Structured Programming

First class

1.1 Introduction

Many people use the Internet to look for information and to communicate with others. This is all made possible by the availability of different software, also known as computer programs. Without software, a computer is useless. Software is developed by using programming languages. The programming language C++is especially well suited for developing software to accomplish specific tasks. Our main objective is to learn how to write programs in the C++programming language. Before you begin programming, it is useful to understand some of the basic terminology and different components of a computer.

A computer is an electronic device capable of performing **commands**. The basic commands that a computer performs are **input** (get data), **output** (display result), **storage**, and performance of **arithmetic** and **logical** operations.

Hardware is the collection of physical components that constitute a computer system. Computer hardware is the physical parts or components of a computer

Software is programs written to perform **specific tasks**. For example, word processors are programs that you use to write letters, papers, and even books. All software is written in programming languages. There are two types of programs: **system programs** and **application programs**.

System programs control the computer. The system program that loads first when you turn on your PC is called the **operating system**. Without an operating system, the computer is useless. The operating system monitors the overall activity of the computer and provides services.

Application programs perform a specific task. Word processors, spreadsheets, and games are examples of application programs. The operating system is the program that runs application programs.

1.2 The Language of a Computer

When you press A on your keyboard, the computer displays A on the screen. But what is actually stored inside the computer's main memory? What is the language of the computer? How does it store whatever you type on the keyboard?

Digital signals are used inside the computer to process information. Digital signals represent information with a sequence of 0s and 1s.

Because digital signals are processed inside a computer, the language of a computer, called **machine language**, is a sequence of **0s** and **1s**. The digit **0** or **1** is called a **binary digit**, or **bit**.

A computer program, or just **a program**, is a sequence of instructions, written to perform a specified task with a computer.

Programming is the process of writing instructions for a computer in certain order to solve a problem

A programming language is a formal constructed language designed to create programs to control the behavior of a computer.

The lowest-level programming language is **Machine language**. It is the only languages understood by computers. While easily understood by computers, **machine languages** are almost impossible for humans to use because they consist entirely of numbers.

Programmers, therefore, use either a **high-level programming language** or an **assembly language**. An assembly language contains the same instructions as a

machine language, but the instructions and variables have names instead of being just numbers.

Programs written in **high-level languages** are translated into assembly language or machine language by a **compiler**. Assembly language programs are translated into machine language by a program called an **assembler**. Figure 1-1 shows these levels

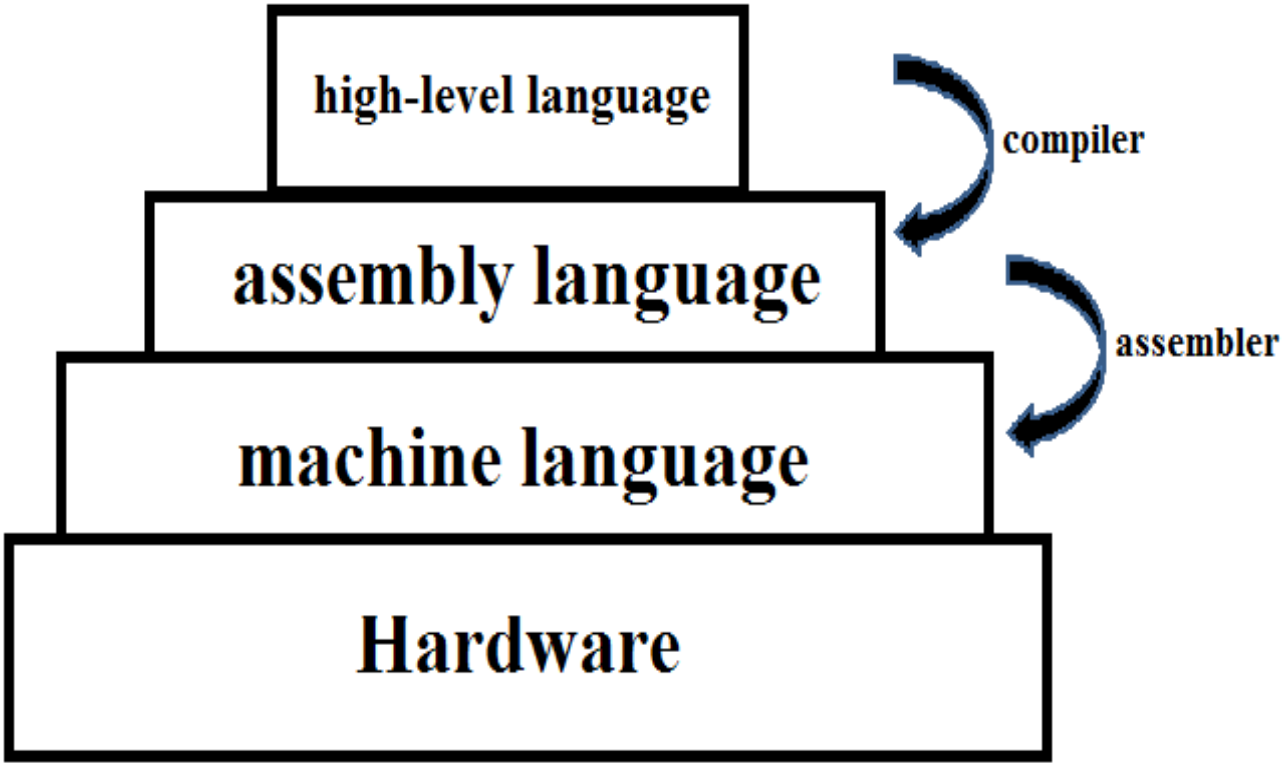


Figure 1-1 The levels of programming languages

EXERCISE 1

1. Write handwritten essay about the evolution of programming language.

1.3 Programming and Algorithm

Programming is a process of problem solving. Different people use different techniques to solve problems. Some techniques are nicely outlined and easy to follow. They not only solve the problem, but also give insight into how the solution was reached.

These problem-solving techniques can be easily modified if the domain of the Problem changes. To be a good problem solver and a good programmer, you must follow good problem solving techniques. One common problem-solving technique includes analyzing a problem, outlining the problem requirements, and designing steps, called an algorithm, to solve the problem.

Algorithm: A step-by-step problem-solving process in which a solution is arrived at in a finite amount of time.

In a programming environment, the problem-solving process requires the following three steps:

1. **Analyze the problem**, outline the problem and its solution requirements, and design an algorithm to solve the problem.

2. **Implement the algorithm** in a programming language, such as C++, and verify that the algorithm works.

3. **Maintain the program** by using and modifying it if the problem domain changes.

Figure 1-2 summarizes this three-step programming process.

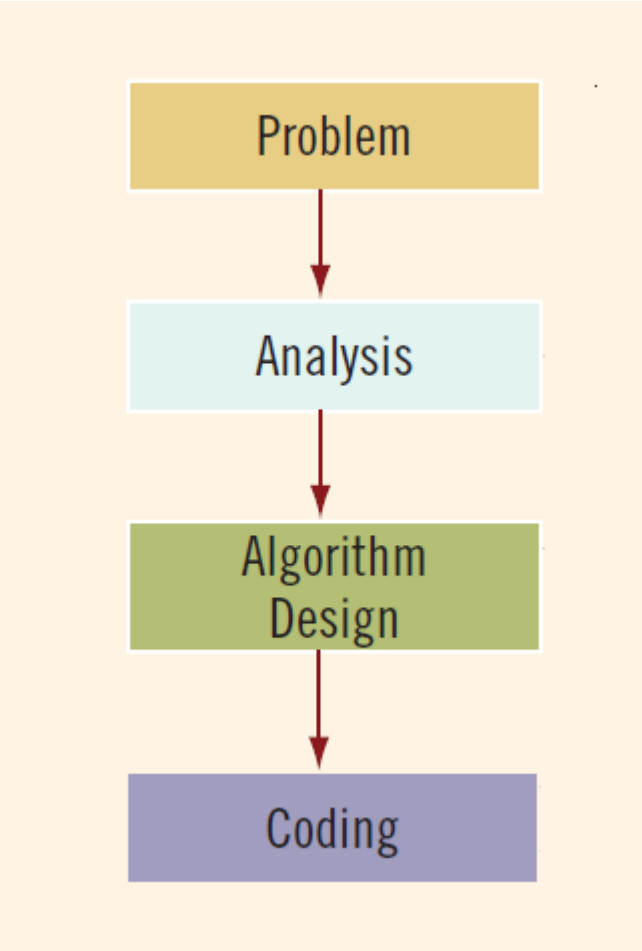


Figure 1-2 programming process

1.4 Algorithms

Informally, an ***algorithm*** is any well-defined computational procedure that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***. An algorithm is thus a sequence of computational steps that transform the input into the output. The algorithm describes a specific computational procedure for achieving that input/output relationship.

