

**Discipline Courses-I**  
**Semester-I**  
**Paper: Programming Fundamentals**  
**Unit-I**  
**Lesson: An Introduction to Programming**  
**Lesson Developer: Rakhi Saxena**  
**College/Department: Deshbandhu College, University**  
**of Delhi**

## Table of Contents

- Chapter 1: **An Introduction to Programming**
  - 1.1: An Introduction to Programming
    - 1.1.1: Learning Objectives
    - 1.1.2: What is a Program?
      - 1.1.2.1: Creating Computer Programs
      - 1.1.2.2: Program Development Cycle
    - 1.1.3: Basic Programming Concepts
      - 1.1.3.1: A Simple Program
      - 1.1.3.2: Running a C++ Program
      - 1.1.3.3: Commenting your Code
  - 1.2: Variables and Constants
    - 1.2.1: Learning Objectives
    - 1.2.2: The Structure of a C++ Program
      - 1.2.2.1: Data Input, Processing and Output
    - 1.2.3: Program Variables
    - 1.2.4: Program Constants
    - 1.2.5: Variable Assignment and Reassignment
    - 1.2.6 Exchanging the Value of Variables
- Summary
- Exercises
- Glossary
- References

## 1.1 An Introduction to Programming

Welcome to Chapter 1! To become a successful programmer, it is important to know what a computer program is before you start writing one. In this chapter you will learn what a program is. You will learn to write a simple C++ program and how to execute it. This chapter also provides a brief overview of the program development cycle.

These concepts will help you understand the structure and syntax of C++ programs.

### 1.1.1 Learning Objectives

After reading this chapter you should be able to:

1. Describe the steps for program creation and develop a simple program.
  - 1.1 Explain what computer programs are.
  - 1.2 Discuss the program development cycle.
  - 1.3 Apply the program development cycle to solve a problem.
  - 1.4 Understand the evolution of the C++ language.
  - 1.5 Describe the basic syntax of a C++ program.
  - 1.6 Write a simple C++ program.
  - 1.7 Compile and execute a C++ program.

### 1.1.2 What is a Program?

Imagine that you have invited your friend to your house. To help her you give her detailed instructions on how to reach your house. Such a set of instructions might look like the following:

Take the bus route number 427.  
Get down at "Saket".  
Cross the road.  
Go straight for 100 meters.  
Take a left turn.  
Go about 50 meters to reach my house at 37, J Block.

For another task such as opening a bank account, you will have another plan of action – another set of instructions to perform that task.

Just like a set of instructions followed by you to carry out a task, a *program* is a set of *instructions* carried out by a computer to accomplish a specific task. The sequence of program instructions, are called *statements*, each of which performs part of the entire task to be executed.

If the task is simple, the computer program will be relatively simple and short. If the task is complicated, the program will also be relatively complex and long.

So when you write a computer program, you are creating a plan of action or a sequence of steps to accomplish a task that a computer can do.

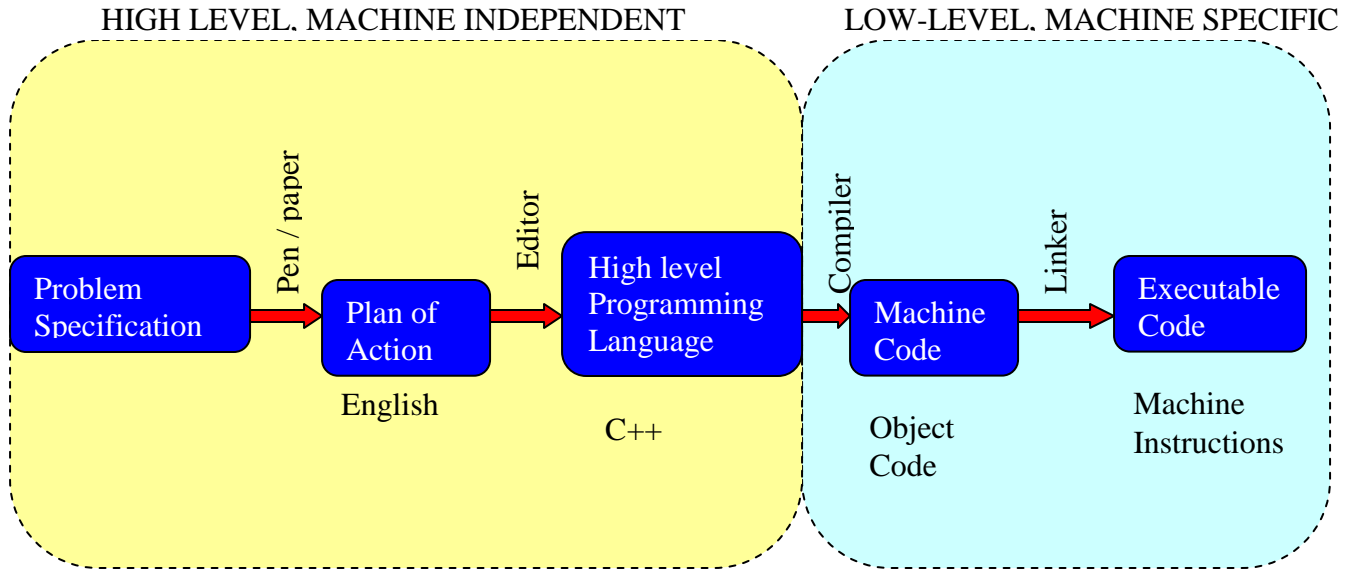
**Programming** is the preparation and writing of detailed set of instructions for computers. It is the art of writing correct and efficient computer programs.

You may have used a number of programs when you work on a computer. A browser, such as, Internet Explorer or Firefox Mozilla, is a computer program that helps you surf the Web. A multimedia player, such as RealPlayer or Amarok, is another computer program that lets you play music or movies.

#### 1.1.2.1 **Creating Computer Programs**

A program has to be written in a language that can be understood by a computer. Programmers write their programs in a high level programming language such as C++, FORTRAN or Java and then use a **compiler** to translate their code into **machine language** code that will run on the machine they are using.

In these lessons you will learn how to develop your own computer programs in a language called C++.



**Figure 1.1: From a Problem to a Program**

Each programming language has its own specific **syntax**. If you don't write your program with the correct syntax, it will not work. Therefore, to write a C++ program you need to learn its syntax.

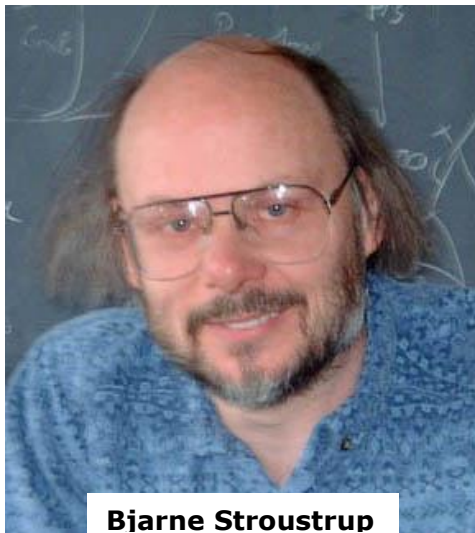
For example, one of the syntax rules for C++ is:

“Every statement must end with a semi colon.”

We will write a program and the syntax of C++ with simple structured C++ programs and build object oriented programs in further lessons.

**Value addition: Did you Know?**

**Heading text: Evolution of C++**



**Bjarne Stroustrup**

C++ was developed by Bjarne Stroustrup of AT&T Bell Laboratories, Murray Hill, New Jersey in the early 1980's, and is a superset of the C language. Stroustrup originally named the language *C with Classes*. It was renamed C++ in 1983.

