about 3368 words
[Street Address]
JBHT 525
479-575-7264
wingning@uark.edu

PROGRAMMING FOUNDATIONS I CLASS NOTES

by

Wing Ning Li

Introduction

Most of you taking this class have the experience of using a computer. For example, smart phones, tablets, laptops, and desktops are all computers. A computer consists of two main parts: hardware and software. For example, keyboards, screens, cameras, hard drives and microprocessors are hardware. Hardware has physical characteristics such as weight and volume. Software is a computer program that allows user to control the hardware to accomplish useful tasks. Unlike hardware, software is not physical but logical. It does not have weight or mass in itself and it is intangible. We may think about software as a sequence of instructions telling the hardware what to do.

Programming Languages are invented for the construction of computer software (Apps). A computer program written in a particular programming language consists of a sequence of statements. Since it is a sequence, there is the first statement, the second statement, and so on until the last statement. Conceptually we may view each statement as one instruction to instruct hardware what to do. Technically a statement of a high level programming language such as the one we will study is translated by a compiler into one or more hardware instructions. The compiler hides from us the messy details of the hardware and makes sure the intent we express in each statement is faithfully carried out by the hardware. Hence, there is no loss of meaning in the translation by the compiler at all. As we can see, a computer program, translated by a compiler into hardware instructions, is a piece of software (an executable file).

Learning computer programming is learning constructing software. Computer programming is the process of coming up with a sequence of statements in a particular programming language so that when the sequence of statements is executed by the hardware, the intended task is accomplished. Let us break down and analyze the above statement further. What is the intended task? How to we describe or articulate the intended task? Without a precise picture of what the intended task is, it is impossible for us to come up with a computer program to accomplish that task! As computer programmers or software developers, our first order of business is to truly understand the intended task to be accomplished by the computer program that we are going to write or develop.

To build a sequence of statements we must understand the nitty-gritty of each statement included. Each statement of a particular programming language has its syntax and semantics. Hence, as computer programmers or software developers, our second order of business is to master the statements that we are going to use. Syntax refers to rules a statement must follow otherwise compilation errors will occur. Semantics refers to the meaning of a statement in the context of hardware execution. Notice that different programming languages have different syntax and semantics. Therefore, we have to study different programming languages in our career so that we may be able to develop software in those languages.

How to come up with a sequence of statements for an intended task is perhaps the most challenging aspect of computer programming. The sequence of statements may be viewed as a computer algorithm. How can we come up with an algorithm to solve a particular problem? The general approach is to break the problem down into smaller sub problems and continue the process until we can figure out how to solve the smaller problems and how to combine their solutions to form the final solution. The study of many known algorithms for various problems will help as well. Trial and error is perhaps the most basic tool in new algorithm development.

Let us assume that we understand the intended behavior of the program to be written, master the statements of the programming language by which we write the program, and come up with a sequence of statements (the program or algorithm). Our next step is to type the sequence of statements, which is our program, and save the program into a file. The file containing the statements is called source code file. After that we need to verify and validate that our program does what it is supposed to do.

As human beings, we all make mistakes. It is particularly true in computer programming. We will experience that in this course very soon. Editing and validating is an iterative process. We create our program so that it does what we want. First we need to make sure that the program is correct syntactically with the help of a compiler. If our program is not syntactically correct, the compiler will generate compilation errors and refuse to produce the executable file. So we will have to get rid of the compilation errors (iteratively editing, compiling, editing, compiling and so on) so that the executable program is produced. Second we need to make sure that the program is correct semantically, that is the program does what we want. We rely on our understanding of the intended behavior the program and compare it against the actual behavior the program. Understanding the intended behavior allows us to derive the correspondence between the input of the program and the expected response of the program. Such correspondences are our test cases and we use them to test our program. Our program may pass some test cases but fail others. For each failed test case, we need to figure out if the mistake is due to our algorithm, or our misunderstanding or misuse of some statements, or typos, or some careless or silly mistakes in translating our idea into code. The process of pinpointing the exact reason for the wrong behavior of our program is called debugging, which is perhaps the most of frustrating aspect of computer programming. It is a painstaking process and demands patience, but once we figure out what is going on it gives us the experience of triumph, as athletes (for example runners) or artists (poets or musician) feel as they accomplish their work.

