Discussion slides on my website

Course Logistics

- Questions → Piazza! (search first, then post question)
- Next step: Office Hours – check course website
- You can attend anyone's OH
- Mine are Mondays from 3-6pm ☺

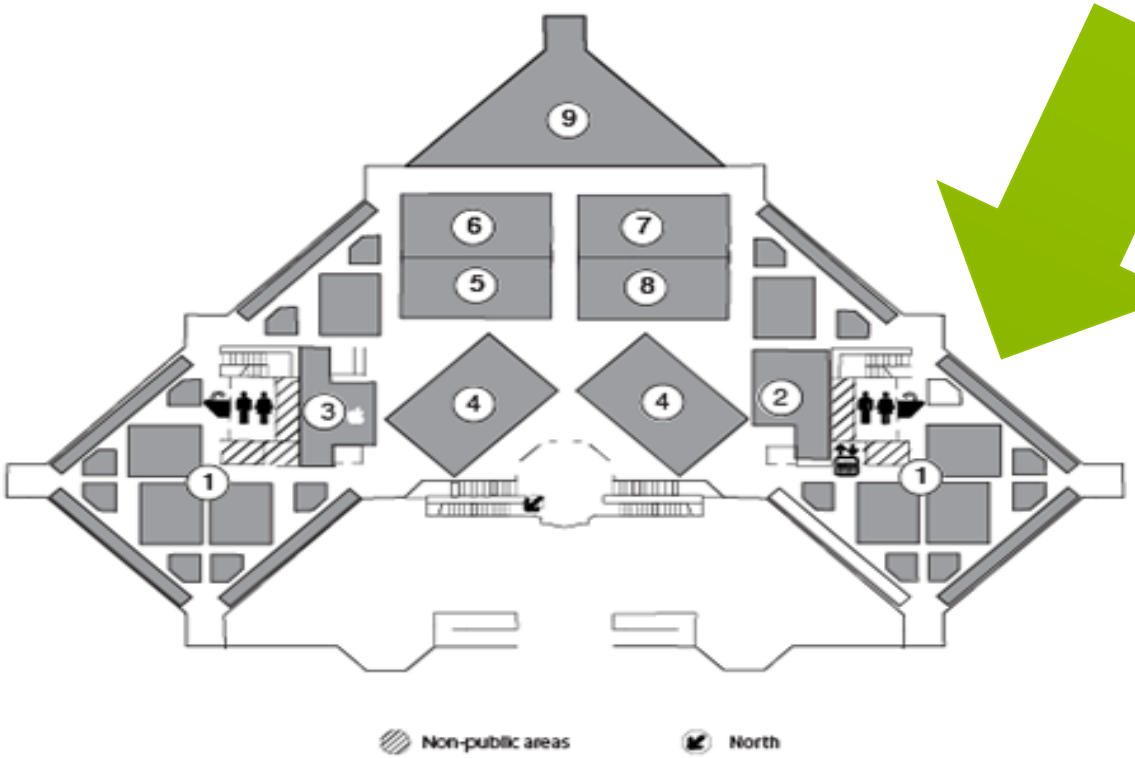
Office Hours

- Office hours this semester will be held on North Campus in the Duderstadt center
- There will always be office hours from:

  3-8 pm M-F on the **third floor, west end** (diagram on next slide)
- There will *sometimes* be additional office hours on the weekend – the will be posted on the website
- There will be **a paper sign up sheet** that will arrive with the first staff member, and they will start promptly at 3, and **end exactly at 8** (give or take 5 minutes)
- The website will indicate which staff members are present at which time, and this will remain mostly constant

Office Hours Location

Level 3 Floor Plan (pdf.)



Upcoming Deadlines

- **Zyante reading** – due before each lecture

- **Codelab & Survey**:

This Friday (Jan 15) at 11:59 pm

- **Project 1: Next Friday (Jan 22) by 6:00 pm (accepted until 11:59 pm)**
  - **Submit Projects early for extra credit**
    - **+ 5% 2 days early (still 6 pm)**
    - **+ 2.5% 1 day early (6 pm)**

- **Always check the website for updates**

Upcoming Event!