For Example:

Example 1: Algorithm Sam to go to school

Input: Sam in house

Output: Sam in school

Step1: start

Step2: Sam is leaved the house.

Step3: he walks down the street.

Step4: he Arrive to school

Step5: he goes through the door into the school

Step6: end

Example 2: Algorithm to find the sum of two numbers

Input: Two numbers.

Output: the sum of two numbers.

Step1: start

Step2: Read the Value of A and B.

Step3: SUM = A+B.

Step4: Display SUM

Step5: end

Example 3: Algorithm to find the multiply of two numbers

Input: Two numbers.

Output: the multiply of two numbers.

Step1: start

Step2: Read the Value of A and B.

Step3: PRODUCT = A*B.

Step4: Display PRODUCT

Step5: end

Example 4: Algorithm to find the division of two numbers

Input: Two numbers.

Output: the division of two numbers.

Step1: start

Step2: Read the Value of A and B.

Step3: DIV = A/B.

Step4: Display DIV.

Step5: end

Example 5: Algorithm to find the area of rectangle

Input: the length and the width of rectangle.

Output: the area of rectangle.

Step1: start

Step2: Read the Value of length and the width.

Step3: AREA = length * width.

Step4: Display AREA.

Step5: end

EXERCISE 2:

1. Design an algorithm to find the sum of three numbers
2. Design an algorithm to find the multiply of three numbers
3. Design an algorithm to find the square of any number
4. Design an algorithm to find the root of any natural number
5. Design an algorithm to find the area of square
6. Design an algorithm to find the area triangle
7. Design an algorithm to find the circumference of circle
8. Design an algorithm to find the value of Z

Where z =3X+5Y

9. Design an algorithm to find the value of Z

   Where z =X^3+X^2

10. Design an algorithm to find the value of Z :Where z =$X^3+X^2/2$

1.5 C++Programming Languages:

C++ (pronounced cee plus plus) is a general purpose programming language. C++is not a completely new language. It can be thought of more as an evolutionary advancement of C. Both languages share the fundamental concepts for using statements, data types, operators, function definition.

EXERCISE 3:

1. Write a handwritten essay about the history of C++.

1.6 C++Program Development Process (PDP):

C++programs typically go through six phases before they can be executed. These phases are:

1. **Edit**: The programmer types a C++**source program**, and makes correction, if necessary. Then file is stored in disk with extension (.cpp).

   ((**Source program**: A program written in a high-level language)).

2. The C++program given in the preceding section contains the statement **#include <iostream>.** In a C++program, statements that begin with the symbol # are called preprocessor directives. These statements are processed by A program called **preprocessor**.
3. the next step is to verify that the program obeys the rules of the programming language—that is, the program is syntactically correct—and translate 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.

((**Object program:** The machine language version of the high-level language program.))

4. **Linking:** The programs that you write in a high-level language are developed using an integrated development environment (IDE). The IDE contains many programs that are useful in creating your program. For example, it contains the necessary code (program) to display the results of the program and several mathematical functions to make the programmer's job somewhat easier. Therefore, if certain code is already available, you can use this code rather than writing your own code. Once the program is developed and successfully compiled, you must still bring the code for the resources used from the IDE into your program to produce a final program that the computer can execute. This prewritten code (program) resides in a place called the **library**. A program called a **linker** combines the object program with the programs from libraries.

((**Linker**: A program that combines the object program with other programs in the library and is used in the program to create the executable code.))

5. **Loading:** You must next load the executable program into main memory for execution. A program called a loader accomplishes this task.

((**Loader:** A program that loads an executable program into main memory.**))**

6. The final step is to **execute** the program. Figure 1-2 shows how a typical C++ program is processed.

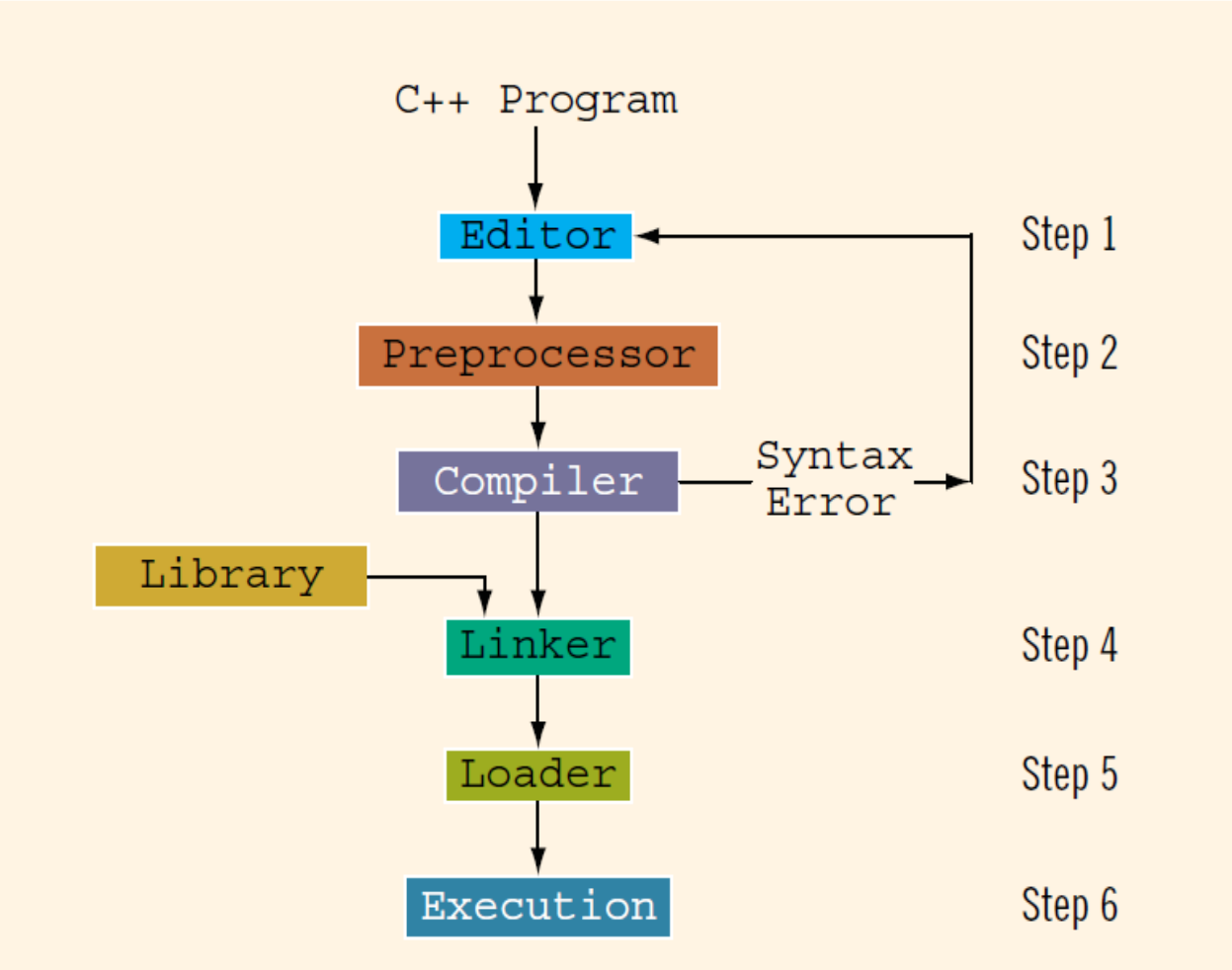


FIGURE 1-2 processing a C++program

1.7 Structure of C++ program

Atypical C++ program is consists of two parts:

1. **The global part** contains necessary declarations, definitions, and files needed by the entire program.
2. **The main function**, referred to as **main(),** is the most important part, the program execution starts with this function.

