The Beginner's Guide to C++

James Kelley

The Beginner's Guide to C++
James Kelley
© 2014

Author's Web Site: www.gurus4pcs.com

If you find an error, have a question or have something to add to this publication, email me at jkelley742@gmail.com.

Dedications:

My son Mike who field tested this book in his High School class.
My wife Joan who put up with my many hours at the keyboard writing this text.

Preface

This course was designed to work with Microsoft Visual C++, Microsoft Visual C++ Express and DevC++ a free download C++ compiler. Note that there are examples that show the use of all the statements covered in the text. Enter this code, make it work, understand what each statement does and when a particular statement is used.

There are 32 lessons in this tutorial. If you are a complete beginner start with lesson 1 and proceed through each lesson in sequence. If you just need to refresh on a certain topic, there is a table of contents that you can search for the lesson that contains the material you need.

The way to learn a programming language is by doing. Do the sample code, use the code to create your own programs. Go on the internet and find other programs and modify them to do another task or change the way they do the task they were assigned to do. Use what you are learning and you will master C++. Remember, no one learned to ride their two-wheel bike by just reading about how to ride. At some point you have to get up there and take some bumps. Just like riding the bike, you are not a good confident rider until you have practiced riding over and over, you will not be a good programmer until you have written many programs. You will be successful if you apply the same persistence as you applied to riding a two-wheel bike to learning how to write C++ programs.

This course was designed to be conducted using smaller segments of instruction. The emphasis is on the "hands-on" portions of the class.

If you would like the test bank, source code for the examples and the lab exercises I use, please email me and all I ask is: (1) how you are using the book (class, personal, reference, etc.); (2) where you live; (3) your comments on the book (likes, dislikes, need to add, etc.). My email address is: jkelley742@gmail.com.

Table of Contents

SECTION I - Using C++
- Lesson 1 - Overview of C++
  - Lesson 1.1- Using Visual C++
  - Lesson 2.2- Why Learn C++?
  - Lesson 1.3- Where is the C++ Language Used?
  - Lesson 1.4- Different C++ Compilers
  - Lesson 1.5- Compiling a Program
  - Lesson 1.6- The Programming Process
  - Lesson 1.7- Programming Style
  - Lesson 1.8- C++ Template
  - Lesson 1.9- Summary
- Lesson 2 - Problem Solving I – Tools
  - Lesson 2.1- Problem Solving
  - Lesson 2.2- Tools
  - Lesson 2.3- Flowcharting
  - Lesson 2.4- Pseudocode
  - Lesson 2.5- IPO Charts
  - Lesson 2.6- IPO Chart
  - Lesson 2.7- Documentation
  - Lesson 2.8- Summary
- Lesson 3 - Problem Solving II - Word Problems
  - Lesson 3.1- Word Problems
  - Lesson 3.2- Real World Example
  - Lesson 3.3- A Complex Problem
  - Lesson 3.4- Why IPO Charts?
  - Lesson 3.5- Sample IPO Chart and Program
  - Lesson 3.6- Summary
- Lesson 4 - Problem Solving III - Generating Test Data
  - Lesson 4.1- Test Data
  - Lesson 4.2- Building the Test Data
  - Lesson 4.3- Case Study
  - Lesson 4.4- Maintenance
  - Lesson 4.5- Importance of Test Data
  - Lesson 4.6- Summary

SECTION II - Basics
- Lesson 5 – Data Types and Variables
  - Lesson 5.1- Variables
  - Lesson 5.2- Data Types
  - Lesson 5.3- Declaring Variables
  - Lesson 5.4- Initializing Variables
  - Lesson 5.5- Casting (Type Conversion)
  - Lesson 5.6- Sample Program
  - Lesson 5.7- Summary
- Lesson 6 – Constants and Arithmetic Operators
  - Lesson 6.1- Constants
  - Lesson 6.2- Declaring Constants
  - Lesson 6.3- Comments
  - Lesson 6.4- Arithmetic Operators
  - Lesson 6.5- Calculations in C++
  - Lesson 6.6- Assignment Operators
  - Lesson 6.7- Operator Precedence

Lesson 6.8- Sample Program
Lesson 6.9- Summary
Lesson 7 – Common Output
Lesson 7.1- Common Output
Lesson 7.2- Formatting
Lesson 7.3- Debug your Program
Lesson 7.4- Sample Code
Lesson 7.5- Summary
Lesson 8 – Common Input
Lesson 8.1- Common Input
Lesson 8.2- Reading a Character
Lesson 8.3- Using cin.ignore
Lesson 8.4- Sample Program
Lesson 8.5- Summary
Lesson 9 – Strings
Lesson 9.1- Strings
Lesson 9.2- String Concatenation
Lesson 9.3- Sample Code
Lesson 9.4- Summary
Lesson 10 – File Input and Output
Lesson 10.1- File Input/Output
Lesson 10.2- File Input
Lesson 10.3- File Output
Lesson 10.4- Using Both Input and Output Files in a Program
Lesson 10.5- Creating Input Files with Notepad
Lesson 10.6- Reading Output Files with Notepad
Lesson 10.7- Sample Program
Lesson 10.8- Summary

SECTION III - Decisions
Lesson 11 - If and If / Else Statement
Lesson 11.1- Decisions
Lesson 11.2- Relational Operators
Lesson 11.3- Simple IF Statement
Lesson 11.4- Sample Code
Lesson 11.5- Summary
Lesson 12 - Nested If Statements
Lesson 12.1- IF...ELSE Statement
Lesson 12.2- Nested IF...ELSE Statement
Lesson 12.3- Logical Operators
Lesson 12.4- Sample Program
Lesson 12.5- Summary
Lesson 13 - Switch Statement
Lesson 13.1- Switch Statement
Lesson 13.2- Sample Program
Lesson 13.3- Summary

SECTION IV - Repetition Structure
Lesson 14 - For Loop
Lesson 14.1- Repetition Structure
Lesson 14.2- For Loop
Lesson 14.3- Sample Program
Lesson 14.4- Summary
Lesson 15 - While Loop and Do Loop
Lesson 15.1- Repetition Structure

- Lesson 15.2- While Loop
- Lesson 15.3- Do...While Loop
- Lesson 15.4- Sentinel Controlled Loop
- Lesson 15.5- Flag Controlled Loop
- Lesson 15.6- EOF Controlled Loop
- Lesson 15.7- Sample Program
- Lesson 15.8- Summary