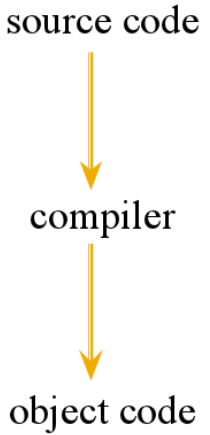
- EECS CARNIVAL
- Thursday, January 21st
- North Campus (Bob and Betty Beyster Building)
- Stay posted on the website and piazza for more info

STYLE POINTS

- Every semester students lose a bunch of points for incorrect style
- There are 10 points reserved for style on each project – people often get a 0 on this just because they don’t check the style guidelines
- Style guidelines are on the course website!

Basics of Programs

- You will write your programs in an IDE (Xcode or Visual Studio)
- These provide a code editor (where you write and edit code)
- The code you write is sent through a compiler
- The compiler turns your code into object code (machine language)



Starting a new project

XCode:

1. Choose File > New > New Project.
2. Select “Command Line Application”, and click Next.
3. Enter a name and details for your new project, and click Next.
4. Click Save.

Visual Studio:

1. Choose File > New > Project
2. Choose **Visual C++** on the left side, then choose **Empty Project** in the middle.
3. Enter a name and choose a location for your new project, and click **OK**.

Hello, World!

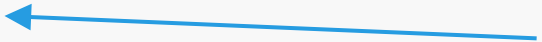
Hello World is a programming tradition. It's the canonical example program given when starting a new programming language, whether it's your first or your 50th language.

Program Goal

Print the words "Hello, World!" to the screen.

Hello, World!

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main() {
5.      cout << "Hello, World!" << endl;
6.      return 0;
7.  }
```



main() is where any C++ program will begin!

Hello, World!

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main() {
5.      cout << "Hello, World!" << endl;
6.      return 0;
7.  }
```

main() is where any C++ program will begin!

cout: “C-out”, as in C-language output - “put this on the screen”

Hello, World!

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main() {
5.      cout << "Hello, World!" << endl;
6.      return 0;
7.  }
```

main() is where any C++ program will begin!

cout: “C-out”, as in C-language output - “put this on the screen”

<< means “insert this into” - the direction of the arrows means it’s going into C-output.

Hello, World!

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main() {
5.      cout << "Hello, World!" << endl;
6.      return 0;
7.  }
```

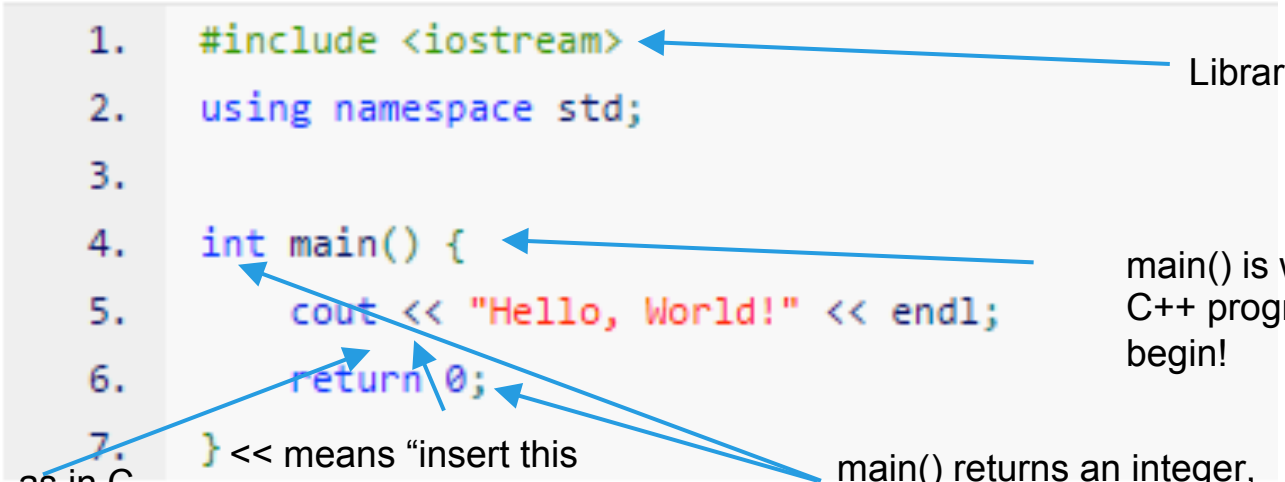
main() is where any C++ program will begin!