Example 6: a C++ program that displays "hello, world" on the screen.

```
#include <iostream.h>            // global part
void main()                      // function heading
{                                // start of function body
   cout << " hello, world ";     // message
}                                                    // end of function body
```

The output of this program is:

```
hello, world
```

The **comment** after **"//"** symbols extends to the end of the line, comments are helpful remarks that appear in the program listing but have no effect on the way the program runs.

C++ is **case-sensitive**; that is, it distinguishes between uppercase characters and lowercase characters. This means you must be careful to use the same case as in the examples. For example, this program uses cout. If you substitute Cout or COUT, the compiler rejects your offering and accuses you of using unknown identifiers. (The compiler also is **spelling-sensitive**, so don't try kout or coot, either.)

Global part include stream I/O header file which, among several other things, contains the definition needs to allow the program to read input and write output.

The function **main()** is the major function that points to the place in the program where execution starts. The body of the function _a group of statements that specify the actions to be performed _ is enclosed between two **curly braces {}.**

The **cout** is used for output, it is to print out the desired message.

Example 7: a C++ program to display your name and age in different lines.

```
#include <iostream.h>            // global part

void main()                      // function heading

{                                // start of function body

   cout << " Ali ";              // name
   cout <<"\n18";                //age

}                                // end of function body
```

The output of this program is

```
Ali
18
```

Example 8: a C++ program to display four lines of text, including the sum of two numbers.

```
#include <iostream>          // global part
void main()                  // function heading
{
cout << "My C++ program." << endl;
cout << "The sum of 2 and 3 = " << 5 << endl;
cout << "7 + 8 = " << 7 + 8 << endl;
}
```

When you compile and execute this program, the following lines are displayed on the screen:

```
My C++ program.
The sum of 2 and 3 = 5
7 + 8 = 15
```

These lines are displayed by the execution of the following statements.

```
cout << "My C++ program." << endl;
cout << "The sum of 2 and 3 = " << 5 << endl;
cout << "7 + 8 = " << 7 + 8 << endl;
```

Next, we explain how this happens. Let us first consider the following statement:

```
cout << "My C++ program." << endl;
```

This is an example of a **C++ output statement**. It causes the computer to evaluate the expression after the pair of symbols << and display the result on the screen.

Usually, a C++ program contains various types of expressions such as:

1. **Arithmetic** expressions are evaluated according to rules of arithmetic operations, which you typically learn in algebra. For example, 7 + 8 is arithmetic expression

2. Anything in double quotes is a **string**. For example, "My C++ program." and "7 + 8 = " are strings. Typically, a string evaluates to itself.

Also note that in an output statement, **"endl"** causes the insertion point to move to the beginning of the next line. (On the screen, the insertion point is where the cursor is.) Therefore, the preceding statement causes the system to display the following line on the screen.

```
My C++ program.
```

Let us now consider the following statement.

```
cout << "The sum of 2 and 3 = " << 5 << endl;
```

This output statement consists of two expressions. **The first expression** (after the first <<) is "The sum of 2 and 3 = " and **the second expression** (after the second <<) consists of the number 5.

The expression "The sum of 2 and 3 = " is a string and evaluates to itself. The second expression, which consists of the number 5 evaluates to 5. Thus, the output of the preceding statement is:

```
The sum of 2 and 3 = 5
```

Let us now consider the following statement.

```
cout << "7 + 8 = " << 7 + 8 << endl;
```

In this output statement, the expression "7 + 8 = ", which is a string, evaluates to itself. Let us consider the second expression, 7 + 8. This expression consists of the numbers 7 and 8 and the C++ arithmetic operator +. Therefore, the result of the expression 7 + 8 is the sum of 7 and 8, which is 15. Thus, the output of the preceding statement is:

```
7 + 8 = 15
```

EXERCISE 3:

1- Write a C++ program that displays your name, your age and your birthday.

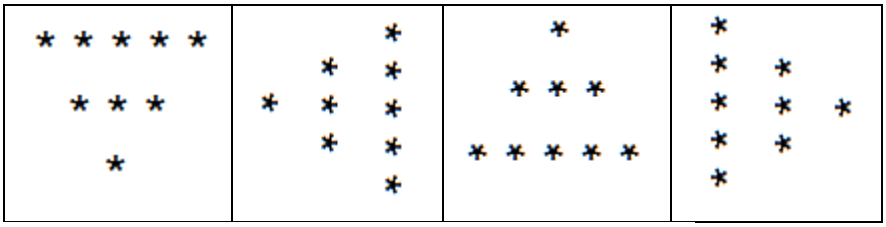
2- Write a C++ program to solve the following expression:

200+500	30*76	40+50-7	7/3	500-600*40	10+43-87*40

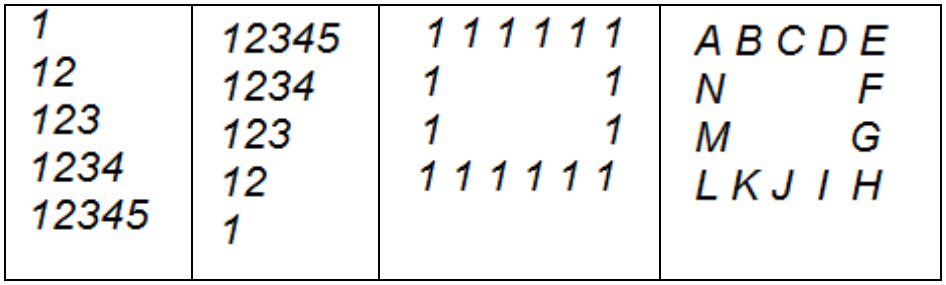
3- Write a C++ program that displays that pattern:

```
1*2*3*4*5*6
```

4-Write a C++ program that displays that pattern:



5-Write a C++ program that displays that pattern:



1.8 Token

The smallest individual unit of a program written in any language is called a token. C++'s tokens are divided into **special symbols**, **keyword**, and **identifiers**.

1.8.1 Special symbols

Following are some of the special symbols:

1. Mathematical symbols for addition, subtraction, multiplication, and division

+	-	*	/

2. Punctuation marks taken from English grammar.

.	;	?	,

In C++, commas are used to separate items in a list. Semicolons are used to end a C++ statement.

3. Blank is also a special symbol. You create a blank symbol by pressing the space bar (only once) on the keyboard.
4. Tokens made up of two characters that are regarded as a single symbol. No character can come between the two characters in the token, not even a blank.

<=	!=	==	>=

1.8.2 Reserved Words (Keywords)

Reserved words are also called keywords. The letters that make up a reserved word are always lowercase. Like the special symbols, each is considered to be a single symbol. Furthermore, word symbols cannot be redefined within any program; that is, they cannot be used for anything other than their intended use. Some of C++Language Reserved Words:

break	case	char	cin	cout
delete	double	else	enum	false
float	for	goto	if	int
long	main	private	public	short
sizeof	switch	true	union	void

1.8.3 Identifiers

Identifiers are names of things that appear in programs, such as **variables**, **constants**, and **functions**. All identifiers must obey C++'s rules for identifiers. A C++ identifier consists of letters, digits, and the underscore character (_) and must begin with a letter or underscore. Examples of Illegal Identifiers:

Illegal Identifier	Description
employee Salary	There can be no space between employee and Salary
Hello!	The exclamation mark cannot be used in an identifier.
one + two	The symbol + cannot be used in an identifier.
2nd	An identifier cannot begin with a digit.

1.9 Simple Variables

Programs typically need to store information. To store an item of information in a computer, the program must keep track of three fundamental properties:

- Where the information is stored(name of variable)
- What value is kept there
- What kind of information is stored(data type)

Example 9:

```
int count =5;
```

Where **int** is the kind of information, **count** is name of information, **5** is value of information.

