بریلینٹ کالج آف آئی ٹی

Page

COMPUTER PROGRAMMING IN C/C++

Brilliant

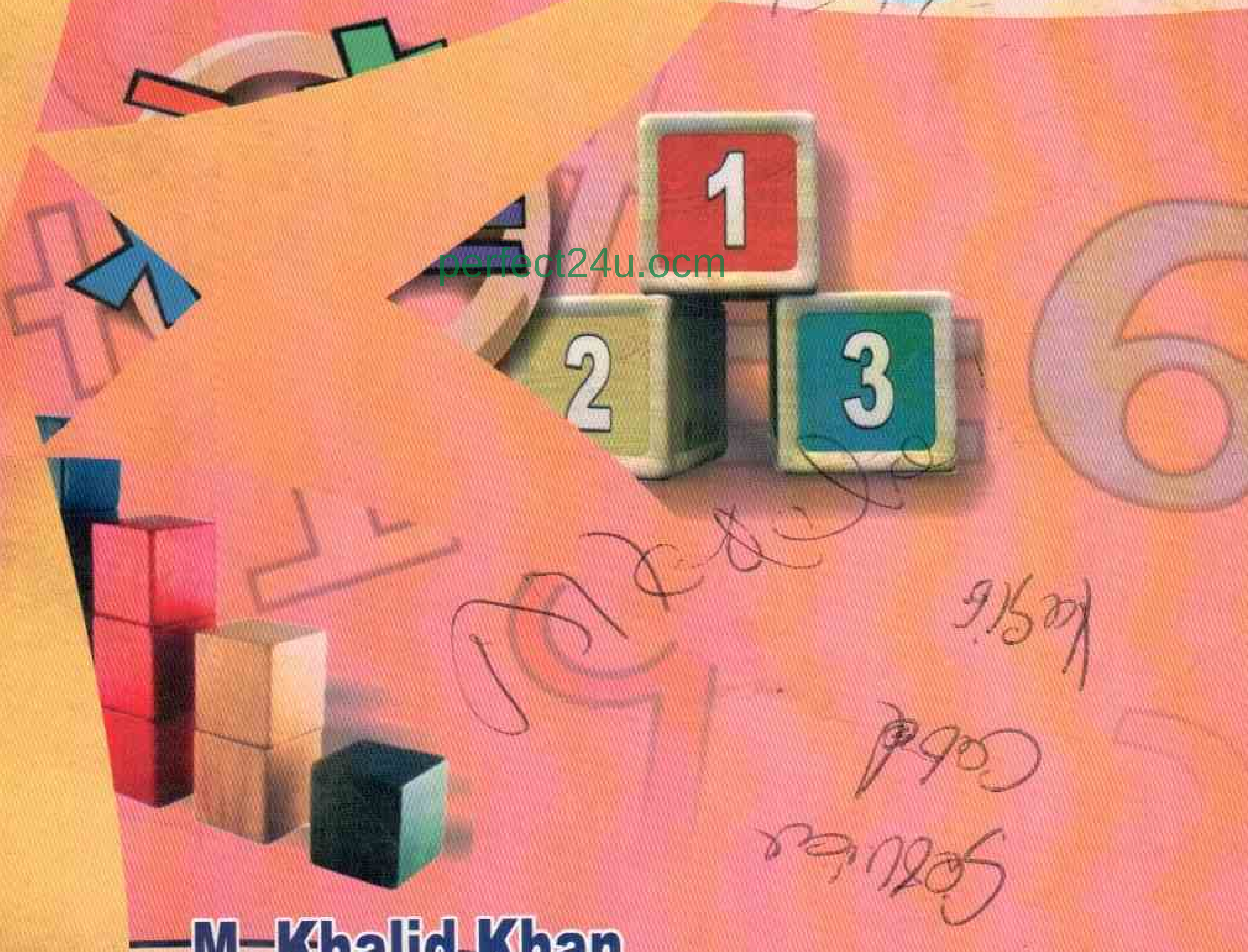
BTE

NEW COURSE OF BTE

DIT-I

C/C++

DIT



[illegible]

M. Khalid Khan

M.Sc. Computer Science (Gold Medalist)

Recommended By: BTE / TTB, KPK Peshawar

COMPUTER PROGRAMMING IN C/C+

According to New Syllabus of DIT
Khyber Pakhtunkhwa Board of Technical Education
Khyber Pakhtunkhwa Trade Testing Board

3rd Edition May, 2016

By
M. Khalid Khan
M.Sc Computer Science (Gold Medalist)
Associate Professor in Computer Science
Govt: College Of Management Sciences, Peshawar.

Can be had from

Nasir Book Depot Qissa Khawani Bazaar Peshawar Phone No: 091-2580145
Marwat Stationer Mandian Chowk Abbotabad Ph-No: 0992-383004
University Book Agency Manin Chowk Swabi
City Books Agency PRC Market Mardan.0937-867042
Nasir Book Agency Iqbal Plaza Near City Center Dhaki Naghalbandi Qissa Khuwani Bazar Ph # 091257227
Kakar Brothers Book Seller and whole Sale Dealer Insaf Market Dhanki Naighal Bandi Ph # 091-2591816

ACKNOWLEDGMENTS

The author wishes to acknowledge the assistance of the following reviewers whose valuable suggestions helped in shaping the form and content of this text.

Mr. Hazrat Zubair, Assistant Professor in Computer Science, GCMS, Peshawar

Mr. Fazal Hussain, Assistant Professor in Computer Science, GCMS, Abbotabad.

Mr. Waqar, Lecturer in Computer Science GCMS, Charsadda

Fazli Akbar, Lecturer in Computer Science GCMS, Mardan

Mr. Naveed Iqbal, Lecturer in Computer Science GCMS, Peshawar

Mr. Riaz, Lecturer in Computer Science GCMS, Sangota Swat

Mr. Riaz Assistant Professor in Computer Science GCMS, Mardan

Mr. Rahim Shah Lecturer in Computer Science GCMS, Peshawar

Mr. Maskeen Shah, Lecturer in Computer Science GCMS, Haripure

Mr. Wasim, Lecturer in Computer Science GCMS, Haripure.

Mr. Latif Ullah Lecturer GCMS Ring Road Peshawar

Mr. Mujjeb ur Rahman Lecturer in Computer Science GCMS, Peshawar

Mr. Faisal Rahim,, Lecturer in Computer Sciences, Govt College of Management Sciences, Peshawar

Mr. Rahat Ullah Lecturer in Computer Science GCMS Mardan 2

Nasar Jamal, Lecturer in Computer Science GCMS, Swabi

I am grateful to all, for their valuable prefatory remarks they expressed about this book.

M. Khalid Khan

COURSE CONTENTS

2- Assembly Language (second generation language):

Assembler Language is the next level of Programming Languages. It was developed to overcome some of the problems of machine language. This is another low-level language in which symbols are used instead of binary code these symbols are called mnemonics codes. For example MUL instruction is used to subtract two numbers. The assembler converts assembly language instruction into machine code instruction. It is called middle level language or it is called second-generation language.

3- High-Level Language (Third Generation language)

A type of language that is close to human language is called high-level language. High Level Language gives formats to English Language and use common mathematical notation. Program in High Level Language are much easier than Low-level language because the program instruction are similar to those in every day English and they contain commonly used mathematical notation. High level Language tells the computer not only what to do but how to do it. The programmer spends less time developing software with a high level language. High-level Language is also called compiler language or problem oriented language or procedural language.

High-level languages use instructions, which are called statements that use brief statements or arithmetic expressions. Translator programs called compilers or interpreters convert high-level language instruction into machine code.

The following are High Level Language. 1) Basic 2) C-Language 3) Java Language 4) Cobol Language etc.

4-Fourth Generation Language (Non-procedural language):

The fourth-generation language is also known as very high-level language. In non-procedural language, user only needs to tell the computer "what to do" "how to do". They are nonprocedural languages and conversational than prior languages. An important advantage of non-procedural language is that they can be used by non-technical user to perform a specific task. These languages accelerate program process and reduce the coding error. 4GL are normally used database application and report generation. Natural language is 4GLs that are very close to English or other human language. 4GLs languages are less efficient in form of processing speed and amount of storage capacity needed. Fourth generation languages need approximately one tenth the number of statement that a high level language needs to achieve the same result because they are so much easier to use than third generation language. For example 1-Query languages 2- Report generator 3- Application generator 4- decision support system. Some example are Word, Excel, Access, Oracle etc come under these categories.

5-Object-Oriented Language

OOP is a technique in which programs are written on the basis of objects. An object is a collection of data and functions. Object may represent a person, thing or place in real world.
In OOP, data and all possible functions on data are grouped together. Object oriented program are easier to learn and modify. C++ and Java are two most popular Object-oriented languages.

Computer Languages and Packages:- In computer languages we define the problems and give our own instructions to the computer, e.g. BASIC, FORTRAN, PASCAL,C/C++ etc. But in computer packages we can solve our problems without giving our instructions to the computer. They need only a few keystrokes and ready-made commands to help in solving our specific problems. Computer languages are versatile to solve the problems where as the computer packages are not versatile and are specific to problem solutions.

HISTORY/INTRODUCTION TO C++:

What *C Programming Language (C)* mean?

C is a high-level and general purpose programming language that is ideal for developing firmware or portable applications. Originally intended for writing system software, C was developed at Bell Labs by **Dennis Ritchie** for the Unix Operating System (OS) in the early 1970s.
Ranked among the most widely used languages, C has a compiler for most computer systems and influenced many popular languages - notably C++.

BCIT

BciT
Maadan
Pakistan

vs India
vs Indis

```
cout<<"The Area is :::"<<area;
getch();
}
```

// This program is to demonstrates some simple C++ Constants and Variables and Comments declarations.

```
#include <iostream.h>
void main()
        {
        char    x;          // These lines are used to declare the
                            //given five different variables.
        int     k;
        int     i, j;
        float   z;
        float   y;
        x =  'Q';           // In this a character constant is assign to a
                            //character variable.
        i  = j;            // In this the value of one variable i.e. j will
                            //be assign to variable i.
        k = j + 9;          //Similarly, the vale of j will be added //to 9
                            and the resultant sum will be //then stored in
                            variable k.
        z = 3.355;          // In this statement a real constant will
                            //be assign to a real variable.
        y = z + 9.87;
        cout <<endl;
        cout<<x<<endl<<k<<endl<<y<<endl;
        return 0;           // Each function should be terminated
                            //by this statement.
        }
```

The output of this program is to display the values of variable x , k , and y on the screen.

- All the names and other words should be written in lowercase letters because uppercase letters have some special meaning for screen, printer and disk files.
- The part of the C++ program that begins with main() and followed by an opening brace is called the main function. A C++

program consists of one or more than one different functions. The function main() is a required function and is always the first function executed.

- The above program contains only a single function i.e main() and hence the entire function is a single block because there is only one pair of braces.
- Each statement is terminated by a semicolon that shows the end of each statement.
- The lines with main() and braces do not have semicolon because these lines simply define the beginning and ending of the function and do not actually executed by the computer.
- The readability of a program can be improved by putting comments throughout the program listings. A comment is a message that explains what is going on at that point in the program. The compiler ignores comments , so they do not add to the file size at execution time of the executable program. Comments start with a double slash symbol (//) and terminate at the end of a line. A comment can start at the beginning of the line or on the same line following a program statement. Both possibilities are shown in the above example.

ADVANTAGES OF C LANGUAGE

1. C language is a building block for many other currently known languages.
   C language has variety of data types and powerful operators. Due to this, programs written in C language are efficient, fast and easy to understand.
2. C is highly portable language. This means that C programs written for one computer can easily run on another computer without any change or by doing a little change.
3. There are only 32 keywords in ANSI C and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs.
4. Another important advantage of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library this makes us easier to add our own functions to C li-

PREPROCESSOR DIRECTIVE: It is also known as compiler directive. It is an instruction to the compiler before the execution of the program.