Lesson 16 - Nested Loops
- Lesson 16.1- Nested Loops
- Lesson 16.2- Break and Continue Statements
- Lesson 16.3- Sample Program
- Lesson 16.4- Summary

SECTION V - Functions

Lesson 17 - Using Functions and Built-In Functions
- Lesson 17.1- Functions Defined
- Lesson 17.2- Why Use Functions
- Lesson 17.3- Built in Functions
- Lesson 17.4- Sample Program
- Lesson 17.5- Summary

Lesson 18 - Value Returning Functions
- Lesson 18.1- User Defined Functions
- Lesson 18.2- Value Returning Functions
- Lesson 18.3- Function Prototypes
- Lesson 18.4- Actual and Formal Parameters
- Lesson 18.5- Sample Program
- Lesson 18.6- Summary

Lesson 19 - Void Functions
- Lesson 19.1- Void Functions
- Lesson 19.2- Using Void Functions
- Lesson 19.3- Passing Parameters to Void Functions
- Lesson 19.4- Sample Program
- Lesson 19.5- Summary

Lesson 20 - Pass by Value vs. Pass by Reference
- Lesson 20.1- Pass By Value
- Lesson 20.2- Pass By Reference
- Lesson 20.3- Pass By Reference in Value Returning Functions
- Lesson 20.4- Sample Program
- Lesson 20.5- Summary

Lesson 21 - Void vs. Value Returning Functions and Nested Functions
- Lesson 21.1- Void vs. Value Returning Functions
- Lesson 21.2- Nested Functions
- Lesson 21.3- Sample Program
- Lesson 21.4- Summary

Lesson 22 - Programming With Functions a Review
- Lesson 22.1- Void and Value Returning Functions
- Lesson 22.2- Pass By Value and Pass By Reference
- Lesson 22.3- Programming with Functions
- Lesson 22.4- Sample Program
- Lesson 22.5- Summary

SECTION VI - Random Numbers and Files

Lesson 23 - Creating Random Numbers
- Lesson 23.1- Random Number - Theory and Use
- Lesson 23.2- Coding Random Numbers

Lesson 23.3- Using Random Numbers
Lesson 23.4- Sample Program
Lesson 23.5- Summary
Lesson 24 - File Input and File Output
Lesson 24.1- Sequential Data Files
Lesson 24.2- Getting Input from a File
Lesson 24.3- Writing Data to a File
Lesson 24.4- Writing String Data
Lesson 24.5- Ignore Function
Lesson 24.6- Passing Files to Functions
Lesson 24.7- Sample Program
Lesson 24.8- Summary

SECTION VII - Arrays
Lesson 25 – One Dimensional Arrays
Lesson 25.1- Introduction to Arrays
Lesson 25.2- Declaring Arrays
Lesson 25.3- Syntax
Lesson 25.4- Manipulating Data in Arrays
Lesson 25.5- Sample Program
Lesson 25.6- Summary
Lesson 26 – Parallel Arrays
Lesson 26.1- Parallel Arrays
Lesson 26.2- Using Parallel Arrays
Lesson 26.3- Calculations in Parallel Arrays
Lesson 26.4- Sample Program
Lesson 26.5- Summary
Lesson 27 – Multi-Dimensional Arrays
Lesson 27.1- Two-Dimensional Arrays
Lesson 27.2- Using Two-Dimensional Arrays
Lesson 27.3- Manipulating Two-Dimensional Arrays
Lesson 27.4- Sample Program
Lesson 27.5- Summary
Lesson 28 – Arrays and Functions
Lesson 28.1- Arrays and Functions
Lesson 28.2- Passing an Array to a Void Function
Lesson 28.3- Using the Function in a Program
Lesson 28.4- Function Call
Lesson 28.5- Passing an Array to a Value Returning Function
Lesson 28.6- Sample Program
Lesson 28.7- Summary

SECTION VIII - Advanced Concepts
Lesson 29 - User Defined Data Types (enum)
Lesson 29.1- Enumerated Data Types
Lesson 29.2- Using the Enumerated Data Type
Lesson 29.3- Working with Enumerator Values
Lesson 29.4- Using Enumerated Data Types as Values
Lesson 29.5- Enum and the Switch Statement
Lesson 29.6- Some Last Thoughts on Enumerated Data Types
Lesson 29.7- Sample Program
Lesson 29.8- Summary
Lesson 30 – C-Strings
Lesson 30.1- Overview of C-Strings
Lesson 30.2- Strings Stored in Arrays

- Lesson 30.3- Using Library Functions with C-Strings
- Lesson 30.4- Reading and Writing Strings
- Lesson 30.5- Sample Program
- Lesson 30.6- Summary

Lesson 31 – Structs
- Lesson 31.1- Abstract Data Types
- Lesson 31.2- Define Structures
- Lesson 31.3- Access Structure Members
- Lesson 31.4- Comparison and Structures
- Lesson 31.5- Declare Arrays in Structures
- Lesson 31.6- Use Arrays in Structures
- Lesson 31.7- Use Arrays of Structs
- Lesson 31.8- Use Structs with Functions
- Lesson 31.9- Sample Program
- Lesson 31.10- Summary

Lesson 32 – An Introduction to Pointers
- Lesson 32.1- Review of Variables
- Lesson 32.2- Address of Operator
- Lesson 32.3- Pointers
- Lesson 32.4- Using Pointers and Address of Operators
- Lesson 32.5- Sample Program
- Lesson 32.6- Summary

Appendix A – Answers to End of Lesson Quizzes

Appendix B – Using Visual C++.NET 2010
Using DevC++
Using g++

Appendix C - IPO Chart Material

Appendix D - C++ Keywords

SECTION I - Using C++

Lesson 1 - Overview of C++

Lesson 2 - Problem Solving I - Tools

Lesson 3 - Problem Solving II - Word Problems

Lesson 4 - Problem Solving III - Generating Test Data

CHAPTER 1
Overview of C++

Objectives:
- Identify various C++ compilers
- Describe Microsoft's Visual C++
- List the steps to compile an executable file
- Explain the programming process.
- Describe the statements common to most C++ programs.

Lesson 1.1 USING VISUAL C++

The Microsoft Visual Studio Programming Suite contains Visual C++, Microsoft's implementation of the C++ programming language. The Microsoft C++ IDE (Integrated Development Environment) provides several tools to make programming in C++ much easier. First there is the Editor which helps you format your program and color codes various segments of your program. Next, the ability to compile and debug your programs, point and click. Should there be errors in the way you have written your code statements (*syntax errors)* in your program, the IDE will try to point you to the approximate line number where the error was detected (Note the word approximate). Appendix A will explain how to build your first C++ program.

