Table of Contents

Course Overview

Procedural programming is the process of solving programming challenges by breaking large problems into smaller ones. These sub-problems are called procedures.

Goals

The goal of this class is that each student will be able to solve problems in C++ and have a solid foundation in software development methodology. By the end of the semester…

- You will be well prepared for CS 165 and the other course in computing majors. This class is the foundation on which all Computer Science (CS), Electrical Engineering (EE), and Electrical & Computer Engineering (ECEN) courses are based.
- You will have confidence in your ability to solve problems with C++. This class will include tons of opportunities for hands-on programming. By the end of the semester, you will have written more than thirty programs.
- You will possess a tool-bag of different approaches to solve software problems. Not only will we learn the mechanics of the C++ programming language, we will also learn how to solve programming problems using procedural tools.
- You will have a firm foundation in the basic constructs of procedural C++. All the components of procedural programming (except structures) will be learned in CS 124. Object Oriented Programming, the second half of the C++ language, is the topic of next semester.

These goals will be explored in the context of C++ using the Linux operating system.

Course Layout

The course will be broken into four sections:

1. **Simple Programs**. How to write your first program and begin thinking like a programmer. At the end of this section, you will be able to write a program with multiple functions, IF statements, and complex mathematical operations.
2. **Loops**. The goal of this section is to be able to solve problems using loops. Additionally we will learn several tools enabling us to solve hard programming problems and manage programs with large amounts of complexity.
3. **Arrays**. Next we will learn how to handle and manipulate large amounts of data in increasingly complex data structures. Specifically, we will learn about arrays, pointers, and file I/O.
4. **Advanced Topics**. The final section of the course will involve learning about a collection of specialized topics (dynamic memory allocation, multi-dimensional arrays, jagged arrays, and instrumentation to name a few) in the hopes they will help us understand the main topics of the semester better. In other words, the goal is not to learn these specialized topics so much as to make sure we have nailed the fundamentals.

How to Use This Textbook

This textbook is closely aligned with CS 124. All the topics, problems, and technology used in CS 124 are described in detail with these pages.

Sam and Sue

You may notice two characters present in various sections of this text. They embody two common archetypes representative of people you will probably encounter in industry.

Sue's Tips

Sue is a pragmatist. She cares only about getting the job done at a high quality level and with the minimum amount of effort. In other words, she cares little for the art of programming, focusing instead on the engineering of the craft.

Sue's Tips tend to focus on the following topics:

- Pitfalls: How to avoid common programming pitfalls that cause bugs
- Effort: How to do the same work with less time and effort
- Robustness: How to make code more resistant to bugs
- Efficiency: How to make code execute with fewer resources

Sam's Corner

Sam is a technology nerd. He likes solving hard problems for their own sake, not necessarily because they even need to be solved. Sam enjoys getting to the heart of a problem and finding the most elegant solution. It is easy for Sam to get hung up on a problem for hours even though he found a working solution long ago.

The following topics are commonly discussed in Sam's Corner:

- Details: The details of how various operations work, even though this knowledge is not necessary to get the job done
- Tidbits: Interesting tidbits explaining why things are the way they are

Neither Sue's Tips nor Sam's Corner are required knowledge for this course. However, you may discover your inner Sam or Sue and find yourself reading one of their columns.

Needing Help

Occasionally, each of us reaches a roadblock or feels like help is needed. This textbook offers two ways to bail yourself out of trouble.

If you find you are not able to understand the problems we do in class, work through them at home before class. This will give you time for reflection and help ask better questions in class.

If you find the programming problems to be too hard, take the time to type out all the examples by hand. Once you finish them, work through the challenge associated with each example. There is something about typing code by hand that helps it seep into our brains. I hope this helps.