1.9 .1 Integer Variables

Integer variables are variables that can have only values that are whole numbers. The number of players in a football team is an integer. You already know that you can declare integer variables using the keyword int . Variables of type int occupy 4 bytes in memory and can store both positive and negative integer values. The upper and lower limits for the values of a variable of type int correspond to the maximum and minimum signed binary numbers, which can be represented by 32 bits. Here is an example of defining a variable of type int :

int num=0;	int length;
int sum;	int toeCount = 10;

Example 10: A C++ program to add two numbers.

```
#include <iostream.h>                    // global part

void main()                              // function heading

{
int first_num;
int second_num;
first_num=5;
second_num=6;
cout<<" first_num+ second_num=";
cout<< first_num+ second_num;
}
```

The output of this program is

```
first_num+ second_num=11
```

1.9 .2 Character Data Types

The char data type serves a dual purpose. It specifies a one - byte variable that you can use either to store integers within a given range, or to store the code for a single **ASCII** character, which is the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. You can declare a char variable with this statement:

```
char letter = 'A';
```

This declares the variable with the name letter and initializes it with the constant ‘ A ’ . Note that you specify a value that is a single character between single quotes.

Because the character ‘ A ’ is represented in ASCII by the decimal value 65, you could have written the statement as:

```
char letter = 65; // Equivalent to A
```

Examble11: A C++ program to display ASCII number for M

```
#include <iostream>
void main()
{
 char c = 'M';       // assign ASCII code for M to c
 int i = c;          // store same code in an int
 cout << "The ASCII code for " << c << " is " << i << "\n";
 cout << "Add one to the character code:\n";
 c = c + 1;
 i = c;
 cout << "The ASCII code for " << c << " is " << i << '\n';
}
```

Here is the output:

```
The ASCII code for M is 77
Add one to the character code:
The ASCII code for N is 78
```

1.9 .3The Boolean Type

Boolean variables are variables that can have only two values: a value called true and a value called false. The type for a logical variable is bool, named after George Boole, who developed Boolean algebra, and type bool is regarded as an integer type. Boolean variables are also referred to as **logical variables.** Variables of type

bool are used to store the results of tests that can be either true or false, such as whether one value is equal to another. You could declare the name of a variable of type bool with the statement:

```
bool testResult;
```

Of course, you can also initialize variables of type bool when you declare them:

```
bool colorIsRed = true;
bool isready = true;
bool isready = false;
bool isready = -100;       // isready = true
bool isready = 0;          // isready = false
```

1.9 .4 Floating - Point Types

Values that are not integral are stored as **floating - point** numbers. A floating - point number can be expressed as a **decimal value** such as 112.5, or with an **exponent** such as 1.125E2 where the decimal part is multiplied by the power of 10 specified after the E (for **E**xponent). Our example is, therefore, 1.125 × 10 2 , which is 112.5.You can specify a floating - point variable using the keyword double , as in this statement:

```
double in_to_mm = 25.4;
```

A variable of type double occupies 8 bytes of memory and stores values accurate to approximately 15 decimal digits.

If you don ’ t need 15 digits’ precision, and you don ’ t need the massive range of values provided by double variables, you can opt to use the keyword float to declare floating - point variables occupying 4 bytes. For example:

```
float pi = 3.14159
```

Example 12: A C++ program to add two floating – point numbers.

```
#include <iostream.h>                    // global part

void main()                              // function heading

{
double first_fnum;
double second_fnum;
first_fnum=5.3;
second_fnum=6.5;
cout<<" first_fnum+ second_fnum=";
cout<< first_fnum+ second_fnum;
}
```

The output of this program is

```
first_fnum+ second_fnum=11.8
```

1.10 Input

The main objective of a C++ program is to perform calculations and manipulate data. Recall that data must be loaded into main memory before it can be manipulated. In this section, you will learn how to put data into the computer's memory. Storing data in the computer's memory is a two-step process:

1. Instruct the computer to allocate memory.
2. Include statements in the program to put data into the allocated memory.

1.10.1 Allocating Memory with Constants and Variables

When you instruct the computer to allocate memory, you tell it not only what names to use for each memory location, but also what type of data to store in those memory locations. Knowing the location of data is essential, because data stored in one memory location might be needed at several places in the program. Knowing

what data type you have is crucial for performing accurate calculations. It is also critical to know whether your data needs to remain fixed throughout program execution or whether it should change. Some data must stay the same throughout a program. For example, the pay rate is usually the same for all part-time employees. A conversion formula that converts inches into centimeters is fixed, because 1 inch is always equal to 2.54 centimeters. When stored in memory, this type of data needs to be protected from accidental changes during program execution. In C++, you can use a **constant** to instruct a program to mark those memory locations in which data is fixed throughout program execution.

Constant: A memory location whose content is not allowed to change during program execution.

To allocate memory, we use C++'s declaration statements. The syntax to declare a Constant is:

```
const dataType identifier = value;
```

In C++, **const** is a reserved word.

Example 13: Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

The first statement tells the compiler to allocate memory (eight bytes) to store a value of type double, call this memory space CONVERSION, and store the value 2.54 in it.

Throughout a program that uses this statement, whenever the conversion formula is needed, the memory space CONVERSION can be accessed. The meaning of the other statements is similar.

Using a named constant to store fixed data, rather than using the data value itself, has one major advantage. If the fixed data changes, you do not need to edit the entire program and change the old value to the new value wherever the old value is used. Instead, you can make the change at just one place, recompile the program, and execute it using the new value throughout. In addition, by storing a value and referring to that memory location whenever the value is needed, you avoid typing the same value again and again and prevent accidental typos. If you misspell the name of the constant value's location, the computer will warn you through an error message, but it will not warn you if the value is mistyped.

In some programs, data needs to be modified during program execution. For example, after each test, the average test score and the number of tests taken changes. Similarly, after each pay increase, the employee's salary changes. This type of data must be stored in that memory cells whose contents can be modified during program execution. In C++, memory cells whose contents can be modified during program execution are called variables.

Variable: A memory location whose content may change during program execution.

The syntax for declaring one variable or multiple variables is:

```
dataType identifier, identifier, . . .;
```

Example 14:

Consider the following statements:

```
double amountDue;
int counter;
char ch;
int x, y;
```

The first statement tells the compiler to allocate enough memory to store a value of

the type **double** and call it amountDue. The second and third statements have similar conventions. The fourth statement tells the compiler to allocate two different memory spaces, each large enough to store a value of the type **int**; name the first memory space x; and name the second memory space y.

1.10.2 Putting Data into Variables

Now that you know how to declare variables, the next question is: How do you put data into those variables? In C++, you can place data into a variable in two ways:

1. Use C++'s assignment statement: it takes the following form:

```
variable = expression;
```

In an **assignment statement**, the value of the **expression** should match the **data type** of the **variable**. The expression on the right side is evaluated, and its value is assigned to the variable (and thus to a memory location) on the left side. A **variable** is said to be initialized the first time a value is placed in the variable. In C++, **"="** is called the **assignment operator**.

Example 13: a C++ program illustrates how data in the variables are manipulated.

```
#include <iostream>
void main()
{
int num1, num2;                 //statement 1
double sale;                    //statement 2
char first;                                 //statement 3
num1 = 4;                                   //statement 4
cout << "num1 = " << num1 << endl; //statement 5
num2 = 4 * 5 - 11;                          //statement 6
cout << "num2 = " << num2 << endl; //statement 7
sale = 0.02 * 1000;                                 //statement 8
cout << "sale = " << sale << endl; //statement 9
first = 'D';                    //statement 10
cout << "first = " << first << endl; //statement 11
}
```

The output of this program is

```
num1 = 4
num2 = 9
sale = 20
first = D
```

Let us take a look at the output statement:

```
cout << " num1 = " << num1 << endl;
```

This output statement consists of the string " num1 = ", the operator <<, and the variable num1. Here, first the value of the string " num1 = " is output, and then the value of the variable **num1** is output. The meaning of the other output statements is similar.

The following table shows the values of the variables after the execution of each statement.