cout: “C-out”, as in C-language output - “put this on the screen”

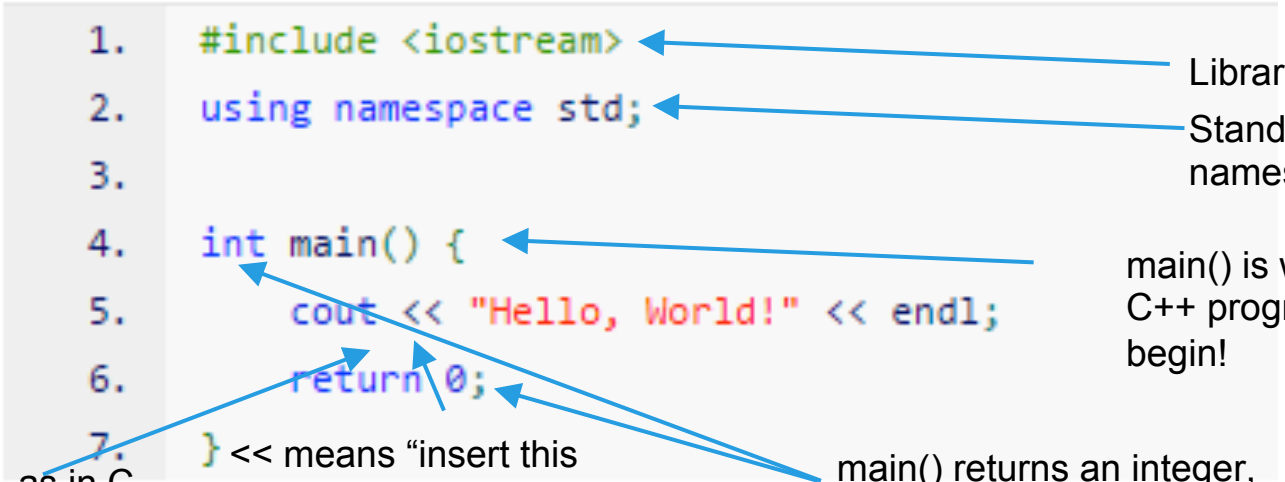
<< means “insert this into” - the direction of the arrows means it’s going into C-output.

main() returns an integer, which indicates whether the program succeeded or failed. 0 means success!

Hello, World!


1. #include <iostream>
2. using namespace std;
3.
4. int main() {
5. cout << "Hello, World!" << endl;
6. return 0;
7. }
Library inclusion
main() is where any C++ program will begin!
cout: “C-out”, as in C-language output - “put this on the screen”
<< means “insert this into” - the direction of the arrows means it’s going into C-output.
main() returns an integer, which indicates whether the program succeeded or failed. 0 means success!

Hello, World!


1. #include <iostream>
2. using namespace std;
3.
4. int main() {
5. cout << "Hello, World!" << endl;
6. return 0;
7. }
Library inclusion
Standard namespace
main() is where any C++ program will begin!
cout: “C-out”, as in C-language output - “put this on the screen”
<< means “insert this into” - the direction of the arrows means it’s going into C-output.
main() returns an integer, which indicates whether the program succeeded or failed. 0 means success!

Review from Lecture…

What is a variable?

- An element in code that has a specific *type*, and **holds or stores values**
- Name should be descriptive

```
int num_students = 40;
```

Data type ***Name*** ***Value***

What is a variable?