Here is a quote from a great computer scientist, Donald Knuth:

> "I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me."

A typical process of debugging involves the following steps:

- Isolate one failed test case (we get rid of bugs one at a time).
- Use a process of elimination to identify the portion of the program that is not working correctly. And continue this process until we can isolate the statements that are causing the issues.
- Understand the real reason as best as we can why the statements do not work.
- Make changes to our program based on our understanding why the statements do not work. And sometimes it turns out that it is our algorithm that is not correct and we may need to adjust that.
- Once corrected, test to see if the previously failed test case passes this time. If it does not pass, we have to repeat the same process. We should also make sure that the test cases that are correct earlier are still correct after the changes made.

You will discover it soon, as some the experienced programmers have learned, that it is very challenging and time-consuming to identify the few statements that cause the incorrect behaviors of the program. So it is a better strategy, which is particularly true at the beginning of our programming career, that we implement and debug our code a few statements at a time. The statements correspond to a sub problem that we try to solve. A typical process of developing our program iterates the following steps:

- Type a few statements that solve a particular sub problem and make sure they are correct syntactically (with the help of the compiler)
- Run the executable code and make sure the expected behavior is achieved.

Since we develop our program incrementally, the syntactic and semantic mistakes should only involve the last few statements that are under construction, since we have tested all the earlier statements of our program.

A few simple C++ programs

The first program is the typical Hello, World program that illustrate the basic structure of a C++ program.

```
// Include statements
#include <iostream>
using namespace std;

// Main function or program
int main()
{

   // Print an output message
   cout << "Hello, World!" << endl;

   return 0 ;
}
```

The second is program a variation of the first program, where the word Hello is stored in a variable. Functionally, this program and the previous program are identical!

```
// Include statements
#include <iostream>
using namespace std;

// Main function or program
int main()
{
   // Initialize variables
   string name = "Hello, World!";

   // Print an output message
   cout << name << endl;

   return 0 ;
}
```

The third program is a variation of the second program, where we ask the user to enter the value for our variable. Now the behavior of the program depends on the user input! We may replace the name of the variable “message” to “name”, which is used in the previous program. If we do so, the behavior of the new program will be identical. The naming of the variable is irrelevant as far as the behavior of the program (program execution) is concern.

```
// Include statements
#include <iostream>
#include <string>
using namespace std;

// Main function or program
int main()
{
   // Initialize variables
   string message;

   // Get the value of message from the user
   cout << "Type your greeting:" << endl;
   getline(cin, message);

   // Print an output message
   cout << message << endl;

   return 0 ;
}
```

The fourth program introduces the conditional statement to handle the situation where the user does not enter anything for the variable.

```
// Include statements
#include <iostream>
#include <string>
using namespace std;

// Main function or program
int main()
{
   // Initialize variables
   string message;

   // Get the value of message from the user
   cout << "Type your greeting:" << endl;
   getline(cin, message);

   // Print an output message
   if ( message.empty() )
      // Print default message, since user typed nothing
      cout << "Hello, World!" << endl;
   else
      // Print the message that the user typed
      cout << message << endl;// Print an output message

   return 0 ;
}
```

The fifth program introduces the iterative statement that allows the program to continuously display what the user types until the user wants to quit it by not typing anything.

```
// Include statements
#include <iostream>
#include <string>
using namespace std;

// Main function or program
int main()
{
   // Initialize variables
   string message;

   // Get the value of message from the user
   cout << "Type your greeting:" << endl;
   getline(cin, message);

   // Print message loop
   while ( message.empty() != true ) {

      // Print the user message
      cout << message << endl;

      // Get the value of message from the user again
      cout << "Type your greeting:" << endl;
      getline(cin, message);
   }

   return 0 ;
}
```

Variables (how to introduce them and manipulate and use their values), conditional statements (which allow us to select different portion of the code to execute), iterative statements (which allow us to repeat the execution of the same portion of the code, not that even though the code is the same but the value of the variables involved may have different values in each iteration), and the default behavior that the code is executed one statement at a time sequentially build up the foundation for computational thinking and computer programming. We will come back to these topics later and provide a more thorough discussion. Other topics that we will consider such as function, class, and library facilities are for making the software easier, faster, and less challenging to build and maintain.