The Direct Path to C++ Programming Language

Hamidullah Lutfy & Wahidullah Lutfy

Table of Contents

Chapter 5

- C++ Program Layout
- Compiling C++ Programs

Chapter 6

- C++ PreProcessors Directives
- The C++ I/O iostream module (class)
- The Scope Resolution “using namespace std”

Chapter 7

- Flow of Control in C++
- Branching in C++ (if statement and switch statement)
- Looping in C++ (for, while, and do while statements)

Chapter 8

- Functional Programming in C++
- Object Oriented Programming C++
- System predefined functions
- User defined functions

Chapter 9

- Introduction to Object Oriented using C++
- C++ Modules (Classes)
- C++ Objects (instances of classes)
- Using member functions of C++ objects

Chapter 10

- File System in general
- Input and Output I/O with C++ fstream module/class
- The ifstream module/class for Input Files
- The ofstream module/class for Output Files
- Compiling C++ Programs

Chapter 11

- Introduction to Data Structures
- Arrays in C++
- The string module/class
- Vector in C++
- Class in C++
- Struct and Link List in C++
- Map or Data Dictionary in C++

Chapter 12

- Comparison of C++ Data Structures with Python

Chapter 13

- Pointers in C++
- References in C++
- What is the Difference between pointers and reference

Chapter 14

- C++ Compilers
- C++ Integrated Development Environment (IDE)
- Compilation of C++ in different platforms

Chapter 15

- Debugging C++ source code using mdb
- Automatic compilation in C++ using make utility

Chapter 16

- Introduction to Web Programming using C++
- Using C++ to run any operating System Commands

Chapter 17

- The C++ Object Oriented (OO) Paradigm
  1) Abstraction (Data Hiding)
  2) Encapsulation
  3) Inheritance
  4) Polymorphisim

Chapter 18

- Exception Handling
- How to handle exception
- Ways and when to Throw an Exception
- Multiple Throws and Catches
- The try-catch block
- Assertion and Debugging your program

Chapter 19

- The C++ Template or Class
- Template to for overloading functions
- Template for data hiding and abstraction

Chapter 20

- The C++ references
- Finding C++ Documentation using IDE
- Finding C++ documentation using the internet
- Finding C++ documentation using UNIX, Linux, Mac

Chapter 21

- The future of C++ Programming Language
- C++ still evolving
- Further Research in C++ Programming

APPENDICES

- The ASCII Character Set
- The Unicode Character Set
- The C++ Reserved keywords
- The C++ Precedence of Operators
- Some of the standard C++ Modules (Classes)
- Some of the standard or predefined C++ Functions
- The C++ Function Overloading
- The C++ Operator Overloading
- The Inline Function
- The Friend Function
- The Static Function
- The C++ Recursion Function

INDEX

Preface

- **About this book**

In this class “The Direct Path to C++ Programming Language”, I would like to provide you with a clear path to learning the “C++ Programming Language” My Objective is to first introduce you to the “syntax and semantics” of the “C++ Programming” , as well as provide you with a precise definition of the topic I am covering in each section. Then, by providing further examples and explanation I make sure you have a clear path to understanding the C++ programming language. You will also be given lab activities that you could do it on your computer with your favorite operating system and C++ compiler of choice or C++ Integrated Development Environment (IDE) that you prefer to use.

While it is physically not possible to mentor you with the lab activities while you are using my book, it is absolutely essential for you to be your own mentor whenever you do the lab activities or learning anything in life. Generally, there is no better mentors or coaches than ourselves!

The first path to successful learning in C++ starts by memorizing what we read in each chapter, practice them by writing the snippet of the source code, compile and run it. If we see the sample dialogue in the output of our program that matches the output of that particular example from the book, then we are learning successfully and should continue memorizing what we did and test it using a different operating system and a different C++ compiler on a different day, if we still are successful at it, congratulation you are doing great!

However, if you find out the sample dialogue output of your program is different than the textbook, maybe you inadvertently had a typo that is having a syntax error during the compilation, or maybe you are not getting the same output due to your program has a logical error or semantic error, which your code meaning is different than what you intended to do. For instance you wrote if (x = y), when you meant to use the comparison “==” operator and instead you used the assignment “=” operator. The compiler will not report that as an compilation error since the syntax is correct, but at run time it will have a different meaning when it runs dynamically. By the way, I just explained the difference between “Syntax errors or static errors or compile time errors” vs the “Semantic errors or dynamic error or run time errors”. More about that will be explained in subsequent chapters.

