CHAPTER – 9

C++

1. WRITE ABOUT THE BINARY OPERATORS USED IN C++?

ARITHMETIC OPERATORS:

☑ Arithmetic operators perform simple arithmetic operations like addition, subtraction, multiplication, division etc.,

OPERATOR	OPERATION	EXAMPLE
+	Addition	10 + 5 = 15
-	Subtraction	10 – 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2 (Quotient of the division)
%	Modulus (Reminder of the division)	10 % 3 = 1(Remainder of the division)

RELATIONAL OPERATORS:

☑ Relational operators are used to determine the relationship between its operands.

☑ When the relational operators are applied on two operands, the result will be a Boolean value i.e 1 or 0 to represents True or False respectively.

☑ C++ provides six relational operators.

OPERATOR	OPERATION	EXAMPLES	
>	Greater than	a > b	4>5 TRUE
<	Less than	a < b	4<5 FALSE
>=	Greater than or equal to	a >= b	4>=5 FALSE
<=	Less than or equal to	a <= b	4<=5 TRUE
==	Is Equal to	a == b	4==5 FALSE
!=	Not equal	a != b	4!=5 TRUE

LOGICAL OPERATORS:

☑ A logical operator is used to evaluate logical and relational expressions.

☑ The logical operators act upon the operands that are themselves called as **Logical Expressions.**

☑ C++ provides three logical operators.

Operator	Operation	Description
&&	AND	It combines two different relational expressions in to one. It returns 1 (True), if both expression are true, otherwise it returns 0 (false).

||	**OR**	It combines two different relational expressions in to one. It returns 1 (True), if either one of the expression is true. It returns 0 (false), if both the expressions are false.
!	**NOT**	It works on a single expression / operand. It simply negates or inverts the truth value. i.e., if an operand / expression is 1 (true) then this operator returns 0 (false) and vice versa

BITWISE OPERATORS:

- ☑ Bitwise operators work on each bit of data and perform bit-by-bit operation.
- ☑ There are three kinds of bitwise operators,
  - ✓ **Logical bitwise operators,**
  - ✓ **Bitwise shift operators, &**
  - ✓ **One's compliment operators.**

ASSIGNMENT OPERATOR:

- ☑ It is used to assign a value to a variable which is on the left hand side.
- ☑ = (equal to) is commonly used as the assignment operator.
- ☑ It is also a binary operator.

2. EXPLAIN THE BITWISE OPERATORS WITH EXAMPLE?

- ☑ Bitwise operators work on each bit of data and perform bit-by-bit operation.
- ☑ There are three kinds of bitwise operators,
  - ✓ **Logical bitwise operators,**
  - ✓ **Bitwise shift operators, &**
  - ✓ **One's compliment operators.**

LOGICAL BITWISE OPERATORS:

OPERATOR	OPERATOR NAME	DESCRIPTION
&	Bitwise AND (Binary AND)	It will return 1 (True) if both the operands are having the value 1 (True); Otherwise, it will return 0 (False)
|	Bitwise OR (Binary OR)	It will return 1 (True) if any one of the operands is having a value 1 (True); It returns 0 (False) if both the operands are having the value 0 (False)

^	Bitwise Exclusive OR (Binary XOR)	It will return 1 (True) if only one of the operand is having a value 1 (True).If both are True or both are False, it will return 0 (False).

TRUTH TABLE FOR BITWISE OPERATORS (AND, OR, XOR)

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

BITWISE SHIFT OPERATORS:

- ☑ There are two bitwise shift operators in C++, **Shift left** (<<) and **Shift right** (>>).
- ☑ **Shift left (<<)–** The value of the left operand is moved to left by the number of bits specified by the right operand. Right operand should be an unsigned integer.
- ☑ **SHIFT RIGHT (>>)–** The value of the left operand is moved to right by the number of bits specified by the right operand. Right operand should be an unsigned integer.

BITWISE ONE'S COMPLIMENT OPERATOR:

- ☑ The bitwise One's compliment operator **~(Tilde)**,inverts all the bits in a binary pattern, that is, all 1's become 0 and all 0's become 1. This is a unary operator.

3. DESCRIBE THE I/O OPERATOR WITH EXAMPLES?

INPUT OPERATOR:

- ☑ C++ provides the operator >> to get input. It extracts the value through the keyboard and assigns it to the variable on its right. Hence, it is called as **"Stream Extraction"** or **"Get From"** operator.
- ☑ It is a binary operator i.e., it requires two operands. The first operand is the pre-defined identifier cin that identifies keyboard as the input device. The second operand must be a variable.
- ☑ To receive or extract more than one value at a time, >> operator should be used for each variable. This is called **Cascading of Operator.**

EXAMPLE:

cin >> a;	Pre-defined object cin extracts a value typed on keyboard and stores it in variable **a.**
cin >>x >> y;	This is used to extract two values. cin reads the first value and immediately assigns that to variable x; next, it reads the second value which is typed after a space and assigns that to y. Space is used as a separator for each input.

OUTPUT OPERATOR:

- ☑ C++ provides << operator to perform output operation. The operator << is called the **"Stream Insertion"** or **"Put To"** operator. It is used to send the strings or values of the variables on its right to the object on its left. << is a binary operator.
- ☑ The first operand is the pre-defined identifier cout that identifies monitor as the standard output object. The second operand may be a constant, variable or an expression.
- ☑ To send more than one value at a time, << operator should be used for each constant/variable/expression. This is called **Cascading of Operator**.

EXAMPLE:

cout << "Welcome";	Pre-defined object cout sends the given string "Welcome" to screen.
cout << "The sum = " << sum;	First, cout sends the string "The Sum = "to the screen and then sends the value of the variable sum.
cout <<"\n The Area: " <<3.14*r*r;	First, cout sends everything specified within double quotes except \n to the screen, and then it evaluates the expression 3.14*r*r and sends the result to the screen.
cout << a + b ;	cout sends the sum of a and b to the monitor.