Lesson 1.2 Why learn C++?

The C++ language is one of the most widely used programming languages in the industry. It is extremely flexible in its ability to solve problems from basic business applications to complex scientific applications. This wide range of application make it a popular language in education, scientific, engineering, and the business world. You can find the current ranking of C++ as a programming language choice by going to the web site www.tiobe.com.

Lesson 1.3 Where is the C++ language used?

The C++ programming language can be used to develop a wide range of applications. It can be used by any local business to write a program to do their payroll or solve some other business problem requiring a custom built solution. It may be used by a software developer to create a commercial package for a common application like a word processor or an accounts receivable package. It is also used in Government, Military, aerospace and the scientific community. Most engineering majors are required to take a course in C++ programming as it is widely used in engineering applications.

C++ is a compiled language, so it is used where speed of processing is important. Compiled means that the program contains all of the commands required by the program in the ones and zeros that the computer needs to do its processing.

Lesson 1.4 Different C++ Compilers

Microsoft Visual Studio (Visual C++) is a commercial product that is part of the Visual Studio.NET suite of development software. It comes with an excellent editor, compiler, debugger and runtime environment. Microsoft offers an Express version which is a free download from their Dreamspark web site for Express software.

DevC++ is a free C++ compiler available from http://www.bloodshed.net, Bloodshed Software and you can download it to your PC. It is an Open Source offering so it is the full version that will not expire or will you ever be asked to make any purchase. There is limited support available on the internet.

gcc Compiler The gcc compiler is included in almost every Linux Operating System implementation. C++ is widely used in the Linux community for software development. Many of the applications available for the Linux operating system were written in C++ and some downloaded software will require compilation before they can be used.

Lesson1.5 Compiling a Program

Create Source Code
Enter the code required to solve the problem. This is a plain text file that consists of the C++ framework containing C++ commands. These commands are stored in a file with a .cpp extension.
Compile the Code
The compiler turns the source code into the ones and zeros that the computer understands. This is then written to a file with an .obj extension
Link in the Libraries
Some commands we use in C++ require additional code to complete their function. This code is stored in files called libraries. This is tested code that you benefit from because it can greatly reduce the number of commands you need to write by using pre-written and tested code. The result of this process is a file with a .exe extension.
Executable
Once you have an executable file it can then be transported to any other PC with a suitable operating system and run, just as well as it did on the computer that created it. Note that programs created by Visual C++ will require that the computer running the executable has the current .NET framework installed.
Finding Files
When you have everything working you may want to find the source and executable files. By default, Microsoft puts a project folder in the Project folder within the Visual Studio folder found in the user's document folder. There are several sub folders in the project folder and depending on the version, the .cpp (source) and .exe (executable) will be found in one of these folders.

Lesson 1.6 The Programming Process

Understand the Problem
Before the programmer can do anything they must have a solid understanding of the problem to be solved by the computer program they are about to create.
Analysis
The programmer must break the problem down into the logical steps and algorithms required to solve the problem. This is generally accomplished with flowcharts, pseudocode, IPO charts and other analysis tools.
Code
This is the step where the programmer writes the source code to solve the application problem. When coding is complete, the programmer compiles the program, fixes any errors and repeats the process until the compiler can produce an executable file.
Test
Now that we have an executable file the programmer must make sure the program produces accurate and consistent results. There should be a set of test data that tests all possible conditions the program may encounter in the course of normal (or even abnormal) operating conditions.
Implementation
Once the program is free of errors and has been proven to produce accurate and consistent results, it can be put into production. Users trained to operate the program, documentation written to explain how the program works and what it does, and placed on the computers or servers that will host the application program.

Maintenance
Even after a program is in use, the users will find problems like, errors in calculation and often add new things the program should do that were never thought of before. This is the maintenance phase and is ongoing throughout the life of the program. Even such well known application programs like Windows, Word, Excel, etc. all have periodic updates done to fix problems found.

Lesson 1.7 Programming Style

Exercising care in how you lay out your programs is an important step in making your programs easy to read and maintain. As we introduce various elements of C++, we will demonstrate a particular style of programming. When you develop a style of programming it should be used consistently. Some of the elements of good programming are good use of comments, proper indentation, consistent naming styles for variables and constants and proper attention to case (uppercase and lowercase letters).

Proper use of comments means that you put comment lines before your program that identify the name of the source file, the author, the date of the program and the purpose of the program. You also place comments at key places in the code to explain what a particular section of code has been designed to do.

Proper indentation means that you indent lines of code consistently to add readability to your code. There is nothing harder to follow than a program that does not use indentation consistently. Indentation helps you track where a section of code starts and ends. This is a big help when you are trying to determine where a statement block starts and ends.

Consistent naming styles for variables and constants. One style for constants is to make them in ALL UPPERCASE. Then when you are looking at code you know that when you encounter a variable name in all uppercase it is a constant. Naming your variables should be done in one of three ways, pick one and stick with it. (1) all lowercase characters; (2) Camel case starting with a lowercase letter and using uppercase to separate words, for example, payrate would be written payRate. (3) Using underscores in place of spaces to separate words like payrate would be written pay_rate. We will discuss this later when we cover variables and constants.

Most C++ programmers stick with lowercase whenever possible. Many programming languages are case sensitive and a programmer is always safe in using lowercase exclusively. C++ variable names are case sensitive, so when writing variable names you need to write them the same way in the code as you declared them initially. Short meaningful names using one of the three styles we mentioned earlier are the best way to avoid the dreaded "variable name not declared" error message.

```
// programname.cpp
// Date:
// Author:
// This program demonstrates good style and comments
#include <iostream>
using namespace std;

int main()
{
  // declare variables
        int num1;
        int num2;
        int answer;
  // get data from users
        cout << "Enter a Number";
        cin >> num1;
        cout << "Enter a Number";
        cin >> num2;
 // processing here
        if (num1 < num2)            // determine if num2 is larger than num1
              answer = num2 - num1;    // if true subtract num1 from num2
        else                        // if it is false
               answer = num1 - num2;    // subtract num2 from num1
 // display results
        cout << "The answer is: " << answer << endl;
 return 0;
}  // end of main function
```

Lesson 1.8 C++ TEMPLATE

All C++ program exercises and examples in this book will use the same basic template. There is a group of statements common to all C++ programs we will be working with.