Chapter 1

- Introduction to Computer Programming
  - Why we program?
  - What is a workflow?
  - What is an algorithm?
  - What are the different types of computer programming languages?
  - What is a source code?
  - What is a compiler and what is an IDE?
  - What is an Operating System?
  - What are examples of Software?

Chapter 2

- Setup your C++ Development Environment
- How to setup your sandbox for Windows platform
- How to setup your sandbox for UNIX and Linux platform

Chapter 3

- Introduction to C++ Programming
- Your first “Hello, World!” Program in C++
- Writing your source-code using the C++ Compiler Collection
- Naming your first program “helloWorld.cpp”
- Compiling and running your first program using the Gnu C++ “g++” compiler
- Running your first program
- The details of the “helloWorld.cpp” is explained line by line
- Your first “Hello, World!” Program in C++

Your first **“Hello, World!”** Program in C++

Example 1:
Here is the **C++ source code** for your first **“helloWorld.cpp”** program.

```
// This is a single line comment in C++
// This is  my first C++ program

/*
I can also write multi-line comment  using the /* … */  to document my first program
Program Name: helloWorld.cpp
Program Description: This is my first program that will display the string “Hello, World!”
Written by: Wahidullah Lutfy
*/
#include <iostream>
using namespace std;

int main()
{
    cout << “Hello, World!” << endl;
    return (0);
}
```

Example 2:

Here is the same program, this time we are commenting the **“using namespace std”** and use the scope resolution “**std::** ” to resolve naming conflicts for both objects such as console output **“cout”** and **“endl”**.

```
/ This is a single line comment in C++
 // This is  my first C++ program

 /*
 I can also write multi-line comment  using the /* … */  to document my first program
 Program Name: helloWorld.cpp
 Program Description: This is my first program that will display the string “Hello, World!”
 Written by: Wahidullah Lutfy
 */
 #include <iostream>
 //using namespace std;

 int main()
 {
      std::cout << “Hello, World!” << std::endl;
      return (0);
 }
```

Your first **“Hello, World!”** Program in C++

Now, in our third example we remove all the comments (both single line comment ‘//’ and multi-line /* … */ comment including the line that we commented in the example 2 that was for “// using namespace std:”.

Example 3:

```
#include <iostream>

int main()
{
	std::cout << “Hello, World!” << std::endl;
	return (0);
}
```

As you can see from the above example 3, we have removed all the comments. We use comments to make our program more readable by adding the necessary comments and proper indentation our program is much more readable and understandable to ourselves when we look at it at a later date and are more appreciated by other developers who may work with us on the same project or development team.

Lab Activity: For each of the examples 1-3, we like you to write similar programs and for each of the example 1 through 3 replace the string **“Hello, World!”** to “**Welcome to C++ Programming!**”.

Chapter 4

- C++ Basics
- Variables and Statements

Chapter 5

- C++ Program Layout
- Compiling C++ Programs

Chapter 6

- C++ PreProcessors Directives
- The C++ I/O iostream module (class)
- The Scope Resolution “using namespace std”

Chapter 7

- Flow of Control in C++
- Branching in C++ (if statement and switch statement)
- Looping in C++ (for, while, and do while statements)

Chapter 8

- Functional Programming in C++
- Object Oriented Programming C++
- System predefined functions
- User defined functions

Chapter 9

- Introduction to Object Oriented using C++
- C++ Modules (Classes)
- C++ Objects (instances of classes)
- Using member functions of C++ objects

Chapter 10

- File System in general
- Input and Output I/O with C++ fstream module/class
- The ifstream module/class for Input Files
- The ofstream module/class for Output Files
- Compiling C++ Programs

Chapter 11

- Introduction to Data Structures
- Arrays in C++
- The string module/class
- Vector in C++
- Class in C++
- Struct and Link List in C++
- Map or Data Dictionary in C++

Chapter 12

- Comparison of C++ Data Structures with Python

Chapter 13

- Pointers in C++
- References in C++
- What is the Difference between pointers and reference

Chapter 14

- C++ Compilers
- C++ Integrated Development Environment (IDE)
- Compilation of C++ in different platforms

Chapter 11

- Introduction to Data Structures
- Arrays in C++
- The string module/class
- Vector in C++
- Class in C++
- Struct and Link List in C++
- Map or Data Dictionary in C++

Chapter 11

- Introduction to Data Structures

Chapter 11

- Arrays in C++

Chapter 11

- The string module/class

Chapter 11

- Vector in C++

Chapter 11 (Vector in C++)

We type the command below to find out which machine we are using.