- **Always** initialize variables to 0 (or their initial values)
- To avoid undefined behavior/problems later!

- Make sure your variables aren't **reserved words** (see Lecture slides)

Cin and cout

- To include these in our program we need

  #include <iostream>

  at the top of our program

- This gives us access to the *iostream library* (where **cin** and **cout** come from)

Cin and cout

What does **cout** do?

Cin and cout

What does **cout** do?

→ Prints output to the standard output

Cin and cout

What does **cout** do?

→ Prints output to the standard output

What does **cin** do?

Cin and cout

What does **cout** do?

→ Prints output to the standard output

What does **cin** do?

→ Reads in user input from the keyboard

Cin and cout

What does **cout** do?

→ Prints output to the standard output

What does **cin** do?

→ Reads in user input from the keyboard

iostream = input/output stream

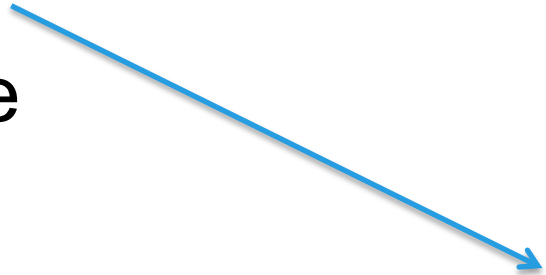
Data Types – Match Them

- int
- double
- char
- string
- bool

- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – Match Them

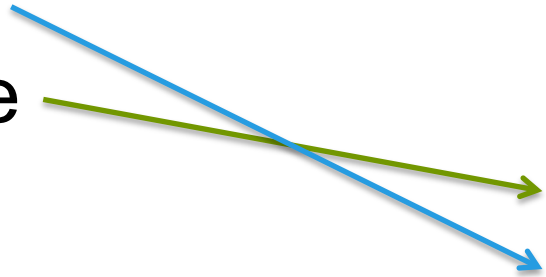
- int
- double
- char
- string
- bool



- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – Match Them

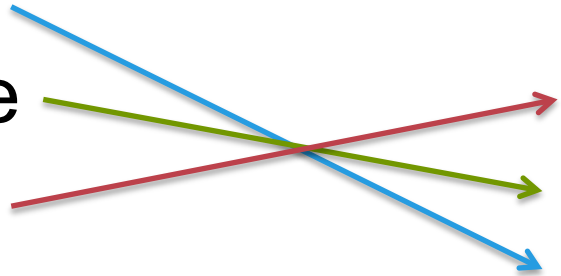
- int
- double
- char
- string
- bool



- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – Match Them

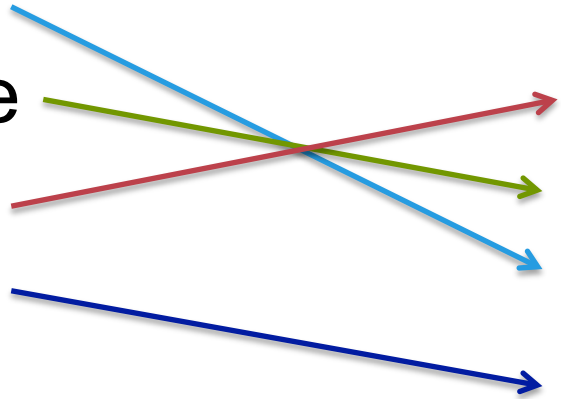
- int
- double
- char
- string
- bool



- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – Match Them

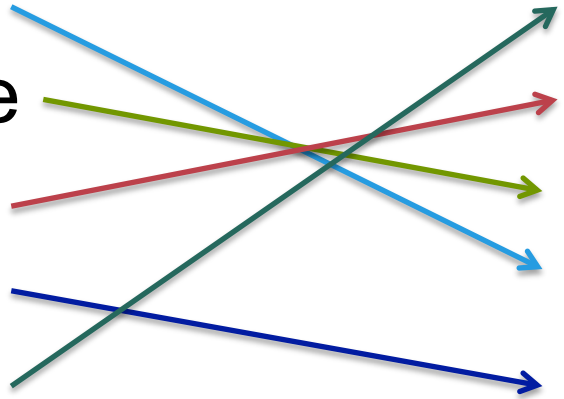
- int
- double
- char
- string
- bool



- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – Match Them

- int
- double
- char
- string
- bool



- True or false
- ‘a’, ‘!’
- 2.0
- 3
- “Hello, world!”, “a”