A skeleton C++ program is shown below. This is the minimum code for all of your C++ programs. Putting the comments up front (all the lines that begin with //) is important to identifying your program. I have also listed names for the files you create in all exercises, naming conventions are extremely important to writing clear, maintainable application programs.

Template for a C++ program:

```
// programname.cpp
// author
// date

#include <iostream>
      ** other include statements may be required **
using namespace std;
      ** structs **
      ** function prototypes **
      ** constants **
      ** global variables **
int main()
{
      ** program statements go here **
      return 0;
}  // end of main function
```

You can use this template to begin many of your coding projects. A consistent look for all of your programs shows good coding discipline. Using a proper structure, indenting, consistent use of upper case and lower case can make your programming easier to debug for you and anyone who needs to maintain your code at a later date.

Lesson 1.9 Summary

C++ is a widely used programming language which makes it an important language for a programmer to master. The programmer should understand the steps to completing a C++ programming project as well as the way a C++ compiler goes about creating an executable file. A good C++ program should be properly commented, spaced and indented mainly for ease of maintenance. Good programming style usually indicates a well structured program.

End of Lesson Quiz

The plain text file that contains the C++ statements is called the _________ file.

The phase where the programmer breaks the problem down into logical steps is called the ___________ phase.

You indent lines of code to increase ____________.

CHAPTER 2
Problem Solving I

Objectives:

- Describe the tools available to plan a programming task.
- List the symbols used in flowcharting and their meaning.
- Explain how to use pseudocode to plan a programming task.
- Create an IPO chart to solve a programming problem.
- Define the steps in creating an IPO chart.

Lesson 2.1 Problem Solving

The ability to solve word problems is, perhaps, the most important tool in the programmers toolbox. Every program begins as a word problem. Someone expresses a problem either verbally or written, the programmer takes that information, analyzes it, develops the algorithm to solve the problem, and turns it into a working program or system of interrelated programs.

Often developing the algorithm is the hardest part of solving the problem. This is where the logic of programming becomes the "brick wall" in programming. Too often beginning programmers do not take the time to analyze the problem and try to develop a working algorithm "on the fly". This may work for the simple problems at the beginning but makes the more advanced problems more difficult.

This is why we devote some time at the beginning of this book to the mechanics of solving problems and using problem solving tools. Developing these skills at the beginning and following them throughout the course will make for a better learning experience.

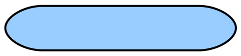
Lesson 2.2 Tools

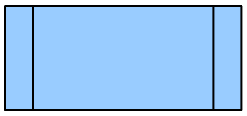
Now we need to examine some of the tools used to analyze word problems and produce the proper analysis of the programming problem to assist the programmer in producing accurate code. We will look at several of these tools: flowcharting; pseudocode; IPO Charts; and Documentation. While some of these have the same function, we will talk about each of them and as your expertise develops, you will learn which of these are best for you.

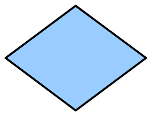
Lesson 2.3 Flowcharting

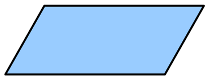
Once we have learned to break out a problem, it is often necessary to draw a picture of the process required to solve the problem. This picture is called a flowchart. It is a series of symbols connected by lines that depict the flow of the data to solve the problem. While there are quite a few different symbols that can be used, we will focus on a basic set of seven symbols that may be used to describe the flow of most programs.

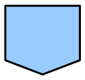
- Terminal
- Process
- Decision
- Input/Output
- Page connector
- Off Page Connector
- Flow Line

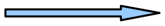
SYMBOL FOR TERMINAL: 

SYMBOL FOR PROCESS: 

SYMBOL FOR DECISIONS: 

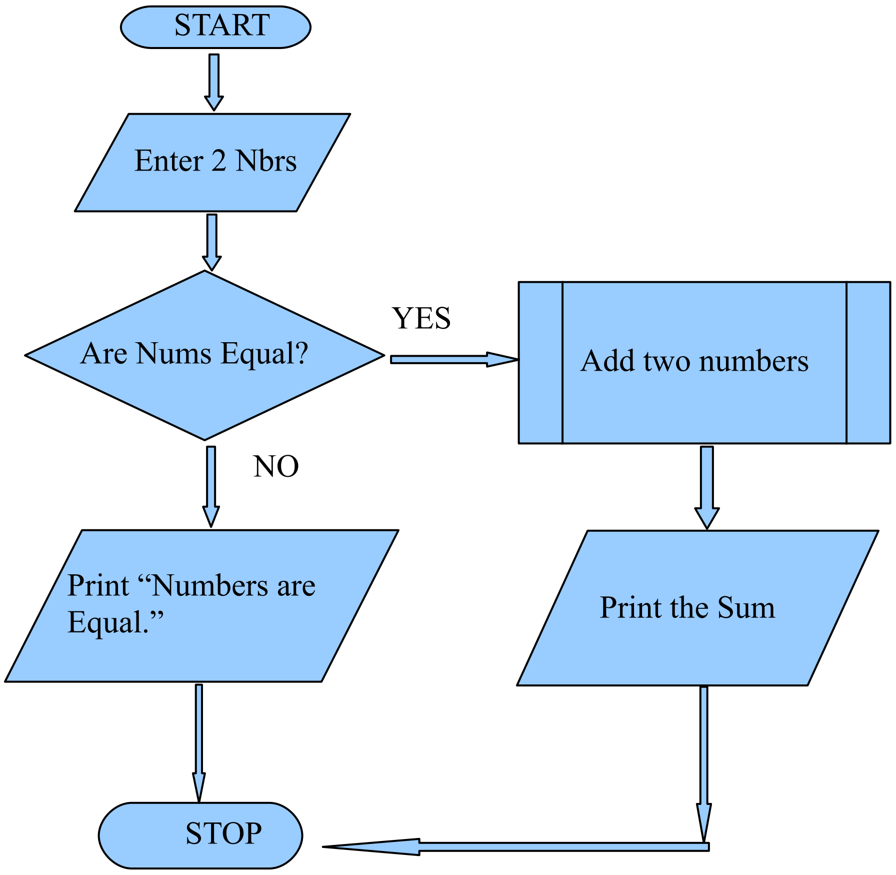
SYMBOL FOR INPUT/OUTPUT: 

SYMBOL FOR PAGE CONNECTOR: 

SYMBOL FOR FLOW LINE: 

There are many more symbols but these are the important symbols for the purposes of this book.