$ hostname

Algorithm

We type the command **‘uname -a’** to get all related info on Operating System and platform.

$ uname -a

Linux algorithm 5.4.0-72-generic #80~18.04.1-Ubuntu SMP Mon Apr 12 23:26:25 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux

We use the command below to find out the OS name and version information.

$ cat /etc/os-release | egrep -i 'name|version'

NAME="**Linux Mint**"
VERSION="**19.3 (Tricia)**"
PRETTY_NAME="**Linux Mint 19.3**"
VERSION_ID="19.3"
VERSION_CODENAME=tricia
UBUNTU_CODENAME=bionic

Chapter 11 (Vector in C++)

We type the command **‘uname -n’** to get the node or hostname.

$ uname -n
Algorithm

We type the command **‘uname -p’** to get the platform in this case X86_64 bits.

$ uname -p
X86_64

We type the command **‘uname -m’** to get the machine architecture.

$ uname -m
x86_64

We type the command **‘uname -s’** to get the OS type in this case Linux.

$ uname -s
Linux

We type the command **‘uname -r’** to get the OS kernel build or revision.

$ uname -r
5.4.0-72-generic

Chapter 11 (Vector in C++)

We use the command below to find the Processor ‘model name’ and type.

$ cat /proc/cpuinfo | grep -i "model name"

model name : AMD A9-9425 RADEON R5, 5 COMPUTE CORES 2C+3G
model name : AMD A9-9425 RADEON R5, 5 COMPUTE CORES 2C+3G

We use the command below to find out how much memory we have, in this case about 16GB.

$ cat /proc/meminfo | grep -i "memtotal"

MemTotal: 15843232 kB

We use the command below to print the current working directory.

$ pwd

/home/wlutfy/Programs/C++

We use the command below to make sure we have enough space for writing.

$ df -h .

Filesystem Size Used Avail Use% Mounted on
/dev/sda2 916G 140G 777G 16% /

Chapter 11 (Vector in C++)

We use the command below to make sure we have enough space for writing.

$ df -h .

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       916G  140G  777G  16% /
```

We use the command below to find out our user name and remote client IP Address

$ who am i

```
wlutfy   pts/1        2021-12-10 00:49 (192.168.254.132)
```

We use the command below to get our User ID, Group ID and other groups we belong.

$ id

```
uid=1000(wlutfy) gid=1000(wlutfy) groups=1000(wlutfy),4(adm),24(cdrom),27(sudo),30(dip)
```

We use the command below to get the long listing of our source program **‘myvec1.cpp**’

$ ls -ld myvec1.cpp

```
-rw-rw-r-- 1 wlutfy wlutfy 2939 Dec 10 21:23 myvec1.cpp
```

We use the command below to find out the file type for our source program ‘**myvec1.cpp’**.

$ file myvec1.cpp

```
myvec1.cpp: C++ source, ASCII text
```

Chapter 11 (Vector in C++)

Example 1: Program Name: myvec1.cpp

We use the '**cat n myvec1.cpp**' command to display the content of the file **'myvec1.cpp**' on the terminal showing the line numbers using the **'-n'** flag of the concatenate '**cat'** command.

$ cat -n myvec1.cpp

```
 1  /*
 2   * Program Name: myvec1.cpp
 3   * Description:
 4   *          To read some int values from the console input until the first negative
 5   *          number is entered. Store all the values read into the Vector myvec1,
 6   *          then, display the values that were provided as input.
 7   *
 8   * Written by: Wahidullah Lutfy
 9   *
10   */
11
12  #include <iostream>
13  #include <vector>
14  #include <iomanip>
15
16  using namespace std;
```

Chapter 11 (Vector in C++)

The rest of the content of the file **‘myvec1.cpp’** is continued on this page.

```
17
18    int main()
19    {
20          // myvec1 is of int data type vector
21          vector<int> myvec1;
22
23          // lastvec1 vector is used to store last elements popped from myvec1.
24          vector<int> lastvec1;
25
26           // last is temp int for storring the last element from myvec1.pop_back()
27          // so it can be inserted into lastvec1.push_back(last)
28          int last;
29
30          // the int number is used to read the data from the standard input console terminal.
31          int number;
32
33          cout << "Enter a list of positive numbers ending with a negative number. \n";
34          cin >> number;
```

The rest of the content of the file **'myvec1.cpp'** is continued on this page.