Data Types – A couple of notes…

- chars are denoted by **single** quotes
- ‘a’, ‘!’

- strings are denoted by **double** quotes
- “Hello, world!”, “a”

- Another way to say true/false is with 1 and 0
- True = 1, False = 0
- This is called **binary code**
- ***** Any number that isn’t 0 is also true***

Integer Division

When you divide two ints, the value is **truncated** (rounded down)

How to handle this? → use double type!

- What is 9 / 2?
- What is 9.0 / 2.0?
- What is 9 / 2.0?

Integer Division

When you divide two ints, the value is **truncated** (rounded down)

How to handle this? → use double type!

- What is 9 / 2? **4**
- What is 9.0 / 2.0? **4.5**
- What is 9 / 2.0? **4.5**
- Because double is higher in the type hierarchy than int, division occurs as if with two doubles (called **implicit type conversion** or **coercion**)

Type Casting

- Converting one type to another
- What is 11 / 2? → (think integer division)
- What is (double) 11 / 2?
- * Can also be written as double (11) / 2

Type Casting

- Converting one type to another
- What is 11 / 2? (think integer division) **5**
- What is (double) 11 / 2? → type casted to a double! **5.5**
- * Also can be written as double (11) / 2
- The (double) tells the compiler to treat 11 as type double, allowing double division
- **WARNING:** Do not write “double (11 / 2)” → does 11 / 2 first, which evaluates to 5, and then tries to cast to double (5 as a double is 5.0, not 5.5)

Operations and Logical Operators

+ - / * %

&& || !

(we'll talk about these three later)

Modulus %

What does this do?

Modulus %

What does this do?

Gives you the **remainder** of a division

What is 8 % 5?

Modulus %

What does this do?

Gives you the **remainder** of a division

What is 8 % 5? **3**

** Think “how many times does 5 go into 8?”
** It goes in **once**, and there are **3** left over

What is 5 % 8? → *tricky*

Modulus %

What is 5 % 8? **5**

** If the second number does not go into the first at all, you are left with the first value (you couldn’t do anything with it)

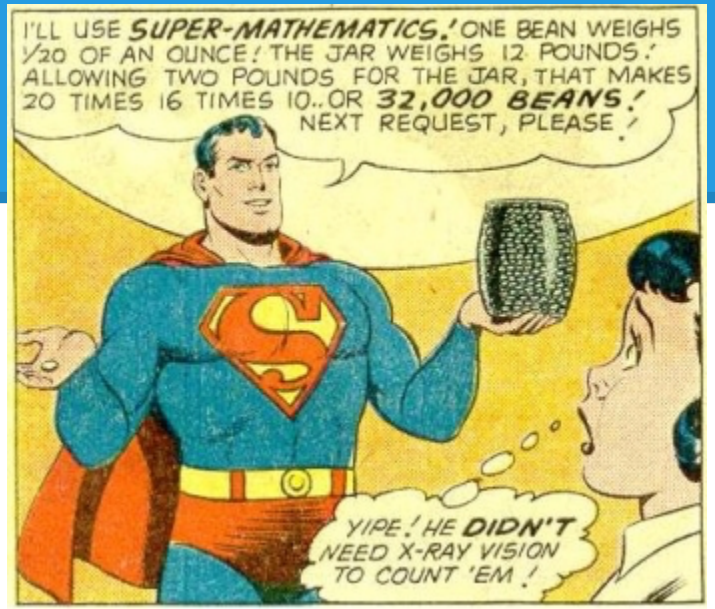
Magic Numbers – Avoid them!