Number of statement	Values of the Variables	Explanation
After statment1	**?** **?** **num1 num2**	Allocate memory to num1 and num2
After statment2	**?** **?** **?** **num1 num2 sale**	Allocate memory to sale
After statement3	**?** **?** **?** **?** **num1 num2 sale first**	Allocate memory to first
After statement4	**4** **?** **?** **?** **num1 num2 sale first**	value is assigned to num1

After statement6	**4 9 ? ?** **num1 num2 sale first**	4*5 =20: 20-11=9 This value is assigned to num2
After statement8	**4 9 20 ?** **num1 num2 sale first**	0.02*1000=20 This value is assigned to sale
After statment10	**4 9 20 D** **num1 num2 sale first**	value is assigned to first

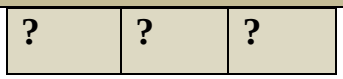
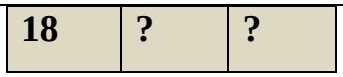
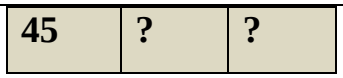
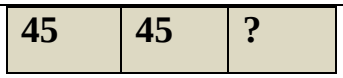
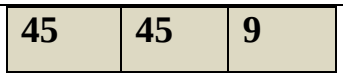
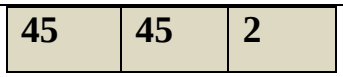
Example 14: a C++ program illustrates how data in the variables are manipulated.

```
#include <iostream>
void main()
{
int num1, num2,num3;                          //statement 1
num1 = 18;                                    //statement 2
num1 = num1 + 27;                             //statement 3
num2 = num1;                                  //statement 4
num3 = num2 / 5;                              //statement 5
num3 = num3 / 4;                              //statement 6
cout << "num1 = " << num1 <<" ,num2= "<<num2<<" ,nnum3= "<<num3;
}
```

The output of this program is

num1 = 45, num2 = 45, and num3 = 2.

The following table shows the values of the variables after the execution of each statement.

Number of statement	Values of the Variables	Explanation
After statment1		Allocate memory to num1,num2 and num3
After statement2		value is assigned to num1
After statement3		num1 + 27 = 18 + 27 = 45. This value is assigned to num1, which replaces the old value of num1.
After statement4		Copy the value of num1 into num2
After statement5		num2 / 5 = 45 / 5 = 9. This value is assigned to num3. So num3 = 9.
After statment6		num3 / 4 = 9 / 4 = 2. This value is assigned to num3, which replaces the old value of num3.

2- Input (Read) Statement:

Previously, you learned how to put data into variables using the assignment statement. In this section, you will learn how to put data into variables from the standard input device, using C++'s input (or read) statements.

When the computer gets the data from the keyboard, the user is said to be acting interactively. Putting data into variables from the standard input device is accomplished via the use of**cin** and the operator >>.

The syntax of **cin** together with >> is:

```
cin >> variable >> variable ...;
```

This is called an input (read) statement.

Example 15: A C++ program illustrates how input statements work

```
#include <iostream>
void main()
{
int feet;
int inches;
cout << "Enter two integers separated by spaces: ";
cin >> feet >> inches;
cout << endl;
cout << "Feet = " << feet << endl;
cout << "Inches = " << inches << endl;
}
```

The output of this program is (the user input is shaded.)

```
Enter two integers separated by spaces: 23 7
Feet = 23
Inches = 7
```

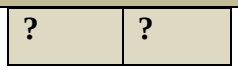
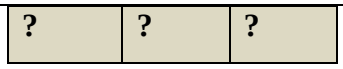
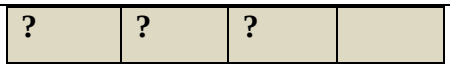
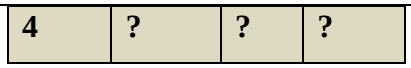
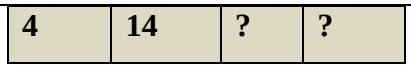
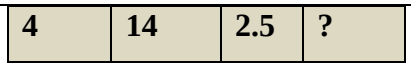
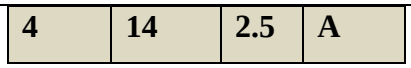
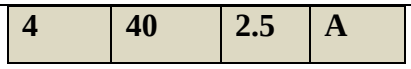
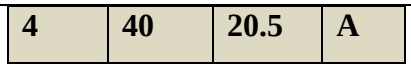
Example 16: A C++ program illustrates how assignment statements and input statements manipulate variables.

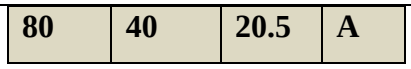
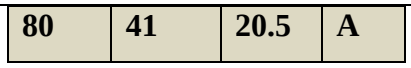
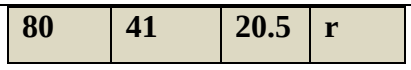
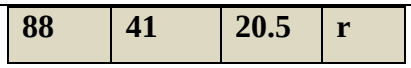
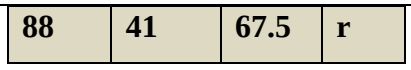
```
#include <iostream>
void main()
{
int fNum, sNum;       //statement 1
double z;             //statement2
char ch;              //statement3
fNum = 4;             //statement4
sNum = 2 *fNum + 6;   //statement5
z = (fNum + 1) / 2.0;  //statement6
ch = 'A';             //statement7
cin >> sNum;          //statement8
cin >> z;             //statement9
fNum = 2 *sNum;       //statement10
sNum = sNum + 1;      //statement11
cin >> ch;            //statement12
fNum = fNum + 8;      //statement13
 z = fNum - z;        //statement14
cout<<"the output of that program is : ";  //statement15
cout<< fNum<<" "<<sNum<<" "<<z<<"  "<<ch }
```

The output of this program is (the user input is shaded.)

```
40 20.5 r
the output of that program is : 88 41 67.5 r
```

The following table shows the values of the variables after the execution of each statement.

After st.	Values of the Variables	Explanation
St1		Allocate memory to fNum1 and sNum2
St2		Allocate memory to fNum1 , sNum2 and z
St3		Allocate memory to fNum1 , sNum2 , z and ch
St4		Store 4 into firstNum.
St5		2 * firstNum + 6 = 2 * 4+ 6 = 14. Store 14 into secondNum.
St6		(firstNum + 1) / 2.0 = (4 + 1) / 2.0 = 5 / 2.0 = 2.5. Store 2.5 into z.
St7		Store 'A' into ch.
St8		Read a number from the keyboard (which is 40) and store it into sNum. This statement replaces the old value of sNum with this new value.
St9		Read a number from the keyboard (which is 20.5) and store this number into z. This statement replaces the old value of z with this new value.

St10		2 * sNum =80 This statement replaces the old value of fNum with this new value
St11		sNum + 1 = 40 + 1 = 41. Store 9 into secondNum.
St12		Read the next input from the keyboard (which is r) and store it into ch. This statement replaces the old value of ch with the new value.
St13		fNum+1=80+1=88 This statement replaces the old value of ch with the new value.
St14		fNum-z=88-20.5=67.5 This statement replaces the old value of ch with the new value.

EXERCISE 4:

1- Resolve the problems in ” exercise 2” using C++ language, use read and write statements
2- Design an algorithm to takes distance between two cities in km and converts it in meter, cm, inch and feet.
3- Write a C++ program which takes distance between two cities in km and converts it in meter, cm, inch and feet

4- What is the output of the following program?

```
#include <iostream>
void main()
{
int fNum, sNum;       //statement 1
double z;             //statement2
char ch;              //statement3
fNum = 100;           //statement4
sNum = (200+fNum)/ 10;   //statement5
z = (fNum + 3) / 2.0;  //statement6
ch = 'b';
ch=ch+1;              //statement7
ch=ch+1;              //statement8
ch=ch-2;              //statement9
cin >> sNum;          //statement10
fNum = 10*sNum;       //statement11
sNum = fNum;      //statement12
cin >> ch;            //statement13
ch=ch+1;              //statement14
fNum = fNum + 8;      //statement15
 z = z*2.7;         //statement16
cout<<"the output of that program is : ";
cout<< fNum<<" "<<sNum<<" "<<z<<"  "<<ch }
```

Show the values of the variables after the execution of each statement.