SAMPLE FLOWCHART:



This flowchart represents a program that asks the user for two numbers. Then compares those numbers and if they are equal the program prints "Numbers are Equal". If the numbers are not equal, the program adds the two numbers and prints the sum.

For some programmers this type of visual representation is important to understand the flow of the code they will need to generate. It can be a time consuming process to develop a neat flowchart that is accurate. Often a flowchart is drawn in rough symbols on paper and then at the end, when all has been considered, drawn in a Word processor or with a flowcharting tool such as Microsoft's Visio or Dia (a free tool similar to Visio).

Lesson 2.4 Pseudocode

Pseudocode is another way to develop the logic of a program. It is merely writing down the steps to solve the problem in the correct and logical order. The process is similar to doing an outline before you write a report. It helps you organize your thoughts on writing the applications. There are no keywords or specific syntax for the pseudocode. Pseudocode for a problem requiring the solution to the simple interest algorithm, Interest = Principal Times Rate Times Term may be expressed in pseudocode as follows:

1. Ask the user for the principal amount.
2. Ask the user for the interest rate as a decimal.
3. Ask the user for the length of the loan in years.
4. Calculate the Interest by multiplying the principal times the rate and the result of that by the term of the loan.
5. Display the annual interest for the loan.

These simple sentences outline a step by step process for solving the algorithm.

Lesson 2.5 IPO Charts

Creating an IPO chart to analyze your program can eliminate many problems when you are ready to code. IPO is an acronym for Input, Processing and Output. Each letter represents a column in this three column form. First column lists the variables for the Inputs, the second column shows any processing items (variables created within the code) as well as the pseudocode or flowchart used to solve the problem. The third column list the variables that are to be output. There is a template for an IPO chart in Appendix B.

When using the IPO chart, you must first analyze the word problem and find out what is the goal or the output or the answer to the problem. That is, What is the answer we are looking for? When you discover what that is, write the goal in the output (O) column.

Next we look at the word problem and decide what is the algorithm or the formula to arrive at the answer we are looking for. Perhaps it is given as part of the problem or we may have to do some research for an algorithm that will solve the problem.

Now that we know the algorithm, we need to examine the word problem for input items. Has sufficient data been given to us to solve the algorithm or the formula or the equation we found to solve the problem? We need to make a list of all the inputs required, note those values that have been supplied by the word problem and place them in the Input (I) column of the IPO chart.

Then, back to the Processing column. Now we need to write the pseudocode or flowchart the processes required to solve the problem. These must be presented in the proper and logical order. First we need to gather any information required from the user that was not supplied by the problem. Next we need to plug all of the inputs into the algorithm. Finally, we need to display the output. If we need to create variables to hold the results of interim calculations that have not been listed in either the input (I) column or the output (O) column, we place those at the top of the processing (P) column in an area labeled (PI) for Processing Items.

Example:
Let's examine the following word problem:

Write a program to calculate the cost of the interest on a home improvement loan where a one year loan is secured at the rate of 5.85%. Ask the user for the estimate of the cost of the improvement and calculate the interest based on the amount entered into the program by the user at run time. How much interest will be paid?

Step 1: We need to discover the goal or the output for this word problem. The last sentence is the key, "How much interest will be paid?". Interest is the number we have to come up with. So, we write "Interest" in the Output (O) column of the IPO Chart.

Step 2: Next we should decide on the formula for calculating the interest on a loan. A little research will provide us with the formula: Interest is equal to the Principal amount of the loan multiplied by the rate of interest multiplied by the term of the loan. The actual formula is: I = P * R * T.

Step 3: Back to the word problem and see if we are given information about the Principal, the Rate and the Term. In the Input (I) column we need to put: Principal = user will enter, Rate = .0585, and Term = 1.

Step 4: Now we go back to the Processing (P) column and do the pseudocode for solving the problem.

1. Get user input for the Principal amount, rate and term of the loan.
2. Plug the inputs into the algorithm I = P * R * T.
3. Display the output (interest) on the screen.

It is a good idea at this point to practice using the IPO chart. You should practice several word problems using the IPO chart. This will be helpful in analyzing word problems and turning them into C++ programs.

Lesson 2.6 IPO CHART

All C++ program exercises and examples require an IPO chart prepared before beginning the code phase.

The IPO chart for the problem is shown below. It is important to map out how to solve the problem before beginning to write the code. An anonymous programmer once wrote: *"Weeks of programming can save hours of planning"*. Without a proper plan for your code it will take you many times longer to prepare a program than if you first took the time to plan your process to solve the problem and then coded it.

IPO Chart for Interest Problem:

```
       I(Input)            P(Processing)                               O(Output)
-----------------+----------------------------------------+---------------------
   Principle     | 1. Get the user to input the principle amount of  | Interest
   Rate          |    the loan.                                       |
   Term of Loan  | 2. Get the user to input the interest rate.       |
                 | 3. Get the user to input the term of the loan.    |
                 | 4. Calculate the interest using the algorithm     |
                 |      Interest = Principle * Rate * Term           |
                 | 5. Display the output on the screen.              |
```

Lesson 2.7 Documentation

In the world of programming the dreaded question is "Where's the documentation?". Every programming project in the real world should be accompanied by some human readable content to explain various aspects of the program. We create IPO charts which certainly are a part of documentation, we insert comments into our code, another form of documentation but, we also need to provide a written narrative of the flow and technical details of the program.

Various programming shops have different requirements for such written narratives, and most expect this prior to the acceptance of the final product. If you are working for a company that writes commercial software, you would be expected to provide a user manual to explain how to use the product, an administrators manual to explain how to install and maintain the program and possibly a developers manual to help the programming department understand how to write programs to interface with the product or to alter the product to fit company needs. Then there should be a document that describes the various algorithms found within the program to insure the user knows how certain results were obtained from the input data.

Some companies will insist on training manuals and training materials to assist in bringing their workers up to speed on the program. Developers may keep track of the weekly progress of the development and the problems encountered in the development cycle. Also, we will talk about a series of test data that will be used to validate the results of the program during testing.

The word documentation strikes fear in the heart of many programmers. Programmers feel they are employed to write code not to write stories about their code. Many resist writing the documentation required and even to the point where they would rather leave their job than write documentation. Larger software companies have special documentation specialists to write the necessary documentation, but often it is left to the person who wrote the code.