Computers & Programs

Sam is talking with an old high school friend when he find out he is majoring in Computer Science. "You know," said his friend, "I have no idea how a computer works. Could you explain it to me?" Sam is stumped for a minute by this. None of the most common analogies really fit. A computer is not really like a calculator or the human brain. Finally, after much thought, Sam begins: "Computer programs are like recipes and the computer itself is like the cook following the instructions…"

Objectives

By the end of this class, you will be able to:

- Identify the major parts of a computer (CPU, main memory, etc.).
- Recite the major parts of a computer program (statements, headers, etc.).
- Type the code for a simple program.

Overview of Computers and Programs

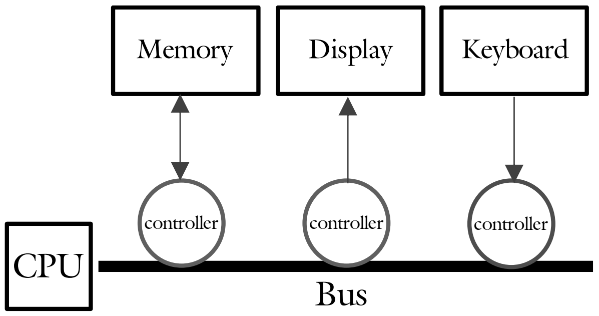
Consider a simple recipe for making cookies. The recipe consists of the ingredients (the materials used to make the cookies) as well as the instructions (such as "stir in the eggs"). If the recipe was precise enough then any two cooks would be able to produce an identical set of cookies from the provided ingredients. Most recipes, however, are ambiguous. This requires the cook to improvise or use his best judgment. This analogy holds well to the relationship between computers and programs with one exception: the computer is unable to improvise:

- **Cook**: The cook is the computer executing the instructions.
- **Recipe**: The recipe is the computer program to be executed.
- **Ingredients**: The ingredients are the input to the program.
- **Cookies**: The cookies correspond to the output of the program.
- **Chef**: The chef is the person who came up with the recipe. This is the role of the programmer.

Your job this semester is to derive the recipe (computer program) necessary to produce the desired cookies (output) from the provided ingredients (input).

Computers

Computers today are all based on the basic model developed by Dr. John von Neumann. They consist of a CPU (the part that executes instructions), the Memory (the part where programs and data are stored), the bus (the wire connecting all components), and the peripherals (input and output devices).



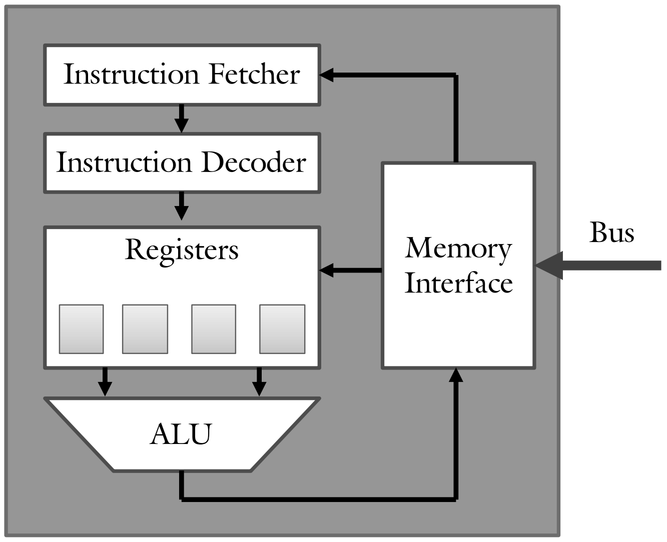
Please view the following movie to learn more about the parts of a computer:

[Parts of a Computer]

Possibly the most important part of a computer is the CPU. This has several components: the memory interface (operating much like a controller, it puts data on the bus and takes data off the bus), instruction fetcher (determines which instruction is next to be executed), the instruction decoder (interpreting what a given instruction does), the registers (keeping temporary data), and the ALU (Arithmetic Logic Unit, performing math and logical operations). Please view the following movie to learn more about the parts of a computer:

[The CPU]

Finally, programs consist of simple instructions that are executed by the CPU.



