Introduction to Software Development - CS 6010 Lecture 2 – Basic Programming

Master of Software Development (MSD) Program

Varun Shankar

Fall 2022

Lecture 2 – Basic Programming

- Topics
  - Hardware and Software
  - Variables and Datatypes
  - Lab – Road Trip Calculator
  - Lab – Variables and Expressions
  - Homework – Vending Machine

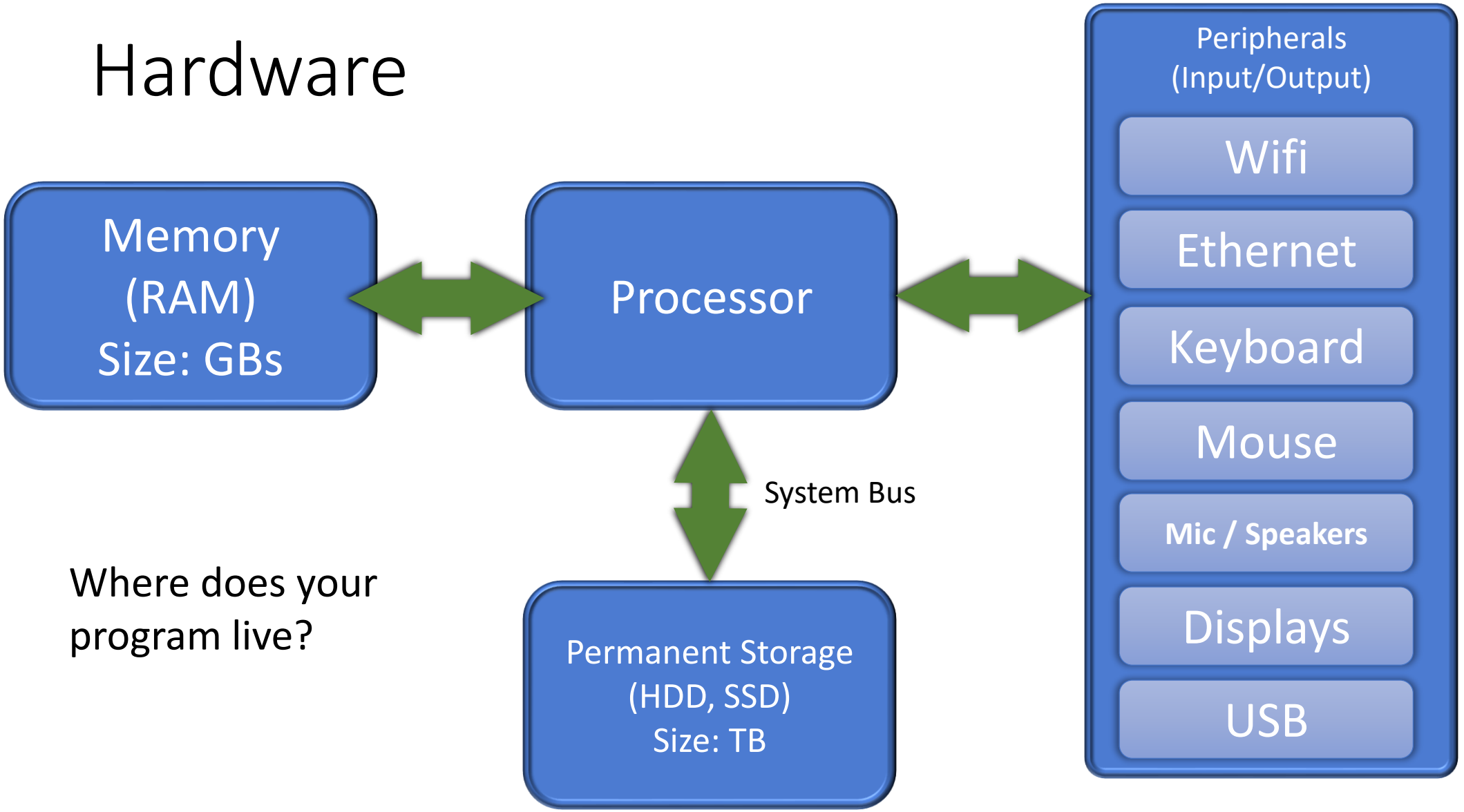
From Yesterday

- Error message: local repo is out of date.
  - Something changed on the remote side… did you edit your repo using the web browser? Did someone else edit (and commit) files to your repo?
  - git pull // pull down the updates to bring your local repo in sync with the remote.
- Committing HelloWorld not working
  - XCode created a “repo” inside your repo. Uncheck the box when creating a new project.
- Repo not found – Try regenerating your token.
  - To update existing GitHub keychain,
  - Open Keychain Access on your mac (you can use spotlight)
  - Search for github.com
  - Select the github.com keychain item
  - Edit or delete your GitHub account credentials.
- Other issues?