Lesson 2.8 Summary

Programs are written to solve problems. Taking a “word problem”, developing an algorithm to solve the problem, writing the code and finally testing and debugging the code is the mission. Some of the tools used to develop the algorithm are: Flowcharts, Pseudocode, and IPO charts as well as combination of these tools. If done properly, the coding, testing and debugging of the program will be much quicker and the documentation of the final program will be easier and more accurate.

End of Lesson Quiz

Every program begins as a ______ ____________.

_________ is drawing a picture of the process required to solve a problem.

_________ is the written steps to solve the problem in the correct and logical order.

IPO is an acronym for __________ __________ __________.

CHAPTER 3
Problem Solving II

Objectives:

- Define the output required in a word problem.
- List inputs required by a word program.
- Create an algorithm to solve the word problem.
- List the steps needed to program a solution to the problem.
- Explain the importance of using IPO charts.

Lesson 3.1 Word Problems

In algebra we learned about solving for unknown variables like:

$$X = Y + Z$$

In this equation we assign the sum of the value of Y added to the value of Z to the variable X. So, if Y has the value 3 and Z has the value 2, the value of X will be 5.

$$X = 3 + 2$$
$$X = 5$$

We can translate this into a C++ program as follows:

```
// Algebra.cpp
// This is a sample program
#include<iostream>      // Required to read/write
using namespace std;    // Always required
int main()              // Always required
      {                 // Open Bracket Always required

// Declare Variables
// A variable is an unknown value
// Unknown at the time of programming but
// may be assigned an initial value that may
// or may not be changed.

int Y = 3;  // A variable named Y
                  // with a value of 3

int Z = 2;  // A variable named Z
                  // with a value of 2

int X = 0;  // A variable named X
                  // with a value of 0

// Calculation
// X is assigned the value of Y plus the value of Z

X = Y + Z:

// The variable X now contains the sum of Y plus Z (5)
// and can be displayed
cout << "The answer is: " << X << endl;

return 0;   // Always required
}  // Closing Bracket - End of Main Function - Always required
```

The section of the program under // Declare Variables has three entries. We declare the variable Y and give it a value of 3, a variable called Z and give it a value of 2 and a variable called X and give it a value of 0 because we do not know the result of the calculation yet.

A variable is a chunk of memory that is allocated by the program to store a value. The value may, or may not, be known at the time the program is written. The value stored in a variable may be changed in the course of the program execution. You give that chunk of memory a name so your program can reference the correct area of memory to get the correct value for its calculation.

The section of the program under // Calculation shows the calculation of adding the amount stored in Y to the amount stored in Z and putting the result in the variable X. The expression to the right of the equal sign (assignment operator) is evaluated left to right and when complete, the result is stored in the variable to the left of the equal sign (assignment operator).

Lesson 3.2 Real World Example

Now, let's put a real life situation to this equation.

John has 3 Apples, Mary has 2 Apples, if they give all their apples to Mom, how many Apples will Mom have?

From this problem we develop an equation:

Mom's Apples = John's Apples + Mary's Apples

Our IPO chart should look like this:

```
         I                           P                               O
--------------------+---------------------------------+----------------------
Johns Apples = 3    |                                 |     Moms Apples
Marys Apples = 2    |   add Johns apples to           |
                    |         Marys apples giving     |
                    |               Moms apples       |
```

Or in our example program from above:</p>

```
      // Apples.cpp
      // This is a sample program
      #include<iostream>
      using namespace std;
      int main()
      {
      // Declare Variables

      int JohnApples = 3;
      int MaryApples = 2;
      int MomApples = 0;

      // Calculation

      MomApples = JohnApples + MaryApples:
      // The variable MomApples has now been
      // assigned the value of JohnApples plus
      // the value of MaryApples.

      // Display the result (MomApples)
```

```
        cout << "Mom has " << MomApples << " apples" << endl;
        return 0;
        } // End of Main Function
```

We use exactly the same program but give the variables more explicit names. The program simply takes the values stored in JohnApples and the value stored in MaryApples, adds the values together and puts the sum in the variable called MomApples.

At the end of this process the contents of the chunk of memory assigned to each variable will look like:

Chunk called JohnApples =	3
Chunk called MaryApples =	2
Chunk called MomApples =	5

Lesson 3.3 A Complex Problem

Now we are ready to tackle an even more complex problem.

You purchase a new text book for $79.95 and the sales tax is 5%. You pay for the book with a $100 bill. How much change will you receive from this transaction?

Again we must first develop the formula or algorithm for solving this problem. It will really consist of several steps of formulas with the answer from one providing the data for subsequent steps.

We need a little more work on our IPO chart here to produce a chart that accurately reflects what we need to accomplish.

First, What is the output? We need to solve for the change received from our transaction.

Second, What are the inputs given? The price of the textbook $79.95. The rate of the sales tax which is 5%. Lastly, we pay with a $100 bill.

Notice that before we can solve for the change received we must know (a) the amount of the sales tax and (b) the total amount of the sale including tax. The storage areas for SalesTax and TotalCost are neither input or output. They are listed under Processing (P) as Processing Items (PI).

Next, we must solve for how much sales tax will be charged. This calculation is:

SalesTax = $79.95 times .05 OR
    *SalesTax = PriceOfBook * TaxRate*

Next we need to find the total cost of our purchase.
TotalCost = $79.95 plus SalesTax OR
    TotalCost = PriceOfBook + SalesTax

Last we need to calculate the change due from our $100 bill.
ChangeDue = $100 - TotalCost OR

ChangeDue = Payment - TotalCost

Our IPO chart should look like this:

```
      I                 PI(processing items)                  O
--------------------+----------------------------------------+--------------
                    |   SalesTax                             |
                    |   TotalCost                            |
                    |----------------------------------------|
                    |               P                         |
--------------------+----------------------------------------+--------------
PriceOfBook = 79.95 |     SalesTax=PriceOfBook * TaxRate     | ChangeDue
Tax Rate = .05      |    TotalCost=PriceOfBook + SalesTax  |
Payment = 100.00    |     ChangeDue=Payment - TotalCost      |
```