1.11 Operators

When you write a program, you use variable names to create locations where you can store data. You can use assignment and input statements to provide values for the variables, and you can use output statements to display those values on the screen. In most programs that you write, you want to do more than input and output variable values. You also might want to evaluate those values and create

expressions that use the values. For example, you might want to perform arithmetic with values, or base decisions on values that users input.

In this section, you learn to use the C++ operators to create arithmetic expressions and study the results they produce. You also learn about the valuable shortcut arithmetic operators in C++. Then you concentrate on Boolean expressions you can use to control C++ decisions and loops—the topics of the next two chapters.

1.11.1 USING C++ BINARY ARITHMETIC OPERATORS

C++ provides five simple arithmetic operators for creating arithmetic expressions:

operator	description
+	**the addition operator**
-	**the subtraction operator**
*	**the multiplication operator**
/	**the division operator**
%	**the modulus operator**

Each of these symbols is an **arithmetic operator**—a symbol that performs arithmetic. Each is also a **binary operator**—an operator that ***takes two operands***, one on each side of the operator, as in 12 + 9 or 16.2 * 1.5.

The results of an arithmetic operation can be used immediately or stored in computer memory, in a variable.

Example17:

```
#include<iostream>
void main()
{
cout << 12 + 9 << endl; // displays the value 21
int sum = 12 + 9; // calculates sum whose value becomes 21
cout << sum << endl; // displays the value of sum
}
```

each cout statement in the program produces the value 21 as output, as you can see in Example17.

1- In the first statement within the main()function, the result, 21, is calculated within the cout statement and output immediately, without being stored.
2- In the second cout statement, the value of a variable is shown. The advantage to this approach is that the result of the addition calculation is stored in the variable named **sum**, and can be accessed again later within the same program, if necessary. For example, you might need sum again if its value is required as part of a subsequent calculation.

Example18:

```
#include<iostream>
void main()
{
int sum = 12 + 9; // calculates sum whose value becomes 21
int avg=sum/2;
cout << avg<< endl; // displays the value of sum
}
```

Addition, subtraction, multiplication, division, or modulus of any two integers results in an integer. For example, the expression 7 + 3 results in 10, 7 – 3 results in 4, and 7 * 3 results in 21. These results are the ones you might expect, but the result of integer division is less obvious. For example, the expression 7 / 3 results

in 2, not 2.333333. When two integers are divided, the result is an integer, so any fractional part of the result is lost.

If either or both of the operands in addition, subtraction, multiplication, or division is a **floating-point number** (that is, at least one operand is a float or a double), then the result is also a floating-point number. For example, the value of the expression:

3.2 * 2 is the floating-point value 6.4

because at least one of the operands is a floating-point number. Similarly, the result of:

3.2 / 2 is 1.6.

When at least one of the operands in division is a floating-point value, then the result is floating point, and the fractional part is not lost.

Examble19: C++ program shows some arithmetic examples and explains the computed results .

```
#include<iostream>
void main()
{
int a = 2, b = 4, c = 10, intResult;
double d = 2.0, e = 4.4, f = 12.8, doubleResult;
float g = 2.0f, h = 4.4f, i = 12.8f, floatResult;
intResult = a + b;
cout << intResult << endl;
// result is 6, an int
// because both operands are int
intResult = a * b;
cout << intResult << endl;
// result is 8, an int
// because both operands are int
```

```
intResult = c / a;
cout << intResult << endl;
// result is 5, an int
// because both operands are int
intResult = c / b;
cout << intResult << endl;
// result is 2 (losing the decimal fraction),
// an int, because both operands are int
floatResult = g / a;
cout << floatResult << endl;
// result is 1.0, a float,
// because the operands are int and float
floatResult = h / g;
cout << floatResult << endl;
// result is 2.2, a float,
// because both operands are floats
doubleResult = a * d;
cout << doubleResult << endl;
// result is 4.0, a double
// because the operands are int and double
// However, the result displays without the
// unneeded decimal point and fractional part
doubleResult = f / a;
cout << doubleResult << endl;
// result is 6.4, a double
// because the operands are int and double
doubleResult = e + h;
cout << doubleResult << endl;
// result is 8.8, a double,
// because operands are float and double
}
```

USING MODULUS

The modulus operator (%) gives the remainder of integer division; it can be used only with integers. The expression:

7 / 3 results in 2

The expression:

```
7 % 3 results in 1
17 % 5 is 2
```

You can pronounce 7 % 3 as "seven modulus three" or "seven mod three."

The modulus operator proves useful in a variety of situations. For example:

1-if you determine the remainder of dividing by 2, you can determine whether any value is even or odd. Any number with a remainder of 0 when divided by 2 is even, and any number with a remainder of 1 when divided by 2 is odd.

2-in time calculations, any integer number of minutes divided by 60 indicates the number of hours, and the result of modulus by 60 indicates the number of minutes left over. In the following code, the value of hours is 3 and the value of extra Minutes is 7:

```
int minutesOnTheJob = 187;
int hours = minutesOnTheJob / 60;
int extraMinutes = minutesOnTheJob % 60;
```

3- Similar calculations can be made to determine how many dozens (12), scores (20), or thousands (1000) are contained in a number, and how many remain

4-When using the decimal numbering system, if you need to know a number's last digit, you can perform modulus by 10. For example, 6543 % 10 is 3 and 6789 % 10 is 9. Similarly, you can extract the last two digits from a decimal number by using modulus 100.

PRECEDENCE AND ASSOCIATIVITY OF ARITHMETIC OPERATIONS

When more than one arithmetic operator is included in an expression, then multiplication, division, and modulus operations always occur before addition or subtraction. Multiplication, division, and modulus are said to have higher **arithmetic precedence**; that is, they are performed first in an arithmetic statement with multiple operations. Addition and subtraction operations have lower precedence. When two operations with the same precedence appear in an arithmetic expression, the operations are carried out in order from either left to right or right to left based on their **associativity**—the rule that dictates the order in which an operator works with its operands. The associativity of arithmetic operations (if there are no parentheses in the expression) is from left to right.

When C++ evaluates a mixed arithmetic expression, the following steps occur:

1. The leftmost operation with the highest precedence (*, /, and %) is evaluated. If the data types of the operands on both sides of the operator are the same data type, the result is the same data type. If the operands are different data types, then C++ performs an implicit cast and the result is the same type as the one that occupies more memory.
2. Each subsequent *, /, or % operation is evaluated in the same manner from left to right.
3. The leftmost operation with the lower precedence (+ and –) is evaluated. If the data types of the operands on both sides of the operator are the same, the result is the same data type. If the operands are different data types, then C++ performs an implicit cast and the result is the same type as the one that occupies more memory.

4. Each subsequent + or – operation is evaluated in the same manner from left to right.

Expression	Value	Comments
2 + 3 * 4.0	14.0	3 * 4.0 results in 12.0; then 2 + 12.0 results in 14.0
3 / 4 + 2.2	2.2	The result of integer division 3 / 4 is 0 (not 0.75); then 0 + 2.2 produces 2.2
3.0 / 4 + 2.2	2.95	The result of floating-point division 3.0 / 4 is 0.75; then 0.75 + 2.2 produces 2.95
3.0 * 1.1 / 2	1.65	The multiplication result is 3.3; then the division result is 1.65
3 / 2 * 5 / 2	2	The first integer division, 3 /2, results in 1 (not 1.5); then 1 * 5 produces 5 and 5 / 2 results in 2 (not 2.5)

1.11.2 SHORTCUT ARITHMETIC OPERATORS

In addition to the standard binary arithmetic operators for addition, subtraction, multiplication, division, and modulus, C++ employs several shortcut operators. The two categories of shortcut arithmetic operators are compound assignment operators and increment and decrement operators.

1-Compound assignment (+=, -=, *=, /=, %=)

Compound assignment operators modify the current value of a variable by performing an operation on it. They are equivalent to assigning the result of an operation to the first operand:

expression	Equivalent to...
y += x;	y = y + x;
x -= 5;	x = x - 5;
x /= y;	x = x / y;
price *= units + 1;	price = price * (units+1);

2-Increment and decrement (++, --)

The increment and decrement operators. These operators are used frequently by C++ programmers and are useful programming tools.