The **C Preprocessor (CPP)** is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do require pre-processing before the actual compilation. We'll refer to the C Preprocessor as CPP.

It starts with a symbol called hash #.The preprocessor directive are written at the start of the program.

Some of the preprocessor directive is the following.

1:# Include preprocessor: this preprocessor directive is used to include the different header files in the program before the execution of the program.

Syntax:

```
#include<iostream.h>
#include <stdio.h>
#include "myheader.h"
```

2:#define preprocessor: This preprocessor directive is used to define a constant and a macro.

Syntax: #define pi=3.14;

C/C++ RESERVED WORDS/ KEYWORDS

In C, we have 32 ++ reserved words /keywords, which have their predefined meaning and cannot be used as a variable name.. It is good practice to avoid using these keywords as variable name. These are auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

USER DEFINED WORDS / IDENTIFIERS

User defined words / Identifiers are names for entities in a C program, such as variables, arrays, functions, structures, unions and labels. An identifier can be composed only of uppercase, lowercase letters, underscore and digits, but should start only with an alphabet or an underscore.

An identifier is a string of alphanumeric characters that begins with an alphabetic character or an underscore character that are used to represent various programming elements such as variables, functions, arrays, structures, unions and so on. Actually, an identifier is a user-defined word. There are 53 characters, to represent identifiers. They are 52 alphabetic characters (i.e., both uppercase and lowercase alphabets) and the underscore character. The underscore character is considered as a letter in identifiers. The underscore character is usually used in the middle of an identifier. There are 63 alphanumeric characters, i.e., 53 alphabetic characters and 10 digits (i.e., 0-9).

Rules for constructing identifiers

1. The first character in an identifier must be an alphabet or an underscore and can be followed only by any number alpha bets, or digits or underscores.
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct. That is, identi fiers are case sensitive.
4. Commas or blank spaces are not allowed within an identi fier.
5. Keywords cannot be used as an identifier.
6. Identifiers should not be of length more than 31 characters.
7. Identifiers must be meaningful, short, quickly and easily typed and easily read.

Valid identifiers: total sum average _x y_
mark_1 x1

Invalid identifiers

1x	-	begins with a digit
char	-	reserved word
x+y	-	special character

Note: Underscore character is usually used as a link between two words in long identifiers.

Differentiate between Keywords words and identifiers

Keyword	**Identifier**
Predefined-word	User-defined word
Must be written in lowercase only	Can written in any lowercase and uppercase
Has fixed meaning	Must be meaningful in the program
Whose meaning has already been explained to the C compiler	Whose meaning not explained to the C compiler
Combination of alphabetic character	Combination of alphanumeric characters
Used only for it intended purpose	Used for required purpose
Underscore character is not considered as a letter	Underscore character is considered as a letter

Difference between variable and identifier

A variable having the data type and name, an identifier is the name of the variable

for example

int x; //int x is the variable and x is the identifier

VAIABLE AND THEIR DIFFERENT DATA TYPES

VARIABLE:-Variables are the most fundamental part of any language. A variable is a symbolic name that can be given a variety of values. Variables are stored in particular places in the computer's memory. When a variable is given a value, that value is actually placed in the memory space occupied by the variable. Most popular languages use the same general variable types, such as integers, floating-point numbers and characters.

In variable name we can use upper case and lower-case letters, and the digits from 0-9. We can also use the underscore. The first character must be a letter or underscore. Variable names can be as long as we need , but only the first 32 characters will be recognized. Compiler distinguishes between upper case and lowercase letters , so **var** is not the same as **Var** or **VAR**.We cannot use a C++ keyword as a variable name. A keyword is the identifier for a language feature, like

int, class , if while, and so on. To declare a variable, we must understand its characteristics/properties, i.e

- Each variable has a name.
- Each variable has a type.
- A variable stored a value in memory, which is assign to it.

Rules for constructing variable names:

a) A variable name is combination of 1...8 alphabets, digits, or underscore. Some compiler allow variable name up to 40 characters.

b) The first character in the variable name must be an alphabet.

c) No comma or blank space is allowed with in a variable name.

d) No special symbol other the underscore can be use in variable name.

Valid variable names	Invalid variable names
i) a	i) 2s
ii) a123	ii) q-4
iii) z_1_2	iii) cout
iv) *a*_ b	iv) a+b

we can declare a variable in two places:

- After the opening brace of a block of code (usually at the top of a function) and these variables are called local variable.
- Before a function name, such as before main(). In this case the variables are called global variables.

A variable can hold different types of data. Due to this we use different types of variable names. For example integer, real, char etc.

DIFFERENT DATA TYPES OF VAIABLE

1-Defining Integer Variables:

Integer variables are used to stored numeric data as integer constants and exist in different sizes, but the most commonly used is type **int.** This types requires two bytes of storage and holds whole numbers in the range -32,768 to 32,767. Consider the following program .

// This program demonstrates integer variables.

```
#include <iostream.h>
void main()
        {
        int     a1, a2;
        int     b1;
        a1= 20;
        a2=a1 + 10;
        cout << "a1 +10 is = ";
        cout << a2;      }
```

2-Defining Character Variables:

Character variables are used to stored character data as character constants and exist in different sizes, but the most commonly used is the type **char.** This types requires one byte of storage and holds only a single character . This type of integer variable stores integer data that range in value from –128 to 127. Character variables are sometimes used to store numbers that confine themselves to this limited range., but they are most commonly used to store ASCII characters.

As we know that the ACSII character set is way of representing characters such as 'a', 'B', '$', '3', and so on, as numbers. These numbers range from 0 to 127 . Some computers extend this range to 255 to accommodate various foreign-language and graphics characters. Consider the following program.

// This program demonstrates character variables.

```
#include <iostream.h>
void main()
        {
        char    c1 , c2;
        char    p1;
        c1 = 'Q';
        c2 = '$';
        p1 = '+';
        cout << c1<<endl <<c2;
        cout << p1<<endl;
        }
```

3-Defining floating-point Variables:

Floating-point variables are used to stored numeric data as real constants and exist in different sizes , but the most commonly used is the type **float .** This types requires two bytes of storage and holds a floating-point number in the range -3.4E+38 to 3.4E+38 .

We can also write floating-point constants using exponential notation. For example, the number 1234.56 would be written 1.23456E3 in exponential notation. This is the same as 1.23456 times 10^3 . The number following the E is called the exponent. It indicates how many places the decimal point must be moved to change the number to ordinary decimal notation.

The exponent can be positive or negative. The exponential number 6.35239E-5 is equivalent to 0.0000035239 in decimal notation. Consider the following program.

```
// This program demonstrates Floating-point  variables.
#include  <iostream.h>
void main( )
        {
        float     z1 ,  z2;
        float     z3;
        z1 =  66.89;
        z2 =   9;
        z3 =   0.5 ;
        cout << z2<<endl <<z2;
        cout << z3<<endl;
        }
```

Identifier:

Various data items with symbolic names in C++ is called as Identifiers. Following data items are called as Identifier in C++ .

1. Names of functions
2. Names of arrays
3. Names of variables
4. Names of classes

The rules of naming identifiers in C++ :

1. C++ is **case-sensitive** so that Uppercase Letters and Lower Case letters are different.
2. The name of identifier **cannot begin with a digit**. However, Under-score can be used as first character while declaring the identifier.
3. Only **alphabetic characters, digits and underscore** (_) are permitted in C++ language for declaring identifier.
4. Other **special characters** are not allowed for naming a variable / identifier
5. **Keywords** cannot be used as Identifier.

Valid Identifier	**Invalid Identifier**
1) SNO	1) 3A
2) S_NO	2) Net-pay
3) N_I_C	3) 'X'
4) ADD	4) ITEM NUMBER
5) A32_78	5) switch

CONSTANTS

A constant is a fixed value or number , which do not change their values during the program execution, and retains its value throughout the program. Different types of variables are used to stored different type of Constants. They are mainly divided into the following two types.

DIFFERENT DATA TYPES OF CONSTANT:

1. Numeric Constants:

They are numeric values or numbers. It is again divided into the following two types.

a. Integer constants:

These constants are whole number i.e having no fractional part. For example 34, 78, 9, etc. Integer variables are used to stored integer constants. Integer constants may be stored either as **int** or **long int**. The range for int is –32768 to 32767 and it requires two bytes of memory. To stored a constant as long int , the range is –2147483648 to 2147483647 and it requires four bytes of memory.

- **Special integer constant:-** It is of two types.

  i. **Octal constants:-** These constants have base **8** and uses symbols **0,1,2,3,4,5,6,7.** To represent a number as octal constant we put a leading **0** before the number.
  For example, **023** is not an integer constant but due to leading zero computer interpret it as octal constant.

  ii. **Hexadecimal constants:-** These constants have base **16** and uses symbols **0,1,2...9,A,B,C,D,E,F.** To represent a number as hexadecimal integer constant , we apend the **0x** to it . For example , **0x23, 0x2b3, 0xFFF , 0x9.**

b. **Floating-point Constants:-** These constants have fractional part. For example 34.6, 77.0, 33. Floating-point variables are used to store real constants. These constants are stored as **float, double, and long double.** The rang, type and the memory space for the above constants are given as below.

Type	Range	No. of bytes
Float	3.4E+38 TO 3.4E+38	Four bytes
Double	-1.7E+308 to 1.7E+308	Eight bytes
Long double	-3.4E+4932 to 3.4E+4932	Ten bytes

Rules for constructing float constants:

a) A float constant must have at least one digit
b) it must have decimal point.
c) It could be +ve or –ve.
d) Default sign will be +ve.
e) No comma or blank space is allowed between float constant

The allowable range of float constant is –3.4E+38 to 3.4E+38

- **Exponential Real Constant/Express a real constant in Exponential form**

A real constant is combination of a **whole number** followed by a **decimal point** and the **fractional part**. If the value of a constant is

either too small or too large, exponential form of representation of real constants is usually used.
In exponential form, the real constant is represented in **two** parts.

Mantissa - The part appearing before **e,** the mantissa is either a real number expressed in decimal notation or an integer.

Exponent - The part following **e,** the exponent is an integer with an optional plus or minus sign followed by a series of digits. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase.

Example:
0.000342 can be represented in exponential form as 3.42e-4
7500000000 can be represented in exponential form as 7.5e9 or 75e8

Rules for Constructing Real Constants in Exponential Form

1. The mantissa part and the exponential part should be separated by letter in exponential form
2. The mantissa part may have a positive or negative sign.
3. Default sign of mantissa part is positive.
4. The exponent part must have at least one digit, which must be a positive or negative integer. Default sign is positive.
5. Range of real constants expressed in exponential for is -3.4e38 to 3.4e38.

2. Character Constants:

Character variables are used to store character constants. Character constants use single quotation marks around a character. When the C++ compiler encounters such a character constant , it translates it into the corresponding ACSII code and stores that number in the program. The constant 'a' appearing in a program, for example, will be stored as 97.

// This program demonstrates character variables and constants.

```
# Include <iostream.h>
 void main( )
            {
            char  aa;
            aa = 'k';
            cout <<endl<<aa<<endl;
            }
```

```
OUTPUT
k
```

Rules for constructing character constants:

- **a)** A characters constant is a single alphabet, single digit , or a single special symbol enclosed with in a single inverted comma eg 'd' , '$' , '1'.
- **b)** The maximum length of character constant can be 1 character.

3-String Constants:-

The combination of characters including blank space , enclose in double quotation marks is called string constant. Its value is set when the program is written and it retains this value throughout the program's existence.

```
#include <iostream.h>
void main()
      {
      cout <<endl<<" Every age has a language of its own " <<endl;
      }
```

KEYWORDS OR RESERVED WORDS:

Keywords or reserved words are those words that are reserved for specific purpose and can not be used for other purposes. They have special meanings and compiler Knows their meanings. In C/C++ all keywords are in lower-case, therefore the keywords must be entered in lowercase.