```
35          while (number > 0)
36          {
37              myvec1.push_back(number); //LIFO Last-In-First-Out Stack
38              cout << "After adding " << number << '\t' << "myvec1.size() is = "
39                  << '\t' << myvec1.size() << endl;
40              cin >> number;
41          }
42
43          cout << endl;
44          cout << "You entered the following numbers:\n";
45          cout << setw(15) << " Memory address "
46               << setw(15) << " Element Number "
47               << setw(15) << " Element Value \n";
48
49          // We use the unsigned int i to work with non-negative numbers.
50          for (unsigned int i=0; i < myvec1.size(); i++)
51                  cout << setw(15) << &myvec1[i]
52                      << setw(15) << "myvec1[" << i << "]"
53                      << setw(15) << myvec1[i] << endl;
54          cout << endl;
```

The rest of the content of the file **'myvec1.cpp'** is continued on this page.

```
55
56          cout << "The elements were pushed on the top of the stack LIFO:\n";
57          cout << "Now, we show the elements in the LIFO order\n";
58
59          cout << setw(15) << "  Memory address  "
60              << setw(15) << "    Element Number (index) \n";
61          for (unsigned int i= myvec1.size(); i > 0; i--)
62                cout << setw(15) << &myvec1[i-1]
63                   << setw(15) << "myvec1[" << i-1 << "]"
64                   << setw(15) << myvec1[i-1] << endl;
65
66          cout << endl;
67          cout << "After popping the last two elements:\n";
68          cout << "and inserting it into new vector lastvec\n\n";
69
70          cout << setw(15) << " Memory address "
71              << setw(15) << " Element Number "
72              << setw(15) << " vec1.pop_back() \n";
```

The rest of the content of the file **'myvec1.cpp'** is continued on this page.

```
73          for (unsigned int i= myvec1.size(); i > myvec1.size()-2; i--)
74          {
75                last = myvec1.back();
76                 cout << setw(15) << &myvec1[i-1]
77                    << setw(15) << "myvec1[" << i-1 << "]"
78                    << setw(15) << myvec1[i-1] << endl;
79                for (unsigned int k=0; i<lastvec1.size(); k++)
80                   {
81                   lastvec1.push_back(last);
82                    cout << setw(15) << &lastvec1[k]
83                    << setw(15) << "lastvec1[" << k << "]"
84                    << setw(15) << lastvec1[k] << endl
85                    << setw(15) << lastvec1[k] << endl;
86                   }
87          }
88          // Add two new-line characters at the end of our program
89          // to separate our output from programs output.
90          cout << endl << endl;
91          return 0; //return success == 0
92    }
```

The last line number of our source file **'myvec1.cpp'** ends at line number 92.

Chapter 11 (Vector in C++)

Now are ready to compile and generate the executable file **'myvec1'** out of our **source-code 'myvec1.cpp'** using the '**Gnu C++ Collection compiler'** called **'g++**'.

Before we use our **'g++**' command let's check some information on our compiler version.

We use the command **'g++**' below command using the **'-o'** option to create the output **Executable Link Format (ELF)** Dynamically Linked Library/Shared Objects to the **C++** Standard Libraries.

```
$ g++ -o myvec1 myvec1.cpp
$ echo $?
0
$
```

The above status of **$?** equal zero means our **main() function** return successfully.

We use the command 'which' to find out if we have the 'g++' compiler in out executable $PATH. In the below example we see the "Gnu C++ Collection compiler 'g++'" is installed in *"/usr/bin" directory as the output of the next two commands reveals that information.*