Some expression can be shortened even more: the increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively.

Suppose **count** is an **int** variable. The statement:

```
count = count + 1;
```

increments the value of count by 1. To execute this assignment statement, the computer first evaluates the expression on the right, which is count + 1. It then assigns this value to the variable on the left, which is **count**.

As you will see later, such statements are frequently used to keep track of how many times certain things have happened. To expedite the execution of such statements, C++ provides the ***increment operator***, ++, which increases the value of a variable by 1, and the decrement operator, – –, which decreases the value of a variable by 1

Increment and decrement operators each have two forms, pre and post. The syntax of the increment operator is:

Pre-increment: ++variable

Post-increment: variable++

The syntax of the decrement operator is:

Pre-decrement: —variable

Post-decrement: variable—

Let's look at some examples. The statement:

```
++count;
or:
count++;
```

increments the value of count by 1. Similarly, the statement:

```
—count;
or:
count—;
```

decrements the value of count by 1.

Because both the increment and decrement operators are built into C++, the value of the variable is quickly incremented or decremented without having to use the form of an assignment statement.

Now, both the pre- and post-increment operators increment the value of the variable by 1. Similarly, the pre- and post-decrement operators decrement the value of the variable by 1. What is the difference between the pre and post forms of these operators? The difference becomes apparent when the variable using these operators is employed in an expression.

Suppose that x is **an int variable**. If **++x** is used in an expression, first the value of x is incremented by 1, and then the new value of x is used to evaluate the expression. On the other hand, if **x++** is used in an expression, first the current value of x is used in the expression, and then the value of x is incremented by 1. The following example clarifies the difference between the pre- and post-increment operators.

Notice the difference:

Example 1	**Example 2**

x = 3;
y = ++x;
// x contains 4, y contains 4
	x = 3;
	y = x++;
	// x contains 4, y contains 3

In Example 1, The first statement assigns the value 3 to x. To evaluate the second statement, which uses the pre-increment operator, first the value of x is incremented to 4, and then this value, 4, is assigned to y. After the second statement executes, both x and y have the value 4.

In Example 2, the first statement assigns 3 to x. In the second statement, the post-increment operator is applied to x. To execute the second statement, first the value of x, which is 3, is used to evaluate the expression, and then the value of x is incremented to 4. Finally, the value of the expression, which is 3, is stored in y. After the second statement executes, the value of x is 4, and the value of y is 3.

Example 20 : C++compound assignment operators

```
#include <iostream.h>
void main()
{
   int num
   cout << "Enter a number: ";
   cin >> num;
   cout << "\n num +=1 :" << num +=1  << "\n";
   cout << "\n num -=2: " << num -=2 << "\n";
   cout << "\n num *=1+3: " << num *=1+3 << "\n";
   cout << "\n num /=2 :" << num /=2 << "\n";
   cout << " num %=4 :" << num %=4 << "\n";
 }
```

If num=4 then the output of this program is:

```
Enter a number:4
num +=1 :5
num -=2 :3
num *=1+3: 6
num /=2 :3
num %=4 :0
```

1.11.3-Relational and comparison operators (==, !=, >, <, >=, <=)

C++ employs the six binary relational operators listed in Table 2-2. **Relational operators** are those that evaluate the relationship between operands. You use these relational operators to evaluate Boolean expressions. A **Boolean expression** is one that is interpreted to be true or false. The relational operators in C++are:

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

1- All false relational expressions are evaluated as 0. Thus, an expression such as: 2 > 9 has the value 0. You can prove that 2 > 9 is evaluated as 0 by entering the statement cout << (2 > 9); into a C++ program. A 0 appears as output.

2- All true relational expressions are evaluated as 1. Thus, the expression 9 > 2 has the value 1. You can prove this by entering the statement cout << (9 > 2); into a C++ program. A 1 appears as output.

3-The unary operator ! is the not operator; it means "the opposite of," and essentially reverses the true/false value of an expression. For example:

cout << (9 2); displays a 1 because "9 is greater than 2" is true. In contrast,

cout << !(9 > 2); displays a 0 because the value of !1 ("not one") is 0.

4- The relational operator = = deserves special attention. Suppose two variables, q and r, have been declared, and q = 7 and r = 8. The statement cout << (q == r); produces 0 (false) because the value of q is not equivalent to the value of r. The statement cout << (q = r);, however, produces 8. The single equal sign does not compare two variables; instead, it assigns the value of the rightmost variable to the variable on the left. Because r is 8, q becomes 8, and the value of the entire expression is 8.

examples:

```
(7 == 5)     // evaluates to false
(5 > 4)      // evaluates to true
(3 != 2)     // evaluates to true
(6 >= 6)     // evaluates to true
(5 < 5)      // evaluates to false
```

Suppose that **a=2**, **b=3** and **c=6**, then:

```
(a == 5)     // evaluates to false, since a is not equal to 5
(a*b >= c)   // evaluates to true, since (2*3 >= 6) is true
(b+4 > a*c)  // evaluates to false, since (3+4 > 2*6) is false
```

Exercise 5:

Exercise 5:

1- 1. Arithmetic operations, such as addition (+), subtraction (–), multiplication (*), division (/), and modulus (%) that take two arguments use _____ operators.

a. unary c. binary

b. summary d. Boolean

2. If either or both of the operands in addition, subtraction, multiplication, or division is a floating-point number, that is, a float or a double, then the result is _____ .

a. always an integer b. usually a floating-point number

c. always a floating-point number d. not mathematically defined

3- If c and d are integers, and d = 34, what is the value of c = d++?

a. 0 c. 34

b. 1 d. 35

4. Multiplication, division, and modulus are said to have _____ than addition and subtraction.

a. lower arithmetic precedence b. higher arithmetic precedence

c. lower associativity d. higher associativity

5. 1. Assume a, b, and c are integers, and that a = 0, b = 1, and c = 5. What is the value of each of the following? (The answers are not cumulative; evaluate each expression using the original values for a, b, and c.)

a. a + b b. a > b

c. 3 + b * c d. ++b

e. b++ f. b <= c

g. a > 5 h. ++a ==b

i. b != c j. b == c

k. b = c l. b / c

m. b % c n. b + c * 4 / 3 o. 22 / (c + 3)

6- Write a C++ program in which you declare a variable that holds an hourly wage. Prompt the user to enter an hourly wage. Multiply the wage by 40 hours and print the standard weekly pay.

7- Write a C++ program in which you declare variables that will hold an hourly wage, a number of hours worked, and a withholding percentage. Prompt the user to enter values for each of these fields. Compute and display net weekly pay, which is calculated as hours times rate, minus the percentage of the gross pay that is withholding.

8- Write a program that allows the user to enter two values. Display the results of adding the two values, subtracting them from each other, multiplying them, and dividing them.

9- Write a program for a bank that allows the user to enter an amount of money in cents. Display the number of whole dollars the bank will give the customer in exchange.

1.12 MAKING DECISIONS

Computer programs often seem smart because of their ability to use selections or make decisions. Consider a medical program that diagnoses your ailment based on a group of symptoms. Some programs make hundreds or thousands of decisions, but no matter how many decisions a program requires, each one is simply a yes-or-no decision. Computer circuitry consists of millions of tiny electronic switches,

and the state of each is on or off. This means that every decision boils down to on or off, yes or no, 1 or 0.

C++ lets us perform selections in a number of ways including using the **if** and **if-else** statements, the **switch** statement, and conditional operators. You also can combine decisions using the logical **AND** and **OR**.

1.12.1 The Single- if

The primary C++ selection structure statement used to perform a single-alternative selection is an if statement. One way to use an if is in a single-alternative selection—one in which an action takes place only when the result of the decision is true. It takes the form:

```
if (Boolean expression)
   action if true;
```

When you write an **if statement**, you use the keyword "**if "**, a Boolean expression within parentheses, and any statement that is the action that occurs if the Boolean expression is true. Figure 1-3 shows a diagram of the logic of the **if statement**.