Reserved words can not be used as variable name and as an identifier names.

Some keywords are used in C/C++ are given below.

i) while ii) cout iii) do iv) if v) int vi) case vii) float viii) else vx) long x) switch. etc

ASSIGNMENT STATEMENT:

It has the following general form.

Variable = expression

These statements are used to assign the values to the variables that lie on the left hand side of the equal sign. The variable may be any variable name of a recognize type. Expression may be a single variable or constant or a combination of constants and variables. For example

```
x = a + b * 55 - 4/2
y ='q';
```

// This program demonstrates the use of assignment statements.

```
#include <iostream.h>
        void main( )
        {
                char   aa;
                int a , b;
                a = 2;
                b = a + 6;
                aa = 'k';
                cout <<endl <a<<endl<<b;
                cout <<endl<<aa<<endl;
                }
```

```
Output
2
8
k
```

MULTIPLE ASSIGNMENT

Multiple assignments are a language property of being able to assign one value to more than one variable. It usually looks like the following:

a = b = c = d = 0 // assigns all variables to 0.

C++ allows for statements like

int a, b, c;

a = b = c = 10;

It is possible to do something like

```
int a = 1;
int b = 2;
int c = 3;

a, b, c += 3;
```

to achieve

```
a = a + 3;
b = b + 3;
c = c + 3;
```

HOW TO ASSIGN STRING

The term *string* generally means an ordered sequence of characters, with a first character, a second character, and so on, and in most programming languages such strings are enclosed in either single or double quotes. In C++ the enclosing delimiters are double quotes. In this form the string is referred to as a *string literal* and we often use such string literals in output statements when we wish to display text on the screen for the benefit of our users. For example, the usual first C++ program displays the string literal "Hello, world!" on the screen with the following output statement:

```
cout << "Hello, world!" << endl;
```

Initializing a C-string variable

```
------------------------------------
char str1[11] = "Call home!";
char str2[] = "Send money!";
char str3[] = "OK";
```

Initializing a C++ string object

```
------------------------------------
string str1("Call home!");
string str2 = "Send money!";
string str3("OK");
```

OPERATORS:

In C++, different operators are used to perform different types of operations. The following different types of operators are used in C++ for different operations.

1. Arithmetic Operators:

C++ uses the following four normal arithmetic operators i.e **+, -, *,** **%** and **/** for addition, subtraction, multiplication remainder(division) and quotient (division). These operators are used with all data types both integer and floating-point.

2. The Remainder Operator:-

There is fifth arithmetic operator that works only with integer variables(char, int and long) . It is called the remainder operator, and is represented by **%** the percent symbol. This operator (also called Modulus operator) finds the remainder when one number is divided by another. Consider the following program.

PRIORITY OF OPERATOR

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right

Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example

Try the following example to understand operator precedence

```
#include <stdio.h>
main() {
  int a = 20;
  int b = 10;
  int c = 15;
  int d = 5;
  int e;
  e = (a + b) * c / d;     // ( 30 * 15 ) / 5
  printf("Value of (a + b) * c / d is : %d\n", e );
  e = ((a + b) * c) / d;   // (30 * 15 ) / 5
  printf("Value of ((a + b) * c) / d is  : %d\n" , e );
  e = (a + b) * (c / d);   // (30) * (15/5)
printf("Value of (a + b) * (c / d) is  : %d\n",  e );
e = a + (b * c) / d;     //  20 + (150/5)
printf("Value of a + (b * c) / d is  : %d\n" ,  e );
 return 0;
}
```

When you compile and execute the above program, it produces the following result

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of (a + b) * (c / d) is  : 90
Value of a + (b * c) / d is  : 50
```

```
// This program demonstrates remainder operator.
#include <iostream.h>
void main()
{
        cout << 22 % 8 << endl
             << 23 % 8 << endl
             << 8 % 8  << endl
             << 9 % 8  << endl
             << 10 % 8 << endl;
}
```

In the above program the numbers 22 through 10 are divided by 8 , using the remainder operator.
The answer are 6, 7, 0, 1, and 2 i.e the remainders of these divisions.

ARITHMETIC ASSIGNMENT OPERATORS:

C++ offers several different ways to shorten and clarify our code. One of these is the arithmetic assignment operator. For example ,The following kind of statement is common in most languages.

```
Total = Total + Item;
```

For such statements C++ offers a condensed approach i.e the arithmetic assignment operator, which combines an arithmetic operator and an assignment operator, and eliminate the repeated operand. For example, consider the following statement.

```
Total   += item;
```

There are arithmetic assignment operators corresponding to all the arithmetic operations, i.e +=, -=, *=, /= and %=. The following program shows the use of such arithmetic operators.

```
// This program demonstrates arithmetic  assignment operators.
#include  <iostream.h>
 void main()
{
          int ans = 27;
          ans  += 10;
          cout << endl << ans;
          ans -= 7;
          cout << endl << ans;
          ans *= 2;
          cout << endl  << ansi;
          ansi  /= 3;
          cout << endl << ansi;
          ansi  %= 3;
          cout << endl << ansi;
}
```

```
OUTPUT
37
30
60
20
2
```

Increment Operator:

Consider the following C++ statement.

Count = Count + 1;

Instead of this statement we can also use the following approach.

Count += 1;

Instead of the above statement, C++ also allow another more condense approach.

++Count;

In this statement the ++ operator is called the increment operator which increments its argument.

Programming example of increment operator

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a;
for(a=1;a<=10;a++)
cout<<a<<endl;
getch();
}
```

Prefix and Postfix operator:
The increment operator can be used in two ways, as a prefix, meaning that the operator precedes the variable; and as a postfix, meaning that the operator follows the variable. What is the difference? Often a variable is incremented within a statement that performs some other operation on it . For example

Totalweight = avg * ++count;

The question here is, the multiplication performed before or after count is incremented? In this case count is incremented first due to the prefix notation is used i.e ++count. If we used
postfix notation, count++, the multiplication would performed first, then count would have been incremented. Consider the following program.

// This program demonstrates the increment operators i.e postfix and prefix.

```
#include <iostream.h>
void main()
{
        int count = 10;
        cout << "count" << count << endl;
        cout << "count" << ++count << endl;
        cout << "count" << count << endl;
        cout << "count" << count++ << endl;
        cout << "count" << count << endl;
}
```

```
OUTPUT
Count=10
Count=11
Count=11
Count=11
Count=12
```

The Decrement (- -)Operator:- The decrement operator , --, behaves very much like the increment operator, except that it subtracts 1 from its operand. It can also be used in both prefix and postfix forms.

// Programming example of Decrement operator.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a;
for(a=20;a<=1;a--)
cout<<a<<endl;
getch();
}
```

TYPE CASTING OPERATOR

Type Casting In C Language. Type casting is a way to convert a variable from one data **type** to another data **type**. For example, if you want to store a long value into a simple integer then you can **type cast** long to int.

```
include <stdio.h>
main() {
  int sum = 17, count = 5;
  float mean;
  mean = (float) sum / count;
  printf("Value of mean : %f\n", mean );
}
```

OUTPUT

```
Value of
mean :
3.40
```

EXPRESSIONS:

It is the combination of operand and operator. Any arrangement of variables and operators that specifies a computation is called an expression. For example A+B - 6 and k*l+h are expressions. When the computations specified in the expression are performed , the result is usually a value. The expressions are not the same as statements. Statements tell the compiler to do something and terminate with a semicolon, while expression specifies a computation. There can be several expressions in a statement.

COMMENTS

In the C Programming Language, you can place comments in your source code that are not executed as part of the program.

Comments provide clarity to the C source code allowing others to better understand what the code was intended to accomplish and greatly helping in debugging the code. Comments are especially important in large projects containing hundreds or thousands of lines of source code.

A comment starts with a slash asterisk /* and ends with a asterisk slash */ and can be anywhere in your program. Comments can span several lines within your C program. Comments are typically added directly above the related C source code.

Adding source code comments to your C source code is a highly recommended practice.

SYNTAX

The syntax for a **comment** is:

/ comment goes here */*

OR

```
/*
 * comment goes here
 */
```

C++ introduced a double slash comment prefix // as a way to comment single lines. The following is an example of this:

```
// This is first C++ program
```

STATEMENT

A **statement** is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or collect input. A computer program is made up of a series of statements.

break statement, if statement, compound statement, null statement, continue statement, return statement, do-while statement, switch statement, expression statement for statement, goto and labeled statements, while statement etc.

```
x = ( y + 3 );          /* x is assigned the value of y + 3  */
x++;                    /* x is incremented                  */
x = y = 0;              /* Both x and y are initialized to 0 */
```

```
proc( arg1, arg2 );       /* Function call returning void   */
y = z = ( f( x ) + 3 );   /* A function-call expression    */
```

C/C++ LIBRARIES

The C standard library is a standardized collection of header files and library routines used to implement common operations, such as input/output and character string handling. Unlike other languages (such as COBOL, Fortran, and PL/I) C does not include builtin keywords for these tasks, so nearly all C programs rely on the standard library.

Example

```
#include <stdio.h>        /* for fprintf() and scanf */
#include <math.h>         /* for arithmetic function */
#include <string.h>       /* for string characterr() */
```

Name	Description
<time.h>	Defines date and time handling functions
<string.h>	Defines string handling functions.
<stdlib.h>	Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
<stdio.h>	Defines core input and output functions
<math.h>	Defines common mathematical functions.

etc

Note: In C++#include <iostream.h> // for cin and cout

INPUT AND OUTPUT FUNCTIONS: The following functions are used for input and output of data.

INPUT function use in C

scanf

The **scanf function** allows you to accept data from input device. You should include <stdio.h> header file for scanf and printf

int uses %d

float uses %f
char uses %c
character strings %s
Ampersand & is used before variable name in scanf() statement as &ch. It is just like in a pointer which is used to point to the variable.

```
#include <stdio.h>
int main()
{
   int a, b, c;
   printf("Enter the first value:");
   scanf("%d", &a);
   printf("Enter the second value:");
   scanf("%d", &b);
   c = a + b;
   printf("%d + %d = %d\n", a, b, c);
   return 0;
}
```

Output function use in C

printf() function:

printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.

```
#include<stdio.h>
        main()
        {
                int a,b;
                float c,d;

                a = 15;
                b = a / 2;
                printf("%d\n",b);
                printf("%3d\n",b);
```

```
Output of the
source
7
  7
007
5.10
```

```
        printf( %03d\n ,b);
        c = 15.3;
        d = c / 3;
        printf("%3.2f\n",d);
}
```

As you can see in the first printf statement we print a decimal. In the second printf statement we print the same decimal, but we use a width (%3d) to say that we want three digits (positions) reserved for the output. The result is that two "space characters" are placed before printing the character. In the third printf statement we say almost the same as the previous one. Print the output with a width of three digits, but fill the space with 0.

In the fourth printf statement we want to print a float. In this printf statement we want to print three position before the decimal point (called width) and two positions behind the decimal point (called precision).

The \n used in the printf statements is called an escape sequence. In this case it represents a newline character. After printing something to the screen you usually want to print something on the next line. If there is no \n then a next printf command will print the string on the same line.

Format	Explaination	Example
%d	Display an integer	10
%f	Displays a floating-point number in fixed decimal format	10.500000
%.1f	Displays a floating-point number with 1 digit after the decimal	10.5
%e	Display a floating-point number in exponential (scientific notation)	1.050000e+01
%g	Display a floating-point number in either fixed decimal or exponential format depending on the size of the number (will not display trailing zeros)	10.5

OUTPUT USING COUT:

Consider the following program statement.

cout<<x<<endl<<k<<endl<<y<<endl;

In this statement the cout identifier is actually an object. It is predefined in Turbo C++ to correspond to the standard output stream. A stream is an abstraction that refers to a flow of data. The standard output stream normally flows to the screen display although it can be redirected to other output devices.

The operator << is called the insertion operator. It directs the contents of the variable on its right to the object on its left. In the above cout statement, it directs the string constants i.e "Every age has its own language" to cout, which sends it to the display. The general form for cout is as follow.

Cout << data [<< data];

The cout is the keyword and the data can be variables, constants, expressions, or a combination of all three. The optional parameter appears in square brackets, [ ] . The text enclosed in square brackets may be present one or more times, or not at all.

INPUT WITH CIN:

cin operator is used to feed data to the computer using keyboard. When this statement is executed in a program , computer wait and the user has to input data for the given list of variables in the cin statement. After providing the complete input data , the computer then process the next statement in the program and produced results. Consider the following program which shows that how cin accomplishes input.

Write a program to demonstrate cin and cout and to find area.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int r;//r radious
float p=3.14;
float area;
cout<<"Enter Value for Radious";
cin>>r;
area=2*p*r*r;                          //To Find area of a circle
cout<<"The Area is :::"<<area;
getch();
}
```

// This program demonstrates the Fahrenheit and centigrams conversion.

```
#include <iostream.h>
void main()
{
        float ftemp,ctemp;
        cout << " Enter temperature in Fohrenheit:  ";
        cin >> ftemp;
        ctemp = (ftemp-32) * 5/9;
        cout << " Equivalent in Celsius is  :    " << ctemp << '\n';
}
```

The statement

```
        Cin >> ftemp;
```

Causes the program to wait for the user to type in a number. The giving number is placed in the variable ftemp. The keyword cin is an object, predefined in C++ to correspond to the standard input stream. The stream represents data coming from the keyboard. The >> operator takes the value from the stream object on its left and places it in the variable on its right.