4. WHAT ARE THE BASIC ELEMENTS OF THE C++ PROGRAM?

1. // (Double Slash) in C++ program to Print a String
This is a comment statement. Any statement that begins with // are considered as comments. Compiler does not execute any comment as part of the program and it simply ignores. If we need to write multiple lines of comments, we can use /* ……. */.
2. # include <iostream>
Usually all C++ programs begin with include statements starting with a # (hash / pound). The symbol # is a directive for the preprocessor. These statements are processed before the compilation process begins. **#include <iostream> statement tells the compiler's preprocessor to include the header file "iostream" in the program.** The header file iostream should be included in every C++ program to implement input / output functionalities. In simple words, **iostream header file contains the definition of its member objects cin and cout. If you fail to include iostream in your program, an error message will occur on cin and cout; and we will not be able to get any input or send any output.**
3. using namespace std;
The line using namespace std; tells the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. Namespaces provide a method of preventing name conflicts in large projects. It is a new concept introduced by the ANSI C++ standards committee.

4. int main ( )
C++ program is a collection of functions. Every C++ program must have a main function. The main( ) function is the starting point where all C++ programs begin their execution. Therefore, the executable statements should be inside the main( ) function.
5. { **6.** cout<<"Welcome to Programming in C++"; **7.** return 0; **8.** }
The statements between the curly braces are executable statements. This is called as a **Block of Code.** In line 6, cout simply sends the string constant "Welcome to Programming in C++" to the screen. Every executable statement must terminate with a semicolon. In line 7, return is a keyword which is used to return the value what you specified to a function. It will return 0 to main( ) function.

5. WHAT ARE THE MAIN STEPS TO CREATE AND EXECUTE THE C++ PROGRAM?

Creating Source code

Creating includes typing and editing the valid C++ code as per the rules followed by the C++ Compiler.

Saving source code with extension .cpp

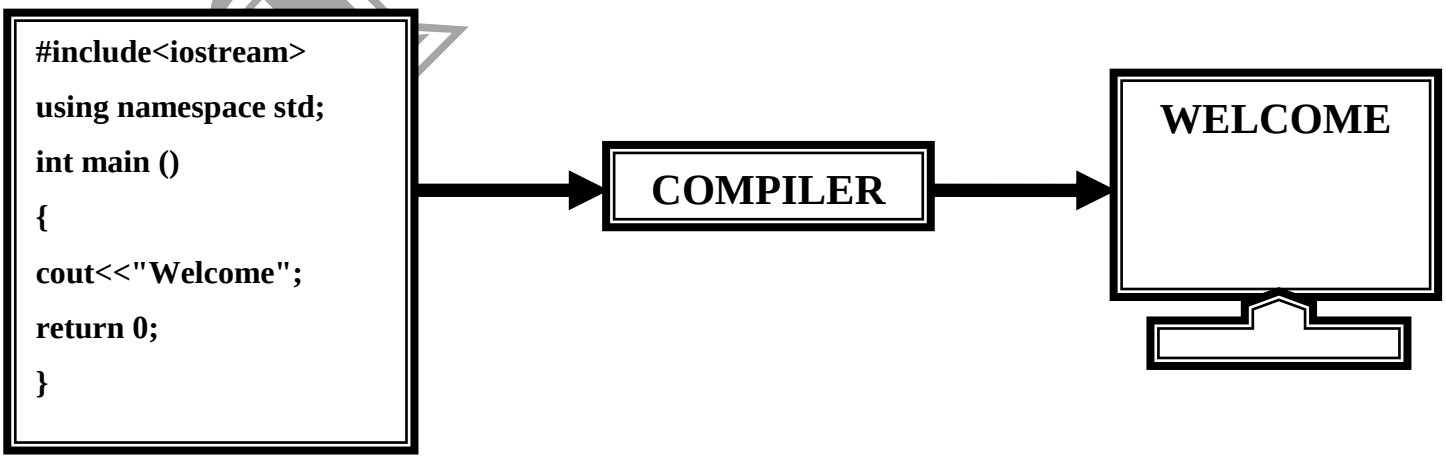
After typing, the source code should be saved with the extension .cpp

Compilation

In compilation, compiler links the library files with the source code and verifies each and every line of code. If any mistake or error is found, it will inform yDou to make corrections. If there are no errors, it translates the source code into machine readable object file with an extension .obj

Execution

This is the final step of construction of a C++ Program. In this stage, the object file becomes an executable file with extension .exe. Once the program becomes an executable file, the program has an independent existence. This means, you can run your application without the help of any compiler or IDE.



6. WHAT ARE THE DIFFERENT TYPES OF ERRORS IN DEV C++?

TYPES OF ERROR	DESCRIPTION
Syntax Error	Syntax is a set of grammatical rules to construct a program. Every programming language has unique rules for constructing the source code. Syntax errors occur when grammatical rules of C++ are violated. Example: if you type as follows, C++ will throw an error. **cout << "Welcome to Programming in C++"** As per grammatical rules of C++, every executable statement should terminate with a semicolon. But, this statement does not end with a semicolon.
Semantic Error	A Program has not produced expected result even though the program is grammatically correct. It may be happened by wrong use of variable / operator / order of execution etc. This means, program is grammatically correct, but it contains some logical error. So, Semantic error is also called as **"Logic Error"**.
Run-time error	A run time error occurs during the execution of a program. It is occurs because of some illegal operation that takes place. For example, if a program tries to open a file which does not exist, it results in a run-time error

CHAPTER – 10

FLOW OF CONTROL

1. WRITE THE SYNTAX FOR

I. IF NESTED INSIDE IF PART:

```
if (expression – 1)
{
        if (expression – 2)
        {
        True_Part_Statements;
        }
        else
        {
        False_Part_Statements;
        }
}
else
{
Body of else part;
}
```