- Magic number = unnamed number with special significance used in program for calculations

- **For example:**
- If your program involves a total number of bags of sugar (ex. 10)…
- Don’t use the integer 10 every time you want to use this!
- **Solution:**
- Make it a constant - a **const** variable!
- const int TOTAL_BAGS_SUGAR = 10;

Super-Mathematics!

```cpp
/*
    kryptonite isn't superman's only weakness
    calculate the number of beans in the jar
*/

#include <iostream>
using namespace std;

// My constants
const int OUNCES_PER_POUND = 16;
const int BEANS_PER_OUNCE  = 20;
const int JAR_POUNDS       = 2;

// calculate how many beans are in 1 lb
// ** use of magic numbers in this case would be:
// const int BEANS_PER_POUND = 20 * 16;
// ^^ DO NOT DO! ^^
const int BEANS_PER_POUND  = BEANS_PER_OUNCE * OUNCES_PER_POUND;

int main(void) {

    // includes jar weight and weight of beans
    int total_pounds = 0;

    // ask user for the weight of the jar (with the beans)
    cout << "Please enter the total weight in pounds: ";
    cin >> total_pounds;

    // calculate how much the beans weigh
    int bean_pounds = total_pounds - JAR_POUNDS;

    // calculate number of beans (based on weight), and print out
    cout << "Total number of beans: " << bean_pounds * BEANS_PER_POUND;

    return 0;
}
```



Definition of a Function

- A function is a *list of statements* that can be executed by referring to the *function's name*
- Functions often take in input values, do some work, and return a value that will be used by whoever called the current function
- There are **library functions** (functions that already exist in different libraries available to you in C++)
- There are also **user-defined functions** (functions you create to develop your program) – *more on these next lectures and discussion*

#include <cmath> functions

- cmath is the library
- #include is how you reference it in your project
- ceil: a function that returns the ‘ceiling’ of the input value
  - The ceiling of a number is the closest integer that is above that number
  - The ceiling of 3.6 is 4, ceil(5.2) is 6
- floor: ?

#include <cmath> functions

- ceil: a function that returns the ‘ceiling’ of the input value
  - The ceiling of a number is the closest integer that is above that number
  - The ceiling of 3.6 is 4, ceil(5.2) is 6
- floor: a function that returns the ‘floor’ of the input value
  - The floor of a number is the closest integer that is below that number
  - The floor of 3.6 is 3, floor(5.2) is 5

Strategy: plan first, code after!

- Figure out how you want to solve the problem → write it out and develop your algorithm

- Good question to ask: “In what order should I do things?” → a “to-do” list
- Break down the problem into smaller, simpler parts → don’t try to think about everything at once

- **For Project 1:**
- Try writing out your variables and their values **first** (you’ll have a lot)
- **Do not** use magic numbers!

Testing

- The purpose of testing is to write a few lines of code that clearly **show you whether your function works** as it should, or your code does what you expect it to do
- The tests should display your program or function's output (using cout)
- The *correct* or *expected* output should be displayed in a comment
- If the output and the expected differ, your test code caught a bug in your code

Testing Example

```
// write a few lines of code that add three numbers together and store them in a variable called sum
int x = 3;
double y = 4.3;
int sum = x + y;

//expected ouput: 7 (why?)
cout << sum << endl;
```

Testing Example

```
// write a few lines of code that add three numbers together and store them in a variable called sum
int x = 3;
double y = 4.3;
int sum = x + y;

//expected ouput: 7 (why?)
cout << sum << endl;
```

since sum is an integer, y is truncated (as occurred with double vs integer division) and then the two are added together

Advice for doing well in EECS 183

- Start projects **early** and go to Office Hours!
- You'll be one of the first in line ☺
- Plan before coding – save yourself a ton of time
- Post on Piazza (post and answer questions)
- Work with others! – talk through logic **out loud** and explain concepts you just learned

Please ask me any questions, and

good luck on Project 1!