The following is a sample instruction set (real CPUs have hundreds of instructions but do essentially the same thing):

Name	Opcode	Description	Example
`NOOP`	0	Do nothing	`NOOP`
`LOAD`	1	Load a value from some memory location into the register	`LOAD M:3`
`SET`	2	Set the register to some value	`SET 1`
`SAVE`	3	Saves the value in a register to some memory location	`SAVE M:10`
`JUMP`	5	Sets the Next Instruction value to some memory location	`JUMP M:0`
`JUMPZ`	6	Same as JUMP except only sets the Next Instruction value if the register is set to zero	`JUMPZ M:0`
`ADD`	8	Adds a value to the register	`ADD 1`
`SUB`	9	Subtracts a value from what is currently in the register	`SUB 1`
`MULT`	10	Multiplies the current value in the register by some value	`MULT 2`
`DIV`	11	Integer division for some value in the register	`DIV 2`
`AND`	12	Returns 1 if the value and the register are both non-zero	`AND 1`
`OR`	13	Returns 1 if the value or the register is non-zero	`OR 1`
`NOT`	14	Returns 1 if the value in the register is zero	`NOT`

Please view the following movie to learn about how these instructions can be used to make a program:

Programs

The last item is an emulator (a program that pretends it is something else. In this case, it emulates the simple computer presented in the previous movies).

Emulator

To run a program:

1. Type or paste the program in the large edit control below the "Load Program" label. Make sure there is one instruction per line or it will not work correctly. You may also need to delete an extra space.
2. Press the Load button. This will load the program into memory and set the "Next Instruction" pointer to zero.
3. Step through the program, one instruction at a time, by pressing the Go button.

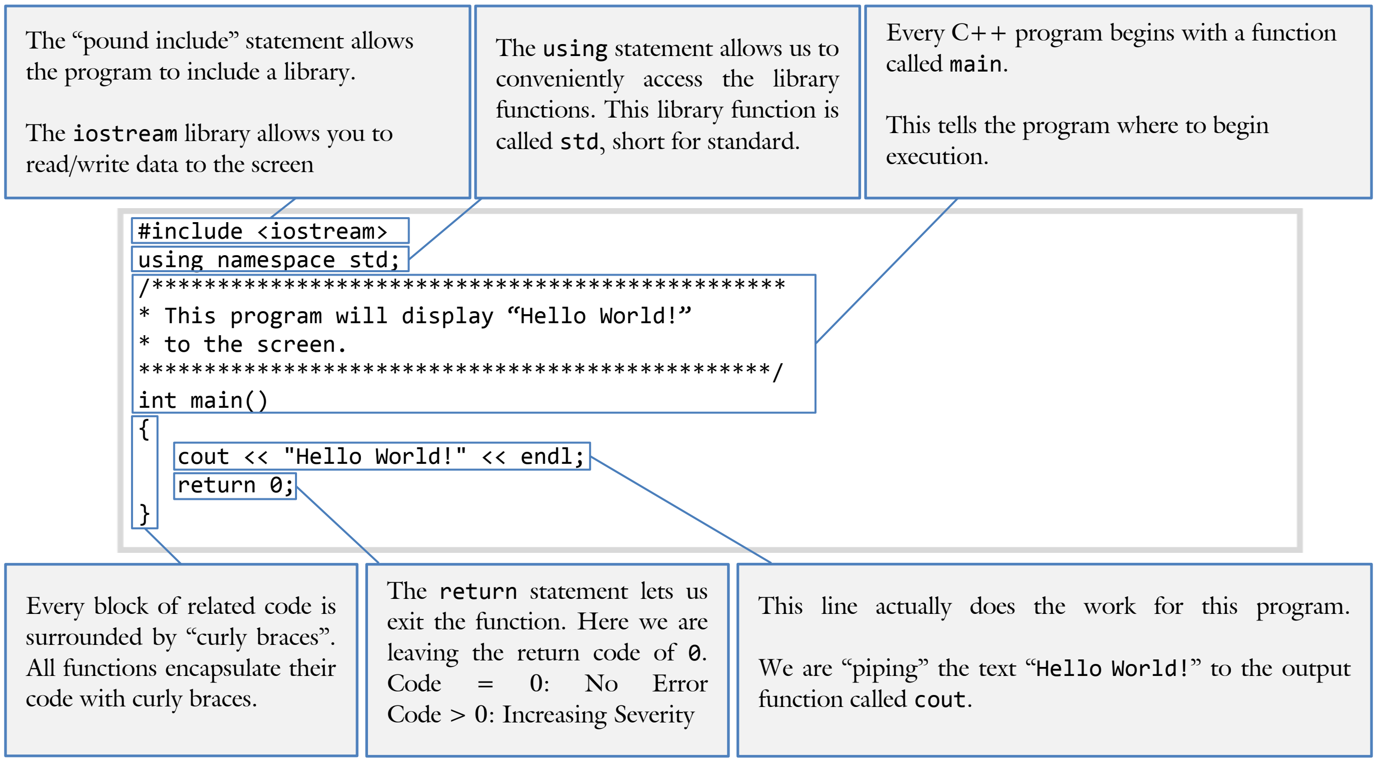
A couple programs you may want to run… what will they do?

```
SET 0xff
SAVE D:0
SET 0x00
SAVE D:0
JUMP M:0
```

```
SET 0xff
SAVE D:0
LOAD M:3
ADD 1
SAVE M:3
JUMP M:0
```