II. IF NESTED INSIDE ELSE PART:

```
if (expression-1)
{
body of true part;
}
else
{
        if (expression)
        {
        True_Part_Statements;
        }
        else
        {
        False_Part_Statements;
        }
```

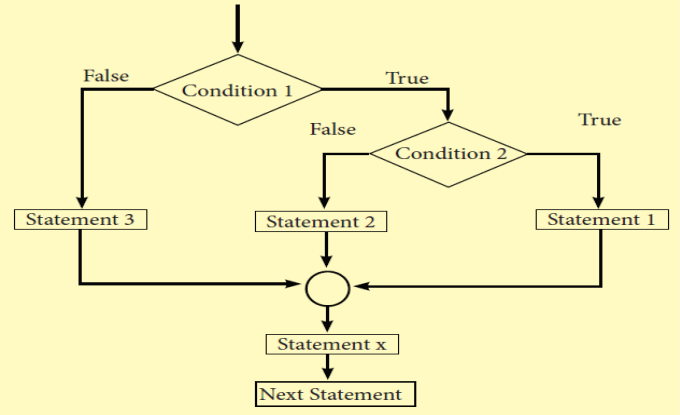
```
}
```

III. IF NESTED INSIDE BOTH IF AND ELSE PART:

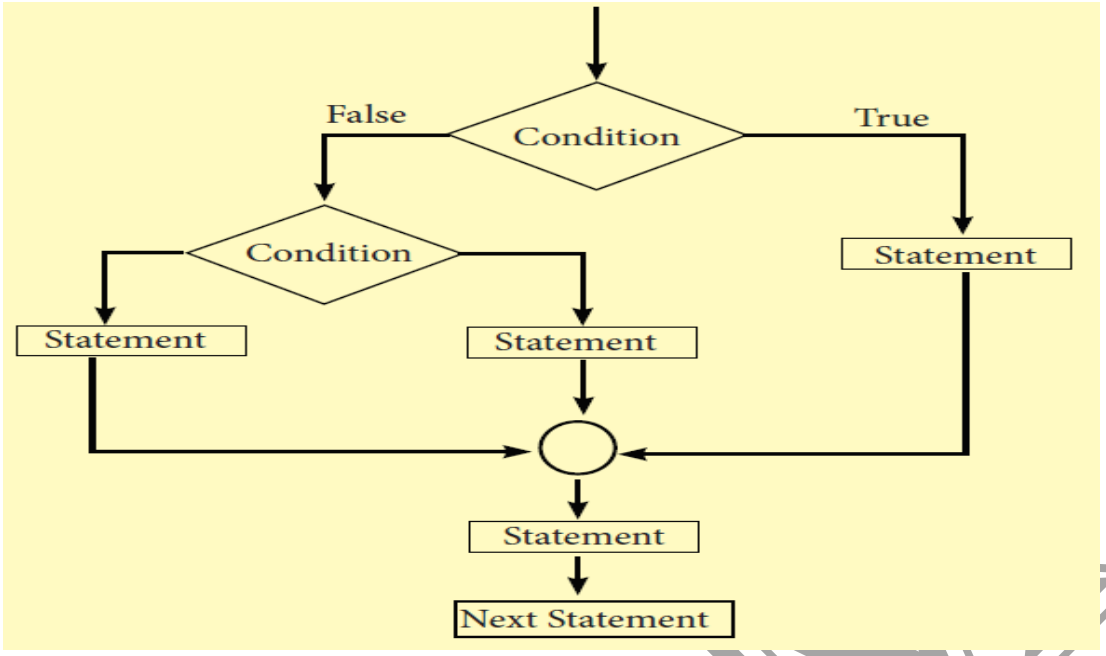
```
if (expression)
{
    if (expression)
    {
    True_Part_Statements;
    }
    else
    {
    False_Part_Statements;
    }
}
else
{
    if (expression)
    {
    True_Part_Statements;
    }
    else
    {
    False_Part_Statements;
    }
}
```

2. DRAW THE FLOWCHART FOR

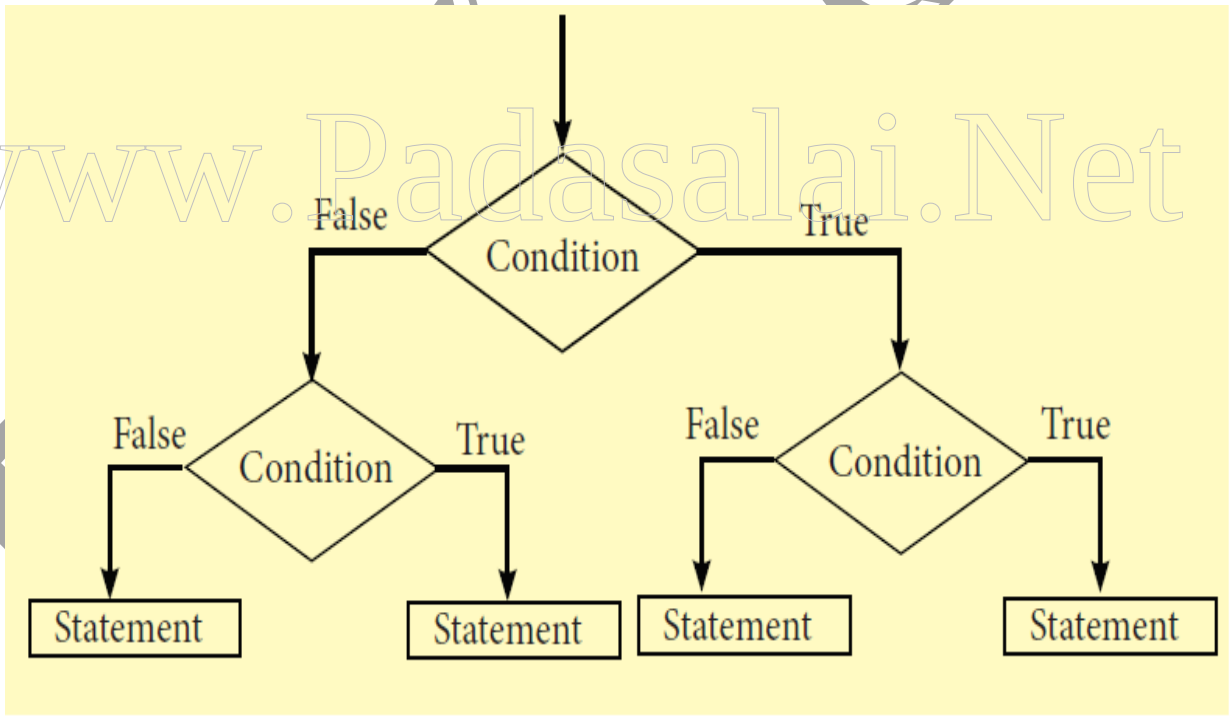
I. IF NESTED INSIDE IF PART



II. IF NESTED INSIDE ELSE PART:



III. IF NESTED INSIDE BOTH IF AND ELSE PART:



3. WRITE THE SYNTAX AND FLOWCHART FOR IF-ELSE-IF LADDER?

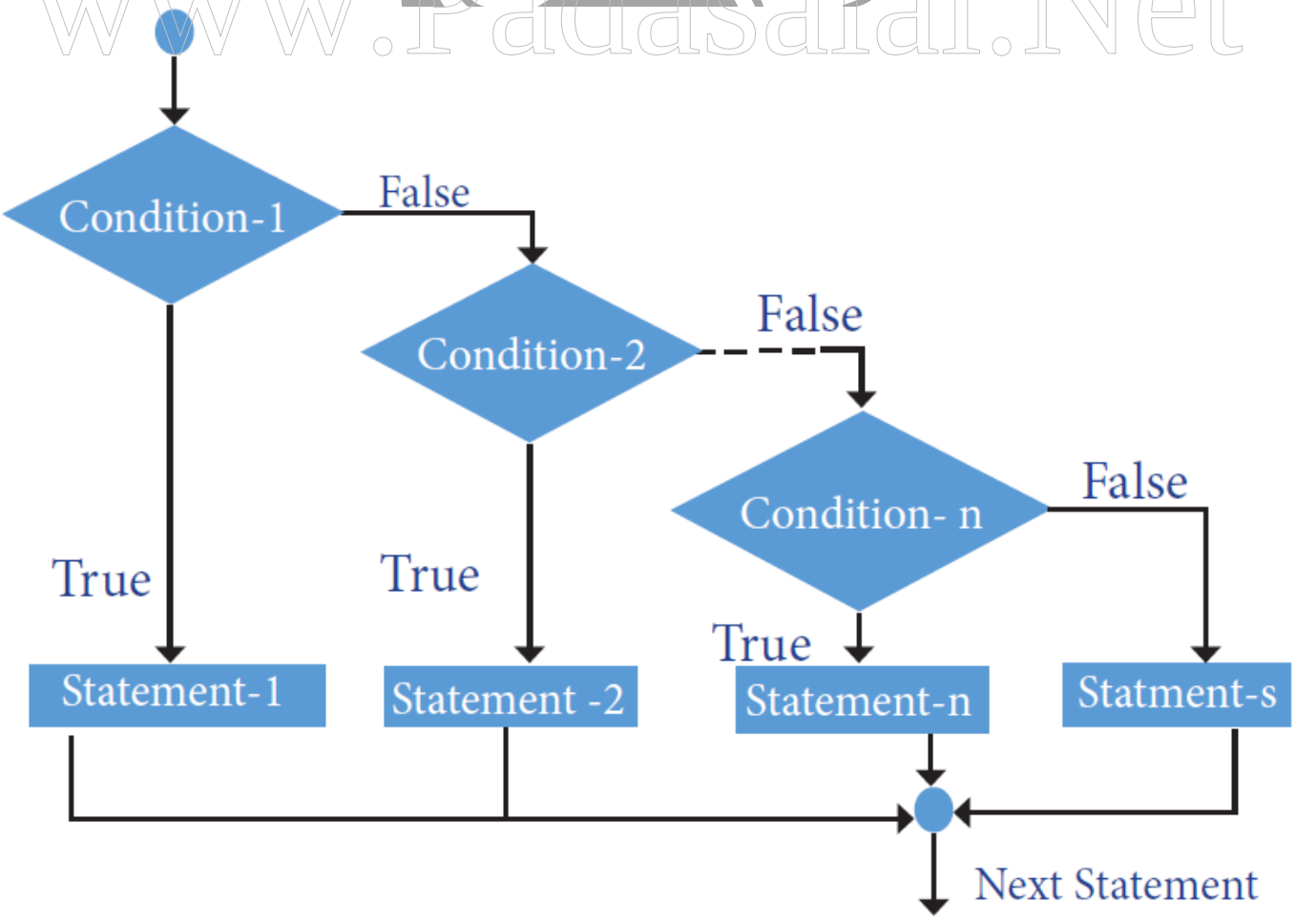
➲ The if-else ladder is a multi-path decision making statement. In this type of statement 'if' is followed by one or more else if statements and finally end with an else statement.

SYNTAX:

if (expression 1)

```
{
Statemet-1
}
    else  if( expression 2)
    {
    Statemet-2
    }
          else if ( expression 3)
          {
          Statemet-3
          }
                else
                {
                Statement-4
                }
```

➲ When the expression becomes true, the statement associated with block is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



4. EXPLAIN SWITCH STATEMENT WITH SYNTAX AND EXAMPLE PROGRAM?

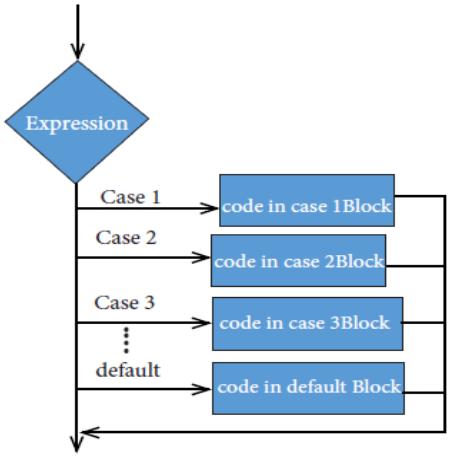
DEFINITION

- This is a multiple branching statement where, based on a condition, the control is transferred to one of the many possible points.
- The switch statement replaces multiple if-else sequences.

SYNTAX

```
switch (expression)
{
case constant1:
        statement – 1;
        break;
case constant2:
        statement – 2;
        break;
case constant3:
        statement – 3;
        break;
case constant4:
        statement – 4;
        break;
default:
        statement – x;
}
```

- The expression is evaluated and if its value matches against the constant value specified in one of the case statements, that respective set of statements are executed. Otherwise, the statements under the default option are executed.



PROGRAM:

```
#include <iostream>
using namespace std;
int main()
{
int n;
cout << "\n Enter week day number: ";
cin >> n;
switch (num)
        {
        case 1 :
        cout << "\n Sunday";
        break;
        case 2 :
        cout << "\n Monday";
        break;
        case 3 :
        cout << "\n Tuesday";
        break;
        case 4 :
        cout << "\n Wednesday";
        break;
        case 5 :
        cout << "\n Thursday";
        break;
        case 6 :
        cout << "\n Friday";
        break;
        case 7 :
        cout << "\n Saturday";
        break;
        default:
        cout << "\n Wrong input";
        }
}
```

OUTPUT:

Enter week day number: 6

Friday

5. DEFINE SWITCH STATEMENT. EXPLAIN THE RULES FOR SWITCH STATEMENT?

DEFINITION:

- This is a multiple branching statement where, based on a condition, the control is transferred to one of the many possible points.
- The switch statement replaces multiple if-else sequences.

RULES FOR SWITCH STATEMENT:

- The expression in the switch statement should result in a constant value, otherwise it is invalid.
- **Duplicate** case values are not allowed.
- The **default** statement is **optional**.
- The **Break** statement is used inside the switch to **terminate** a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
- Nesting of switch statements is also allowed.

6. DESCRIBE THE KEY DIFFERENCES BETWEEN IF – ELSE AND SWITCH?

IF – ELSE	SWITCH
If – else statement decide whether to execute the statements inside if block or under else block.	Switch statement decides which case to execute.
If-else statement uses multiple statements for multiple choices.	Switch statement uses single expression for multiple choices.
If-else statement checks for equality and logical expression.	Switch checks only for equality.
If statement evaluates integer, character, pointer or floating-point type or Boolean type.	Switch statement evaluates only character or a integer data type.
In If – Else statement either if statement or else statement is executed.	In Switch Statement it decides which case to execute.
In If – Else statement, if the condition is FALSE, the else statement is executed.	In Switch Statement, if the condition is FALSE, the default statement is executed.

It is difficult to edit if-else statements as it is tedious to trace where the correction is required.	It is easy to edit switch statements as they are easy to trace.

7. DESCRIBE THE NESTED SWITCH WITH SYNTAX AND EXAMPLE?

➲ When a switch is a part of the statement sequence of another switch, then it is called as **Nested Switch statement.** The inner switch and the outer switch constant may or may not be the same.

SYNTAX

```
switch (expression)
{
case constant 1:
statement(s);
break;
    switch(expression)
    {
    case constant 1:
    statement(s);
    break;
    case constant 2:
    statement(s);
    break;
    ------
    ------
    ------
    default :
    statement(s);
    }
case constant 2:
statement(s);
break;
------
------
------
default :
statement(s);
}
```

8. WHAT ARE THE PARTS OF THE LOOP IN THE ITERATION STATEMENT?

- Every loop has four elements. They are,
  - ✓ Initialization expression
  - ✓ Test expression
  - ✓ Update expression
  - ✓ The body of the loop

INITIALIZATION EXPRESSION(S):

- The control variable(s) must be initialized before the control enters into loop.
- The initialization of the control variable takes place under the initialization expressions.
- It is executed only once in the beginning of the loop.

TEST EXPRESSION:

- The test expression is an expression or condition whose value decides whether the loop-body will be executed or not.
- If the expression evaluates to true (i.e., 1), the body of the loop executed, otherwise the loop is terminated.
- In an entry-controlled loop, the test-expression is evaluated before the entering into a loop.
- In an exit-controlled loop, the test-expression is evaluated before exit from the loop.

UPDATE EXPRESSION:

- It is used to change the value of the loop variable.
- This statement is executed at the end of the loop after the body of the loop is executed.

THE BODY OF THE LOOP:

- A statement or set of statements forms a body of the loop that are executed repetitively.
- In an entry-controlled loop, first the test-expression is evaluated and if it is nonzero, the body of the loop is executed otherwise the loop is terminated.
- In an exit-controlled loop, the body of the loop is executed first then the test-expression is evaluated. If the test-expression is true the body of the loop is repeated otherwise loop is terminated

9. EXPLAIN FOR LOOP WITH THE SUITABLE PROGRAM?

FOR LOOP

- For Loop is an **Entry – Controlled Loop.**
- It contains **three** different statements. They are Initialization, Test Expression, and Update Expression separated by semi – colon.

SYNTAX

```
for (initializations; test expression ; update expression)
{
```

Body of the Loop;

}

- The Initialization is done only once.
- Next, The Test Expression or Condition is checked. If the Test Expression or Condition is TRUE, the body of for loop and the Update Expression is executed again and again.
- When the Test Expression is FALSE, the Statement – X is executed.

PROGRAM

```
#include<iostream>
using namespace std;
int  main()
{
int i;
for(i=1;i<=5;i++)
{
cout<<i<<"\n";
}
}
```

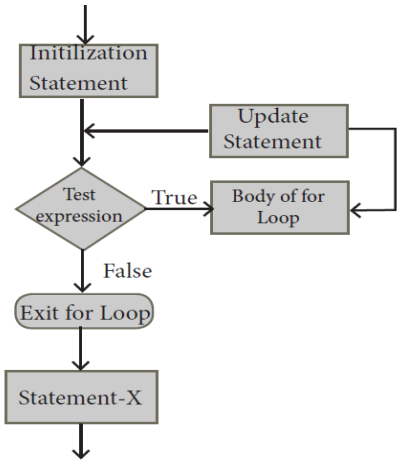
OUTPUT

1
2
3
4
5

EXPLANATION OF THE PROGRAM

- The Control Variable **i** is initialized to **1**.
- The Test Expression **i<=5** is checked i.e., **1<=5** is checked. The condition becomes **TRUE**, then the value of **i** is **printed** and the value of i is updated as **++i** (i=i+1).
- Again, the condition **2<=5, 3<=5, 4<=5, 5<=5** is checked, the value of i is printed again and again.
- When the Test Expression **6<=5** is checked, it becomes **FALSE**. The Loop is **terminated**.

FLOW CHART:



10.EXPLAIN WHILE LOOP? EXPLAIN WITH AN EXAMPLE?

WHILE LOOP

DEFINITION

- While Loop is an **Entry – Controlled Loop.**
- The Test Expression is checked **before** entering into the Body of the Loop.

SYNTAX

```
while (Test Expression)
{
Body of the Loop;
}
Statement – X;
```

- When the Condition or Test Expression is evaluated to **TRUE**, the body of the loop is executed **again and again**.
- When the Condition or Test Expression is evaluated to **FALSE**, the **Statement – X** is executed.

PROGRAM

```
#include<iostream>
using namespace std;
int  main()
{
int i=1;
while(i<=5)
{
cout<<i<<"\n";
++i;
```

```
}
system("pause");
return 0;
}
```

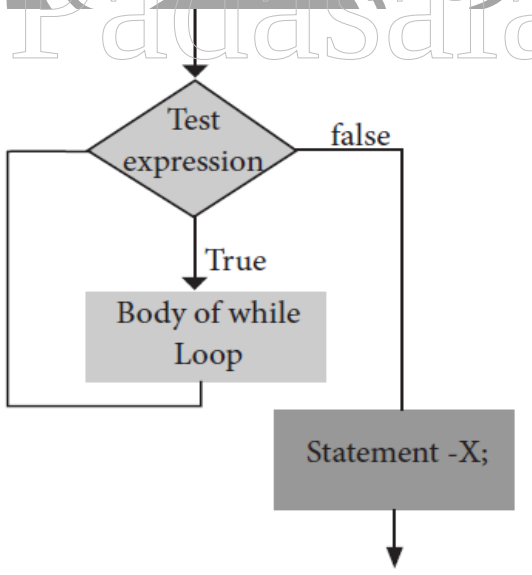
OUTPUT

1
2
3
4
5

EXPLANATION OF THE PROGRAM

- The Control Variable **i** is initialized to **1**.
- The Test Expression **i<=5** is checked i.e., **1<=5** is checked. The condition becomes **TRUE**, then the value of **i** is **printed** and the value of i is updated as **++i** (i=i+1).
- Again, the condition **2<=5, 3<=5, 4<=5, 5<=5** is checked, the value of i is printed again and again.
- When the Test Expression **6<=5** is checked, it becomes **FALSE**. The Loop is **terminated**.

FLOW CHART:



11.WHAT IS DO-WHILE LOOP? EXPLAIN WITH AN EXAMPLE?

DO – WHILE LOOP

DEFINITION

- Do – While Loop is an **Exit – Controlled Loop.**
- The Test Expression is checked **after** executing the Body of the Loop.

SYNTAX

```
do
{
Body of the Loop;
}
while (Test Expression);
```

- The body of the loop is executed first and then the Condition or Test Expression is evaluated. If the condition becomes **TRUE**, the body of the loop is executed **again and again**.
- When the Condition or Test Expression is evaluated to **FALSE**, the **Statement – X** is executed.

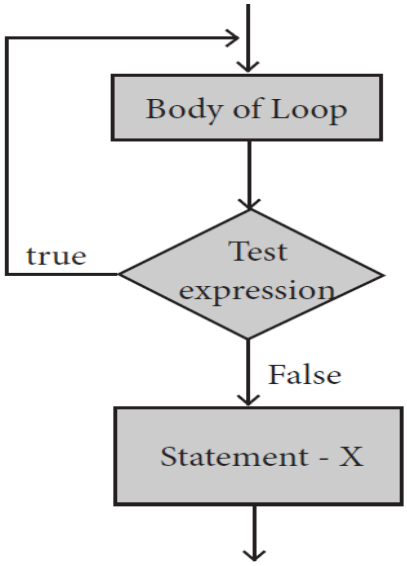
PROGRAM

```
#include<iostream>
using namespace std;
int  main()
{
int i=1;
do
{
cout<<i<<"\n";
i++;
}
while(i<=5);
system("pause");
return 0;
}
```

OUTPUT

```
1
2
3
4
5
```

FLOW CHART:



EXPLANATION OF THE PROGRAM

- The Control Variable **i** is initialized to **1**.
- The value of **i** is **printed** and the value of i is updated as **++i** (i=i+1).
- The Test Expression **i<=5** is checked i.e., **1<=5** is checked. When the condition becomes **TRUE**, then the above statement is executed.
- Again, the condition **2<=5, 3<=5, 4<=5, 5<=5** is checked, the value of i is printed again and again.
- When the Test Expression **6<=5** is checked, it becomes **FALSE**. The Loop is **terminated**.

12. EXPLAIN IF AND IF – ELSE STATEMENT WITH AN EXAMPLE?

IF STATEMENT:

- If statement evaluates a condition, if the condition is true then a true-block (a statement or set of statements) is executed, otherwise the true-block is skipped.

SYNTAX:

```
if (expression)
        True-block;
Statement – X;
```

EXPLANATION:

- The Condition or Test Expression should be given inside the Parenthesis. If the expression is true (nonzero) then the true-block is executed and followed by statement-x are also executed, otherwise, the control passes to statement-x.

PROGRAM:

```
#include <iostream>
using namespace std;
int main()
{
int age;
cout<< "\n Enter your age: ";
cin>> age;
if(age>=18)
        {
        cout<< "\n You are eligible for voting";
        }
cout<< "This statement is always executed.";
system("pause");
return 0;
}
```

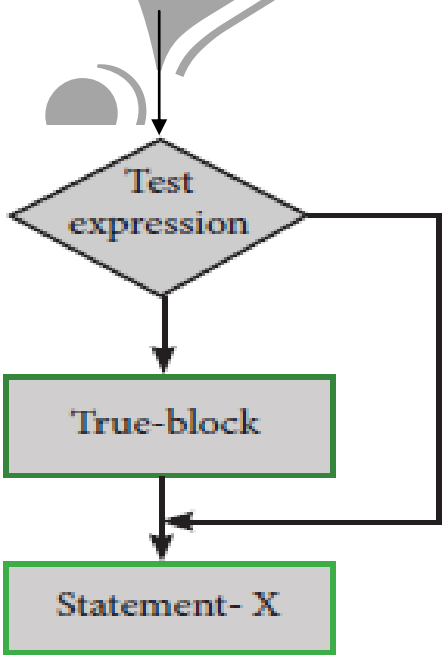
OUTPUT

Enter your age: 23

You are eligible for voting

This statement is always executed.

FLOWCHART:



IF – ELSE STATEMENT:

- In if-else statement, first the expression or condition is evaluated either true of false.
- If the result is true, then the statement inside true-block is executed and false-block is skipped.
- If the result is false, then the statement inside the false-block is executed i.e., the true-block is skipped.

SYNTAX

```
if ( expression)
        {
        True-block;
        }
else
        {
        False-block;
        }
STATEMENT – X;
```

PROGRAM:

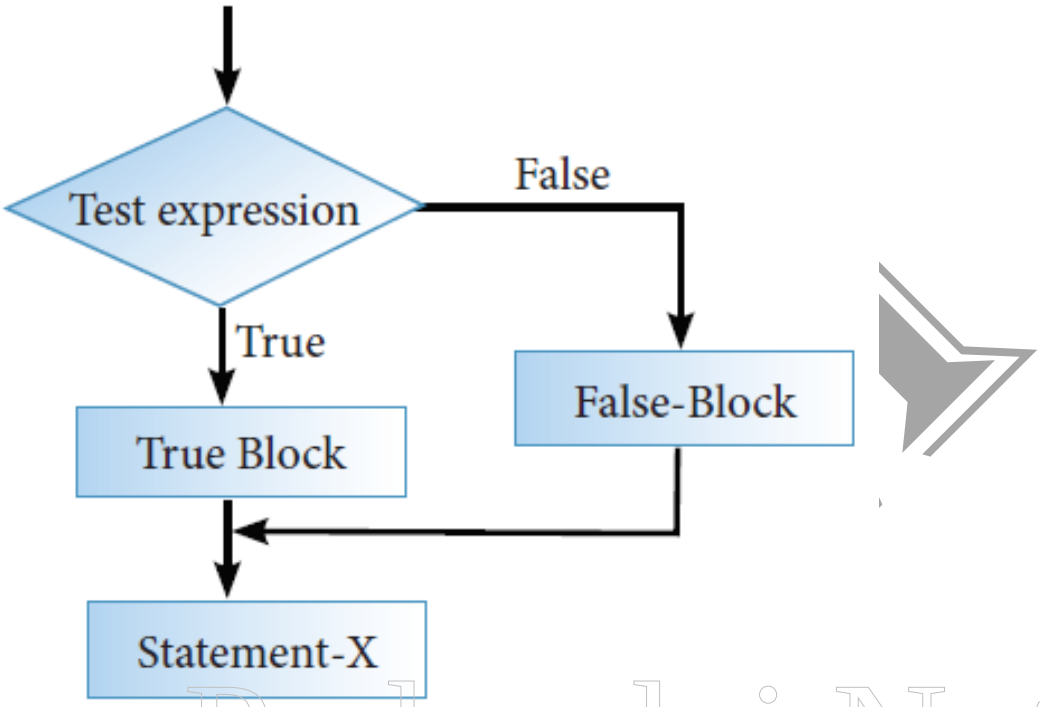
```
#include <iostream>
using namespace std;
int main()
{
int num, rem;
cout<< "\n Enter a number: ";
cin>>num;
rem = num % 2;
if (rem==0)
        cout<< "\n The given number" <<num<< " is Even";
else
        cout<< "\n The given number "<<num<< " is Odd";
system(“pause”);
return 0;
}
```

OUTPUT:

Enter number: 10

The given number 10 is Even

FLOWCHART:



CHAPTER – 11

FUNCTIONS

1. EXPLAIN CALL BY VALUE METHOD WITH EXAMPLE?

DEFINITION:

- In call by value method, any change in the formal parameter is not **reflected** back to the actual parameter.

EXPLANATION:

- This method copies the value of an actual parameter into the formal parameter of the function.
- In this case, changes made to formal parameter within the function will have **no effect** on the actual parameter.

PROGRAM:

```
#include<iostream>
using namespace std;
void display(int x)
{
        cout<<"\n The Value of x="<<x*x;
}
int main()
{
        int a;
        cout<<"\n Call by Value";
        cout<<"\n Enter the Value for a=";
        cin>>a;
        cout<<"\nThe Value of a="<<a;
        display(a);
        cout<<"\n Back to main function= "<<a;
        system("pause");
        return 0;
}
```

OUTPUT:

Call by Value

Enter the Value for a=2

The Value of x=4

Back to main function=2

2. EXPLAIN CALL BY REFERENCE METHOD WITHEXAMPLE?

DEFINITION:

- In call by reference method, any change in the formal parameter is **reflected** back to the actual parameter.

EXPLANATION:

- This method copies the address of the actual argument into the formal parameter.
- In this case, changes made to formal parameter within the function will have **effect** on the actual parameter.

PROGRAM:

```
#include<iostream>
using namespace std;
void display(int &x)
{
        cout<<"\n The Value of x="<<x*x;
}
int main()
{
        int a;
        cout<<"\n Call by Reference";
        cout<<"\n Enter the Value for a=";
        cin>>a;
        cout<<"\nThe Value of a="<<a;
        display(a);
        cout<<"\n Back to main function= "<<a;
        system("pause");
```

```
    return 0;
}
```

OUTPUT:

Call by Reference

Enter the Value for a=2

The Value of x=4

Back to main function=2

EXPLANATION:

- The **&(AMPERSAND)** symbol in the declaration of the parameter **x** means that the argument is a reference variable and hence the function will be called by **Passing Reference.**
- Hence when the argument **a** is passed to the **display()** function, the variable **x** gets the address of **a** so that the location will be shared.
- In other words, the variables **x** and **a** refer to the same memory location.
- We use the name **a** in the main() function, and the name **x** in the **display()** function to refer the same storage location.
- So, when we change the value of **x**, we are actually changing the value of **a**.

3. DEFINE SCOPE? EXPLAIN THE VARIOUS TYPES OF SCOPES IN C++ LANGUAGE?

- **Scope** refers to the **accessibility** of a variable.
- There are **4** types of scopes in C++. They are:
  - ✓ **Local scope**
  - ✓ **Function scope**
  - ✓ **File scope**
  - ✓ **Class scope**

LOCAL SCOPE

- A local variable is defined within a block.
- The scope of a local variable is the block in which it is defined.
- A local variable cannot be accessed from outside the block of its declaration.
- Local variables are not known outside their own code block.
- A block of code begins and ends with curly braces { }.

- Local variables exist only while the block of code in which they are declared is executing.
- A local variable is created upon entry into its block and destroyed upon exit.

FUNCTION SCOPE

- The scope of variables declared within a function is extended to the function block, and all sub-blocks therein.
- The life time of a function scope variable, is the life time of the function block.
- The scope of formal parameters is function scope.

FILE SCOPE

- A variable declared above all blocks and functions (above main( )) has the scope of a file.
- The scope of a file scope variable is the entire program.
- The life time of a file scope variable is the life time of a program.

PROGRAM FOR THE SCOPE RULES FOR VARIABLES

```
#include<iostream.h>
#include<conio.h>
int n1=10;      // File Scope
class a
{
        private:
                int b;          // Class Scope
        protected:
                int c;
                void add();
        public:
                int d;
                void sub();
};
int main()
{
        int n2 =20;    // Function Scope
        if(n1>n2)
        {
```

```
        int temp;      // Local Scope
        temp=n1;
        n1=n2;
        n2=temp;
    }
    cout<<"\n"<<n1<<"\n"<<n2;
    system("pause");
    return 0;
}
```

OUTPUT

10

20

4. DESCRIBE THE INLINE FUNCTION WITH SYNTAX, ADVANTAGES AND PROGRAM?

DEFINITION:

- An inline function looks like normal function in the source file but inserts the function's code directly into the calling program.
- To make a function inline, one has to insert the keyword **inline** in the function header.

SYNTAX:

```
inline returntype functionname(datatype parametername1, … datatype parameternameN)
{
statements;
}
```

ADVANTAGES OF INLINE FUNCTIONS:

- ✓ Inline functions execute faster but require more memory space.
- ✓ Reduce the complexity of using STACKS.

PROGRAM:

```
#include <iostream>
using namespace std;
inline float simple_interest(float p1,float n1, float r1)
{
```

```
        return (p1*n1*r1)/100;
    }
    int main ()
    {
        float si,p,n,r;
        cout<<"\nEnter the Principle, Interest and Number of years";
        cin>>p>>n>>r;
        si=simpleinterest(p,n,r);
        cout << "\nThe Simple Interest = Rs."<<si;
        system("pause");
        return 0;
    }
```

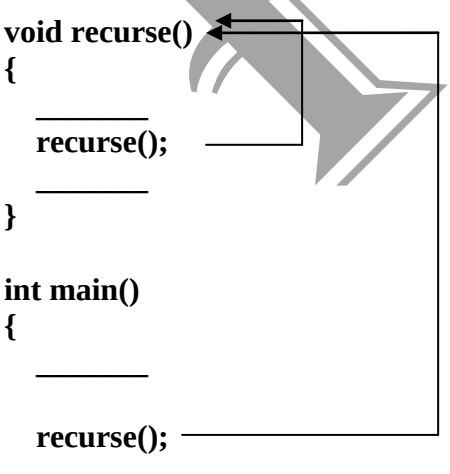
OUTPUT:

```
Enter the Principle, Interest and Number of years
60000
10
5
The Simple Interest = Rs.30000
```

5. WHAT IS RECURSION? WRITE A PROGRAM TO FIND GCD USING RECURSION.

- ✓ A function that calls itself is known as recursive function. And, this technique is known as recursion.

WORKING OF RECURSION:



}

PROGRAM:

```
#include <iostream>
using namespace std;
int factorial(int); // Function prototype //
int main()
{
        int no;
        cout<<"\nEnter a number to find its factorial: ";
        cin >> no;
        cout << "\nFactorial of Number " << no <<" = " << factorial(no);
        return 0;
}
int factorial(int m)
{
        if (m > 1)
        {
        return m*factorial(m-1);
        }
        else
        {
        return 1;
        }
}
```

OUTPUT:

Enter a number to find its factorial: 5

Factorial of Number 5 = 120