```
$ which g++
/usr/bin/g++

$ echo $PATH
/home/wlutfy/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
```

Chapter 11 (Vector in C++)

*To find out the type of file '**g++**' is, we use the command below and the output tells us it is a symbolic link to another file namely **'g++7'** as the output of our two commands shows.*

$ file /usr/bin/g++
/usr/bin/g++: symbolic link to g++-7

$ ls -ld /usr/bin/g++
lrwxrwxrwx 1 root root 5 May 20 2019 **/usr/bin/g++ -> g++-**7

Further checking on the symbolic link file g++-7 reveals that is symbolic link to the actual Linux binary executable file '**x86_64-linux-gnu-g++-7'.**

$ ls -ld /usr/bin/g++-7
lrwxrwxrwx 1 root root 22 Dec 4 2019 /usr/bin/g++-7 -> **x86_64-linux-gnu-g++-7**

Further checking on the type of file for '**x86_64-linux-gnu-g++-7'**, it shows that is the actual executable GNU Linux C++ Compiler.

$ file /usr/bin/x86_64-linux-gnu-g++-7
/usr/bin/x86_64-linux-gnu-g++-7: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=c6d8a52fddc774e05a7ab39859c160fa0b2d184b, stripped

Chapter 11 (Vector in C++)

To find out the version of the GNU C++ Collection Compiler 'g++', we use the command below and the output tells us we are using the ***"g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"***

$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

To find out if 'gcc' is installed on this host.

$ which gcc
/usr/bin/gcc

If you are interested to find out the version of the GNU C Collection Compiler 'gcc', we use the command below and the output tells us we are using the ***"gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"***

$ gcc --version
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPO**SE.**

Chapter 11 (Vector in C++)

Example 2: Program Name: myvec2.cpp

We use the '**cat n myvec2.cpp**' command to display the content of the file **'myvec2.cpp**' on the console terminal showing the line numbers using the **'-n'** flag of the concatenate '**cat'** command.

$ cat -n myvec2.cpp

```
 1  /*
 2   * Program Name: myvec2.cpp
 3   * Description:
 4   *    To read some int values from the standard input console (terminal), and
 5   *    display them back. We also, to display the content of the vector in
 6   *    reverse order. In this case we are using the Last In First Out (LIFO)
 7   *    the 'Stack Data Structure' also known as 'LIFO' using vector myvec1.
 8   *
 9   * Written By:     Wahidullah Lutfy
10   */
11  #include <iostream>
12  #include <vector>
13
14  using namespace std;
15
```

The rest of the content of the file **'myvec2.cpp'** is continued on this page.

```
16    int main()
17    {
18          vector<int> myvec1;
19          cout << "Enter a list of positive numbers. \n"
20              << "Place a negative number at the end.\n";
21          int number;
22          cin >> number;
23          while (number > 0)
24          {
25                myvec1.push_back(number); //LIFO Last-In-First-Out Stack
26                cout << "After adding" << '\t' << number << '\t'
27                    << "myvec1.size is = " << '\t' <<  myvec1.size() << endl;
28
29                // Make sure to read a negative number to end the while loop.
30                cin >> number;
31          }
32          cout << "\nYou provided the following numbers:\n";
33
34          for (unsigned int i=0; i < myvec1.size(); i++)
35                cout << &myvec1[i] << '\t' << "myvec1[" << i << "] = " << myvec1[i] << endl;
36          cout << endl;
```

The rest of the content of the file **'myvec2.cpp'** is continued on this page.

```
36          cout << endl;
37
38          cout << "Displaying the vector from end() to begin() myvec1:\n";
39          for (vector<int>::iterator it = myvec1.end()-1 ; it != myvec1.begin()-1; --it)
40              std::cout << &it << '\t' << *it << endl;
41
42          cout << endl << endl; // To print two new line display so our output is clean.
43          return 0;
44      }
```

The last line number of our source file **'myvec2.cpp'** ends at line number 44, and now are ready to compile and generate the executable file **'myvec2'** out of our **source-code 'myvec2.cpp'** using the '**Gnu C++ Collection compiler'** called **'g++**'.
We use the command **'g++**' below command using the **'-o'** option to create the output **Executable Link Format (ELF)** Dynamically Linked Library/Shared Objects to the **C++** Standard Libraries.

```
$ g++ -o myvec2 myvec2.cpp
$ echo $?
0
$
```

The above status of **$?** equal zero means our **main() function** return successfully.

Chapter 11 (Vector in C++)

We use the below command to get a long list of the executable file ‘myvec2’.
The g++ compiler created the ‘myvec2’ with read, write execute for the owner, and group, and read, and executable by others, the write permission is not granted to the world.

$ ls -ld myvec2

-rwxrwxr-x 1 wlutfy wlutfy 23080 Dec 10 23:41 myvec2

We use the **‘file**’ command on **‘myvec2’** executable to determine the file type and architecture of the processor the **Executable Link Format (ELF)** dynamically linked with Linux **‘ld’** loader **x-86-64 bits** shared object **“.so”** file.

$ file myvec2

myvec2: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=c9e3464f23591512e8112f929f46205c8df4d646, not stripped

We use the **‘ldd’** command to find out the shared object dependencies for our ‘**myvec2**’ binary.

$ ldd myvec2