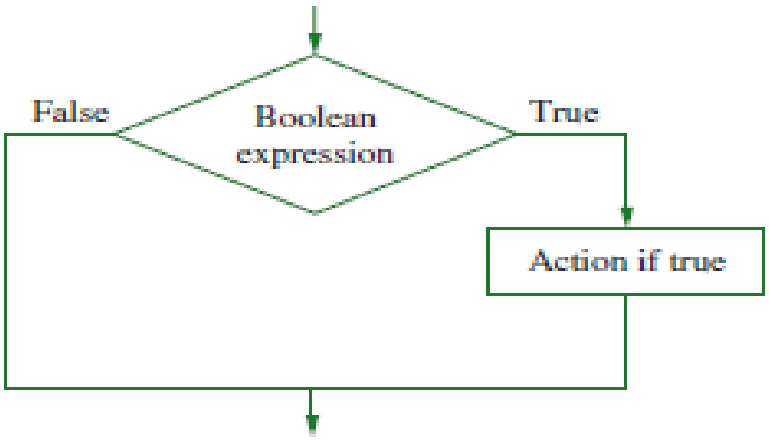


Figure1-3 Flowchart diagram of an `if` statement

Example 21: Algorithm to display a number if that number is negative

Input: number.

Output: display negative number only

Step1: start

Step2: Get the Value of A.

Step3: Find the negative number using the following formula:
if (A is smaller than 0)
Display A

Step4: end

Example 22: Algorithm to test if the student passes the exam

Input: Degree of the exam.

Output: display if the student is pass in exam

Step1: start

Step2: Get the Value of Degree.

Step3: Test the Degree:
if (Degree is greater or equal than 50)
The student is passing

Step4: end

Example 23:C++ program to display only even number.

```
#include <iostream.h>
void main()
{    int number;
   cout << "Enter an integer: ";
   cin >> number;
   if ( number %2 =0)
        cout << endl<<"You entered an even number: " << number << endl;
   cout << "This statement is always executed.";    }
```

If number =4 then the output of this program is:

```
Enter an integer: 4
You entered an even number: 4
This statement is always executed.
```

If number =3 then the output of this program is:

```
Enter an integer: 3
This statement is always executed.
```

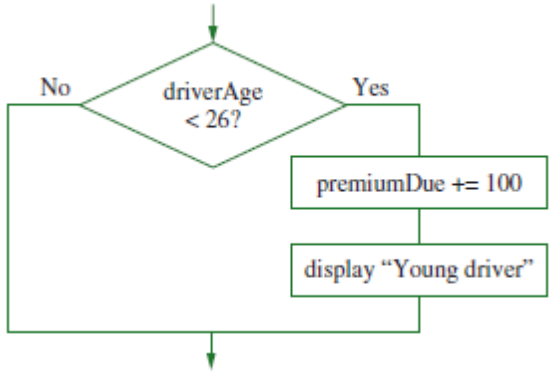
Example 24: Consider the program shown in below. An insurance policy base premium is set as $75.32. After the program prompts for and receives values for the driver's age and number of traffic tickets, several (shaded) decisions are made.

```
#include<iostream>
using namespace std;
void main()
{
int driverAge, numTickets;
double premiumDue = 75.32;
cout << "Enter driver's age ";
cin >> driverAge;
cout << "Enter traffic tickets issued ";
cin >> numTickets;
if(driverAge < 26)
    premiumDue += 100;
if(driverAge > 50)
    premiumDue -= 50;
if(numTickets == 2)
    premiumDue += 60.25;
cout << "Premium due is " << premiumDue << endl;
}
```

if the expression in the parentheses of any of the if statements is **true**, then the statement following the if executes. For example, if the **driverAge** is less than 26, then 100 is added to the **premiumDue**.

Sometimes you want to perform multiple tasks when a condition is met. For example, suppose that when a driver is under 26, you want to add $100 to the premium, but also display a message. If the execution of more than one statement depends on the selection, then the resulting statements must be placed in a block with curly braces as shown in below

```
if(driverAge < 26)
{
premiumDue += 100;
cout << "Young driver";
}
```



1.12.2The if-else structure

"If" takes one action when its ***Boolean expression*** is evaluated as true, and uses an **else** clause to define the actions to take when the expression is evaluated as false. It takes the form:

```
if (Boolean expression)
    action if true;
else
   action if false
```

Example 25: Program to check whether an integer is positive or negative

```
#include <iostream>
void main()
{
   int number;
```

```
cout << "Enter an integer: ";
cin >> number;

if ( number >= 0)
      cout << endl<<"You entered a positive integer: " << number << endl;
else
   cout << "You entered a negative integer: " << number << endl;
cout << "This line is always printed.";
}
```

If number =-4 then the output of this program is:

```
Enter an integer: -4
You entered a positive integer: -4
This line is always printed.
```

If number =7 then the output of this program is:

```
Enter an integer: 7
You entered a negative integer: 7
This line is always printed.
```

Example 26: program uses an if-else structure. In the program, after the user enters a character in the **genderCode** variable, the variable is tested to see if it is equivalent to the character '**F**'. If it is, the output is the word "**Female"**, if it is not, the output is "**Male**".

```
#include<iostream.h>
void main()
{
char genderCode;
cout << "Enter F for female or M for male "<<endl;
cin >> genderCode;
if(genderCode == 'F')
    cout<< endl <<"Female"  ;
else
   cout << endl << "Male" << endl;                    }
```

If genderCode =F then the output of this program is:

```
Enter F for female or M for male
F
Female
```

If genderCode =s then the output of this program is:

```
Enter F for female or M for male
s
Male
```

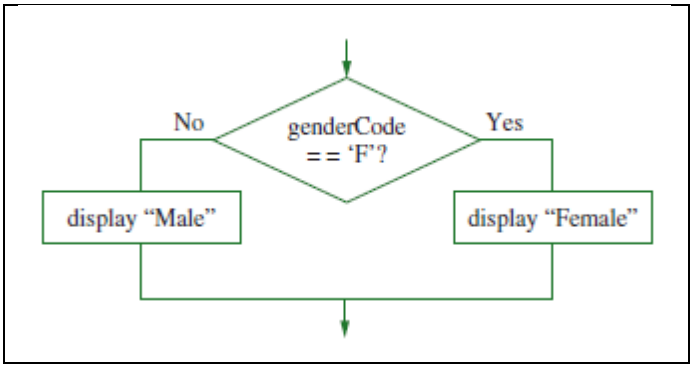


Diagram of logic of (if-else) statement in Example 26

Exercise 6:

1-What is the output after executing the following segment of code?

```
int num = 10;
if (num > 10)
cout << "Yes" << endl;
else
cout << "No" << endl;
```

a. Yes b. No c. Yes *and* No d. nothing

2-What is the output after executing the following segment of code?

```
int num = 5;
if (num > 10)
cout << "Yes" << endl;
cout << "No" << endl;
```

a. Yes b. No c. Yes *and* No d. nothing

3- Design an algorithm to find the largest number between two numbers

4-write a C++ program that allows the user to enter two double values. Display one of two messages: “The first number you entered is larger”, or “The first number you entered is not larger”.

5. Write a program that allows the user to enter two double values. Display one of three messages: “The first number you entered is larger”, “The second number you entered is larger”, or “The numbers are equal”.

6-W rite a program that allows the user to enter two numeric values. Then let the user enter a single character as the desired operation: ‘a’ for add, ‘s’ for subtract, ‘m’ for multiply, or ‘d’ for divide. Perform the arithmetic operation that the user selects and display the results.

7-write a program for a college admissions office. The user enters a numeric high school grade point average (for example, 3.2), and an admission test score. Print the message “Accept” if the student meets either of the following requirements:
» A grade point average of 3.0 or above and an admission test score of at least 60
» A grade point average below 3.0 and an admission test score of at least 80
If the student does not meet either of the qualification criteria, print “Reject”.

8-write a program that asks the user to type a vowel from the keyboard. If the character entered is a vowel, display “OK”; if it is not a vowel, display an error message. Be sure to allow both uppercase and lowercase vowels.