Our program will need more variables and three different calculations as we see above. Note the change in the IPO Chart. In the middle column there is a section marked PI (Processing Items). Processing Items are variables that need to be set up for interim calculations. They are neither input or output items. They are needed for storing the interim results of calculations in the course of processing. Hence the name Processing Items. In our example above, there are two processing items: SalesTax and TotalCost. As you can see looking at the three calculations, SalesTax holds the amount of Sales tax calculated and TotalCost holds the cost of the book plus the sales tax. Since the output is only the change due, these values are not considered output but are necessary to store the results of these calculation so that the ChangeDue can be calculated.

```
// Change.cpp
// This is a sample program
#include<iostream>
using namespace std;
int main()
{
// Declare Variables
// Variables known from data in the problem
double PriceOfBook = 79.95;
double TaxRate = .05;
double Payment = 100.00;

// Variables that will be calculated in the
// course of running the program.
double SalesTax = 0;
double TotalCost = 0;
double ChangeDue = 0;

// Calculations

// Sales Tax Calculation
SalesTax = PriceOfBook * TaxRate;

// Total Cost Calculation
TotalCost = PriceOfBook + SalesTax;

// Change Due Calculation
ChangeDue = Payment – TotalCost;

// Display the change due
cout << "Change: " << ChangeDue << endl;
return 0;
} // End of Main Function
```

In the calculations section we can see how we must order the calculations properly so the result is correct. First, we must calculate the sales tax. Next we calculate the total cost by using the price of the book plus the value we calculated for sales tax in the previous step. Last, we calculate the change due by taking the amount of payment and subtracting the value calculated in the total cost step. If these are done in any other order the result will be incorrect.

At the end of this process the contents of the chunk of memory assigned to each variable will look like the table below:

Chunk called PriceOfBook =	79.95
Chunk called TaxRate =	0.05
Chunk called Payment =	100
Chunk called SalesTax =	4
Chunk called TotalCost =	83.95
Chunk called ChangeDue =	16.05

Now you should understand why your ability to solve word problems is critical to understanding how to develop a computer program. You need to be able to solve the problem once, manually before you can teach the computer to solve the problem by giving it the steps, by which, you solved the problem correctly. Yes, a program merely teaches or instructs the computer to do things in the order dictated by the programmer.

Another point often missed by beginning programmers is the remarkable similarity of these small programs. Programs require that the programmer present the computer with directions in very specific order and syntax. There must always be a line:

int main() {

This opens the programs main function. This is how the computer knows where the instructions begin. Processing of a C++ program always begins at the main function. There must also be two lines at the very end:

return 0;
} // End of Main Function

This is how the computer knows when it is finished processing the main function of the program.

Everything between the curly braces must also be in a logical order. For your simple, beginning programs we can work with a template that will most often work for many of the programs you will be writing in the beginning. Then, as you learn the rules you can start to develop your own "style" of programming. However, be advised that in many programming shops you will be required to adhere to standards and styles of that shop and you may not be able to be as creative in style as perhaps you would like.

In our samples above, we first declared all our variables. This is important to do before the calculations because it tells the computer what the names of the variables are before the computer needs to use them. We certainly want to name the memory areas and load them with data before we do any calculations.

The second step of our programs was to do the calculation or calculations required to solve the problem. So, our template for all the examples above were a simple, two step, template to declare variables and then do calculations.

Lastly, we need to display the results of our calculations.

Now you are ready to move on to more complex problems and examples. To convert these problems into code it is often necessary to diagram the logic necessary for the computer to solve the problem. There are many tools to accomplish this task but we have discussed three of these and learned to use them to organize our thoughts. The three we have illustrated in this chapter are: Flowcharting, Pseudocode and IPO charts. Practice using these tools and it will help to insure your success in the remainder of this course.

Lesson 3.4 Why IPO CHARTS

Before beginning to code a C++ program, it is advisable to require an IPO chart be prepared to facilitate the code phase.

Organizing your word problems into IPO charts may seem to be a complete waste of time but, they do serve a vital service to the programmer. This is your road map to creating an efficient, well written program. It is not evident in the simple problems we have looked at so far, but in the not too distant future, you will be writing more complex programs and user-defined functions. The lack of an IPO chart will make your programming and debugging of these programs far more difficult.

Lesson 3.5 Sample IPO Chart and Program

PROBLEM:
Tom wants to borrow $1,000 for one year to purchase a new laptop. His bank will give him the money at twelve percent interest for one year. Tom wants to know how much interest he will pay on this loan.

IPO CHART:

I	P	O
Principal = 1000		Interest
Rate = 12%	Interest = Principal	
Term = 1	times Rate times Term	
	Display Interest	

PROGRAM:

```
// Interest.cpp
// This is a sample program
#include <iostream>
using namespace std;
int main()
{
// Declare Variables
// Variables known from data in the problem
double principle = 1000.00;
double rate = 0.12;
double term = 1.0;
double interest = 0.0

// Interest Calculation
interest = principle * rate * term;
// Display the result
cout << "Interest on loan: " << interest << endl;
return 0;
} // End of Main Function
```

Lesson 3.6 Summary

Turning a word problem int an algorithm to solve the problem is often one of the most difficult tasks. First, you need to read the problem carefully and decide what information you need and what information you should throw out. If you can't solve the problem on paper, how will you write the code to solve the problem? The more complex the problem, the more complex the analysis and planning. IPO charts seem to be one of the best tools for assisting beginning programmers in organizing their analysis of a word problem.

End of Lesson Quiz

A __________ is a chunk of memory allocated by the brogram to store a value.

The computer starts to execute at the __________ function.

The programmer's ability to solve _______ ___________ is critical to understanding how to develop a computer program.

CHAPER 4

Problem Solving III

Objectives:

- Explain the importance of a set of test data.
- Create test data for a program.
- Use test data for testing and problem solving.

Lesson 4.1 Test Data

Preparing to test your program is often a task that is completely ignored. However, this may be one of the most important steps in software development. Using a set of test data to test your program not only makes sure your program does what it was intended to do but also provides a means of conducting future tests after modifications have been made that insure you don't fix one problem and create a new problem.

A well crafted set of test data will test all of the possible values and conditions that the program is likely to encounter in the user environment. The test data should not only be good data but also data that tests the error handling of the program. You can count on the user entering a value that is out of range or the wrong data type. It is important that your program handle these errors and does not "blow up" on the user. Minimally, test data should test boundaries, one over a boundary, one under a boundary, and illegal data on each field processed.

Test data also comes into play when you make modifications to an existing program. You should be able to use the same test data to validate the changes you made did not affect the existing functions of the program. When you find a problem with a program and fix the problem that test data should be added to your original test data. The test data then becomes a document that grows as your application program changes.