Hardware

- What parts of a computer can you name?
  - CPU
  - Memory – What are bits, bytes, words, kilobytes, mega, giga, tera, peta…
  - Peripherals
    - Hard drive, SSD
    - GPU
    - Keyboard, Mouse, Display
    - Network

Hardware



Where does your program live?

Software

- Operating System
  - OSX, Windows, Linux
- Programs
  - Programming Languagues
    - High Level: C++, C#, Java, Python

      Compiles to: (using gcc, clang, etc)
    - Low Level: Assembly

      Assembles to: (using an assembler but usually happens without your notice)
    - Lowest Level: Machine Code – “Programs”, “Binaries”, (”Apps” - Sigh)

      Binary – 1s, 0s that control electric signals

Algorithm

- An Algorithm is an unambiguous list of steps that tells us how to accomplish a goal. Examples:
  - a cooking recipe
  - directions to get to a restaurant
  - a computer program
- Programs are the implementation of “Algorithms”
- Algorithms (and programs, depending on the language they are written in) can be written at a very high (abstract) level, or be very detailed.

Algorithm Execution Rules

- Start from the Top
- Proceeds one line at a time
  - From Top to Bottom
- Finish each step before starting the next step.
- *Later on (but soon) we will add to this with repetition and branching…

Programs

- Programs consist of two things:
  - Data (the Variables)
  - Code (the Algorithm)

- Temporalness (Time of existence)
  - Data – Persists throughout the entire program*
  - Code – Exists only one line at a time*

A Program

```
/* Programs need comments that tell programmers what the
   program is doing. */

#include <iostream>  // Brings in / allows us to use someone else's code…
int main() {
        // The program goes here
        std::cout << "Hello World!" << std::endl; // << means "move data this
way"

} // Curly braces start and end blocks of code (in many programming
languages).
```

How do we quantify the *best s*olution (the best program)?

1. Least complex
2. Easy to debug / maintain / update
3. Fastest (may contradict above)

There is a trade off between time it takes for the developer to write, and how long it takes a user to use. As a professional code developer you should err on the side of minimizing the amount of time it takes the *user* to use your code.

Variables

- A *Variable* is the *name* that we (the programmers) agree to use to represent a specific piece of information. Most information in a program *changes* over time, hence the term “variable” (ie: changing).
- Variables must contain enough information “inside the computer” to represent facts about what is “outside the computer”.
- Variables are used to represent a specific piece of data.
  - int number_of_students = 44;

Variable Example (Assigning a Value)

- `<type> <Name> [= Value]; // <- Memorize this!`
- `float gpa = 3.7;`
- Is this a good variable name?
- What does it tell us?
- Can the value change as the program executes?
  - Yes, and almost always does.
- Can the name of the variable change?
  - No!
- What is the ‘;’ (semi-colon) for?
  - Tells the compiler that a line of code is ending

Datatypes – The type of data a variable represents.

- The computer only knows about the following 5 Data Types*,***:
- Basic (3 types – store a single value per variable)
  - Numbers** (1, -2, 3.14, 99.9999999)
  - Characters (‘A’, ‘z’, ‘!’) % notice the ‘’ (quotes)
  - Booleans (true/false)
- Combinations (2 types – store multiple pieces of information in a single variable)
  - Arrays (lists of any <u>one</u> type of data – eg: arrays of characters, array of numbers, etc. [9, 5, 18, -3])
    - We use [ ] for arrays.
  - Structures/Objects (groupings of many data types into a single variable)
    - (eg: car. A car has mpg, number of doors, current speed, amount of gas, make, model, etc, etc)

* There are actually more types, but we will focus on these 5 for now.

** Technically numbers are broken down into several sub-categories – see next slide.

*** Even more technically a computer only knows 1s and 0s – grouped into bits, bytes

Numbers

- 1, 3.1415, -50, 2.4
- I'll use the term *Numbers* (for now) but we really need to know:
  - Integers (Whole Numbers)
    - Examples: 1, -5, 3002, 0
    - Signed / Unsigned, # of bits (8, 16, 32, etc)
  - Floats (also known as *singles*)
    - Have a decimal
    - Examples: 2.4, 3.141
    - Note: 1.0 is still a float
  - Doubles (2x more precise Float)
    - Have a decimal, twice as many digits as a float
    - Example: 2.7182818284590452

Characters

- Everything on the keyboard (plus some)
- alphabet: ‘a’, ‘b’, …, ‘z’, ‘A’, ‘B’, …, ‘Z’
  - Note that ‘a’ is not the same as ‘A’
- digits: ‘1’, ‘2’, ‘3’,… (these are NOT numbers – pay attention to the quotation marks!)
- punctuation, math symbols, *space ‘ ‘*, tab
  - Question: How many *spaces* in a tab?
- *new line*, *carriage return*, etc.
  - Note, some characters you can’t “see”.
- What is more than one character together called? How to denote?
  - A string [of characters]
  - “Hello World” – Notice the double quotes!

Booleans

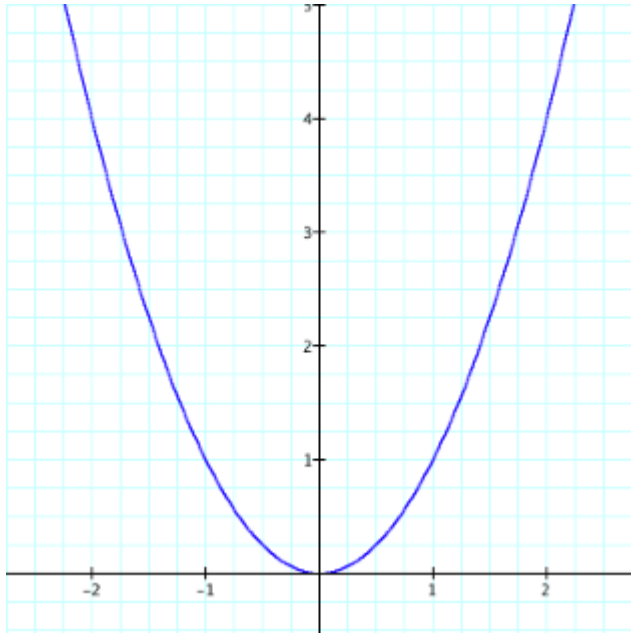
- Variables with a value of `true` or `false`
- **Boolean variables "answer a question"**
- How are Boolean values stored inside the computer?
- Note to programmers: Do NOT use 1 or 0 even though the computer does internally. (Why?)

```
money = 0; // BAD!! What type is this variable?
```

vs

```
has_money = false; // Good! Using Boolean, also better variable name!!
```

Let's make sure we know what '=' means...



- In *math*, '=' means equality.
  - Y = X^2
- In programming, '=' means assignment.
- `X = 3  // Some languages actually use: X <= 3`
  - Take the value 3, and place it into the variable X (so that we can use the value at some point in the future).
- `X == 3 // What does this mean?`
  - Checks for equality – this is a *Boolean statement* that can evaluate to one of two values:
    - **`true`** or **`false`**

Mixing data types

- float numStudents = 7; // Issues with this variable?
  - What value is numStudents actually given (inside the computer)?
    - 7.0
  - In most cases, you won't have a fractional student… so what type should numStudents be?
  - int
- int numStudents = '3'; // What is '3' vs 3?
  - '3' is a character
  - Syntax error? Well not exactly, as the compiler will actually allow this.
  - Why does the compiler allow this??
- int gpa = (3 + 4) / 2; // What is the *value* of the variable gpa?
  - 3 // Those numbers are all integers, so the computer does integer math.
- int gpa = (3 + 4) / 2.0; // What is the *value* of the variable gpa now?
  - Still 3. Note (3+4)/2.0 itself is a float.

Let's make sure we've got this…

- `course_gpa = quiz_average + test_average;`
  - What are `course_gpa, quiz_average, and test_average`?
    - Variables – they are storing data we wish to process.

- `course_gpa = A;`
  - What is `A`?
  - Could be: `course_gpa = 'A'; // Assuming course_gpa is what type?`
  - Tick marks distinguish between variables and characters

- What does the "=" above mean?
  - Assignment of a value into the given variable.

How variables are stored in the computer

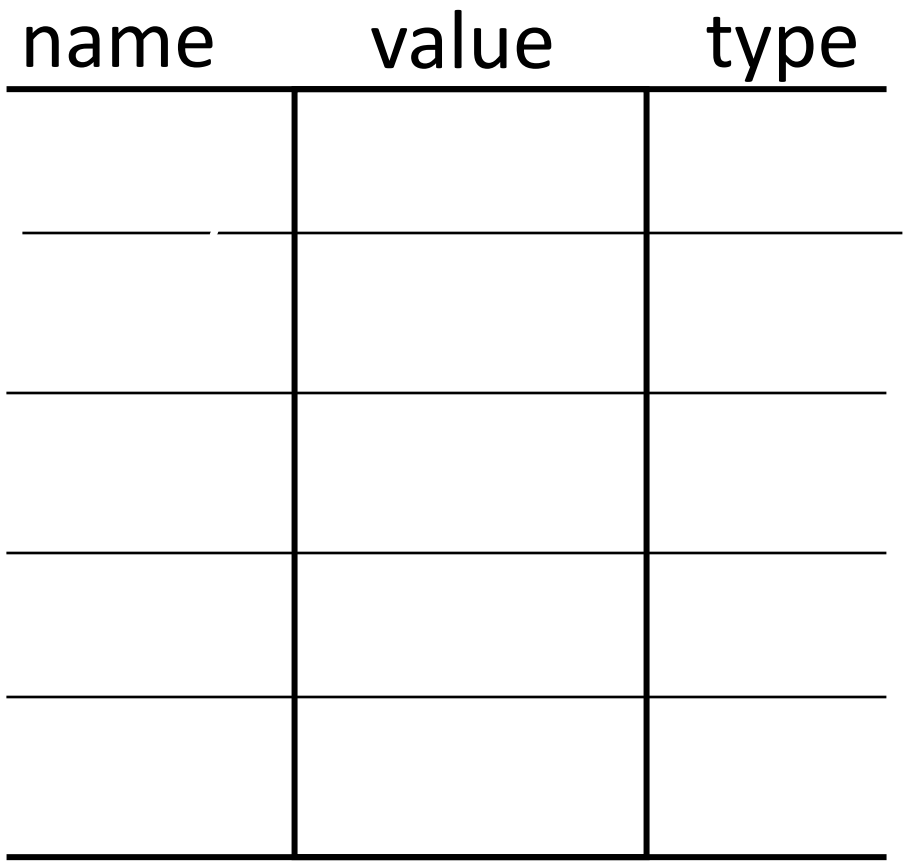
- Where is the data stored?
  - RAM (Memory)
- How can a programmer visualize the storage of data (while writing their program)?
  - Ladder Diagram
  - This is a “On paper” representation of the data in our program
    - Quick question: How is data stored in a program?
      - With variables!
  - Remind us of what three things we need to consider when creating a variable
    - Type, Name, Value
  - Ladder diagrams additionally help us keep track (again, on paper) of what is happening to our data as the program executes.

Ladder Diagram Example

Ladder Diagram

Each “Row” represents one “piece of information” known by the computer.

One “Variable”!



Creating Variables (in C++)

- <type> <name> [= value]; // Do you remember what <> and [ ] mean?
- What types have we seen so far?
  - int, float, bool, char, etc
- What is a valid name for your variable?
  - Technically, almost any name.
  - In practice, want a name that conveys some information.
    - number = 3; // BAD!!! We already know 3 is a number
    - number_of_apples = 3; //better
  - Starts with a character (letter), can contain letters, digits, _ (underscores)
    - No spaces or other punctuation
  - Variable names ARE CASE SENSITIVE!
    - name and Name are different variables!
  - camelCaseNames vs snake_case_names

Logic vs. Syntax Errors

- Syntax is like grammar and spelling
  - Syntax errors are easily caught by the computer. Given:

```
int numSeniors = 20;
int numJuniors = 35;
```

  - What syntax errors are in the following line of code?

```
ints totalNumStudents = numSeniors ‘+’ numJuniors
```

- Logic is how things work
  - Logic errors are almost never caught by the computer.
    - Example – what is wrong with the following line of code?

```
int totalNumStudents = numSeniors + numJuniors;
```

Looking at basic I/O

- But first, pop quiz: How do you access the data (value) stored in a variable?
  - Just write down the variable's name.
- I/O – What does this stand for?
  - Input / Output
- We use a system library to do I/O
  - It knows how to do it on a given operating system.
  - Makes it easier for us. [That is the point of using libraries.]
- IoStream
  - namespaces – std::
  - cin – read in some data [from the user]
  - cout – display some data [to the user]

#include <iostream>

- std::cout
  - Pronounced “standard C out” or “std C out”
  - “std::” means the function comes from the standard namespace
  - cout stands for “console output”
- Remember, to use functions from a library, you must #include the header file that contains the library.
- For brevity, I will (and you can with a little magic) just write “cout”.
- cout << “Hello, the number of students in this class is: “ << numStudents;
- Think of “cout” as the screen, and the arrows (<<) are pointing the information to the screen.
- To make the output move to the next line, you can use std::endl.
  - end line
  - cout << “Hello” << endl;
  - Note, I’ll probably use “\n” (the magic newline character instead of endl)

std::cin

- cin is the opposite of cout (console input)
- It allows the program to ask the user to input data (which will then be processed)
- int age;
- std::cin >> age; // Reads a number from the console.
- Notice the arrows (>>) again point to where the data is going… in this case into the variable age.

Simple I/O Example Program

```
#include <iostream>  // In order to use cin/cout/endl, we must include the iostream library.

int main() {

        int i;   // What is the value of i right now?  Why is this named i?
        std::cin  >>  I;   // Request that the user type in a number
        std::cout  <<   “The user entered the number “ << i << std::end;   // “\n”

        return 0;
}
// Did you catch the syntax error above? (I vs i)
// How to read in two variables?
// std::cin >> var1 >> var2;
```

Math

- A CPU actually is a bunch of circuits that only do one thing:
  - Math
  - But it is very fast at it
- In C++ (and most languages), the mathematical operators (+, -, *, /) work just like you would expect*.
  - Order of operations work just like what you learned in school
- However, there are a few exceptions:
  - ^ does not mean “raise to the power” (in C++) [It’s bitwise Xor]
  - % is used to mean “modulo” – it returns the remainder after doing a division
    - 10 % 3 == 1
    - 10 % 4 == 2

% and / with integers

- In C++, if you divide two integers, you get an integer back
  - 1/2 gives you 0 (why)
  - 11/3 gives you back 3
- Integer division “truncates,” meaning any fractional part gets thrown away
  - 99/100 == 0
- The % (modulus) operator gives you the remainder after dividing
- For positive ints, a and b, the following is true:
  - int x = a / b; // remember, it rounds down
  - int y = a % b; // this is the part that got “rounded down” in the line above
  - a = x * b + y
- For example:
  - int a = 10; int b = 3; // You can put two lines of code on the same line - separated by ;
  - int x = a / b; // x is 3
  - int y = a % b; // y is 1 because 10 divided by 3 has a remainder of 1
  - int c = b * x + y; // c is 10, the same as a

A few more ‘magic’ operators

- +=, -=, *=, /=
  - x += 4; // This means x = x + 4;
  - x -= 4; // Means x = x – 4; You can remember it is not x =- 4 because this would be x = -4;
- ++, --
  - Increment (++), and Decrement (--) operators.
  - Usually applies only to integers.
  - x = 4;
  - x++; // Now the value of x is 5.
  - Note ++x works too (changes the value of x to 5), but has slightly different semantics to be discussed at a later date.

Today's Assignment(s)

- Lab – Road Trip Calculator
  - ~10:40-11 – 20 minutes on your own
  - Then we will write it together
- Group Lab – Variables and Expressions
  - ~11.10 AM – Noon
  - TAs and I will be around to help until noon.
  - TA office hours today. TAs will be in 3255.
- Labs should usually be finished in class (by noon), but can be turned in on Canvas anytime on the day they are assigned.

- Homework – Vending Machine
  - Due tomorrow before class.