```
linux-vdso.so.1 (0x00007ffeac7a9000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f9584d9e000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f9584b86000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f958478e000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f95843ee000)
/lib64/ld-linux-x86-64.so.2 (0x00007f9585336000)
```

Chapter 11 (Vector in C++)

Now, let's just execute our **'myvec2'** program to see the output.

$ ./myvec2

```
Enter a list of positive numbers.
Place a negative number at the end.
5 10 15 -1
After adding      5    myvec1.size is =     1
After adding      10   myvec1.size is =     2
After adding      15   myvec1.size is =     3

You provided the following numbers:
0x560fd5e98690     myvec1[0] = 5
0x560fd5e98694     myvec1[1] = 10
0x560fd5e98698     myvec1[2] = 15

Displaying the vector from end() to begin() myvec1:
0x7ffe2bd8830815
0x7ffe2bd8830810
0x7ffe2bd883085


$
```

Chapter 11 (Vector in C++)

A second sample run for our program **'myvec2'** is shown below:

```
$ ./myvec2
Enter a list of positive numbers.
Place a negative number at the end.
55 777 -5
After adding      55   myvec1.size is =      1
After adding      777  myvec1.size is =      2

You provided the following numbers:
0x555705ce66b0     myvec1[0] = 55
0x555705ce66b4     myvec1[1] = 777

Displaying the vector from end() to begin() myvec1:
0x7ffecb506f08 777
0x7ffecb506f08 55


$
```

Chapter 11

- Class in C++

Chapter 11

- Struct and Link List in C++

Chapter 11

- Map or Data Dictionary in C++

Chapter 15

- Debugging C++ source code using mdb
- Automatic compilation in C++ using make utility

Chapter 16

- Introduction to Web Programming using C++
- Using C++ to run any operating System Commands

Chapter 17

- The C++ Object Oriented (OO) Paradigm
  1) Abstraction (Data Hiding)
  2) Encapsulation
  3) Inheritance
  4) Polymorphisim

Chapter 18

- Exception Handling
- How to handle exception
- Ways and when to Throw an Exception
- Multiple Throws and Catches
- The try-catch block
- Assertion and Debugging your program

Chapter 19

- The C++ Template or Class
- Template to for overloading functions
- Template for data hiding and abstraction

Chapter 20

- The C++ references
- Finding C++ Documentation using IDE
- Finding C++ documentation using the internet
- Finding C++ documentation using UNIX, Linux, Mac

Chapter 21

- The future of C++ Programming Language
- C++ still evolving
- Further Research in C++ Programming

APPENDICES

- The ASCII Character Set
- The Unicode Character Set
- The C++ Reserved keywords
- The C++ Precedence of Operators
- Some of the standard C++ Modules (Classes)
- Some of the standard or predefined C++ Functions
- The C++ Function Overloading
- The C++ Operator Overloading
- The Inline Function
- The Friend Function
- The Static Function
- The C++ Recursion Function

INDEX

- **Chapter 11 Vector in C+++** (Continuation of myvec1.cpp program)

```
21          int last;
22          cout << "Enter a list of positive numbers ending with a negative number. \n";
23          int number;
24          cin >> number;
25          while (number > 0)
26          {
27              myvec1.push_back(number); //LIFO Last-In-First-Out Stack
28              cout << "After adding " << number << '\t' << "myvec1.size() is = "
29                   << '\t' << myvec1.size() << endl;
30              cin >> number;
31          }
32          cout << endl;
33          cout << "You entered the following numbers:\n";
34          cout << setw(15) << " Memory address "
35               << setw(15) << " Element Number "
36               << setw(15) << " Element Value \n";
37          for (unsigned int i=0; i < myvec1.size(); i++)
38                 cout << setw(15) << &myvec1[i]
39                      << setw(15) << "myvec1[" << i << "]"
40                      << setw(15) << myvec1[i] << endl;
41          cout << endl;
42
43          cout << "The elements were pushed on the top of the stack LIFO:\n";
44          cout << "Now, we show the elements in the LIFO order\n";
45
```