Lesson 4.2 Building the Test Data

First step is to create a chart to track each input, calculation and output.. A spreadsheet is a great tool for keeping track of your test data and can be used to validate your calculations. The alternative is a paper version and hand calculate all your outputs. It might look something like this:

Principal	**Rate**	**Term**	**Interest**	**Tests**

The above form shows inputs and outputs for the Interest calculation problem. The user is asked for the three inputs and the output is calculated. To complete this form, enter a variety of responses for the three inputs, consider ranges, and invalid data. Once you have taken care of the inputs, using the algorithm for the calculation that we developed in our IPO chart, calculate the Interest for each set of input data. Where there is invalid data note that fact in the Interest column.

Lesson 4.3 Case Study

We will now apply what we have learned about IPO charts and test data to an actual program. We need to go though each step carefully and in order to develop a good plan for writing and testing our code.

Payroll
We will create a basic payroll program that will ask the user for the employee number, hours worked (maximum 40, minimum 8), rate of pay (maximum 25.00 minimum 6.35) and number of dependents (maximum 7). It will calculate the gross pay and the federal tax for the employee and lastly calculate the net pay and display the following: Employee Number, Gross Pay, Federal Tax, and Net Pay. Gross Pay is hours worked multiplied by rate of pay. Federal tax is calculated by taking gross pay minus $8 for each dependent times 15%. Net Pay is Gross Pay minus Federal Tax.

Let's create our IPO chart from the Payroll problem description:

First we are told that we need to calculate the Gross Pay, Federal Tax, and Net Pay for the employee.

Second we are told to ask the user to enter employee number, hours worked, rate of pay and number of dependents.

We are given the calculation for Gross Pay (hours worked times rate of pay); Federal Tax (15% of Gross minus $8 times the number of dependents) and Net Pay (Gross Pay minus the Federal Tax).

Let's apply this information to an IPO chart:

```
        I                            PI                                  O
--------------------+-----------------------------------------------------------
                    |                                            |
                    +--------------------------------------------+
                    |                 P                          |
--------------------+--------------------------------------------+--------------
employee number     |   Get Employee Number                      | Gross Pay
hours worked        |   Get hours worked                         | Federal Tax
rate of pay         |       maximum 40 hours, min 8 hours        | Net Pay
number of dependents|   Get rate of pay                          |
                    |       maximum 25.00 minimum 6.35           |
                    |   Get number of dependents                 |
                    |       maximum 7 dependents                 |
                    |   Calculate Gross Pay                      |
                    |       hours worked times rate of pay       |
                    |   Calculate Federal Tax                    |
                    |       Gross Pay times .15 minus            |
                    |         number of dependents times 1,000   |
                    |   Calculate Net Pay                        |
                    |       Gross Pay minus Federal Tax          |
                    |   Display Results                          |
```

Now we need to set up some test data. We should first create values for each input and then calculate each calculated value. When we run our program, we should get exactly the same results.

Inputs				Outputs			Tests
Emp. Nbr	Hrs. Wkd.	Rate of Pay	Dependents	Gross Pay	Federal Tax	Net Pay	
12345	40	10.75	2				Normal 40 hr pay
23456	40	9	0				Normal Pay No Dep
34567	40	10	8				Too many dependent
45678	40	30	3				Invalid Pay Rate
56789	45	10	0				Invalid Hours - Hi
98765	7	10	0				Invalid Hours - Lo
87654	40	25	7				All Max Values
76543	8	6.35	0				All Min. Values
65432	40	6	2				Invalid Pay Rate - Min

Test data should be prepared prior to starting the code phase. Preparing a table like we see above will help us understand the requirements of the programming phase. Thinking of all the possibilities is an exercise that may help clarify the code required to solve the problem.

Lesson 4.4 Maintenance

Once a program is in place, we must realize the world changes and requirements change. One of the companies that use our payroll system have a new requirement. We received the following request from their paymaster.

Payroll Change Request
Acme Corp. has decided to give all of our employees the opportunity to invest in a 401K program. We will allow them to put between 0% and 10% of their Gross Pay into a 401k program. We will deduct this amount pre-tax (before calculating the Federal Tax, this reduces the amount of federal tax paid by the employee). We also require the amount to be put in a 401K program be printed in the final report.

We need to go back and update our IPO chart appropriately to reflect the new changes. Yes, we should always keep our IPO chart as a form of documentation for our programs. Then we should look at our test data and add the appropriate columns and rows to reflect the new changes. Remember, these changes will affect our calculated fields so we will have to re-calculate each to insure proper test results. We will also have to add rows to reflect testing possible scenarios from adding this new requirement. I have shown the adjustments below.

Inputs					Outputs				Tests
Emp. Nbr	Hrs Wkd	Pay Rate	Dependent s	401K	Gross Pay	401K Amt	Federal Tax	Net Pay	
12345	40	10.75	2	0.05					Normal 40 hr Pay
23456	40	9	0	0.05					Normal Pay No Dep.
34567	40	10	8	0.05					Too many dep.
45678	40	30	3	0.05					Invalid Pay Rate max.
56789	45	10	0	0.05					Invalid Hours Hi
98765	7	10	0	0.05					Invalid Hours Lo
87654	40	25	7	0.1					All max. values
76543	8	6.35	0	0					All min. values
65432	40	6	2	0.05					Invalid Pay Rate Low
99943	8	6.35	0	0					No 401K
99932	40	6	2	0.1					Max 401K
99921	40	6	2	0.12					Exceed Max 401K

Lesson 4.5 Importance of Test Data

All C++ program exercises and examples will require test data to be prepared before beginning the code phase.

The importance of test data is often overlooked. It helps us in a number of ways:

- To organize our thoughts on the program requirements.
- To think about input validation issues.
- To validate our algorithms by making the programmer hand calculate outputs.
- To insure changes do not adversely affect other calculations in the program.

So, it is important to take the time to learn how to generate test data. When you get a job in programming, it will more than likely require that you generate a complete suite of test data for your program projects and demonstrate that your code successfully solves all of the test data situations, both legal and illegal inputs.

Lesson 4.6 Summary

Too often a program is not completely tested before it is released to the users. Beginner programmers get into the habit that when it seems to run successfully, it is complete. Building a series of test data is necessary to test ALL of the possibilities before releasing to the users, they will find new problems. The test data will also be used to re-test the program when changes are made to the program to insure the change only affects the changed areas. When changes are made to solve the problem, new test data should be added to the series of test data.

End of Lesson Quiz

A well crafted set of __________ __________ will test all possible values and conditions the program is likely to encounter in the user environment.

Which is not an important issue for test data?

    Organize thoughts on program requirements.
    Demonstrate ability to use spreadsheets.
    Show validation issues.

Test data should be run on working programs when they are _____________.