PROCESSING A C++ PROGRAM

The following are the general steps to execute a C++ program (see figure (1.1)).

1- When a C++ program is written it must be typed into the computer and saved to a file; a **text editor**, which is similar to a **word processor** program, is used for this task. The statements written by the programmer are called **source code**, and the **file** they are saved in is called the **source file**.

2- Source code **lines** that begin with the symbol # are processed by a program called a **preprocessor**.

3- After processing **preprocessor directives**, the next step is to verify that the program obeys the rules of the programming language-that is, the program is syntactically correct- and translates the program into the equivalent machine language. The **compiler** checks the **source program** for **syntax errors** and, if no error is found, translates the program into the equivalent **machine language**. The equivalent machine language program is called an **object program**.(.obj)

4- Although an **object program** contains machine language instructions, it is not a complete program. Because C++ is conveniently equipped with a **library of prewritten code** for performing common operations or sometimes-difficult tasks. For example; the library contains **mathematical functions**, such as calculating the **power**, **absolute** or **square root** of a number. This collection of code, called the **run-time library**, is extensive. Programs almost always use some part of it. When the **compiler** generates an **object program**, however, it **does not include** machine code for any **run-time library routines** the programmer might have used. **A program called a linker combines the object program with the programs form library**, and is used in the program to **create** the **executable code**.(.exe)

5- The **executable program** must next be loaded **into main memory for execution**. **A program called loader accomplishes this task**.

6- The final step is to **execute the program**.

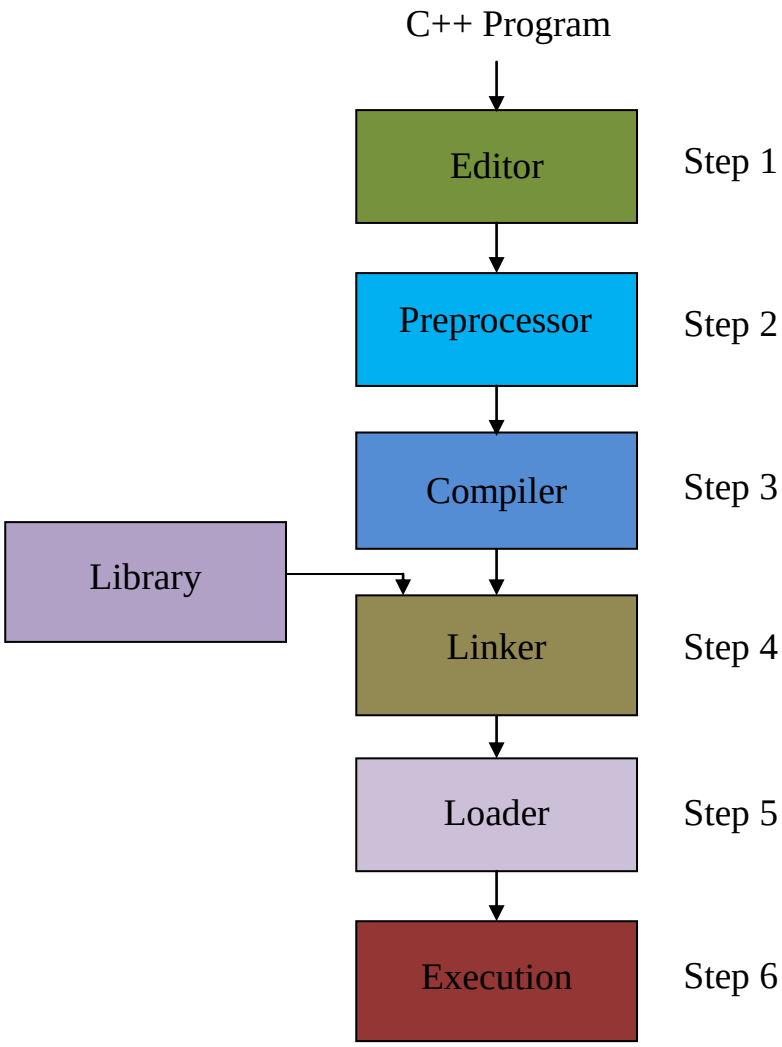

Figure (1.1) steps taken after writing a C++ program to execute it

DATA TYPES

Data type is a type of container that can hold a specific kind of program data. C++ language permits the use of several basic data types, some of these types are listed in table (1.1):

Table(1.1) basic data types

Data type	Meaning	Size
Char	Character	8 bit
Int	Integer	16 bit
long	Long integer	32 bit
float	Floating point	16 bit
double	Double floating point	32 bit
void	Valueless	-

Integer data type (int)

Integers are whole numbers without a fractional part. (zero and negative whole numbers are integers)

Long integer type (long)

It is the same as the integer, except it reserves twice the space for the regular integer.

Floating point data types

The data types that are used to represent real numbers (numbers with fractional parts) are **float** and **double**. The difference between these data types lie in the maximum number of significant digits that is, the number of decimal places in float values is 6 or 7 while the maximum number of significant digits in values belonging to the double is 15. This means that a double can store a number approximately ten times larger than float.

Void data type (void)

The void type specifies a valueless expression.

Character data type (char)

char is the smallest data type, it occupies only 1 byte (eight bits) of memory. The **char** data type is used to represent ASCII characters. When using the char data type, you enclose each character (letter, digit or special symbol) with single quotation marks. Examples: 'A', 'a', '0', '+', ' ', '$'. Each of the 128 values of the ASCII character set represents a different character. For example, the value 65 represents 'A', the value 48 represents '0' and the value 97 represents 'a'.

String data type

The **string** is a collection of characters, numbers or special symbols **enclosed in double quotation marks**. Examples: "Hello!", "1+2=", "a-b=", "what is your name?"