The output of the above program is as below.

```
Enter temperature in fahrenheit :  212
Equivalent in Celsius  is  :    100
```

This cin operator has the following general form.

cin >> values;

get()

The get() function is used to input a single character from the keyboard

Put( )

The put() function is used to output a single character on the screen.

Consider the following program.

```
// This program demonstrates the use of  get() and put().
#include  <fstream.h>
main()
        {
        char  xx;
        char  x1 , x2;
        cout<<endl<<"Enter  your  name   ";
        cin.get(xx);
        x1 = xx;
        cout<<endl<<"Enter  your father  name   ";
        cin.get(xx);
        x2 = xx;
        cout<<endl<<" Your biodata is here  "<<endl;
        cout.put(x1);
        cout.put(x2);
        return 0;
        }
```

The output of this program is as below

```
Enter  your  name  A
Enter  your father  name  B
Your biodata is here
AB
```

Puts( )

The C library function **puts** writes a string including the null character.

Example

The following example shows the usage of puts() function.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char str1[15];
  char str2[15];
  strcpy(str1, " Khalidseries");
  strcpy(str2, " Khalidseries");
```

Output

```
Khalidseries
Khalidseries
```

```
  puts(str1);
  puts(str2);
  return(0);
}
```

gets() function :

Reading or Accepting String From User in C

Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

gets(<variable-name>)

A program to show the use of gets() and puts() functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[30];
puts("Enter the name ");
gets(name);
puts(name);
getch();
}
```

OUTPUT

```
Enter a string : Khalidseries
You entered: Khalidseries
```

Example

The following example shows the usage of gets() function.

```
#include <stdio.h>
int main()
{
  char str[50];
  printf("Enter a string : ");
  gets(str);
  printf("You entered: %s", str);
  return(0);
  getch();
}
```

OUTPUT

```
Enter a string : Khalidseries
You entered: Khalidseries
```

getch()

getch() is used to get a character from console but does not echo to the screen.

getche()

getche() is used to get a character from console, and echoes to the screen.

getchar()

getchar() is used to get or read the input (i.e a single character) at run time.

```
int c;
    c = getchar();
```

putchar

Putchar is a function in the C programming language that writes a single character to the standard output stream

Example

The following example shows the usage of putchar() function.

```
#include <stdio.h>
int main ()
{
  char ch;
  for(ch = 'A' ; ch <= 'Z' ; ch++)
  {
    putchar(ch);
  }
    return(0);
}
```

OUTPUT

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Example for getch() in C/C++

```
#include <stdio.h>
int main()
{
  printf("%c", getch(stdin));
  return(0);
}
```

Run on IDE

Input: g (press enter key)

Output: g

Example for getchar() in C/C++

```
#include <stdio.h>
int main()
{
  printf("%c", getchar());
  return 0;
}
```

Run on IDE

Input: g(press enter key)

Output: g

Example for getch()

```
#include <stdio.h>
#include <conio.h>
int main()
{
  printf("%c", getch());
  return 0;
}
```

Run on IDE

Input: g (Without enter key it is buffered)

Output: Program terminates immediately.

But when you use DOS shell in Turbo C, it shows a single g, i.e., 'g'

Example for getche( )

```
#include <stdio.h>
#include <conio.h>
// Example for getche( ) in C
int main()
{
  printf("%c", getche());
  return 0;
}
```

Run on IDE

Input: g(without enter key it is not buffered)

Output: Program terminates immediately.

But when you use DOS shell in Turbo C, double g, i.e., 'gg'

DECISIONS:

Decisions can be made in C++ in several ways. The most important is with the if ...else statement, which chooses between two alternatives. This statement can be used without the else statement as a simple **if** statement. Another decision statement is called **switch** statement that creates branches for multiple alternatives sections of code depending on the value of a single variable. Finally the conditional operator is used in specialized situations.

The if statement:

The if statement is the simplest of the decision statements. It tests a relationship, using the relational operators, and makes a decision about which statement to execute next , based on the result of that decision. It has the following general format.

If (logical expression)

{ A block of 1 or more C++ statements}

if is the keyword. The condition includes any relational comparison and must be enclosed within parenthesis. The block of the if , sometimes called the body of the if statement contains at least one or more statements. If only one statement follows the if , the braces are not required. The block will execute only if the condition is true. If the condition is false , C++ ignores the block and simply executes the next statement in the program.

<u>The single selection if structure</u>

Consider the following program.

// This program demonstrates the use of if statement.

```
#include <iostream.h>
void main()
{
        int x;
        cout <<endl<< "Enter a number:    ";
        cin >> x;
        if(x >100)
        cout << " The number is greater than 100\n";
        cout<<endl<<"This  is the end of program<<endl;
}
```

The if key word is followed by a test expression in parenthesis. The syntax of the if statement is as follow.

If (logical expression)

```
        Statement;
        If (logical expression)
        {
        statement;
        statement;
        statement;
        }
```

From the syntax of the if statement it is clear that it is very much like that of while . The difference is that the statements following the if are executed only once if the test expression is true; the statements following the while are executed repeatedly until the test expression becomes false.

// This program demonstrates the use of if statement.

```
#include <iostream.h>
void main()
{       int x;
        cout << "Enter a number:    ";
        cin >> x;
        if(x >100)
        {
        cout << " The number is :     " << x;
        cout << " The number is greater than 100\n";
}
cout<<"This is the end of program<<endl;
}
```

Write a program to find that the one number is divisible by anan-other number are not.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int n1; //number to be divide
int n2; //second number divisor n1 / n2
cout<<"Enter a n1=";
cin>>n1;
cout<<"Enter a n2=";
cin>>n2;
if(n1%n2= =0)
{
cout<<"The number  n1 is divisable by n2";
}
else
cout<<"n1 is not divisible by n2";
getch();
}
```

Write a program to find out that the given number is ODD or EVEN.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a;
cout<<"Enter an Integer value:::::";
cin>>a;
if(a%2= =1)
cout<<"It is an odd Number";
else
cout<<"It is an Even Number";
getch();
}
```

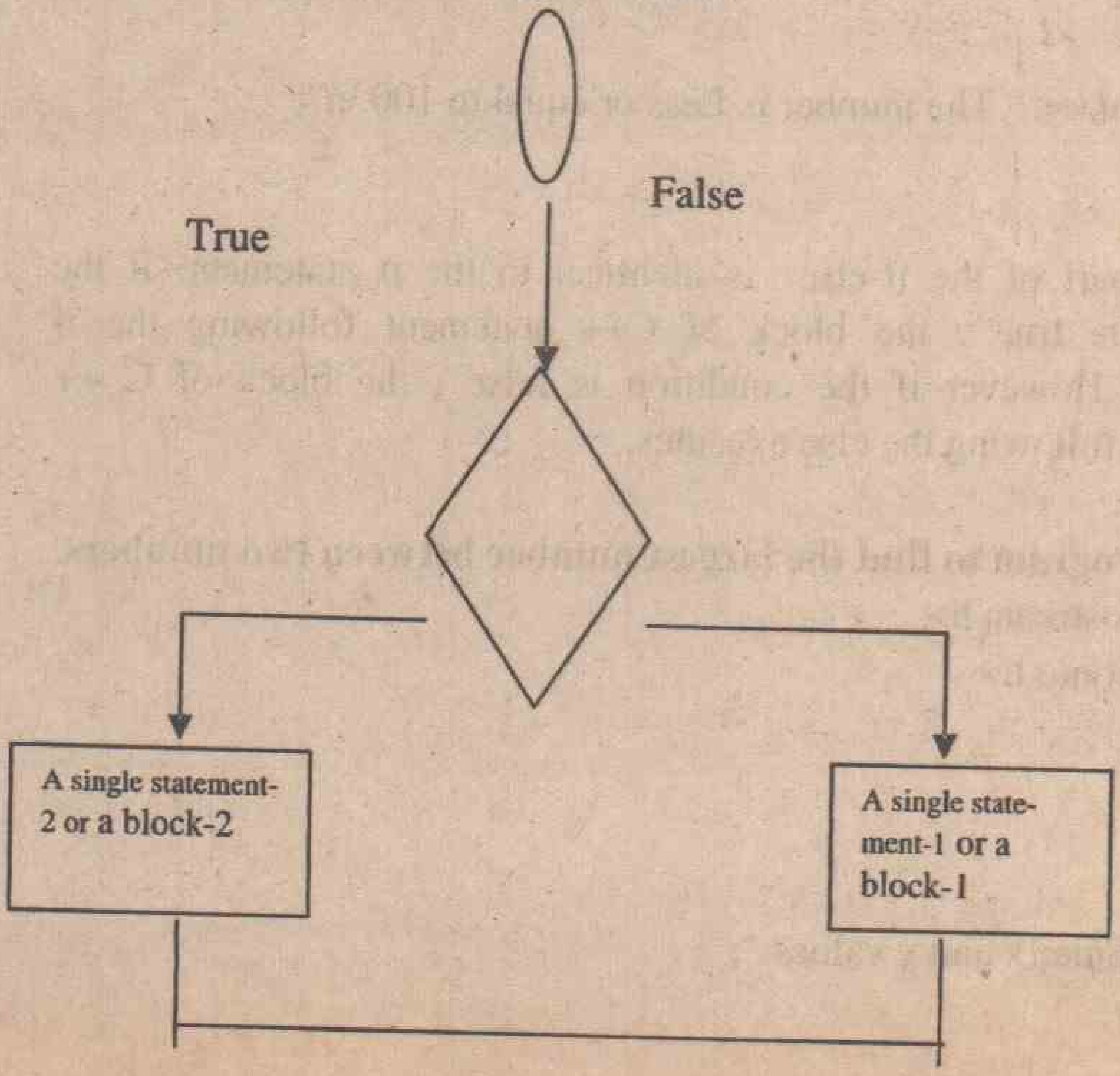
Write a program to find out that you are pass or fail.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int marks = 0;
cout<<"Enter your marks: ";
cin>>marks;
if(marks >=40)
{
cout<< "You are Pass";
else
cout<<"You are Fail";
}
getche();
}
```

The else Statement:
The else statement never appears in a program without if statement. It has the following general form.