Programs

There are many computer languages, each enabling the programmer to write a computer program. In this class, we will be using the C++ language developed by Bjarne Stroustrup in the early 1980's. A C++ program consists of two parts: the header and the functions. The header describes what tools will be used in the program and the functions contain the recipes themselves. The functions consist of individual instructions called statements:



The "pound include" statement allows the program to include a library.

The `iostream` library allows you to read/write data to the screen

The `using` statement allows us to conveniently access the library functions. This library function is called `std`, short for standard.

Every C++ program begins with a function called `main`.

This tells the program where to begin execution.

```
#include <iostream>
using namespace std;
/************************************************
* This program will display "Hello World!"
* to the screen.
***********************************************/
int main()
{
   cout << "Hello World!" << endl;
   return 0;
}
```

Every block of related code is surrounded by "curly braces". All functions encapsulate their code with curly braces.

The `return` statement lets us exit the function. Here we are leaving the return code of `0`. Code = 0: No Error Code > 0: Increasing Severity

This line actually does the work for this program.

We are "piping" the text "`Hello World!`" to the output function called `cout`.

IOSTREAM: This simple program has two parts to the header. The first part indicates that the `iostream` library will be used. The `iostream` library contains many useful tools that we will be using in every program this semester, including the ability to display text on the screen. There are other libraries we will be introduced to through the course of the semester, including `iomanip`, `cassert`, and `fstream`.

NAMESPACE: The second line of code in the header indicates we are to be using the shorthand version of the tools rather than the longer and more verbose versions. This semester we will always have "using namespace std;" at the top of our programs, though in CS 165 we will learn when this may not be desirable.

CURLY BRACES: All the instructions or statements in a C++ program exist in functions. The `{}`s (called "curly braces") denote where the function begins and ends. Most programs have many functions, though initially our programs will only have one. Back to our recipe analogy, a program with many functions is similar to a recipe having sub-recipes to make special parts (such as the sauce). The functions can be named most anything, but there must be exactly one function called `main()`. When a program begins, execution starts at the head of `main()`.

COUT: The first instruction in the program indicates that the text "Hello world" will be displayed on the screen. We will learn more about how to display text on the screen, as well as how to do more complex and powerful tasks later in the semester (Chapter 1.1).

