Programming Primer

- You've now entered and run your first program
- The program doesn't do much, and you probably don't fully understand everything you typed, but we all have to start somewhere
- These notes give you a general overview of elements that are common to most programming languages
- Through all this, remember the heart of what we're doing: programming is a way to describe how to get something done (i.e., how to perform an algorithm)

Basic Concepts

- A *program* is a sequence of *characters* (letters, numbers, punctuation, maybe other symbols) that you type into the computer
- This sequence of characters is expected to conform to a particular *programming language* — a well-defined set of rules and constructs that determine the range of activities that your program can do
- In some ways, a programming language is a "contract" between you and the computer: "If the program says this, then the computer will do this"

Running a Program

- Recall that ultimately, the computer operates solely on bits, both in terms of the instructions that it can perform (machine language) and the information that it manipulates (registers, main memory, mass storage)
- The program that you type — frequently referred to as *code* or *source code* — must be *translated* into this machine language
- This translation is performed by *another* program, which may take on different names: *interpreter*, *compiler*, *virtual machine*, to name a few

- Upon translation, the computer can now *run* your program — that is, perform the tasks that you specified in the source code
- Typically, the tasks you need to perform — the *algorithms* — take in some information (*input*) then provide the result of the work performed (*output*)
- At any point in this process, something might go wrong; these *errors* fall into a broad range of categories:
  - You might have written something that doesn't conform to the programming language
  - You might have provided bad input (e.g., you might have asked the computer to divide by zero, or your might have provided a word where the program expected a number)
  - The algorithm itself is incorrectly specified (e.g., you might have meant for your program to perform an addition, but you mistakenly performed a subtraction)
  - And many more — the range of possible errors is virtually limitless

What Goes in a Program?

- Most programs consist of an ordered sequence of *definitions* (telling the computer what something *is*) and *statements* (telling the computer what to *do*)
- The computer goes through this sequence, well, *sequentially* (or step-by-step) and performs whatever the definition or statement expresses, as stipulated by the programming language that is being used
- A programming language provides several *constructs* that help control the order in which steps are performed, including repetitions and decision-making

- Most programming languages also provide a set of *built-in* definitions or statements — entities that the computer either "knows about" or "knows how to do" automatically, just by virtue of using that particular programming language
- A few other key elements that most programs, regardless of language, have:
  - *Comments* are notes that you make for yourself or for others who will be looking at your program; they are delineated in such a way that the computer will know to ignore them
  - *Delimiters* help specify when certain items, such as definitions and statements, start or stop — unlike human beings, computers need very explicit indicators of when something begins or ends…it's like requiring absolutely precise punctuation
  - *Spacing* and *indentation* refers to how a program is "formatted," in terms of blank lines or spaces — you typically have free reign over this, and generally shoot for clarity and readability, making it easier for fellow human beings to assimilate and understand what your source code is trying to do

From Algorithm to Program

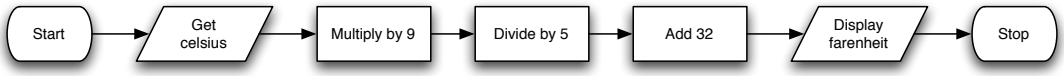
- As mentioned before, computer science has two primary elements: the study of algorithms, and the process of making a machine perform this algorithm
- Let's now do this in a concrete manner, by having a machine perform a particular algorithm
- First, let's state *what* this algorithm does, before figuring out *how* the algorithm does it:

  We want to convert a temperature reading from degrees Celsius to degrees Farenheit

- Knowing the "what," for this particular algorithm, it isn't hard to state the "how" in English:
  - Multiply the Celsius reading by 9
  - Divide it by 5
  - Add 32 to the result
- Of course, there's more than one way to express this algorithm; mathematically, it is quite compact:

$$f = (9c / 5) + 32$$

- We can also use *flowchart* notation:



- In many ways, a program is simply yet another way to express an algorithm

Determining What the Algorithm Needs

Before we get to the program, let's look at what the algorithm needs in order to do its work:

- It needs "places" to store the information that it's working on — in programming languages, these are almost always called *variables*
- It needs to be able to multiply, divide, and add numbers — these are examples of *operations*
- Finally, we need to be able to display the result of the computer's hard work: this is part of a language's *input/output*, or *I/O*, capabilities

Example: JavaScript

Here is our Celsius conversion algorithm written in the *JavaScript* programming language, which comes with all modern Web browsers:

```
// Place the Celsius reading in c.
var c = 4;

var temp = c * 9;
temp = temp / 5;

var f = temp + 32;

// Display the result.
alert(c + "C is " + f + "F.");
```

One way to specify a comment is through two slashes — when the computer encounters this, everything else until the end of the line is ignored

Variables in JavaScript are "declared" using the var *keyword, followed by a name, then an optional initial value*

**, /, and + represent multiplication, division, and addition, respectively*

JavaScript uses semicolons as statement delimiters

alert() *is a built-in* function *that displays the value of whatever is in between its parentheses*

Where We Go From Here

- Converting Celsius to Farenheit is fairly simple — and as written, it's *really* simple, because we have to modify the program to convert a different Celsius reading!
- As we venture further into the course, the algorithms that we will try to ask the computer to perform will grow in sophistication and complexity
- We will see how modern programming languages try to express the full range of "algorithmic needs" by providing additional features that allow us to implement even the most complicated of algorithms

Example: Java

For immediate comparison, here is the same algorithm written in the *Java* programming language:

```
public class CelsiusConverter {
  /**
   * Performs the conversion and displays the result.
   */
  public static void main(String[] args) {
    double c = 4.0;
    double temp = c * 9.0;
    temp = temp / 5.0;
    double f = temp + 32.0;
    System.out.println(c + "C is " + f + "F.");
  }
}
```

Java requires a construct called a class *within which all code must reside*

Everything between "/" and "*/" is a comment*

Whoa, what's all this??? We'll get to them gradually

Curly braces ("{ }") delimit classes and methods

System.out.println() *performs the output duties here*