```
If (logical expression)
        {
        A BLOCK OF ONE OR MORE C++ STATEMENTS
        }
else
        {
        A BLOCK OF 1 OR MORE C++ STATMENTS
        }
```

Consider the following flowchart.


False
True
A single statement-2 or a block-2
A single statement-1 or a block-1

From the flowchart , it is clear that when the logical condition given in the **if-else structure** is true , statement-1 or block-1 is executed and then control is transfer to the statement following the **if structure** otherwise statement-2 or block-2 is executed and then the computer execute the statements of the program in sequential order.
Consider the following program.

```
//This program demonstrates the use of if...else statement.
#include <iostream.h>
#include<conio.h>
void main()
{          int num;
           cout << "Enter any number:::: ";
           cin >> num;
           if (num >100)
                     {
                     cout << "The number is greater than 100 \n";
                     }
           else
                     {
           cout << " The number is Less or equal to 100 \n";
                     }
}
```

The first part of the if-else is identical to the if statement. If the condition is true . the block of C++ statement following the if executes. However if the condition is false , the block of C ++ statements following the else executes.

Write a program to find the largest number between two numbers.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int x,y;
clrscr();
cout<<"\nEnter x and y values:";
cin>>x>>y;
```

```
if(x>y)
{
cout<<"x is greater than y";
}
else
{
cout<<"y is greater than x";
}
getch();
```

Write a program to find out the greatest number between Three numbers .

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a,b,c;
cout<<"Enter First value:::::";
cin>>a;
cout<<"Enter Second Value::::";
cin>>b;
cout<<"Enter Third value:::::";
if(a>b)
   if(a>c)
   cout<<"First number is greater";
   else
   cout<<"Third Number is greater";
   else
   if(b>c)
   cout<<"Second Number is greater";
   else
   cout<<"Third Number is greater";
getch();
}
```

Nested if statements:

If one or more than one **if** statements completely occur inside the other **if** statement, then it is called **nested if structure.**
Consider the following program.

//Write a program to find out the largest number between Three numbers using Nested if statement methods.

```
#include <iostream.h>
#icnlude <conio.h>
void main()
{
int x , y, z, big ;
cout<<endl<<"Enter values for x, y, z";
cin>>x>>y>>z;
    big = x;
            if(y >big)
    big = y;
            if(z>big)
 {
    big = z;
   cout << endl<< big;
 }
 }
```

//Write a program to find out the largest number between Three nubers using Nested if statement methods.

```
#include <iostream.h>
#icnlude <conio.h>
void main()
{
int x , y, z , small;
cout<<endl<<"Enter values for x, y, z";
 cin>>x>>y>>z;
    small = x;
            if(y <small)
   small = y;
            if(z>small)
            {
```

```
            small = z;
cout << endl<< small;
            }
getch();
}
```

// This program demonstrates the use of nested if statements.

```
#include <iostream.h>
#inlude  <conio.h>
void main()
{
        int a,b, x  ,  y,  z ;
        cout<<endl<<"Enter values for x, y, z  ";
        cin>>x>>y>>z;
        if(x >9)
                {
                a = x;
                cout <<endl<<a;
                }
        else
                if(y <9)
                {
                b = y;
                cout << endl<< y;
                }
                else
                cout<<endl << " 'b' has Wrong  value  ";
}
```

Write a program to find out the Grade of student marks.

```
#include <iostream>
#include<conio.h>
        int main()
        {
          int mark;
          cout << "What mark did they get in a test?" << endl;
          cin >> mark;
           if(mark >= 90)
          {
```

```
        cout << "You got an A*" << endl;
        cout << "You Passed!" << endl;
    }
    else if(mark >= 80)
    {
        cout << "You got an A" << endl;
        cout << "You Passed!" << endl;
    }
    else if(mark >= 70)
    {
        cout << "You got an B" << endl;
        cout << "You Passed!" << endl;
            }
    else if(mark >= 60)
    {
        cout << "You got an C" << endl;
        cout << "You Passed!" << endl;
    }
    else if(mark >= 50)
    {
        cout << "You got an D" << endl;
        cout << "You Failed!" << endl;
    }
    else if(mark >= 40)
    {
        cout << "You got an E" << endl;
        cout << "You Failed!" << endl;
    }
    else
    {
        cout << "You got an U" << endl;
        cout << "You Failed!" << endl;

    }
getch();
return 0;
        }
```

Conditional operator (?:)

The conditional operator is the only ternary operator in C++ language. It requires three operands instead of unary and binary operator. It is used to replace **if-else** logic in some cases. Consider the following general form.

Conditional_expression ? statment1 : statment2 ;

In the above structure the computer tests the resultant value of the conditional_expression , if it is true , statment1 is executed and if it is false then statemnt2 is executed. Only one of the expression following the **?** is never executed.

For example.

(a >9) ? (x = 1) : (x = 2) ;

The use of parenthesis is optional .

Consider the following if-else structure.

```
If (a>b)
        {
          x =9;
        }
else
        {
          x =8;
        }
```

The above structure can be written as below using the conditional oprater.

(a > b) ? (x =9) : (x= 8);

we can also write the above statement as

x = (a>b) ? (9) : (8);

Hence any **if-else** statement can be rewritten as a conditional and a conditional can be rewritten as an **if-else** statement.

THE *SIZEOF* OPERATOR:

It is a unary operator because it operates on a single value. This operator is used to produce the size of data or data type in bytes. It has one of the following general forms.

Sizeof data
Sizeof (data type)

For example consider the following program.

```
#include <iostream.h>
#include<conio.h>
void main()
{
cout<<endl<<"The size of integer  data type  =  "<< sizeof(int);
return (0)
}
```

```
OUTPUT
The size of integer  data type  =  2
```

THE COMMA OPERATOR:

This operator is also called sequence point . It is used to put more than one expression on a single line separated by this operator, i.e For example.

```
        int  a = `6  ,  b = 9;    // separates different
                                   //statements on a single line.
```

or

```
        int   a , b , c;           // separates different variable
                                   //names on a single line.
```

or

```
        sq = (n * n) ,  cube = (n * n * n);
```

or

// separates different expressions on a single line.

```
        i= 9;
        j = (i =3,  i + 4);        // at the end  j has a value of  7 .
```

or

```
        x = ( a = 3 , b = 99)  // at the end a has a value of 3
                           //, b has 99 and x has a value of 99 also.
```

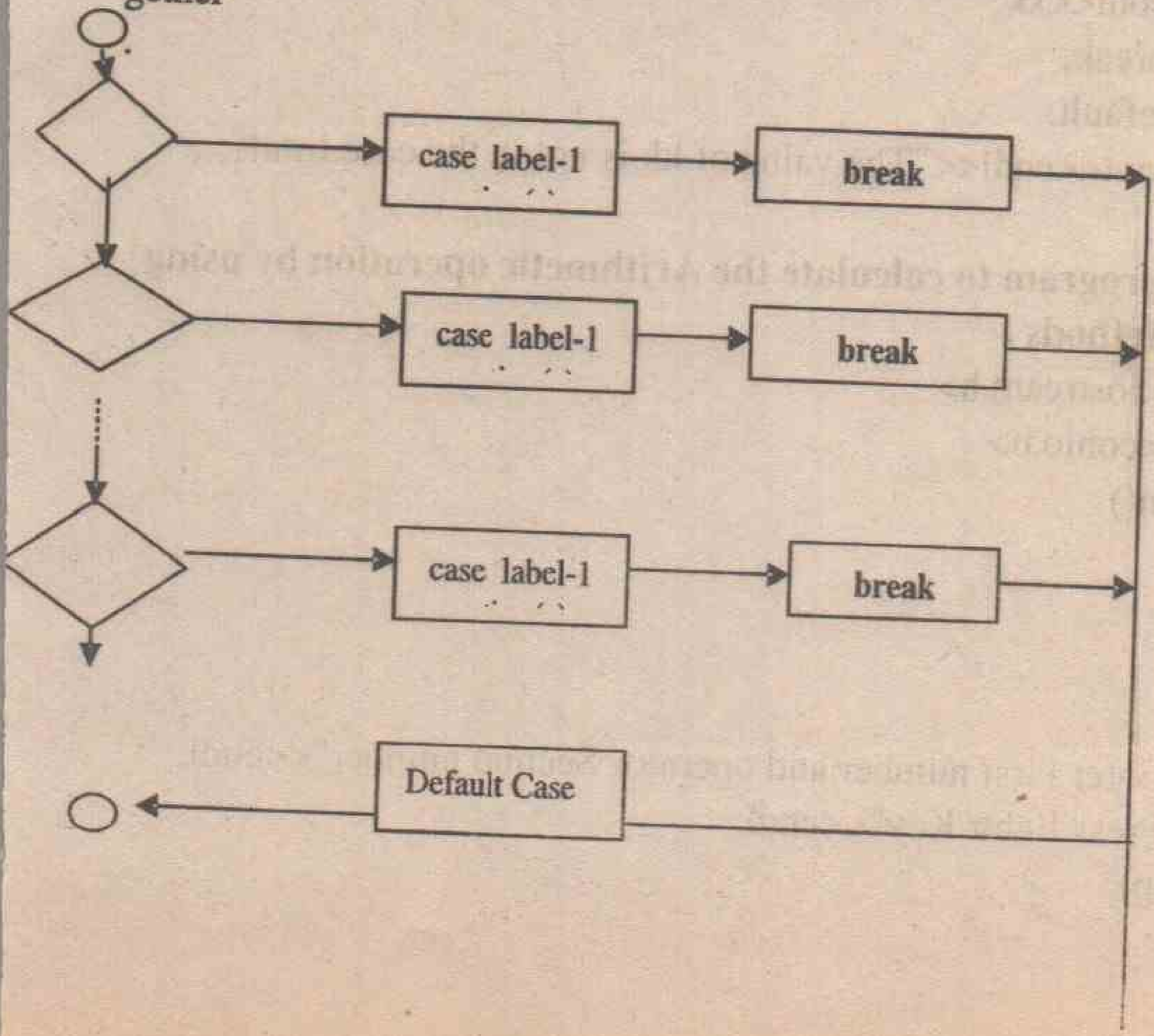
THE SWITCH STATEMENT OR MULTIPLE-SELECTION STRUCTURE :

This structure is used to handle multiple-selection situation during decision-making. It consists of a series of **case** labels and an optional **default** case. Consider the following general form

```
switch ( controlling expression)
case label-1 :    One  statement or a block of statements
                  break;
```

```
case label-2 :  One  statement or a block of statements
                break;
case label-1 :  One  statement or a block of statements
                break;
case label-1 :  One  statement or a block of statements
                break;
case label-1 :  One  statement or a block of statements
                break;
case label-1 :  One  statement or a block of statements
                break;
default : cout << endl<< " This statement or block of statements is executed if the above all fails";
```

The switch is a keyword that is followed by an expression enclosed in parenthesis called controlling expression. The value of this expression is compared with each of the **case** labels. If a match occurs , then the statements for that case are executed. In case of multistatment case, braces are optional. The break statement transfer control to the first statement after the switch structure. The break statement is used because the cases in a switch structure would otherwise run together



Example program:

```
// This program demonstrates the use of switch  structure.
#include <iostream.h>
void main()
        {
        int      kk;
        cout <<endl<<"Enter a value for  kk  :";  cin>>kk;
        switch (kk)
        case   1: cout<<endl<<"The value  entered for kk  =  ";
         cout<<kk;
         break;
        case   2: cout<<endl<<"The value  entered for kk  =  ";
         cout<<kk;
         break;
        case   3: cout<<endl<<"The value  entered for kk  =  ";
         cout<<kk;
         break;
        case   4: cout<<endl<<"The value  entered for kk  =  ";
         cout<<kk;
         break;
        default:
        cout<<endl<<"The value of kk is not in the case limit";
        }
```

Write a program to calculate the Arithmetic operation by using switch Methods .

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a,b;
char op;
cout<<"Enter First number and operator Second number"<<endl;
cout<<"Press Enter Key"<<endl;
switch(op)
{
case '+':
```

```
cout<< Addition= <<(a+b);
break;
case '-':
cout<<"Subtraction="<<(a-b);
break;
case'*':
cout<<"Multiplication="<<(a*b);
break;
case'/':
cout<<"Division ="<<(a/b);
break;
case '%':
cout<<"Reminder="<<(a%b);
break;
default:
cout<<"invaled input";
}
getch();
}
```

Write a program to find out your skill using numbers

```
include <iostream.h>
void main()
 {
   short int number;
  cout << "Enter a number between 1 and 5: ";
  cin >> number;
  switch (number)
  {
    case 0:   cout << "Too small, sorry!";
              break;
    case 5:  cout << "Good job!\n";        // fall through
    case 4:  cout << "Nice Pick!\n";       // fall through
    case 3:  cout << "Excellent!\n";       // fall through
    case 2:  cout << "Masterful!\n";       // fall through
    case 1:  cout << "Incredible!\n";
             break;
    default: cout << "Too large!\n";
```

```
            break;
  }
  cout << "\n\n";
   return 0;
         }
```

```
Enter a number between 1 and 5: 3
Output:
Excellent!
Masterful!
Incredible!
```

RELATIONAL OPERATORS:

A relational operator compares two values . The value can be any built-in C++ data type , such as char, int, and float, or user-defined classes. The comparisons involve such relationships as equal to, less than. greater than, and so on. The result of the comparisons is true or false.

The following list shows different type of relational operators.

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

Consider the following program.

// This program demonstrates the use of operators.

```
#include <iostream.h>
void main()
{        int numb;
         cout << " Enter a number :   ";
         cin  >> numb;
         cout << "numb<10  is " << (numb < 10) << endln;
         cout << "numb>10  is " << (numb > 10) << endln;
         cout << "numb= =10  is " << (numb = =  10) << endln;
getch( );
return 0;
 }
```

```
OUPUT
Enter a number :  20
Numb <10  is   0
Numb >10  is   1
numb= =10 is   0
```

LOGICAL OPERATOR:

These operators are used to form more complex logical expressions by combining two or more than two simple logical expressions .

These logical operators are

1. Logical AND :- It is denoted by &&.
2. Logical OR :- It is denoted by ||.
3. Logical NOT :- It is denoted by !.

Write a Programming example of logical operator.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a;
char op;
cout<<"Enter an integer value"<<endl;
cin>>a;
cout<<"Enter any character"<<endl;
cin>>op;
if((a<0) II (op=='y') II (op=='n'))
cout<<"OK, Condition is true";
else
cout<< "All relational expression are fals";
getch();
return 0;
}
```

Another example of logical operatoer.

```
if (a==2   &&  b>3  )
        {  x  = 2;
        y  =  3;
        }
        if( a <= 4  || b>5)
        {
        z= 5;      is treated as  true.
        k= 6;
        }
      if(a != 6)
          x=6;
else
x=5;
```

Any expression that produces a value can be used in the decision portion of any control structure. If that value is zero , it is treated as false , and if the value is nonzero , it Consider the following table of different operators for priority.

Operators	type
( )	Parenthesis
++ -- + - !	Unary
* / %	Multiplicative
+ -	Additive
<< >>	Insertion/extraction
< <= > >=	Relational
== !=	Equality
&&	Logical AND
||	Logical OR
?:	Conditional
= += -= *= /= %=	Assignment

CONTROL STATEMENTS:

The flow of control jumps from one part of the program to another, depending on calculations performed in the program. Program statements that cause such jumps are called control statements. There are two major categories of control statements , i.e Loops and Decisions.

How many times a loop is executed, or weather a decision results in the execution of a section of code, depends on weather certain expressions are true or false. These expressions are typically involve a kind of operator called a relational operator , which compare two values. Since the operation of loops and decisions is so closely involved with these operators , so we will discuss them first.

COUNTERS:

Counter is very important feature of computer programming. It is technique for controlling a looping process i.e. it is a organized method of counting the number of repetitions.

There are three types of counter; which are given below:

1) Standard Counter.
2) Accumulator Counter.
3) Multiplicative Counter.

1) STANDARD COUNTER

The standard counter is normally used for counting process. With the help of this counter, it is possible to count and control the number of execution of a statement or group of statements. A few examples of Standard counters are given below:

C=C+1; C++; Z=Z+5; N+=10;

Notice that each time a Standard counter is executed a constant (fixed) value is added to the counter.

2) ACCUMULATOR COUNTER

Accumulator Counter is normally used to accumulate the subtotals. Notice that the value of Accumulator counter is not indexed by a fixed value (such as 1,5,10 etc), but by the value of a variable.

The Accumulator counter often looks like this:

A=A+X; Z+=Y; N=N+M;

3) MULTIPLICATIVE COUNTER

Multiplicative counters are used for multiplication purposes. Multiplicative counter, the starting value of the counter must be one. If the values initialize this counter will be zero, because we know that zero multiplied by anything will be zero. Multiplicative counters often look like these: M=M*2; Z *=5; X *=Z;

LOOPS:

Loops are used to repeat a block of code. Loops are used to execute a section of a program repeatedly for a certain number of times. The repetition continues while a condition is true. When the condition becomes false , the loop ends.

FOR LOOP

The for loop executes a section of code for a fixed number of times. It is usually used when we know, before entering the loop, how many times we want to execute the code in the loop.

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

The syntax of a **for** loop

for (index-variable = initial value ; index variable relational operator(s) final value ; increment)

OR

Syntax

The syntax of a **for** loop

```
for ( init; condition; increment )
{
  statement(s);
}
```

Here is the flow of control in a 'for' loop

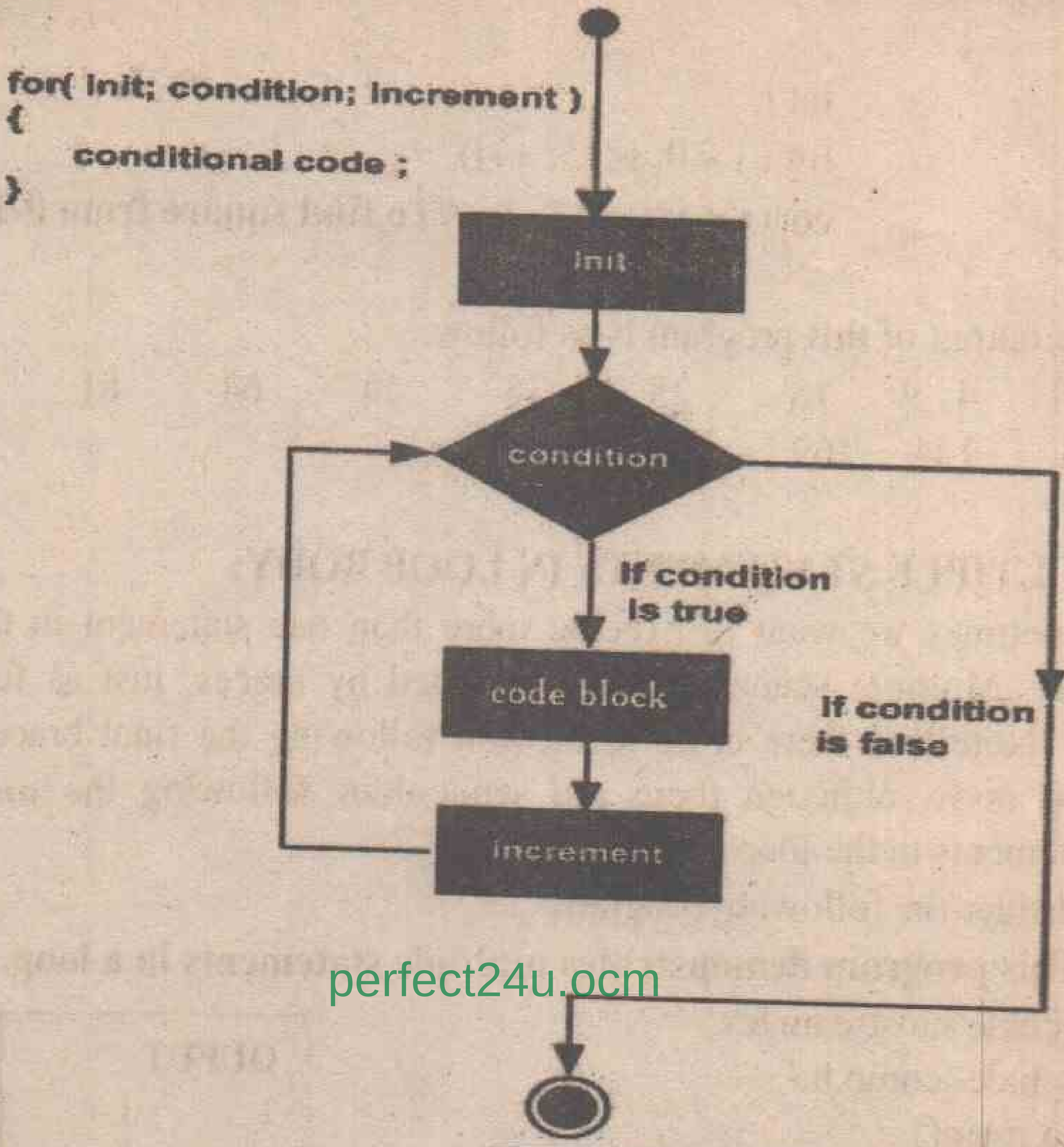
The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.

After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow Diagram



Example

```
#include <stdio.h>
int main () {
 int a;
 /* for loop execution */
 for( a = 10; a < 20; a = a + 1 )
 {
   printf("value of a: %d\n", a);
 }
 return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

THE FOR LOOP REPETITION STRUCTURE

Consider the following program.

```
// This program demonstrates the use of a  for loop & square from 0-15.
#include <iostream.h>
```

```
void main()
        {
                int j;
                for ( j = 0; j<15; ++j)
                cout << j*j << "   "; //To find square from 0-15
        }
```

The output of this program is as follow.

```
0  1   4  9   16    25    36    49    64    81    100
121    144   169   196
```

MULTIPLE STATEMENTS IN LOOP BODY:

Sometimes we want to execute more than one statement in the loop body. Multiple statements are delimited by braces, just as functions are. Note that there is no semicolon following the final brace of the loop body, although there are semicolons following the individual statements in the loop body.

Consider the following program.

// This program demonstrates multiple statements in a loop.

```
#include <iostream.h>
#include<conio.h>
void main()
{
int i;
for (i=;  i<=10;  i++)
    {
        cout << setw(4)  << i;
        int cube = i * i  * i;
        cout << setw(6) << cube << endl;
    }
getch();
return 0;
}
```

```
OUPUT
1      1
2      8
3      27
4      64
5      125
6      216
7      343
8      512
9      729
10     1000
```

In the above program, the loop body, which consists of braces enclosing several statements, is called a block of code. One important aspect of block is that a variable defined inside the block is not visible outside it . In the above program we have defined the variable cube inside the braces. Thus if we placed the statement cube = 10; after the loop body, the compiler would signal an error because the variable cube would be undefined outside the loop. The advantage of restricting the visibility of variables is that the same variable name can be used within different blocks in the same program.

Writ a program to calculate and print the factorial of any number using loop

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a;
long int fact;
cout<<"Enter any Number::::::::";
cin>>a;
fact=1;
for(a>=1;a--)
fact=fact*a;
cout<<"Factorial of "<<a<<" is ="<<fact;
getch();
}
```

Write a program to print first twenty numbers in ascending order

```
#include <iostream.h>
#include<conio.h>
void main()
{
int i;
for (i=1;  i<=20;  i++)
        {
                cout << i<<"\n";
        }
getche();
}
```

Write a program to print first twenty numbers in descending order

```
#include <iostream.h>
#include<conio.h>
void main()
{
int i;
for (i=20;  i>=1;  i--)
        {
                cout << i<<"\n";
        }
getche();
}
```

Write a program to print the even numbers between 0 to 100.

```
#include <iostream.h>
void main()
{
int i;
cout<<" Even Numbers Between 0 to 100";

for (i=0;  i<=100; i=i+2)
        {
                cout << i<<"\n";

        }
}
```

Write a program to Print Odd Numbers Between 1 to 100

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
int i;
cout<<" Odd Numbers Between 1 to 100";

for (i=1;  i<=100; i=i+2)
```

```
        {
                cout << i<<"\n"; //to find square write  cout<<i*i<<\n
        }
}
```

Write a program to find the sum of first 10 numbers.

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr()
int i;
int sum=0;
for (i=1;  i<=10; i=i+1)
        {
        sum=sum+i; // also example of accumulator counter
        }
        cout<<" the sum of first 10 numbers ="<<sum;
}
```

THE WHILE LOOP:

A **while** loop repeatedly executes a statement or group of statements as long as a given condition (logical expression) is true.

Syntax

The syntax of a **while** loop

```
While ( logical expression)
{
  statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **logical expression/condition** may be any expression. The loop iterates while the condition is true. When the condition becomes false, the program control passes to the line immediately following the loop. It contains a test expression but no initialization or increment expressions. As long as the test expression is true, the loop continues to be executed.

Consider the following program.

Write a program to print the sum of odd numbers from 1 to 10 using While Loop.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a,n;
n=1;
while(n<=10)
{
cout<<n<<endl;
n=n+2;
}
cout<<"sum="<<a;
getch( );
}
```

Multiple statements in While loop:

Consider the following program

// This program demonstrates the use of multiple statements in while loop.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
        clrscr();
         int  i =1 ;
         while (i < =6)
         {
         cout<<endl;
         cout<<setw(5)<< i<<setw(10)<<i*i;
         ++i;
         }
 getch();
}
```

```
OUTPUT
1       1
2       4
3       9
4       16
5       25
6       36
```

Write a program to print out the first ten numbers using while loop

```
#include <iostream.h>
#include<conio.h>
void main()
{
int i;
i=1;
while(i<=10)
        {
                cout << i<<"\n";
        i++;
        }
getche();
}
```

Write a program to print first ten numbers in reverse order using While loop.

```
#include <iostream.h>
#include<conio.h>
void main()
{
int i;
i=10
while ( i>=1)
        {
        cout << i<<"\n";
        i--;
        }
getche();
}
```

Write a program to Print Even Numbers Between 1 to 100 using While loop method

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
int i;
cout<<" Even Numbers Between 1 to 100";
i=0;
while (i<=100)
        {
                cout << i<<"\n";
        i=i+2;

        }
getch();
}
```

Wrte a program to Print Odd Numbers Between 1 to 100 using while loop method

```
#include <iostream.h>
void main()
{
int i;
cout<<" Odd Numbers Between 1 to 100";
i=1;
while (i<=100)
        {
                cout << i<<"\n";
        i=i+2;
        }
}
```

Write a program to find the sum of first 10 numbers using while loop method

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
int i;
int sum=0;
i=1
while ( i<=10)
        {
                sum=sum+i;
        i++;

        }
        cout<<" the sum of first 10 numbers ="<<sum;
}
```

Write a program to print the table of any number using While loop.

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int tab,res,c;
cout<<"Enter Value for Table::::";
cin>>tab;
c=1;
while(c<=10)
{
res=tab*c;
cout<<tab<<"*"<<c<<"="<<res<<endl;
c++;
}
getch();
}
```

THE DO-WHILE LOOP:

In **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop. A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

The syntax of a **do...while** loop i

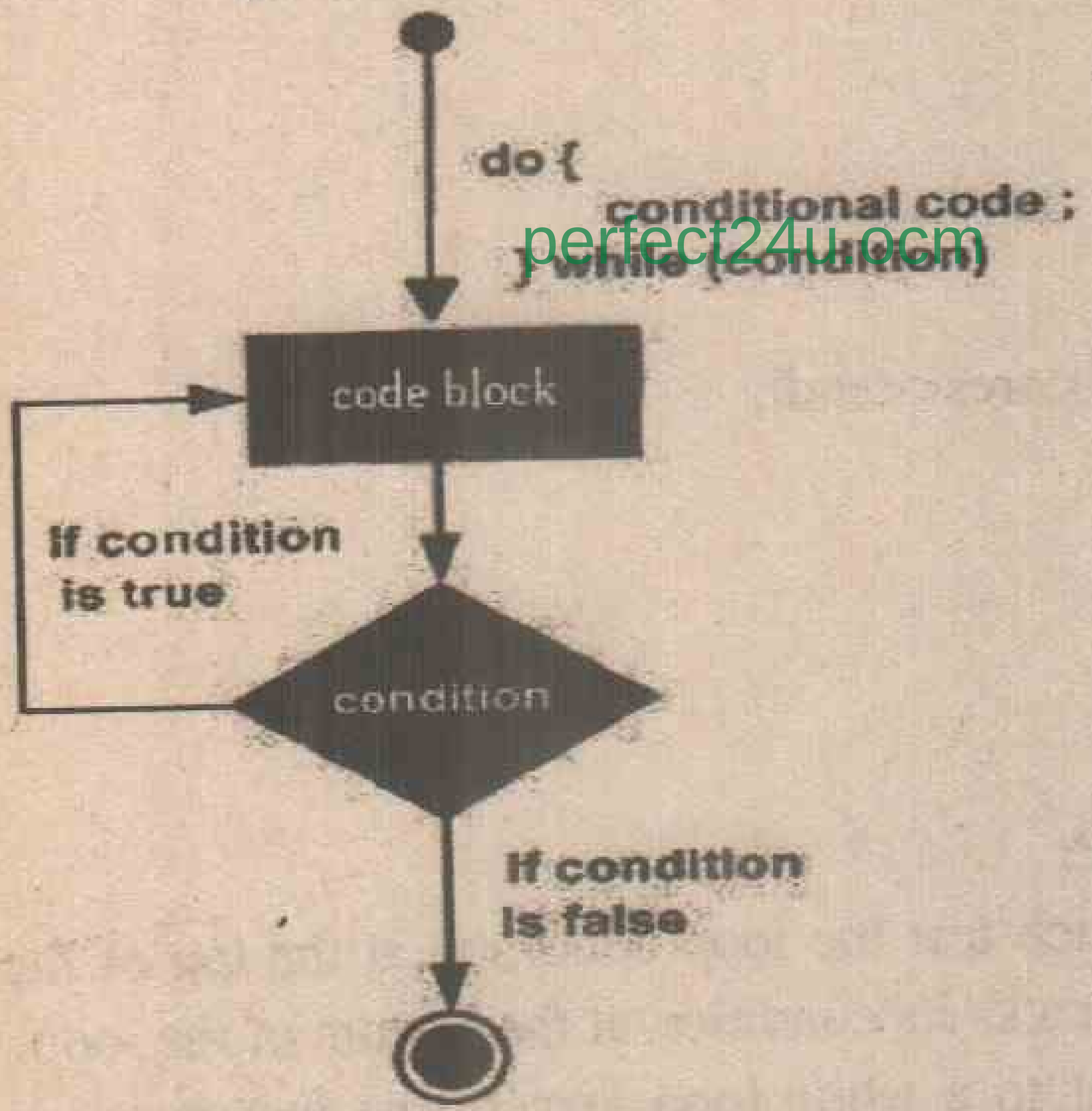
```
do
{
  statement(s);
}
while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

In a while loop, the test expression is evaluated at the beginning of the loop. If the test expression is false, the loop body would not be executed at all. In some situations this is what we want, but in some cases we want to guarantee that the loop body is executed at least once , no matter what the initial state of the test expression . when this is the case we should use the do loop , which place the test expression at the end of the loop. Consider the following flowchart.

Flow Diagram



At the end of the loop the condition is tested, if the it is true the control is transfer to the starting point of the loop, if it becomes false the control is transfer to the next statement in sequence.

The do loop has the following syntax.

Example

```
#include <stdio.h>
 int main () {
   /* local variable definition */
```

```
  int a = 10;

  /* do loop execution */
  do {
    printf("value of a: %d\n", a);
    a = a + 1;
  }while( a < 20 );
   return 0;
}
```

```
OUTPUT
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

This program demonstrates the use of Do-While loop.

```
#include <iostrem.h>
#include<conio.h>
void main()
{
clrscr();
long  a, b;
           char ch;
do
           {
           cout << "Enter the value for a  :   ";
          cin >> a;
           cout << "Enter the value for b :  ";
          cin >> b;
           cout << " Quotient is   " <<  a /b;
           cout << " , remainder  is   " << a % b;
           cout << "\ndo another ? (y/n):  ";
           cin >> ch;
           }
           while( ch = 'y');
}
```

The break statement:- This statement is used to alter the normal flow of the program execution. When it is executed in a loop or in a case structure , it causes control to transfer immediately from that structure, and then continue execution of the program in order. The common uses of this statement are to exit from a structure skipping the remainder statements of the structure. Consider the following program.

// This program demonstrates the use of break statement.

```
#include <iostream.h>
void main()
{
int       i;
for (i =1;  i <=10;   ++i)
{
if( i==6)
 break;          // skip the remaining statements in the loop when I =6.
Cout << endl<<"The value of  i   =   " << i;
 }
Cout<<endl<<The loop has been discontinu  when i =5  "<< i ;
}
```

The continue Statement:- This statement is used to ignore the remaining statements in the body of the **while, for or do/while** loop and proceeds with the next iteration of the loop.

In **while and do/while** structures the loop continuation–test is evaluated immediately after the execution of the continue statement. In the **for loop** , the increment expression is executed first , then the loop continuation test is evaluated . Consider the following program.

// This program demonstrates the use of continue statement.

```
#include <iostream.h>
void main()
        {
        int       i;
        for (i =1;  i <=10;   ++i)
       {
        if( i==6)
continue; // skip the remaining statements in the loop when I =6.
Cout << endl<<"The value of  i   =   " << i;
  }
Cout<<endl<<The loop has been discontinue  when i =5"<< i ;
}
```

Write a program to print first ten numbers using Do-while loop method

```
#include <iostream.h>
void main()
{
int i;
i=1;
        do
        {
            cout << i<<"\n";
             i++;
        }
        while(i<=10);
getche();
}
```

Write a program to print first ten numbers in reverse order using Do-while loop

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
int i;
i=10
        do
        {
        cout << i<<"\n";
        i--;
        } while ( i>=1);

getche();
}
```

Write a program to Print Even Numbers Between 1 to 100 using Do-while loop method.

```
#include <iostream.h>
void main()
{
int i;
cout<<" Even Numbers Between 1 to 100";
i=0;
        do
        {
                cout << i<<"\n";
        i=i+2;

        } while (i<=100),
}
```

Write a program to print Odd Numbers Between 1 to 100 using Do-While loop method.

```
#include <iostream.h>
void main()
{
int i;
cout<<" Odd Numbers Between 1 to 100";
i=1;
        do
        {
                cout << i<<"\n";
        i=i+2;

        } while (i<=100);
}
```

Write a frogram to find the sum of first 10 numbers using Do-While loop method.

```
#include <iostream.h>
void main()
{
int i;
int sum=0;
i=1     do
        {       sum=sum+i;
                i++;
        } while ( i<=10),
        cout<<" the sum of first 10 numbers ="<<sum;
}
```

The goto statement:

A **goto** statement in provides an unconditional jump from the **'goto'** to a labeled statement in the same function.

Syntax

The syntax for a **goto** statement is as follows –

```
goto labél;
......
..
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

LIBRARY FUNCTION

The C language is accompanied by a number of standard *library functions* which carry out various useful tasks. In particular, all input and output operations (*e.g.*, writing to the terminal) and all math operations (*e.g.*, evaluation of sines and cosines) are implemented by library functions.

In order to use a library function, it is necessary to call the appropriate *header file* at the beginning of the program. The header file informs the program of the name, type, and number and type of arguments, of all of the functions contained in the library in question. A header file is called via the preprocessor statement. Library are the following

- **Trigonometric Functions**

Trigonometric functions are available in <math.h> header file.

i) sin()

The sin() function is used to find the trigonometric sine of specified angle. The angle is given in radians as floating point value.

Syntax: sin(x); x indicates the angle in radians.

ii) cos()

The cos() function is used to find the trigonometric cos of specified angle. The angle is given in radians as floating point value.

Syntax: cos(x); x indicates the angle in radians.

iii) tan( )

The tan() function is used to find the trigonometric tangent of specified angle. The angle is given in radians as floating point value.

Syntax: tan(x); x indicates the angle in radians.

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<math.h> {
clrscr();
float n;
cout<<"enter a floating point value";
cin>>n;
cout<<"Trigonometric sine of "<<sin(n)<<endl;
cout<<"Trigonometric cos of "<<cos(n)<<endl;
cout<<"Trigonometric tan of "<<tan(n)<<endl;
getch(); }
```

Output:

```
Enter a Floating point value:  5.5
Trigonometric sine of:0.70867
Trigonometric cos of:0.70554
Trigonometric tan of:-0.995584
```

- **Arithmetic Functions.**

Arithmetic Functions are available in <math.h> header file.

i) abs()

The abs() function is used to find the absolute value of an integer .e.g. abs(-5) = 5

Syntax: abs(x); x indicates the integer whose absolute value is to be found.

ii) fabs()

The fabs() function is used to find the absolute value of a floating point number. e.g. fabs(-3.6) = 3.6

Syntax: fabs(x); x indicates the floating point number whose absolute value is to be found.

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main() {
clrscr();
```

Output:

```
Enter integer:-5
Enter float:-3.6
Absolute value of: 5
Absolute value of: 3.6
```

```
int n;
float f;
cout<<"enter an integer";
cin>>n;
cout<<"enter a float";
cin>>f;
cout<<"The absolute value of "<<n<<"is="<<abs(n)<<endl;
cout<<"The absolute value of "<<f<<"is="<<fabs(f)<<endl;
getch(); }
```

iii) log()

The log(x) calculates the natural logarithm of x

Syntax: log(); x indicates the value whose log to be found.

e.g. log(5.5) = 1.704748

iv) ceil()

The ceil() function rounds a float or double value to the next integer and returns it. e.g. ceil(13.2) = 14

Syntax: ceil(x); x It indicates the float or double value that is to be rounded.

v) floor()

The floor() function rounds a float or double value to an integer and returns it. The rounded value is not greater than the original value.e.g floor(13.2) = 13

syntax: floor(x); x It indicates the float or double value that is to be rounded.

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main() {
clrscr();
double d;
cout<<"enter a number";
cin>>d;
cout<<"the rounded value with ceil() ="<<ceil(d)<<endl;
cout<<"the rounded value with floor() ="<<floor(d)<<endl;
```

Output:

```
Enter a number: 13.2
floor: 13
ceil:  14
```

```
getch(); }
```

vi) pow()

pow( ) function in C is used to find the power of the given number. The power function is used to find the result of one integer raised to the power of second integer

"math.h" header file supports pow( ) function.

Syntax for pow( ) function in C is given below.

pow (base, exponent);

pow(3,2) = 9

Example program for pow() function in C:

```
#include <stdio.h>
#include <math.h>
int main()
{
 printf ("2 power 4 = %f\n", pow (2.0, 4.0) );
 printf ("5 power 3 = %f\n", pow (5, 3) );
 return 0;
}
```

```
OUTPUT
2 power 4 =16.000000
5 power 3 = 125.000000
```

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()   {
clrscr();
Int a,b;
cout<<"Enter first integer:"; cin>>a;
cout<<"Enter second integer:"; cin>>b;
cout<<"The result of pow(a,b) ="<<pow(a,b);
getch();}
```

Output:

```
Enter first integer:3
Enter second integer:2
Result pow(a,b):9
```

pow10( ) function

pow10( ) function in C is used to find the power of 10. The power function is used to find the result of 10 raised to the an integer

```
#include <stdio.h>
#include <math.h>
 int main()
{
  int x = 5;
  double result;
   result = pow10(x);

  printf("Ten raised to %d is %lf\n", x, result);
   return 0;
}
```

vii) sqrt() The sqrt() function is used to calculate the square root of a floating point number.

syntax: sqrt(x);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()  {
clrscr();
float n;
cout<<"Enter a floating point number:";
cin>>n;
cout<<"The square root of "<<n<<"is ="<<sqrt(n);
getch(); }
```

Output:

```
Enter a floating point number:4.5
The square root is:2.121320
```

viii) pow 10()

The pow 10() function is the power (10^P).It has the following general format:

Double pow10(int P);

For example:

pow10(5) → 100000

pow10(-2) → 0.01

ix) exp() The exp function is referred to as exponential function (e**x).The exp is the exponent of a natural log and performs the opposite of the log function.

For example exp(0) = 1

x) rnd( ) (random)

Returns integral number in the range between 0 and RAND_MAX.

Example

The following example shows the usage of rand() function.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i, n;
  time_t t;
   n = 5;
   /* Intializes random number generator */
  srand((unsigned) time(&t));
  /* Print 5 random numbers from 0 to 49 */
  for( i = 0 ; i < n ; i++ )
  {
    printf("%d\n", rand() % 50);
  }
    return(0);
}
```

```
OUTPUT
38
45
29
29
47
```

String Functions

String functions are available in <string.h> header file.

i) strlen() The word 'strlen' stand for word string length. The function is used to find the length of a string. The length includes all characters as well spaces in the string. It does not count the null character.e.g strlen(gcms)=4

Syntax: strlen(str);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()   {
clrscr();
```

```
Output:
Enter String:GCMS
String length:4
```

}

```
char str[10];
int len;
cout<<"enter a string:";
cin>>str;
len=strlen(str);
cout<<"the string contains"<<len<<"characters";
getch();  }
```

ii) strcpy()

The word 'strcpy' stand string copy. The function is used to copy one string to another including the terminating null character.

Syntax:strcpy(str1, str2);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()  {
clrscr();
        char str1[10];  char str2[10];
cout<<"str1";   cin>>str1;
cout<<"str2";   cin>>str2;
strcpy (str1,str2);
cout<<"string copy"<<strcpy(str1,str2);
getch();   }
```

Output:

```
Enter  String1:GCMS
Enter String2:College
string copy:College
```

iii) strncpy ()

The strncpy() function is used to copy one string to another including the terminating null character. It copies only specified number of characters in one string to another.

Syntax: strncpy(str1, str2, n);

iv) strrev()   The strrev() function reverses all characters in a string except the null character.

Syntax:strrev(str); e.g strrev(GCMS)=SMCG

Programming Example.

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

Output:

```
Enter  String:GCMS
string reverse:SMCG
```

```
#include<string.h>
void main()  {
clrscr();
char str[10];
cout<<"str:";
cin>>str;
strrev(str);
cout<<"string reverse:"<<str;
getch();  }
```

v) strlwr ()

The strlwr() function converts all characters of a string to lower-case.

Syntax: strlwr(str);e.g strlwr(GCMS)=gcms

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()   {
clrscr();
char str[10];
cout<<"str";
cin>>str;
strlwr(str);
cout<<"string lower"<<str;
getch();  }
```

Output:

```
Enter  String:GCMS
string lower:gcms
```

vi) strupr()

The strupr() function converts all characters of a string to uppercase.

Syntax: strupr(str);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()  {
clrscr();
char str[10];
```

Output:

```
Enter  String:gcms
string upper:GCMS
```

```
cout<<"Enter string:";
cin>>str;
strupr(str);
cout<<"string upper:"<<str;
getch(); }
```

vii) Strcat()

The strncat() function is used to append the specified number of characters of one string to the end of second string.

Syntax: strncat(str1, str2, n);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()  {
clrscr();
char str1[10];
char str2[10];
cout<<"Enter str1:";
cin>>str1;
cout<<"Enter str2:";
cin>>str2;
strcat (str1,str2);
cout<<"string concatenating"<<strcat(str1,str2);
getch(); }
```

Output:

```
Enter str1: GCMS
Enter str2: College
 String concatenating: GCMSCollege
```

viii) strncat()

The strncat() function is used to append the specified number of characters of one string to the end of second string.

Syntax: strncat(str1, str2, n);

Programming Example.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()  {
clrscr();
char str1[10];
```

Output:

```
Enter number  : 3
Enter str1: GCMS
Enter str2: College
 String concatenating: GCMSCol
```

```
char str2[10];

int n;

cout<<"Enter number :";

cin>>n;

cout<<"Enter str1:";

cin>>str1;

cout<<"Enter str2:";

cin>>str2;

strncat (str1,str2,n);

cout<<"string concatenating :"<<strncat(str1,str2,n);

getch( );

}
```

Khyber Pakhtunkhwa Board Of Technical Education Peshawar
Diploma in Information Technology
(Part-1)
1st Term Examination 2016
Paper: Computer Programming C/C++

Time Allowed: 03 Hours Max. Marks-50

Note: Attempt any five questions. All questions carry equal marks.

Q.1 Differentiate any five pairs from given.
i) Signed and Unsigned Integer
ii) Strlwr () and Strupr()
iii) Break and Continue
iv) Prefix and Post fix Operator
v) Dectoration time value assignment and run time value assignment
vi) C and C++

Q.2 write short notes on any two from the following with example.
i. For loop
ii. Processor Directive
iii. Arithmetic Operator

Q.3 Write body structure for any five from the given.
i. Nested If-else statement
ii. Shile Loop
iii. Goto Statement.
Iv. Declaration of user define function
vi. Printf
vii. C/C++ program structure

Q.4 Write output for any four from the given logic.

```
i. int a=5, b=10;
 Printf("b%a=%dln",b%a);
```

```
ii. int n=10
    char =”*”;
    printf("%d",n);
    Printf("%c",ch);
```

```
iii. int I,j;
  for(i=1;i<5;i++)
     {
  For(j=1;j<=I;j++)
  Printf ("*");
  Printf ("\n")
     }
```

```
iv) Printf("%f",ceil(9.9));
v) Printf("%f",pow(5,2));
```

Part-B

Q.5 Write program that take a number from user and shows whether it is odd or even using if-else statement.

Q.6 Elaborate while and do while loop detail with one example.

Q.7 Elaborate user defines words and reserve words with example.

Q.8 Elaborate input and output statements. also write down two functions of each statement

Khyber Pakhtunkhwa Board Of Technical Education Peshawar
Diploma in Information Technology
(Part-1)
1st Term Examination 2016
Paper: Computer Programming C/C++

Time Allowed: 03 Hours Max. Marks-50

Note: Attempt any five questions. All questions carry equal marks.

Q.1. a) What is the difference between C and C++
b) Write the weakness of C
c) Explain each part of the following .
i) int varint ii) char varch =65;

Q.2: Write a program to find out the factorial of a number 6.

Q.3: Write a program to display the square of even numbers of range with 1 to 20

Q.4: a) Describe the relational and conditional operators with proper examples
b) What will be the output of the following program?

```
Main ()
        {
                Int K = 35;
                Printf( "%d%d%d", k==35, k=50, k>40);
        }
```

Q.5: Explain the following functions

i) Scanf ( )
ii) Printf ( )
iii) Gets ( )
iv) Puts ( )

Q.6: What is the difference between For loop and While loop.

Q.7: Explain with examples the following fuctions.

i) ABS ( ) ii) SQRT ( )
iii) Strlen ( ) iv) exp ( )
v) pow ( )

Q.8: Write a program to display the following outputs using a loop statement

```
1
2
1 2 3
1 2 3 4
```

****