C++ evolved from C. C was developed after the language called B (designed by Ken Thompson in 1970). B itself came from another language called BCPL. BCPL was developed in 1967 by Martin Richards as a language for writing operating systems software and compilers.

C++ allows programmers the ability to comprehend and manage large C programs. It also enables programmers to write reusable code. It is widely used in the software industry as well as for hardware design.

**ANSI C++**

The American National Standards Institution (ANSI) provides standard definitions of many programming languages, including C and C++. A program written only in ANSI C++ is portable – that is, it is guaranteed to run on any computer whose supporting software conforms to the ANSI standard. In practice most versions of C++ include ANSI C++ as a core language, but also include extra machine-dependent features to allow smooth interface with different computers' operating systems. These machine dependent features should be used cautiously.

**Source:** Self made

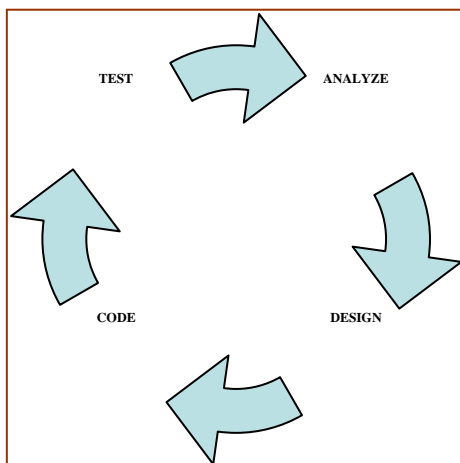
For image: <http://www.cpptutor.com/evolution-of-c++.htm>

**1.1.2.2 Program Development Cycle**

Before you write a program, you have to devise a plan of action to solve the given problem. A general problem solving strategy is as below:

1. **Analyze** the problem - Understand the problem by determining the information that is available, what results are desired and how to proceed to get those results.
2. **Design** a solution - Provide step by step instructions to solve the problem.
3. **Code** the program – Write program code in a high level language to implement the design created in the previous step.
4. **Test** the program – Run the program to verify that the program solves the given problem.

If you discover a flaw in one of steps, maybe your program did not give the right result, you will need to return to a previous step and redo the task. This process of analysis, design, coding and testing is known as the program development cycle.



**Figure 1.2 Program Development Cycle**

**1.1.3 Basic Programming Concepts**

Now that you know the steps to go from the specification of a problem to the development of a program to solve the problem, let's begin by writing a simple computer program. This program will introduce you to the basic syntax of a C++ program.

### 1.1.3.1 A Simple Program

We will start with a program that prints a welcome message when it is executed. The steps for this consist of just one instruction:

Display "Hello World"

These steps must now be coded in C++. The "Welcome Message" C++ program is coded as below:

Line Number	Program Code
1.	#include <iostream>
2.	using namespace std;
3.	int main()
4.	{
5.	cout << "Hello World!";
6.	return 0;
7.	}

Let us now understand each line in the program code one by one:

1. This statement is an **include** directive statement. It tells the compiler and the linker that the program will need to be linked to a library of routines that handle input from the keyboard and output to the screen. The library file is enclosed within angular brackets. In this case it is "iostream".
2. This statement is a using directive statement. It tells the compiler and the linker that the program will be using names that have a meaning defined for them in the "std" namespace. The latest versions of the C++ standard divide names (e.g. cin and cout) into sub-collections of names called **namespaces**.
3. A C++ program consists of a collection of functions each of which is a group of statements written to perform a specific task. A **function** is identified by a function name and a function body. The function name is identified by a word followed by round brackets. The body is enclosed within a pair of curly braces. **main** is the name of a special function that every program in C++ must have. When you run a program, the statements in this function are executed first.
4. The main function's body is enclosed within curly braces. The opening curly brace defines the beginning of a function.
5. This is the statement that does the actual task of the program. It prints the string "Hello World" on the screen. A string is a sequence of characters that is typed within double quotes.
6. This statement tells the computer to return from the main function with an integer value 0 to indicate successful termination. The "int" before the main function name tells that an integer value must be returned by this function.
7. This is how a function ends. The closing curly brace defines the end of a function.

**NOTE:** Older compilers do not support namespaces. If you are using an old compiler (that is, one that does not support ANSI C++) replace the first two lines in the program with

```
# include <iostream.h>
```

<b>Value addition: Common Coding Errors</b>
<b>Heading text: BEWARE!!</b>
<ul style="list-style-type: none"><li>• C++ is case sensitive – follow the lowercase and uppercase exactly as it is given in the program</li><li>• Don't forget to put a semicolon at the end of a statement.</li><li>• Make sure pairs of curly braces match in your program.</li></ul>
<b>Source:</b> <u>Self made</u>

### 1.1.3.2 Running a C++ Program

Once you have written the code for the “Welcome Message” program, you will need to *compile* and *link* it so that an executable file can be created. To *run* your program, you will have to run the executable file.

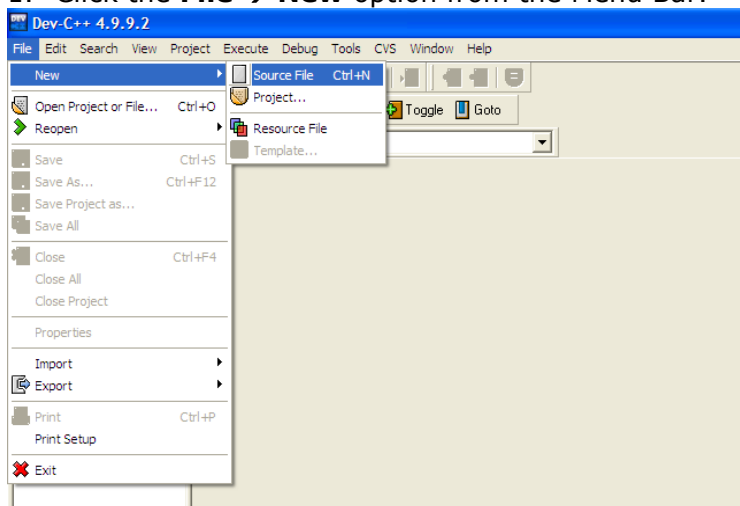
Depending on your computer and your compiler, the process of running your program will vary.

#### **Windows Environment**

You can use the Dev-C++, Turbo C++ or the Borland C++ Integrated Program Development Environments (IDEs). They provide a built-in editor and a menu bar for creating a new file, editing a program, saving, compiling and then running the program.

Following are the steps to create the “Welcome Message” program and execute it with the Dev-C++ IDE:

1. Click the **File → New** option from the Menu Bar.



**Figure 1.3 - Creating a new file**

2. Type the program.

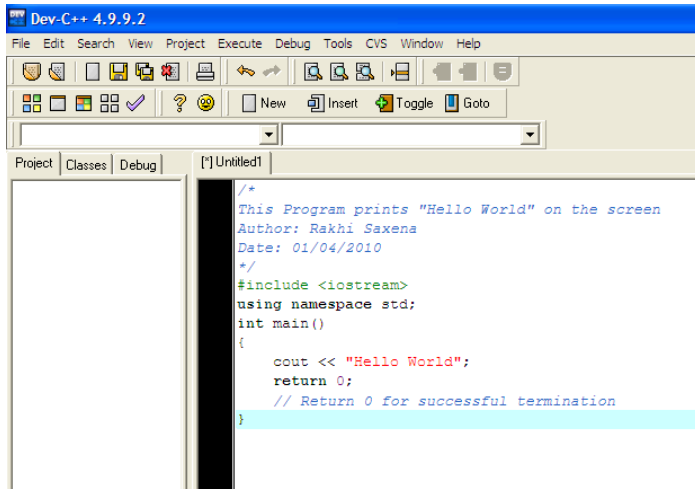


Figure 1.4 Editing a Source File

3. Click the **File** → **Save** option from the Menu Bar.

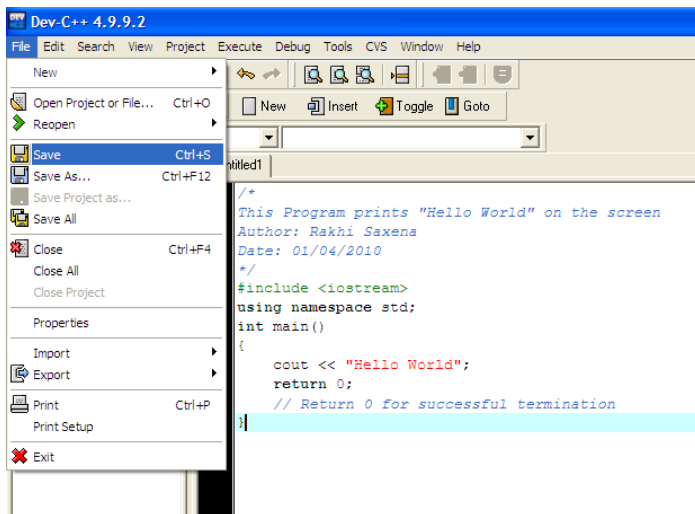
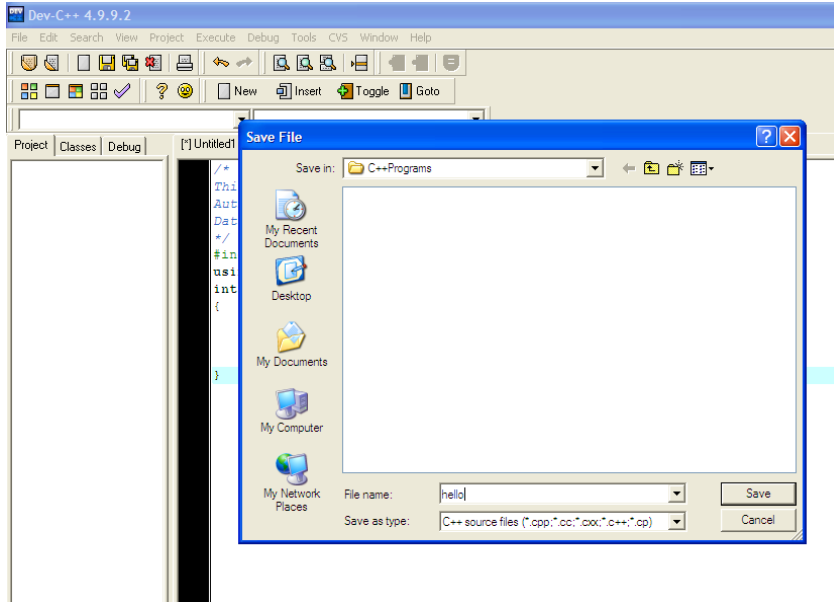


Figure 1.5 Saving a File

4. Save the program as "hello.cpp". The cpp extension tells the computer that this file is a C++ source file.

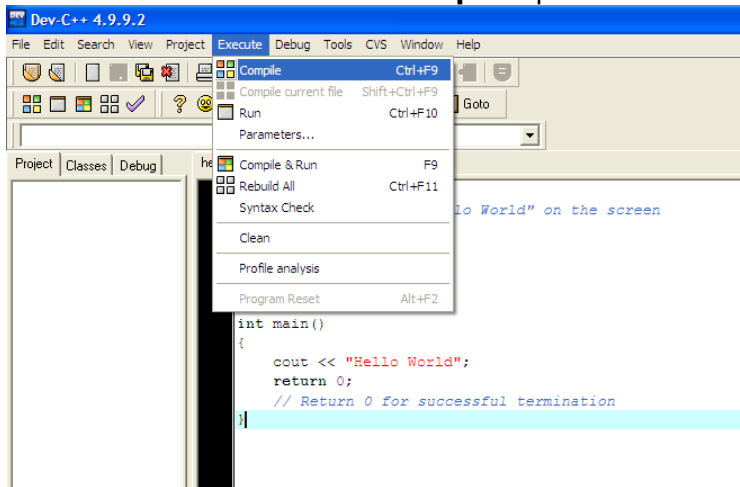


## An Introduction to Programming



**Figure 1.6 Naming the Source File**

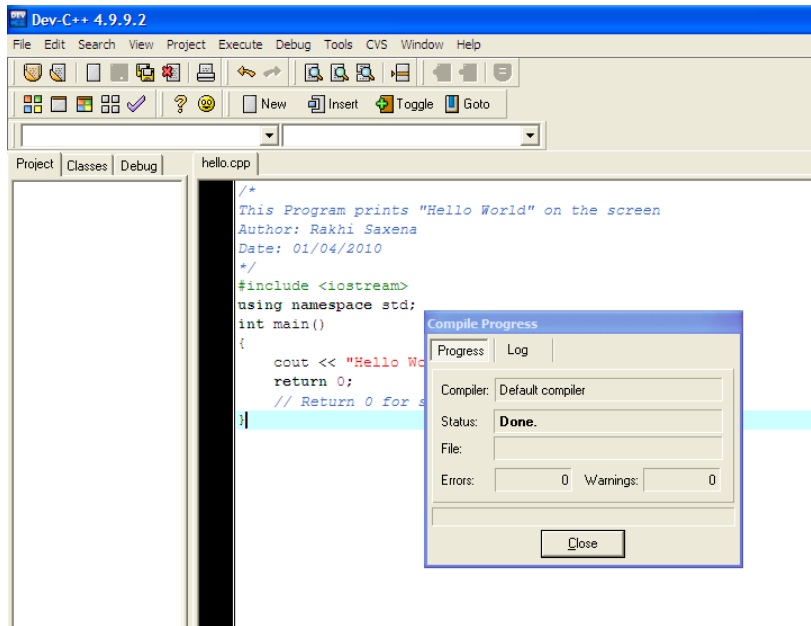
5. Click the **Execute** → **Compile** option from the Menu Bar.



**Figure 1.7 Compiling the Program**

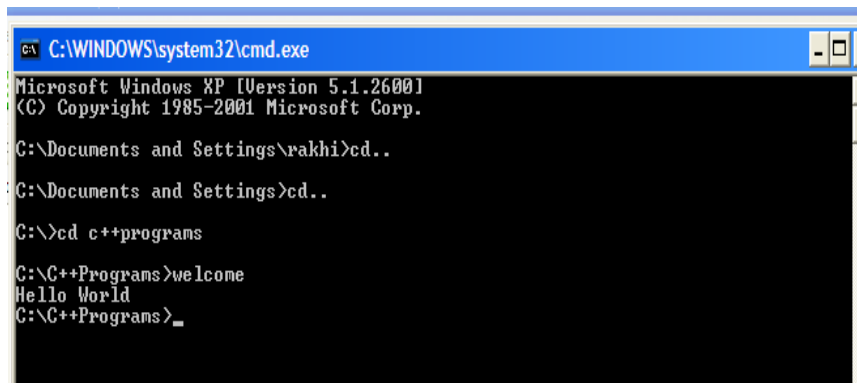
6. If there are no errors in your program, you will see the following screen. Click on the **Close** button.

## An Introduction to Programming



**Figure 1.8 Compilation Completed**

7. Click the **Run** option from the Menu Bar to execute your program. Also, an executable file named "hello.exe" will be created in the same directory as your program. You can execute this file from the command prompt by typing its name.



**Figure 1.9 - Program in Execution**

That's it!! If you have followed the instructions carefully, you will see the words "Hello World!" on your screen. Congratulations, you are now a programmer!

### **UNIX/ Linux Environment**

You can use the g++ compiler. This is a free compiler available for all UNIX/ Linux platforms.

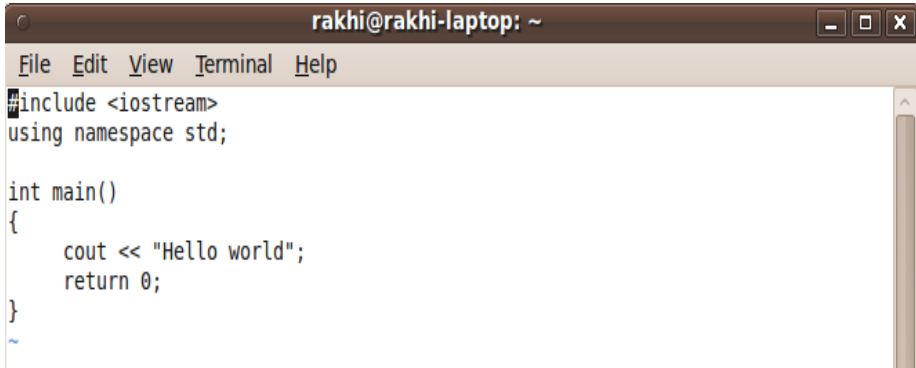
Following are the steps to create the "Welcome Message" program and execute it:

1. Type the program in any word editor (you can use ed, emacs or vi).
2. Save the program as "hello.cpp". The cpp extension tells the computer that this file is a C++ source file.
3. Compile the program with the following command typed at the UNIX prompt.  
g++ hello.cpp -o hello
4. If you did not make any errors while typing, the executable file "hello" should be created.
5. Type "hello" at the UNIX prompt to run this file.

**Value addition: Screenshots**

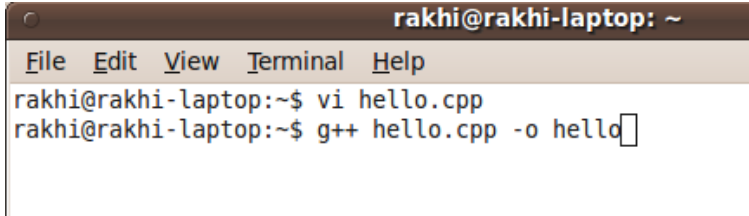
**Heading text: "Welcome Message" program on Linux**

**Editing the program with the "vi" editor**



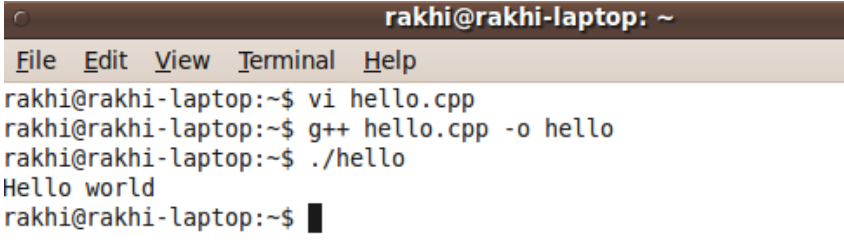
```
rakhi@rakhi-laptop: ~  
File Edit View Terminal Help  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hello world";  
    return 0;  
}  
~
```

**Compiling the program with the g++ compiler**



```
rakhi@rakhi-laptop: ~  
File Edit View Terminal Help  
rakhi@rakhi-laptop:~$ vi hello.cpp  
rakhi@rakhi-laptop:~$ g++ hello.cpp -o hello
```

**Executing the program**



```
rakhi@rakhi-laptop: ~  
File Edit View Terminal Help  
rakhi@rakhi-laptop:~$ vi hello.cpp  
rakhi@rakhi-laptop:~$ g++ hello.cpp -o hello  
rakhi@rakhi-laptop:~$ ./hello  
Hello world  
rakhi@rakhi-laptop:~$ █
```

**Source:Self made**

### 1.1.3.3 Commenting Your Code

## An Introduction to Programming

The program that you saw in the last section is a correct program, it runs successfully. However, if another programmer or a user were to read the program, they wouldn't understand what this program does.

It is a good idea to *document* your program by providing explanatory notes for portions of the code within the program. This is especially useful in the real world, where large programs are written by one programmer and maintained or enhanced by other programmers.

These explanations can be written in a program via **comments**. Comments can be placed in a C++ program in the following two ways:

- Placing a special set of characters (`//` - two consecutive forward slashes) before any line in the code.
- Placing text to be commented between the symbols `/*` and `*/`

The former method is used for single line comments while the latter is generally preferred for multiple line comments.

Line No.	Program Code	
1.	<code>/*</code>	} A Multi line comment
2.	<code>This Program prints "Hello World" on the</code>	
3.	<code>screen</code>	
4.	<code>Author: Rakhi Saxena</code>	
5.	<code>Date: 01/04/2010</code>	
6.	<code>*/</code>	
7.	<code>#include &lt;iostream&gt;</code>	} A Single line
8.	<code>using namespace std;</code>	
9.	<code>int main()</code>	
10.	<code>{</code>	
11.	<code>    cout &lt;&lt; "Hello World";</code>	
12.	<code>    return 0;</code>	
13.	<code>    // Return 0 for successful termination</code>	
14.	<code>}</code>	
15.		

Comments are ignored by the compiler at compile time. They are also ignored by the computer when the program is run.

### Value addition: Style Tips

#### Heading text: STYLE TIP!!

It is a good idea to place the following two types of comments in your program

1. **Header Comments** – These are written at the beginning of a program/ module and include information like purpose of the module, author, version, and so on.
2. **In line Comments** – These are written inside the program and explain the purpose of specific parts of the code.

**Source:** Self made

## 1.2 Variables and Constants

In the previous section, you learnt how to write a C++ program and how to execute the code. In this section you will learn to write simple programs that do a little more than print a welcome message. You will learn the basic building blocks that form the structure of almost all computer programs – data input statements, data processing statements, data output statements, program variables and constants. You will learn to assign and reassign values to variables.

These concepts will help you develop programs that take some input data, process it and display the processed data. You will also explore a program that exchanges the values of two variables with the help of a temporary variable.

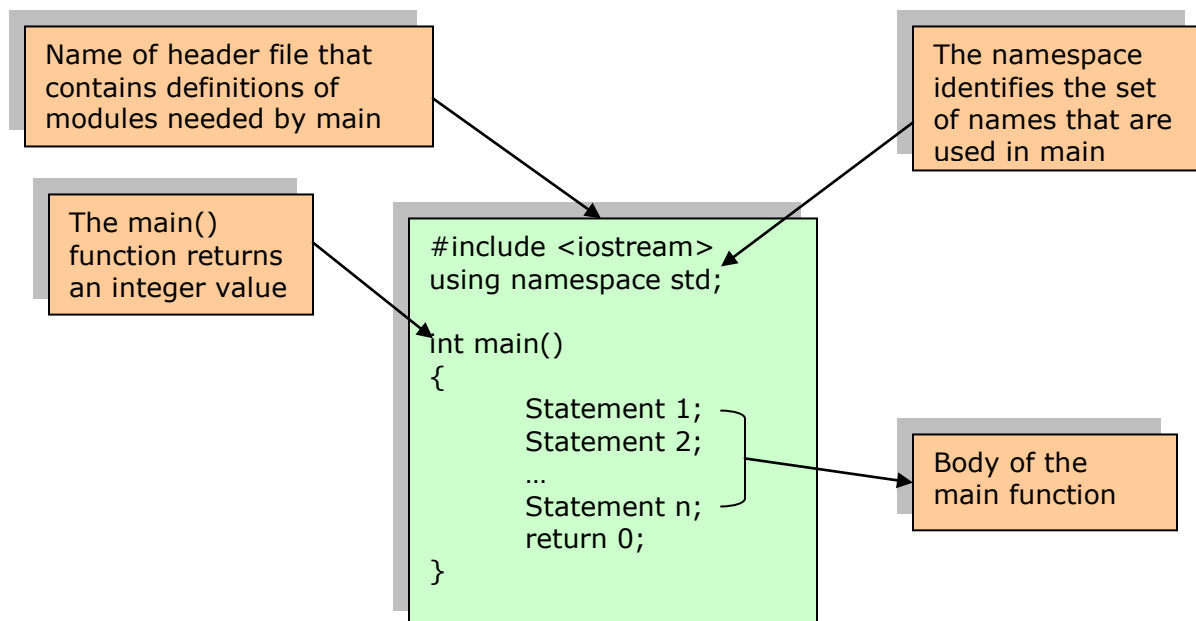
### 1.2.1 Learning Objectives

After reading this chapter you should be able to:

1. Describe the structure of a C++ program.
2. Write a program that performs data input, processing and output.
3. Define and use program variables.
  - 2.1 Perform variable assignment and reassignment.
  - 2.2 Exchange the value of variables using a temporary variable.
4. Define and use literal, defined and memory constants.

### 1.2.2 The Structure of a C++ Program

You have seen a simple C++ program in the last chapter. Figure 1.10 shows the basic structure of a C++ program.



**Figure 1.10 Basic Structure of a C++ Program**

Most often, the statements in a computer program perform three basic tasks:

- Data Input
- Data Processing
- Data Output

The word data means numbers, words or in general any symbols that are processed by a program.

In the next section, we will develop a program that performs all these three tasks.

### 1.2.2.1 Data Input, Processing and Output

Imagine that you have just got your results (hope you did well!) and now you wish to compute the percentage you scored in your exams.

You could impress your friends by writing a computer program that allows your friend to input their obtained marks and the total marks, then performs the steps necessary to calculate percentage and displays the result.

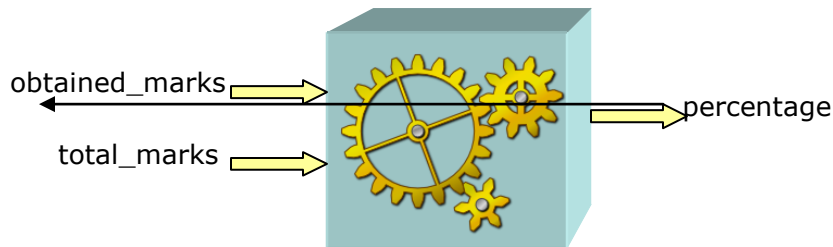


Figure 1.11 Percentage Calculator Program

Value addition: Animation	
Heading text: Data Input, Processing and Output	
The instructions for a computer program to calculate percentage will be as below:	
<b>DATA INPUT</b>	1. Prompt the user for marks obtained in the exam. 2. Prompt the user for total marks.
<b>DATA PROCESSING</b>	3. Calculate percentage = marks obtained / total marks * 100.
<b>DATA OUTPUT</b>	4. Display the percentage.
<b>Source:</b> Self made	

**Data Input:** This part of a program brings in data from an outside source into a program. Most often, this data is typed at the computer keyboard by a user of the program.

The "cout <<" statement in C++ is used to prompt the user to enter some data. The "cin >>" statement causes the program to wait until the user types in something.

For example, if you want to ask the user to enter the marks obtained, you would write,

```
cout << "Enter the marks obtained:";
cin >> obtained_marks;
```

Here, obtained\_marks is a placeholder for whatever value the user enters at the keyboard.

Similarly, to get the total\_marks from the user you would write,

```
cout << "Enter the total marks:";
cin >> total_marks;
```

Again, total\_marks is a placeholder for whatever marks the user enters at the keyboard.

Placeholders are actually called [variables](#) in C++. We will learn more about them in the next section.

**Data Processing:** This part of the program takes the input data and transforms it according to how the program is defined.

For example, to calculate the percentage from the obtained\_marks and the total\_marks, you would write,

```
percentage = obtained_marks/total_marks * 100.0;
```

**Data Output:** This part of the program displays the desired results.

For example, to display the percentage the friend scored, you would write,  
cout << "Percentage scored:" << percentage;

### Value addition: Source Code

#### Heading text: Percentage Calculator Program

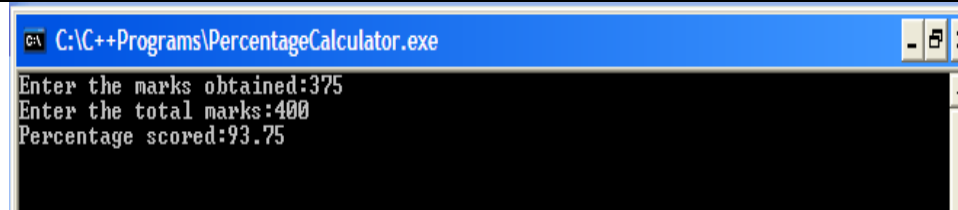
```
#include <iostream>
using namespace std;
int main(void)
{
    int    obtained_marks, total_marks;
    float  percentage;
    cout << "Enter the marks obtained:";
    cin >> obtained_marks;
    cout << "Enter the total marks:";
    cin >> total_marks;
    percentage = 100.0 * obtained_marks/total_marks;
```

```
    cout << "Percentage scored:" << percentage;  
    return 0;  
}
```

**Source:** Self made

### Value addition: Program in Execution

#### Heading text: Percentage Calculator Program



**Source:** Self made

### Value addition: Style Tips

#### Heading text: STYLE TIP!!

1. **Use Input Prompts** - Whenever you want the user to input some data, always provide a prompt telling the user that data is needed and explain what kind of data is required. If you do not prompt the user, the user will not know that execution has paused for user input and might just think that the program has finished execution. Always use a cout statement before a cin statement.
2. **Multiple Input Prompts** - If you want multiple prompts from the user, you can use multiple cin statements. However, if you want, you can ask for multiple inputs with one cin statement also. For example, if you want the user to enter three numbers, you can write

```
    cout << "Enter three numbers:";  
    cin >> number1 >> number2 >> number3;
```

The user can now enter three numbers on the same line separated by blank spaces. The first number entered will be assigned to the variable number1, the second to number2 and the third to number3.

**Source:** Self made

### 1.2.3 Program Variables

The placeholders in the Percentage Calculator program are called program **variables** in C++. A variable is a quantity whose value can change during program execution.

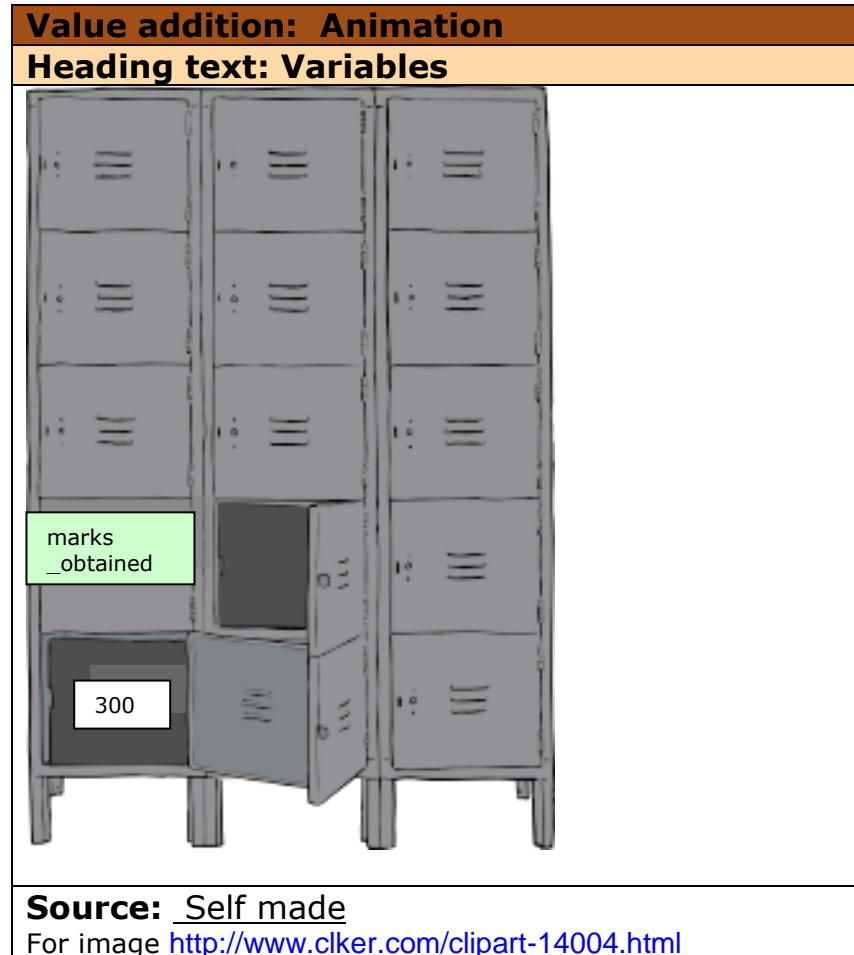
When we write a program, most of the time we don't know the actual numbers or other data that the user will enter when the program is executed. That is why we use variables to assign the input data.



A variable is called so because its values can *vary*. When the same program is executed again, the same variable can take another user input value.

In the price conversion program, *marks\_obtained* and *total\_marks* are input variables and *percentage* is an output variable.

Technically, a variable is the name for a storage location in the computer's internal memory. The *value* of the variable is the contents at that location. You can imagine storages location as a set of lockers. Each variable can be thought of as the name printed on the locker. The value of the variable can be thought of as the contents of the locker.



Variables in C++ have to be **declared** and **defined** before they can be used. A declaration is used to name a variable. Definitions are used to create the variable object. Most often variables are declared and defined in the same statement. For example,

```
int marks_obtained;
```

declares and defines the variable *marks\_obtained*.

The type of the variable has to be specified at the time of declaration. The type of the variables *marks\_obtained* and *total\_marks* in our percentage calculator program is **int**. This means that these variables are of the type integer. Specifying the type

tells the compiler how much memory space to reserve for this variable. It also means that only integer values can be stored in this variable.

You will read about other data types in the next chapter.

### Value addition: Did you Know?

#### Heading text: C++ Rules for variable names

1. Variable names must be one word. Spaces are not allowed in variable names. Underscores are allowed.  
"total\_marks" is fine but "total marks" is not.
2. Variable names can begin with either an alphabetic character or an underscore.
3. Variable names can consist of only alphabets, digits, and underscore.
4. Special characters, arithmetic operators, punctuation symbols, such as #, ^, and so on cannot be used in a variable name.
5. There are some **reserved words** in C++ that cannot be used as variable names.
6. A variable name declared for one data type cannot be used to declare another data type.

**Source:** Self made

### Value addition: Style Tips

#### Heading text: STYLE TIP!!

1. C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with the same spelling but written in small letters. For example, the variable name "percentage" differs from the variable name "PERCENTAGE" or "Percentage".
2. Variable names should be meaningful! The name should indicate the use of that variable. Single character variable names such as i, j, and so on, should be used only for temporary variables.
3. You can define multiple variables of the same type in one statement by separating each with a comma. For example, you can define three integer variables as below:  

```
int num1, num2, num3;
```

**Source:** Self made

## 1.2.4 Program Constants

Constants are data items that have fixed value. Their value does not change during program execution. Constants, like variables, have a data type.

In the statement,

```
number1 = number2 * 100;
```

the number 100 is an integer constant.

Integer constants can be represented in -

- Decimal notation - represented with a number, for example, 100.
- Octal notation - represented with the number preceded by a zero character, for example, 08.
- Hexadecimal notation - represented with the number preceded with the characters 0x, for example 0x12.

### Value addition: Did you Know?

#### Heading text: More on Constants

You can define three types of constants in C++:

1. **Literal Constants** – an unnamed constant. It is a *data value* that you type in a program. Such constants can be used to initialize variables or in program statements. For example,

```
int age = 42;
```

```
area = 3.14 * radius * radius;
```

In these statements 42 and 3.14 are literal constants.

2. **Defined Constants** - a symbolic or named constant. It is defined using a **define** preprocessor directive. For example,

```
#define PI 3.1417
```

A define preprocessor directive is usually written at the beginning of a program. It tells the compiler to replace all occurrences of the identifier PI in the program with 3.1417

It is common convention to write defined constants in uppercase.

A defined constant is used in a program like a variable name:

```
area = PI * radius * radius;
```

3. **Memory Constants** – a named constant. It is defined using a **const type qualifier** in C++ to tell the compiler that value of this data item cannot be changes during program execution. For example,

```
const int pi = 3.1417;
```

A memory constant has to be assigned a value at the time of definition.

**Source:** Self made

### 1.2.5 Variable Assignment and Reassignment

Variables can also be **initialized** at the time of declaration. For example, the statement,

```
int num1 = 10;
```

declares an integer variable named num and tells the compiler that when this variable is created, the contents of the memory location for this variable should take the integer value 10.

When we assign values to a variable using the assignment operator (equals sign), it's called an **assignment**.

Variables can be assigned values in program statements also. For example,

```
num2 = num1 * 53;
```

This statement multiplies the value of num1 with 53 and assigns the result to num2. If the value of num1 at the time of execution of this statement was 10, num2 now takes the value 530. The value of num1 remains unchanged.

Variables can also be reassigned values. For example, the following statement replaces the existing value of variable num2 with the value 62.

```
num2 = 62;
```

The value in the memory location allocated to num2 is erased and replaced with the value 62.

An assignment statement thus assigns the value of the expression to the right of an equals sign to the variable on the left of an equals sign.

### **Value addition: Did you Know?**

#### **Heading text: A Special Assignment!**

The following statement may look a little strange:

```
counter = counter + 1;
```

Lets see how this statement is executed:

- First, the right side of the assignment statement is evaluated.
- 1 is added to the current value of counter.
- The result is then assigned to the variable on the left, counter.

The net result of this statement is to increment the value of counter by 1. For example, if the value of counter before execution of the statement was 12, after execution, the value of counter will be 13.

**Source:** Self made

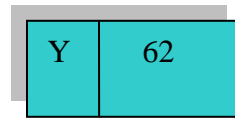
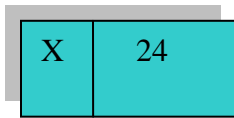
### **1.2.6 Exchanging the Value of Variables**

Now that you know how to assign values to variables, let us write a program that exchanges the values of two variables.

Imagine you have defined two integer variables x and y and initialized them with values 24 and 62 respectively.

## An Introduction to Programming

```
int X = 24, Y = 62;
```



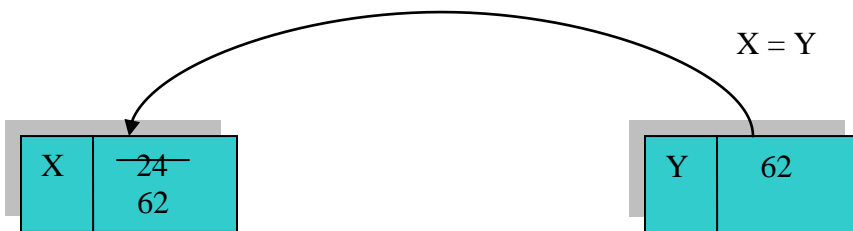
Now we want that the values of the variable to be exchanged, that is, we want X should take the value 62 and Y should take the value 24.

Can we write the following two statements to perform the exchange?

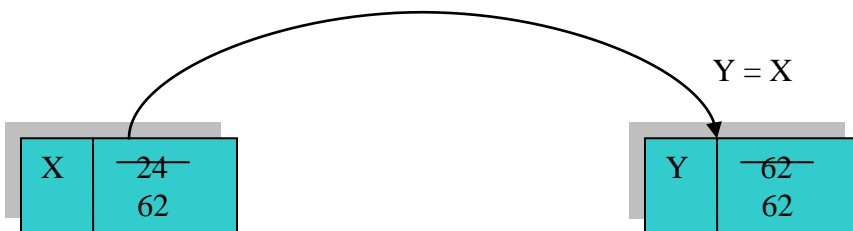
```
X = Y;
```

```
Y = X;
```

Let us see what happens. The first statement assigns the value of Y to X : so X takes the value 62.



Now, Y takes the value of the variable X.



This is not what we wanted!! Both the variables now have value 62. What has happened is that the first step erased the previous value of X and replaced it with the value of Y. The original value in X was lost!

So what can be done? One solution is to store the original value of X in a temporary variable, let us call it Z, before it is erased. We first write the value of X into Z. Then copy value of Y to X and lastly replace the contents of Y with the value in Z.

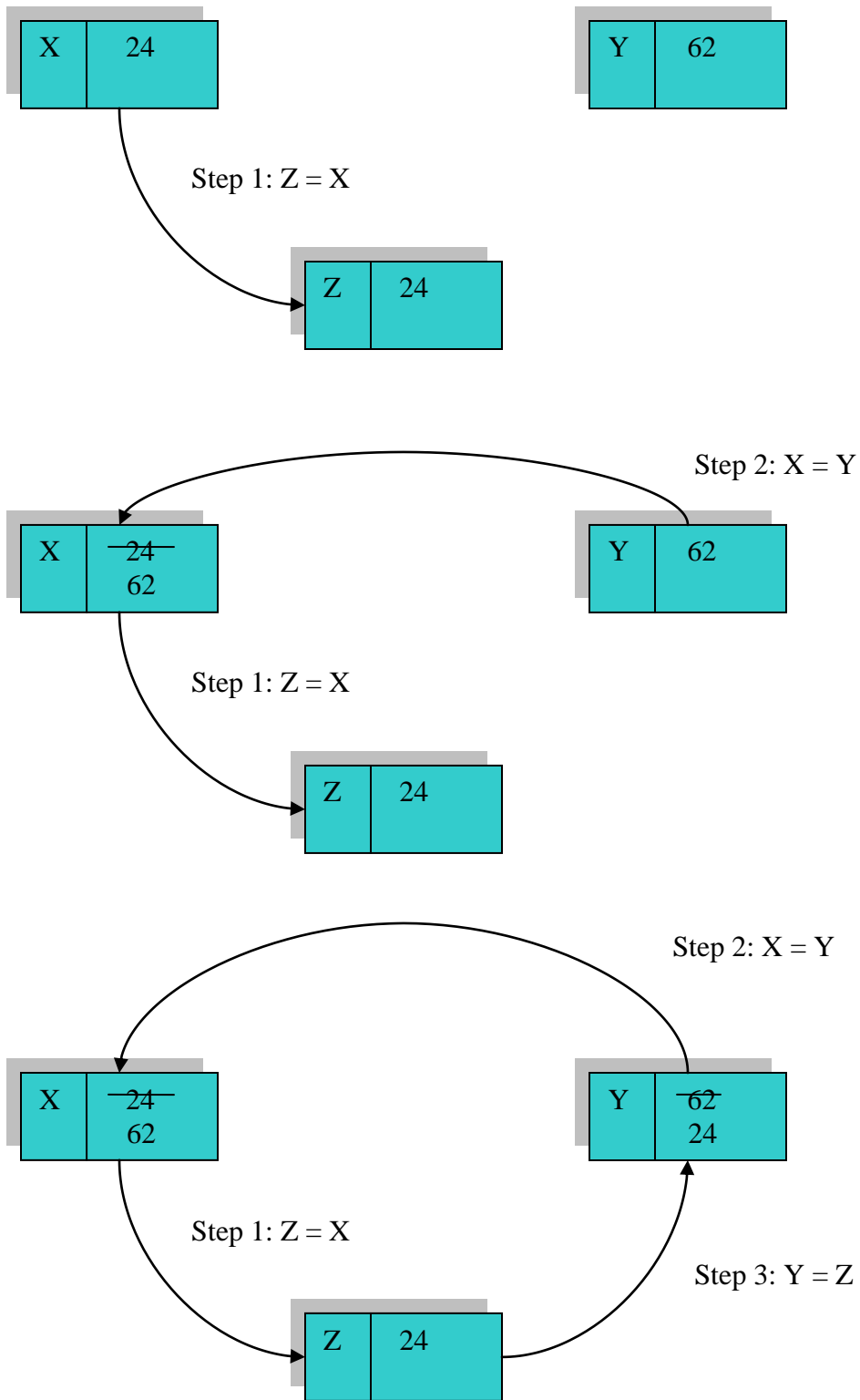
```
int X = 24, Y = 62, Z;
```

```
Z = X;
```

```
X = Y;
```

```
Y = Z;
```

# An Introduction to Programming



So we see that a simple task such as exchanging or swapping the two variables needed another temporary variable to solve the problem.

### Value addition: Source Code

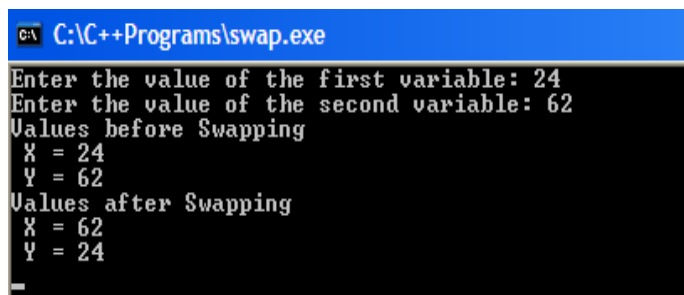
#### Heading text: The Swap Program

```
#include <iostream>
using namespace std;
int main()
{   int x, y, z;
    cout << "Enter the value of the first variable: ";
    cin >> x;
    cout << "Enter the value of the second variable: ";
    cin >> y;
    cout << "Values before Swapping"<< endl;
    cout << " X = " << x << endl; //endl causes next cout
                                //to print in the next line
    cout << " Y = " << y << endl;
    z = x;    // copy value of x into z
    x = y;    // replace value of x with value of y
    y = z;    // copy value of z into y
    cout << "Values after Swapping"<< endl;
    cout << " X = " << x << endl;
    cout << " Y = " << y << endl;
    return 0; // Return 0 for successful termination
}
```

**Source:** Self made

### Value addition: Program in Execution

#### Heading text: The Swap Program



```
CA C:\C++\Programs\swap.exe
Enter the value of the first variable: 24
Enter the value of the second variable: 62
Values before Swapping
X = 24
Y = 62
Values after Swapping
X = 62
Y = 24
```

**Source:** Self made

### Value addition: Common Coding Errors

#### Heading text: Program variables and constants

1. Using a space when declaring a variable name.
2. Forgetting to declare a variable before using it.
3. Declaring multiple variables in sequence. For example,  
int num1, int num2; // Wrong, compiler will give error  
int num1, num2; // Right

4. Declaring two variables of different types on the same line. For example,

```
int num1, float num2; // Wrong, compiler will give error
int num1;             // Right
float num2;           // Right
```

5. Initializing two variables by using one assignment statement

```
int num1, num2 = 10; // Wrong, compiler will not give error, but
                    // only value of num2 is initialized, not of
                    // num1.
int num1 = 10;       // Right
int num2 = 10;       // Right
```

**Source:** Self made

## Summary

- A program is a set of instructions written to solve a specific problem.
- The program development cycle consists of the following steps:
  - Analyze the problem
  - Design the solution
  - Code the program
  - Test the Code
- A program is written in a high level language such as C++ and must be compiled and linked to execute it.
- You must learn the syntax of a programming language to write a program in that language.
- The instructions written in a C++ program are called statements. Each statement in C++ must end with a semi colon.
- A program is built from a collection of functions.
- main() is a special function that all C++ programs must have. This is the function from where the program begins execution.
- To run a program, you need to type it in an editor, then compile and link it to create an executable file.
  
- Most often a computer program performs three basic tasks: Data Input, Data Processing, and Data Output.
- Data input statements transmit data from an outside source into a program.
- Processing statements manipulate data to obtain the desired results.
- Data output statements display the results on the screen.
- The word data means numbers, words or in general any symbols that are processed by a program.
- The "cout <<" statement in C++ can be used to prompt the user to enter some data.
- The "cin >>" statement can be used to accept the value that a user enters. It causes the program to wait until the user types in something.
- A program variable is a named storage location in memory. Its value can change during program execution.
- A variable has a data type and must be declared and defined in a program before it can be used.



- A program constant is a fixed value that cannot change. Three types of constants can be defined – literal constants, defined constants and memory constants.
- An assignment statement assigns values to a variable using the assignment operator (equals sign).
- Variables can be reassigned values during program execution.

### Exercises

- 1.1.1 List the steps in the program development cycle.
- 1.1.2 Write a C++ program that prints "Hello World" five times.
- 1.1.3 What will be the output of the following program?

```
#include <iostream>
using namespace std;
int main()
{
    cout << "It's a Small World After All!";
    return 0;
}
```

- 1.1.4 Identify three errors in the following program:

```
#include <iostream.h>
using namespace std;
int main
{
    cout << "Hello World"
    return 0;
}
```

- 1.2.1 Write a pair of statements that prompts for and inputs a user's age.
- 1.2.2 Write a C++ program that computes and displays the strike rate of a cricket player in a match when the user inputs the number of runs scored and number of balls faced. (Hint:  $\text{StrikeRate} = \text{RunsScored} / \text{BallsFaced} * 100$ )
- 1.2.3 Write a C++ program to exchange the value of two variables without using a temporary variable. (Hint: Use arithmetic operations)
- 1.2.4 What will be the output of the following program:

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10, y = 20;

    cout << " X = " << x << endl;
    cout << " Y = " << y << endl;

    x = y;
    y = x;

    cout << " X = " << x << endl;
    cout << " Y = " << y << endl;
}
```

```
        return 0;  
    }
```

1.2.5 Identify three errors in the following program fragment:

```
int main  
{  
    int percent age;  
    int percentage;  
    int percentage = 100;  
    int num1 = 2, float num2 = 20;  
}
```

## Glossary

**Comment:** Text inserted into a program for explanatory purpose, but ignored by the computer when the program is executed.

**Compiler:** A program that translates a source program written in some high-level programming language into machine code for some computer.

**Constant:** A fixed value that cannot change. Three types of constants can be defined – literal constants, defined constants and memory constants.

**Defined Constant:** a symbolic or named constant; It is defined using a **define** preprocessor directive.

**Function:** A group of statements written to perform a specific task. A function is identified by a function name and a function body. The function name is identified by a word followed by round brackets. The body is enclosed within a pair of curly braces.

**Literal Constant:** an unnamed constant; It is a *data value* that you type in a program

**Machine language:** A processor specific set of binary codes that correspond to actions to be taken by the processor of a machine.

**Memory Constant:** a named constant; It is defined using a **const type qualifier** in C++ to tell the compiler that value of this data item cannot be changes during program execution.

**Program:** A set of instructions carried out by a computer to accomplish a specific task.

**Program Development Cycle:** The process of analysis, design, coding and testing a program.

**Programming:** is the preparation and writing of detailed set of instructions for computers.

**return:** The return statement causes a function to return immediately.

**String:** A sequence of characters that is typed within double quotes.

**Syntax:** The syntax of a computer language is the rules of its usage.

## References

### 1. Works Cited

### 2. Suggested Reading

## An Introduction to Programming

1. B. A. Forouzan and R. F. Gilberg, Computer Science, A structured Approach using C++, Cengage Learning, 2004.
2. R.G. Dromey, How to solve it by Computer, Pearson Education
3. E. Balaguruswamy, Object Oriented Programming with C++ , 4<sup>th</sup> ed., Tata McGraw Hill
4. G.J. Bronson, A First Book of C++ From Here to There, 3<sup>rd</sup> ed., Cengage Learning.
5. Graham Seed, An Introduction to Object-Oriented Programming in C++, Springer
6. J. R. Hubbard, Programming with C++ (2<sup>nd</sup> ed.), Schaum's Outlines, Tata McGraw Hill
7. D S Malik, C++ Programming Language, First Indian Reprint 2009, Cengage Learning
8. R. Albert and T. Breedlove, C++: An Active Learning Approach, Jones and Bartlett India Ltd.

### **3. Web Links**

- 1.1 <http://www.cprogramming.com/begin.html>
- 1.2 [http://www.cplusplus.com/doc/tutorial/program\\_structure/](http://www.cplusplus.com/doc/tutorial/program_structure/)
- 1.3 <http://cnx.org/content/m11863/latest/>
- 1.4 <http://sourceforge.net/projects/dev-cpp/>
- 1.5 <http://bloodshed-dev-c.en.softonic.com/>
- 1.6 <http://www.uniqueness-template.com/devcpp/>
- 1.7 <http://gcc.gnu.org/>
- 1.8 <http://www.programmingforums.org/thread7219.html>
- 1.9 <http://pages.cs.wisc.edu/~beechung/ref/gcc-intro.html>
- 1.10 <http://www.cplusplus.com/doc/tutorial/variables/>