The string data type is a **programmer-defined data type** and is n**ot directly available for use in a program**. To use the string data type, you need to access its

definition form the header file **cstring**. Therefore, to use the string data type in a program you must include the following preprocessor directive:

#include<cstring.h>

The preprocessor directive #include tells the compiler to insert another file into your source file. In effect, the #include directive is replaced by the contents of the file indicated. The type of file usually included by # include is called a **header file**. Using a # include directive to insert another file into your source file is similar to pasting a block of text into a document with your word processor.

VARIABLES

The variable is a **memory location** whose **content may change** during program execution. The general form of a variable declaration is:

```
data type  variable name;
```

NOTE: every C++ statement must end with a semicolon (;).

Examples:

int number**;**

char ch;

string name;

When there is more than one variable in a declaration, the variables are separated by **comma** as illustrated in the following declaration statement:

float a, b, c**;**

int first, middle, last;

There are several rules to name a variable.

- Names may contain letters (A to Z, a to z), numbers (0 to 9), or underscores (_).

- The first character must be a letter or an underscore.
- Names cannot contain any symbols, such as - ! @ # $ % ^ & * ( ) " + = | ' \ , nor can they have any spaces.

CONSTANTS

The constant is a memory location whose content is not allowed to change during program execution.

The syntax to declare a constant is:

```
const data type  constant name = value;
```

Examples:

const int number = 55;

const char letter= 'a';

const string name = "mohammed";

INPUT STATEMENT

The first way to put data into a variable is to use the standard input device (usually the keyboard), this is accomplished via the use of **cin** and the operator >>. The syntax of **cin** and the operator >> is:

```
cin>>variable1>>variable2 ….;
```

This is called an **input** (**read**) **statement**. In C++, >> is called the **stream extraction operator**. When the **cin** statement is executed during the **run of a program**, the computer will pause until the user types in a value and presses the **Enter** key.

Examples:

cin>> number**;**

cin>> ch;

cin>>name;

The **cin** statement can be used to enter more than one value of the **same data type**, it can also be used to enter multiple values of **different data types** as shown in the following **examples**:

1.

string name1, name2;

cin>>name1>>name2;

2.

char a;

double b;

cin>>a>>b;

INITIALIZING VARIABLES IN DECLARATIONS

The second way to put data into a variable is to **initialize** the variable (that is, **give it a value**) at the time that you declare the variable.

Examples:

int cont = 0;

int a=3, b, c=5, d;

char letter = 'm';

string s= "welcome to C++";

Notice that, in the second declaration statement only the variables **a** and **c** are initialized.

ASSIGNMENT STATEMENT

The contents of a variable can vary over time. To change the value of a variable, you use an assignment statement. **The left hand side of an assignment consists of a variable.**

The right hand is an expression that has a value. The value is stored in the variable, overwriting its previous contents.

The = sign does not mean that the left hand side is equal to the right hand side but that the right hand side value is copied into the left hand side variable. You should not confuse this assignment operation with the = used in algebra to denote equality. The assignment operator is an instruction to do something, **namely place a value into a variable**. The mathematical equality states the fact that two values are equal.

The general form of assignment statement declaration is:

```
variable = expression;
```

Examples:

z= 3*x +5;

a= b;

x= m+n;

y=9;

OUTPUT STATEMENT

In C++, output on the standard output device which is usually the screen is accomplished via the use of **cout** and the operator <<. The general syntax of **cout** together with << is:

```
cout<<variable or expression or string or manipulator<<variable or expression or string or manipulator,…;
```

This is called an **output statement**. In C++ << is called **stream insertion operator**. Generating output with **cout** follows two rules:

1. The expression is evaluated and its value is printed at the current **insertion point** (on the screen the insertion point is where the **cursor** is) on the output device.

2. A **manipulator** is used to format the output. The simplest manipulator is **endl** (pronounced "end-line" or "end-L", which causes the insertion point to move to the **beginning of the next line**.

Examples:

```
cout<<z;
cout<<2+3;
cout<<x+y-5;
cout<<"x="<<x<<endl;
cout<<"Hello!";
```

BASIC PROGRAM CONSTRUCTION

Every C++ program must be written under a special structure. Example (1.1) illustrates this structure.

Example 1.1: this example illustrates the basic structure of any C++ program.

```
#incude<iostream.h>
int main()
{
//write your program here
return 0;
}
```

Despite its small size, this program demonstrates the main parts of a C++ program. Let's examine it in detail.

Line 1: #include<iostream.h>

Is a **preprocessor directive** notifies the **preprocessor** to **include** in the program the contents of the **input/output stream header file <iostream.h>** this header must be

included for any program that outputs data to the screen or inputs data from the keyboard using C++ stream input/output.

Line 2: int main()

Defined a **function** called **main. A function is a collection of programming instructions that carry out a particular task**. Every C++ program must have a main function. Most C++ programs contain other functions beside main but, **it begins executing at function main**, even if main is not the first function in the program.

Line 3: the left brace **{**

The left brace must begin the body of every function.

Line 4: //write your program her

Each begin with **//** indicating that the remainder of each line is a **comment**. You insert comments to document your program and to help other people read and understand them. **Comments do not cause the computer to perform any action when the program is run**.

Line 5: return 0;

The return statement denotes the end of the main function. When the main function ends, the program terminates. The zero value is a signal that the program runs successfully.

Line 6: the right brace must end each function's body.

Example 1.2: write a program to calculate the area of a rectangle.

```
#include<iostream.h>
int main()
{
int length,width, area;
cout<<"enter the length: ";
cin>>length;
cout<<"enter the width: ";
cin>>width;
area=length*width;
cout<<"the area of the rectangle is: "<<area;
return 0;
}
```

Program output:

```
enter the length: 2
enter the width: 7
the area of the rectangle is: 14
```