RETURN: The last statement in this example is the return statement. The return statement serves two purposes. The first is to indicate that the function is over. If, for example, a return statement is put at the beginning of a function, all the statements after return will not be executed. The second purpose is to send data from one function to another. Consider, for example, the absolute value function from mathematics. In this example, the output of the absolute value function would be sent to the client through the return mechanism. We will learn more about the use of return later in the semester (Chapter 1.4).

Comments

Strictly speaking, the sole purpose of a recipe is to give the cook the necessary information to create a dish. However, sometimes other chefs also need to be able to read and understand what the recipe is trying to do. In other words, it is often useful to answer "why" questions (example: "Why did we use whole wheat flour instead of white flour?") as well as "how" questions (example: "How do I adjust the recipe for high elevation?"). We use comment to address "why" questions.

Comments are notes placed in a program that are not read by the compiler. They are meant to make the program more human-readable and easier to adjust. There are two commenting styles in C++: line and block comments.

Line comments

Line comments indicate that all the text from the beginning of the line comment (two forward slashes `//`) to the end of the line is part of the comment:

```
{
   // line comments typically go inside functions just before a collection
   //     of related statements
   cout << "Display text\n";        // You can also put a line comment on the end
                                    // of a line of code like this.
}
```

Some general guidelines on comments:

- Comments should answer "Why" and "How" questions such as "Why is this variable set to zero?" or "How does this equation work?"
- Comments should not state the obvious.
- When there are multiple related statements in a function, set them apart with a blank line and a comment. This facilitates skimming the function to quickly gain a high-level understanding of the code.

Block comments

Block comments start with `/*` and continue until a `*/` is reached. They can span multiple lines, but often do not. We typically use block comments at the beginning of the program:

```
/***********************************************************************
* Program:
*    Assignment 10, Hello World
*    Brother Helfrich, CS124
* Author:
*    Sam Student
* Summary:
*    This program is designed to be the first C++ program you have ever
*    written. While not particularly complex, it is often the most difficult
*    to write because the tools are so unfamiliar.
************************************************************************/
```

All programs created for CS 124 need to start with this comment block. Observe how the comment blocks starts with the `/*` at the top and continues until the `*/` on the last line. We will start all of our programs with the same comment block. Fortunately there is a template provided so you won't have to type it yourself.

We also have a comment block at the beginning of every function. This comment block should state the name of the function and briefly describing what it is designed to do. If there is anything unusual or exceptional about the function, then it should be mentioned in the comment block. The following is an example of a function comment block:

```
/**********************************************************
 * MAIN
 * This program will display a simple message on the screen
 **********************************************************/
```

Problem 1

Which part of the CPU performs math operations?

Answer:

____

Please see page 5 for a hint.

Problem 2

If a program is like a recipe for cookies then which of the following is most like the data of the program?

- Ingredients
- Instructions
- Measurements
- Chef

Please see page 4 for a hint.

Problem 3

What is missing from this program?

```
#include <iostream>

int main()
{
   cout << "Howdy\n";

   return 0;
}
```

Answer:

____

Please see page 7 for a hint.

Problem 4

Which of the following is a function?

```
int main()
```

```
#include <iostream>
```

```
return 0;
```

```
using namespace std;
```

Please see page 7 for a hint.

Problem 5

What is wrong with this program?

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Howdy\n";
}
```

Answer:

____

Please see page 7 for a hint.

Assignment 0.2

In I-Learn, please answer the following questions:

1. What does the following assembly program do?

```
SET 0xff
SAVE D:0
SET 0x00
SAVE D:0
JUMP M:0
```

2. What does the following assembly program do?

```
SET 0xff
SAVE D:0
LOAD M:3
ADD 1
SAVE M:3
JUMP M:0
```

3. Write the code to put "Hello world" on the screen:

4. What is the purpose of comments in a program?

5. Give an example of all the ways to write a comment in C++: