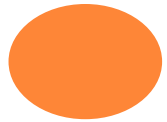
CONTROL STRUCTURES

After this lesson, learners will be able to:

- Understand the concept and usage of selection and iteration statements.
- Know various types of control structures available in C++.
- Analyze the problem, decide and evaluate conditions.
- To analyze and choose an appropriate combination of constructs to solve a particular problem
- Write code that employ decision structures, including those that employ sequences of decision and nested decisions.
- Design simple applications having iterative nature.



FLOW OF CONTROL

The order in which a set of statements are executed in a program is known as its flow of control

Generally the flow of control is sequential

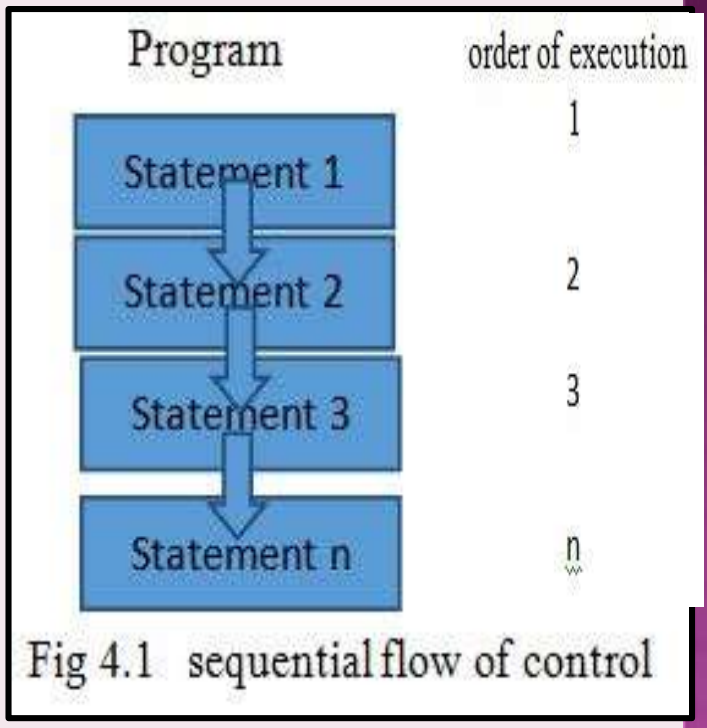


Fig 4.1 sequential flow of control

However at times it is required to change this sequential flow so that only a particular statement or group of statements are executed.

This is achieved **control structures**

WHAT ARE CONTROL STRUCTURES?

Control structures are statements that upon execution alter or control the sequence of execution of statements in a program.

TYPES OF CONTROL STRUCTURES

These can be categorized into the following categories:

Selection Control Statements	Iterative Control Structures	Jump Statements
• if • if.....else • switch.....case	• for loop • while loop • do.....while loop	• break • continue • exit • go to

THE IF STATEMENT

The if statement is used to execute an instruction or a set of instructions only if a condition is fulfilled.

SYNTAX

if (**conditional expression**)
Statement;

where :

- **Conditional expression** is any valid c++ expression having Relational and/or logical operators. Eg (x>y), (x+y !=8)etc
- **Statement** is a valid c++ executable statement such as a cout or calculation etc

THE IF WITH MULTIPLE STATEMENTS

There can be more than one executable statement in an if. In such a case the statements have to be enclosed in curly brackets .

```
if ( conditional expression)
  { Statement  1;
    Statement 2;
    .
    .
  }
```

WORKING OF SIMPLE IF

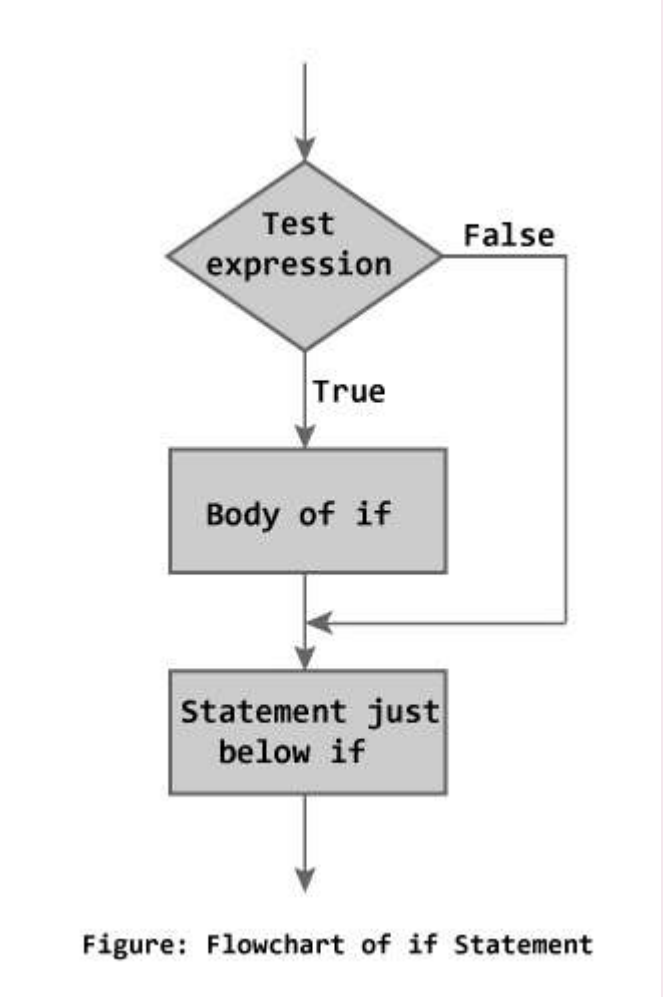


Figure: Flowchart of if Statement

PROGRAM TO ILLUSTRATE THE IF

Program

```
#include <iostream.h>
void main()
{int x,y;
x=9, y=7;
 if(x>y)
cout<<“Condition True”;}
```

Output : Condition True

THE IF..ELSE STATEMENT

The if.. else statement executes the set of statement(s)following if, if the given condition is true otherwise it executes an the set of statement(s) following the else.

SYNTAX

If (**condition expression**)

Statement 1;

else

Statement 2;

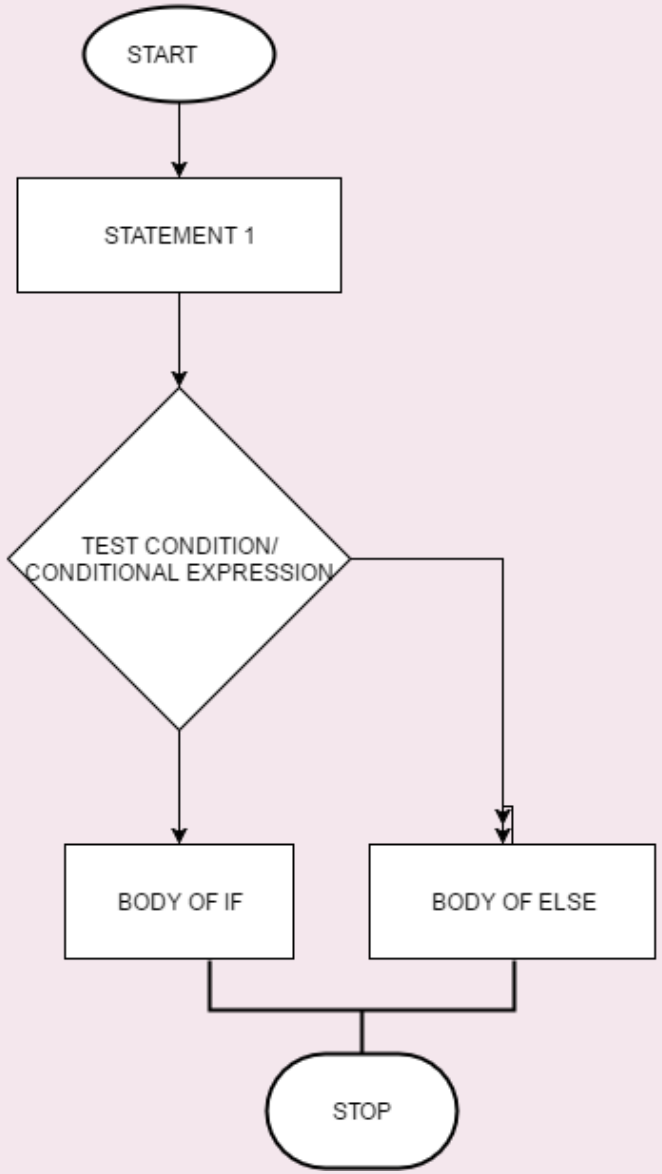
- Only one statement either 1 or 2 will be executed in any case .
- Statement 1 will be executed if conditional expression evaluates to **true**
- Statement 2 will be executed if it evaluates to **false.**

EXAMPLE OF IF ELSE

Program Segment	Output (expression)

1. int m = 15;
if(m > 6)
cout << "Hello";
else
cout<< "Bye";
	Hello
	[as m is more then 6, the first statement is executed]

THE WORKING OF IF ELSE STATEMENT



PROGRAM TO ILLUSTRATE IF ELSE

Program

```
#include <iostream.h>
void main()
{int x,y;
cin>>x>>y;
 if(x>y)
cout<<“Condition True”;
else
cout<<“Condition False”;
}
```

Input: 15,8 Output : Condition True
Input :15, 22 Output : Condition False

PROGRAM: WAP TO INPUT AGE OF A PERSON AND CHECK THE VOTING AGE

```
#include <iostream.h>
void main()
{
 int age;
cout<<“Enter the age of a person”;
cin>>age;
if (age>=18)
cout<<“\nYou can vote”;
else
cout<<“\n you cannot vote”;}
```

```
Enter the age of a person
20
You can vote
```

NESTED IFS AND IF..ELSE STATEMENTS

When we write an if within an if or an if..else within an if what we get are nested if or if else statements. We can have various nesting constructs:

1. if (condtion)
if(condition)]
statement 1;
	2. if (condition)
	if (condition)
	statement 1;
	else
	if (condition)
	statement 2;

3. if (condition)
if(condition)
statement 1;
else
statement 2;
	4.if (condition)
	statement 1;
	else
	if (condition)
	if(condition)
	statement 2;

HOW DOES IT WORK?

- There can be any level of nesting. But the rule is:
- Whenever a condition is encountered it is checked, if it evaluates to true, the statement immediately following it is executed otherwise the next statement is executed.
- If there is an else part, the statement in the else is executed.

CONSTRUCT TYPE 1

. if (condtion)

if(condition)

statement 1;

A simple if within an if with no else part. Statement1 will be executed only if both ifs evaluate to true.

IF WITHIN AN IF

Example program segement1:

```
int x = 13, y = 6, z = 2;
if ( x > y)              // outer if
if ( x > z)             // inner if
cout << " x is greater then y & z";
```

The outer if evaluates to true. So the control goes to the next statement. The if here also gives true, thus the cout is executed and we get the output:

x is greater than y & z

CONSTRUCT TYPE 2

```
. if (condition)
        if (condition)
           statement 1;
  else
      if (condition)
        statement 2;
```

An if in the if part as well as in the else part of an outer if

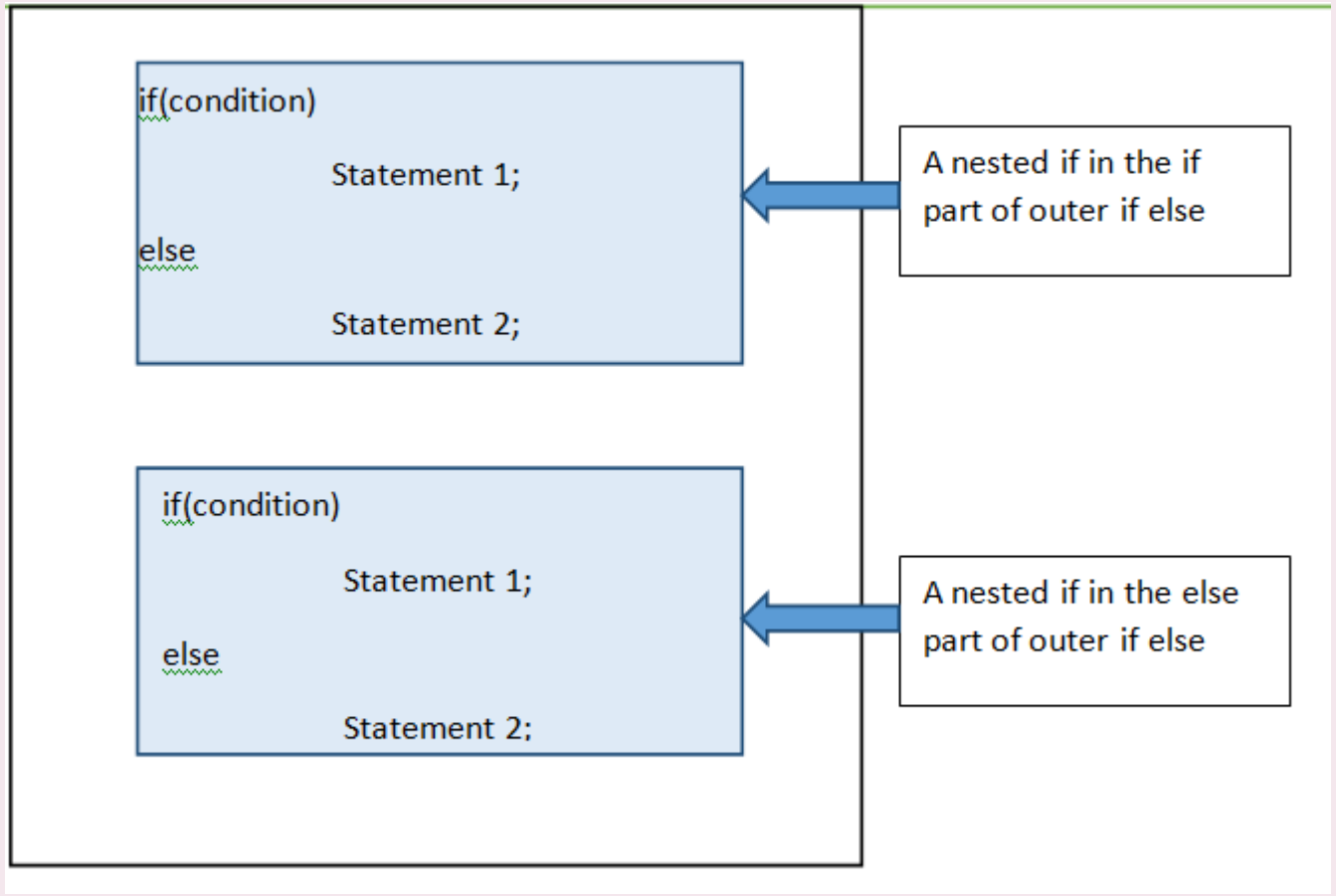
A NESTED IF IN THE IF PART OF IF ELSE

```
int x = 13, y = 16, z = 2;
   if ( x > y)
    if(x>z)
      cout<<"\nx is greatest ";
  else
   if ( y > z)
  cout << "\ny is greatest";
```

output

```
y is greatest
```

A COMPLETELY NESTED IF ELSE



A COMPLETELY NESTED IF ELSE

```
//program to find largest of 3 numbers
#include<iostream.h>
void main()
{ int x,y,z;
cout<<"\nenter 3 numbers:"<<endl;
cin>>x>>y>>z;
if(x>y)
 if(x>z)
   cout<<"\n"<<x<<"is the largest";
else
   cout<<"\n"<<z<<"ïs the largest";
else
 if(y>z)
    cout<<"\n"<<y<<"is the largest";
 else
    cout<<"\n"<<z<<"is the largest"";
```

IF ELSE IF LADDER

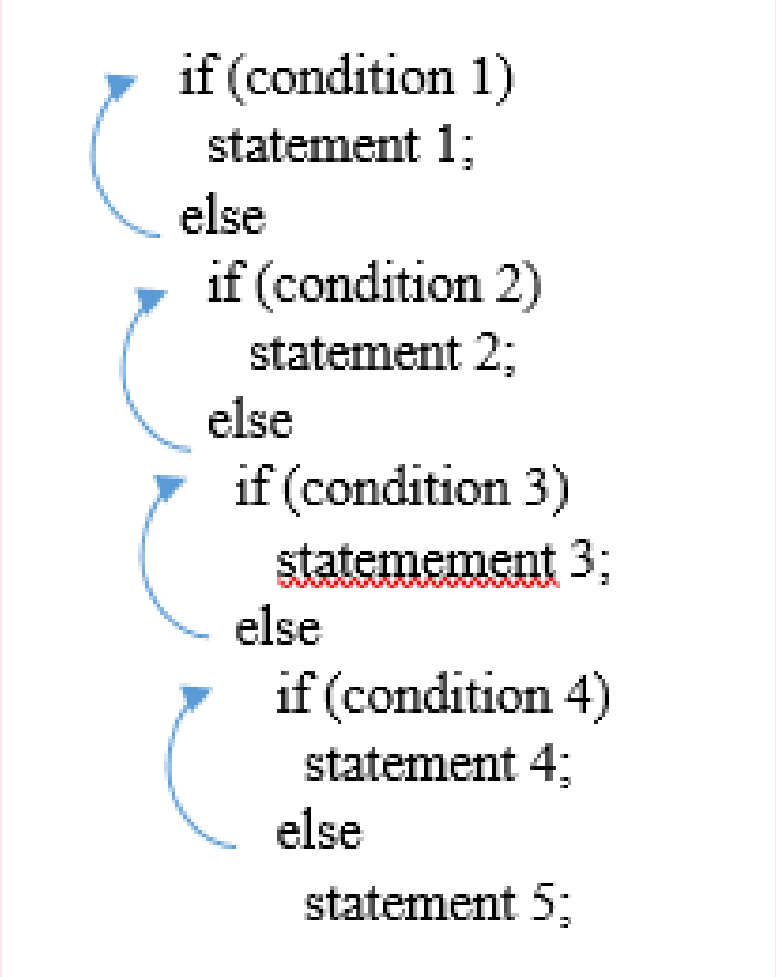
We can have complex structures like the if else if where conditions are tested sequentially and a course of action takes place.

```
if (condition 1)
    statement 1;
else
if (condition 2)
     statement 2;
else
 if (condition 3)
       statement 3;
 else
 if (condition 4)
       statement 4;
 else
       statement 5;
```

MATCHING ELSE WITH IF IN NESTED IF..ELSE

In a nested if..else statement we start from the end and match the first else we encountered with the closest preceding if that does not already have an else.

The next else with the next preceding if and so on. The structure here illustrates this.



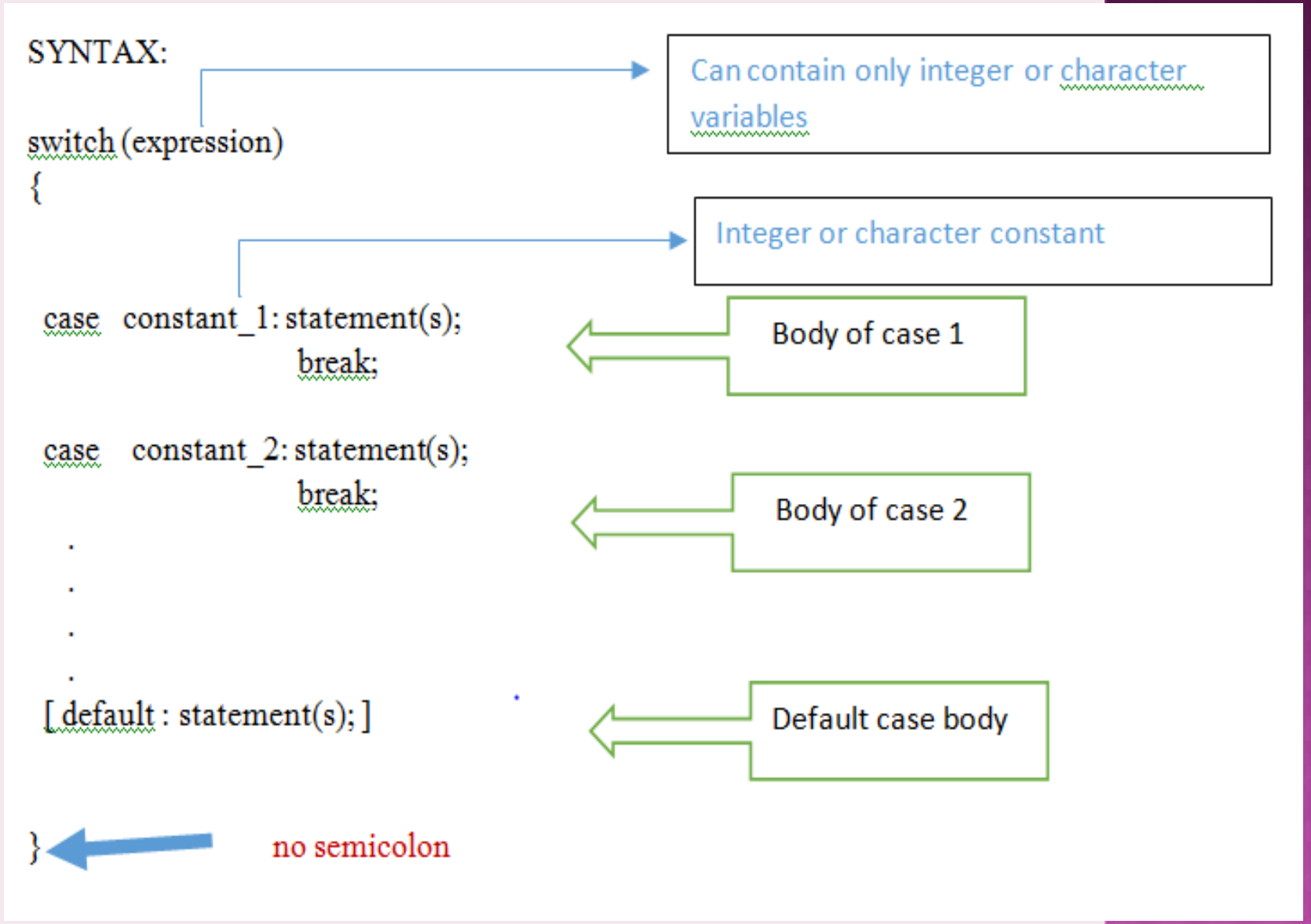
Program to display a grade according to the marks input by the user.

```
#include<iostream.h>
#include<conio.h>
  void main()
{
                clrscr();
                int marks;
                cout << " \n please input the marks";
                cin >> marks;
              if ( marks >= 90)
                    cout << "grade a ";
                else
                if ( marks >= 75)
                    cout << "grade b ";
                else
                if ( marks >= 65)
                    cout << "grade  c";
                else
              if ( marks >= 50)
                   cout << "grade d";
             else
            if ( marks >= 40)
                   cout << "grade e ";
           else
                  cout << "grade f ";
                }
```

THE SWITCH STATEMENT

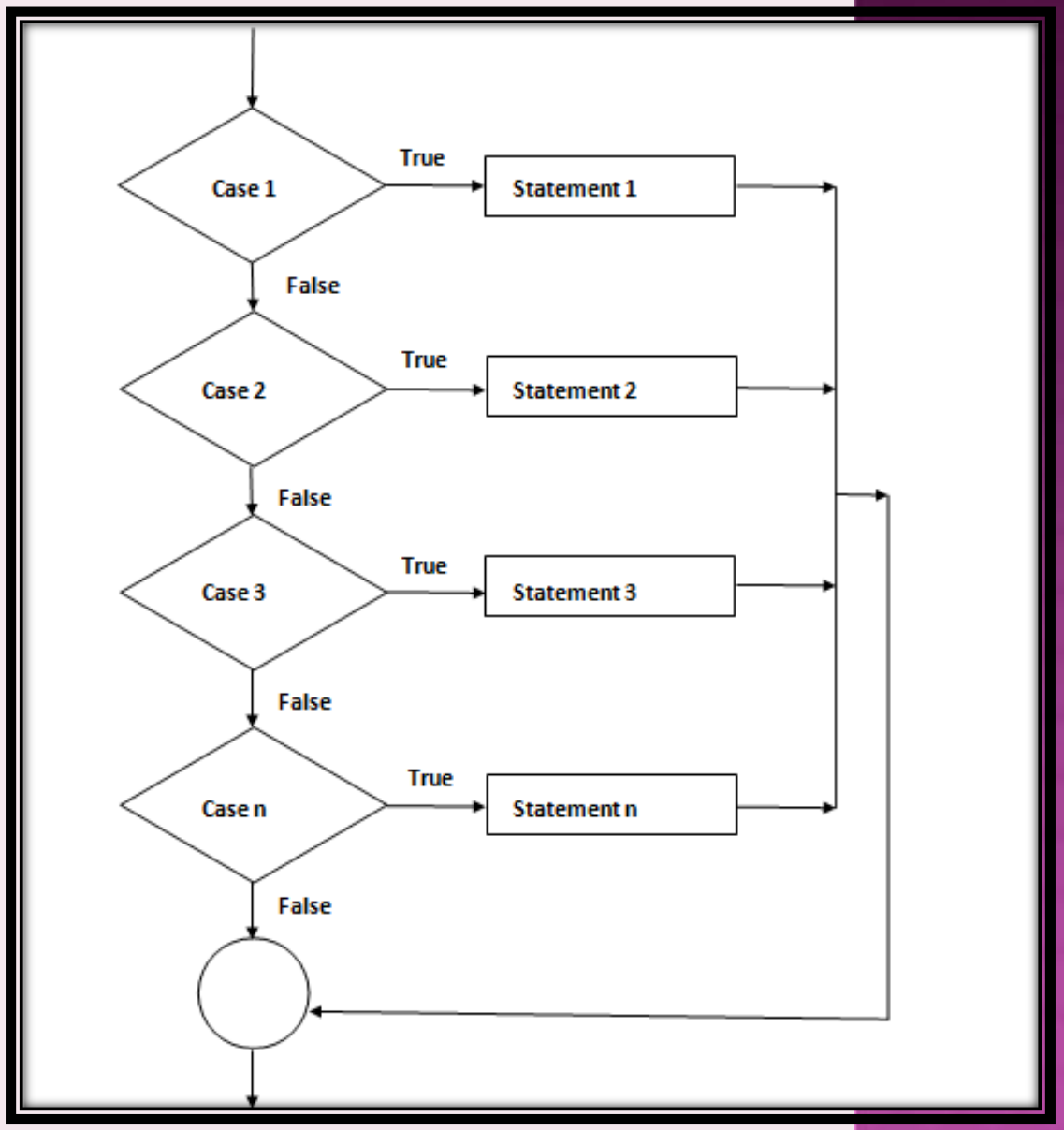
The switch statement is a selection control structure that is used to model branching of code to be executed depending on the value of the switch expression.

The switch statement provides with a no. of options out of which only one is executed depending on its value.



WORKING OF A SWITCH STATEMENT

- First, the switch expression is evaluated.
- If the value matches any of the case labels the control branches to the statement following the matched case
- From here, control proceeds sequentially until either a break statement or the end of the switch statement is encountered.
- If the value of the switch does not match any of the cases then either of the two happens.
  - If there is a default label, the statement following default is executed.
  - If there is no default, control comes out of switch and proceeds to the next statement after the switch.

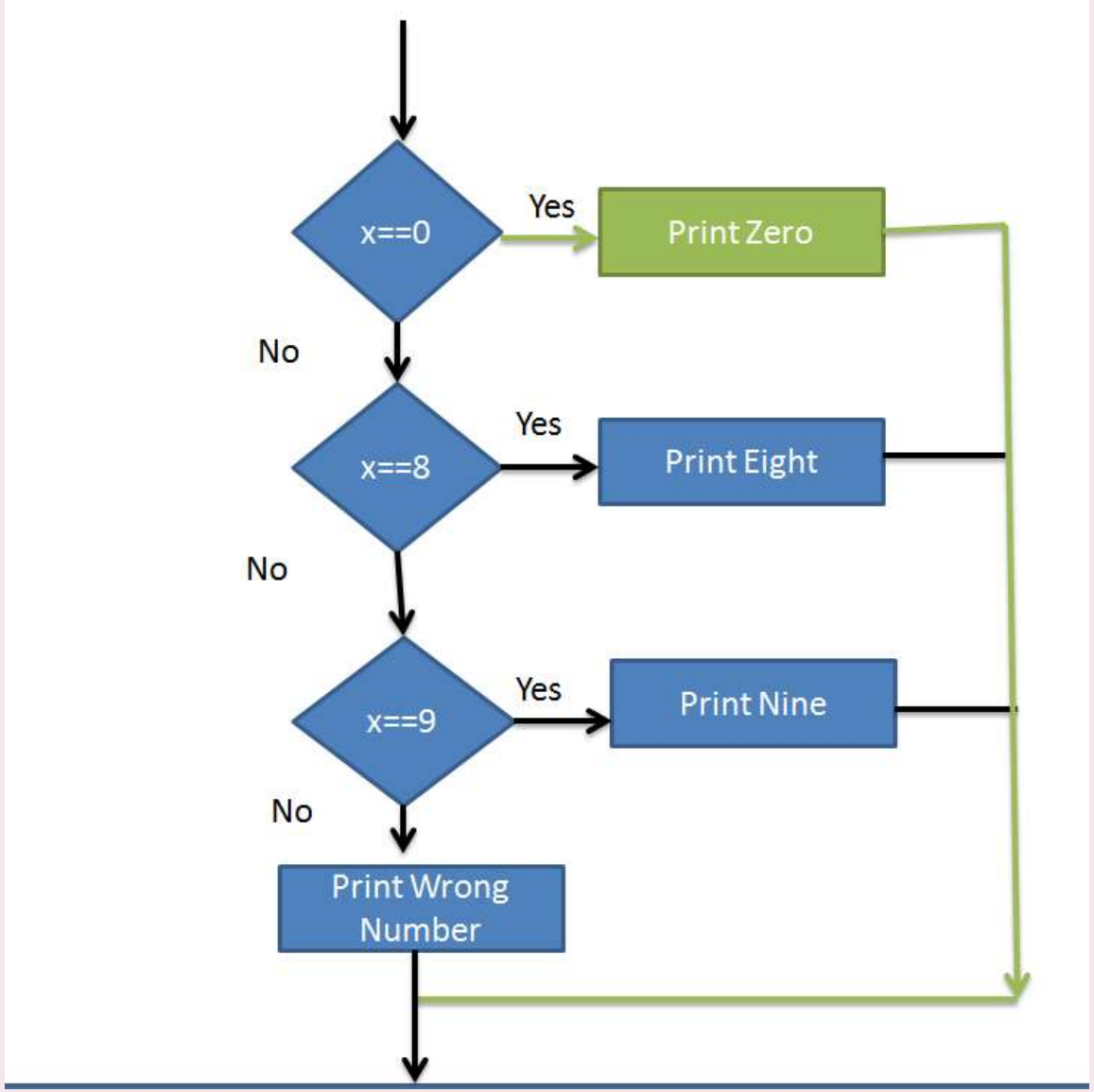


AN EXAMPLE TO EXPLAIN WORKING

```
int x = 0;
switch( x )
{
  case 0 : cout <<” Zero ”;
             break;

 case 8 : cout <<” Eight ”;
             break;

 case 9 : cout <<” Nine ”;
             break;

default : cout <<” Wrong number ”;

}
```

The switch is evaluated on the basis of the variable **x**. Depending on its value a particular case is executed. Now we shall observe the output for some values of x.

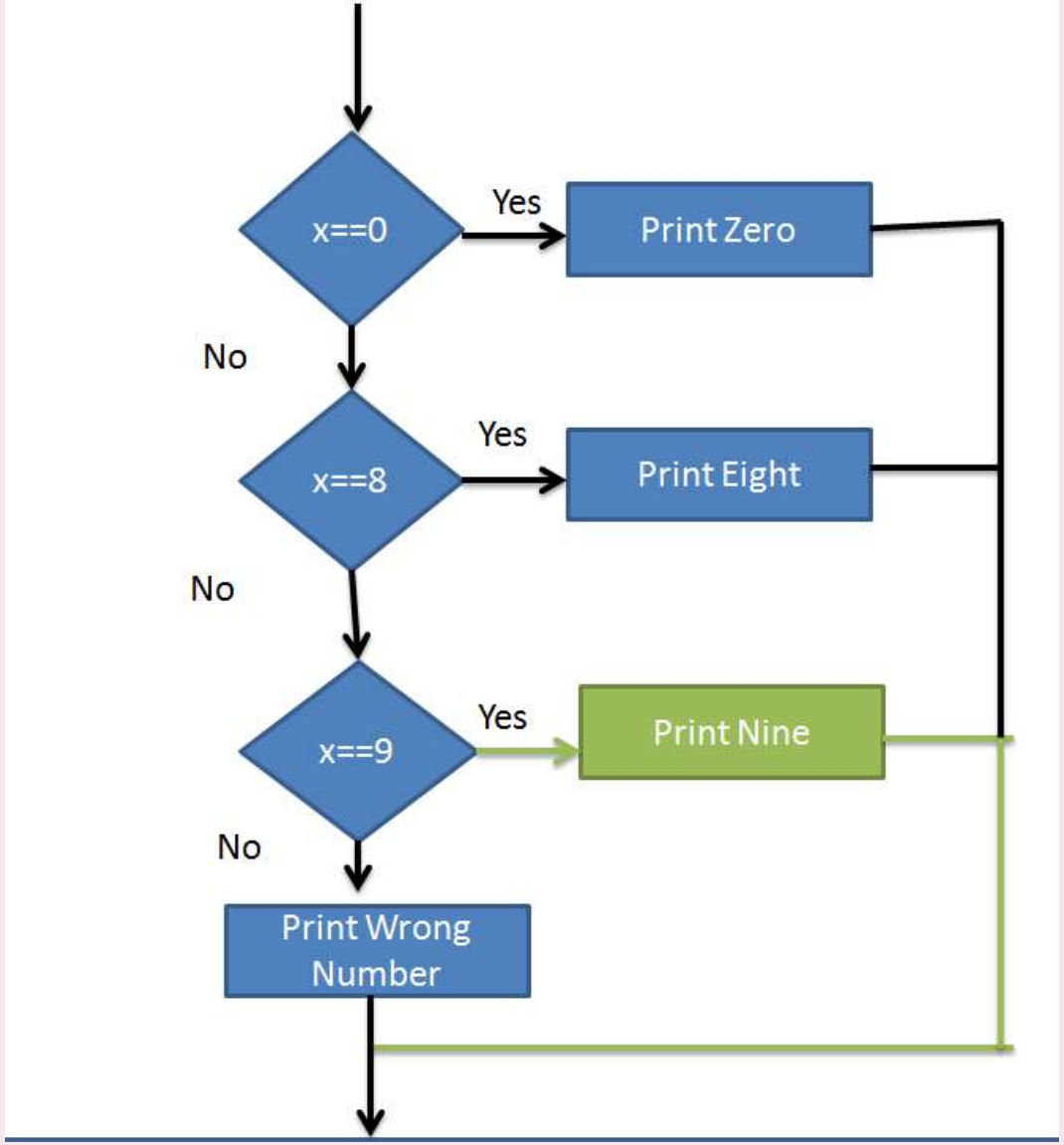
SAMPLE INPUT VALUE X=0



Explanation:

1. The variable in the switch bracket ie, x has the value 0.
2. This value is matched with each of the cases.
3. When a match is found ie, case 0, the statement following the colon is executed.
4. Next the break statement is executed which causes the control out of the switch. There are no other comparisons

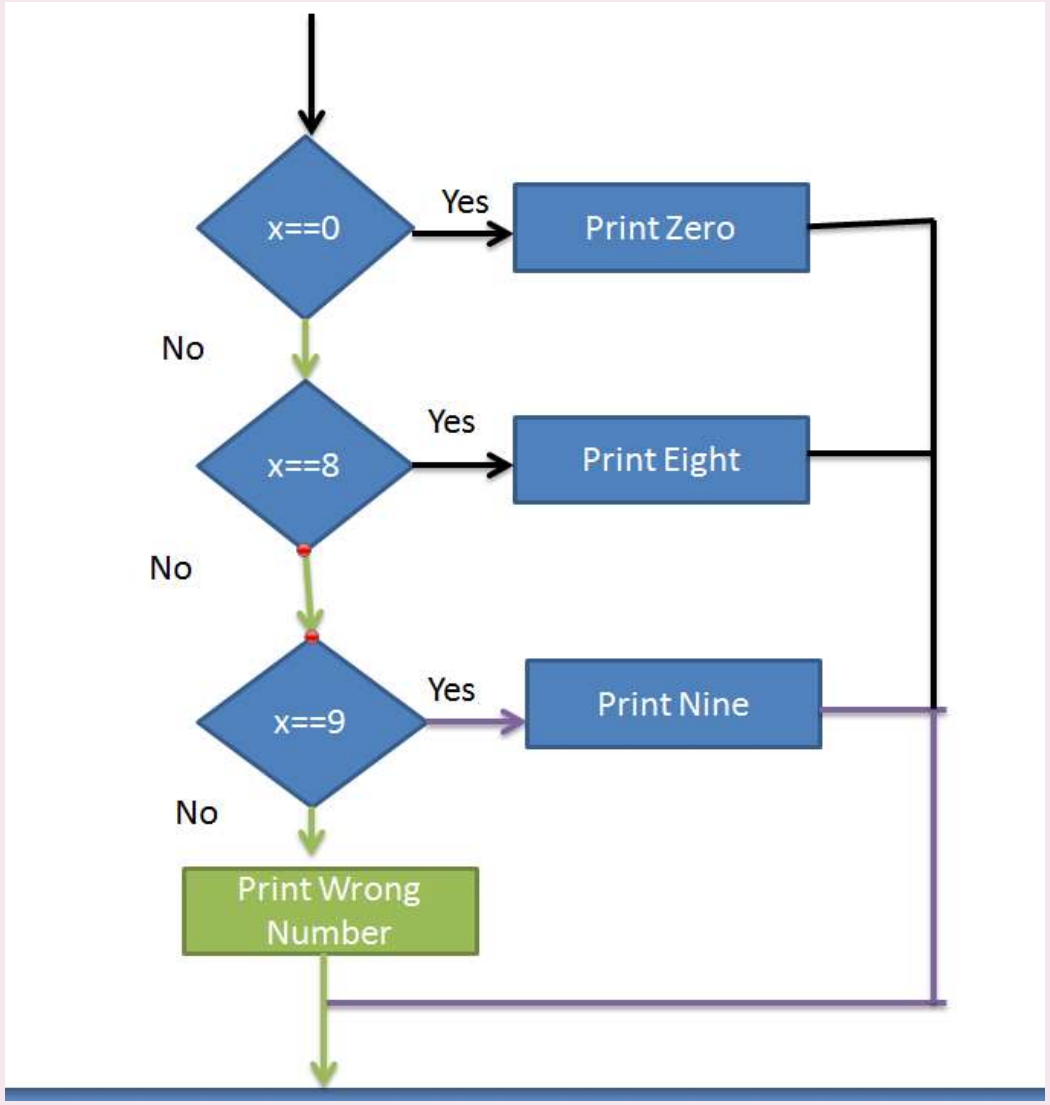
SAMPLE INPUT VALUE X=9



Explanation:

1. The variable in the bracket ie, x has the value 9.
2. This value is matched with each of the cases.
3. When a match is found ie, case 9, the statement following the colon is executed.
4. Next the break statement is executed which causes the control out of the switch. There are no other comparisons

SAMPLE INPUT VALUE X=15



Explanation:

1. The variable in the bracket ie, x has the value 15.
2. This value is matched with each of the cases.
3. Since none of the case value matches the default case is executed.
4. The value Wrong Number is displayed
5. If there is no default in the above input case there shall be no output

INVALID SWITCH CASE LABELS:

1. `case a-b: cout<<“hello”;`

2. `case “Rita”: marks=marks+20;`

3.
```
case 2: cout<<“hello”;
            break;
case 2 : cout<<”Welcome”;
            break;
```

4.
```
default : cout<<“wrong choice”;
default: cout<<“Try again”;
```

5. `case 8.9: cout<<“hello”;`

1. The **case label** cannot be an expression eg. a+b or a-b etc.
2. The case label cannot be a **string** expression eg. “Hello”, “good” etc.
3. Each case constant can appear **only once** in a given switch statement
4. Only **one default** label can be there each switch Statements
5. The label must be a constant of **type integer or character only**. It can not be floating point.
6. The default label is executed only if none of the case labels match the expression in the switch. Suppose in the above example the value of x is 15, then the output would be wrong number

SIGNIFICANCE OF BREAK STATEMENT

```
int x = 0;
switch( x )
{
  case 0 : cout <<” Zero ”;

 case 8 : cout <<” Eight ”;

 case 9 : cout <<” Nine ”;

 default : cout <<” Wrong number ”;

}
```

- If the break is omitted then all of the statements following the selected branch (case) are also executed.
- In the program given if the break is omitted, we get the output:

ZeroEightNineWrongnumber

Program to display day of week in words

```
#include<iostream.h>
#include<conio.h>
void main()
{
      clrscr();
      int day_o_w;
       cout<<"Enter numerical day of the week:"<<"\t";
        cin>>day_o_w;
      switch( day_o_w )
     {
       case 1 : cout <<" Monday ";
                  break;
       case 2 : cout <<" Tuesday ";
                       break;
       case 3: cout <<" Wednesday ";
                       break;
       case 4 : cout <<" Thursday ";
                       break;
       case 5 : cout <<" Friday ";
                       break;
       case 6 : cout <<" Saturday ";
                       break;
       case 7 : cout <<" Sunday ";
                        break;
       default : cout <<" Sorry, not a day of the week ";
         }
}
```

Input value	Output
2	Tuesday
6	Saurday
11	Sorry, not a day of the week
5	Friday

EXPLANATION

1. In the above program, an integer value is input.
2. The value is stored in a variable day_o_w which is placed in the switch expression.
3. Depending on the input the case labels are compared.
4. Upon finding a match the case branch with the matched value is executed.
5. For eg. if the input is 5, case 5 is executed and the output will be Friday. After this the control goes out of the switch and the program ends.

OPERATORS IN SWITCH EXPRESSION

- In a switch statement only exact comparison of values can be done,ie, only equality operator(==) can be implemented. We cannot implement relational operators like >, <, >=, <= in a switch statement.
- The case label in a switch statement must exactly match ie, be equal to the switch expression, in order for the statement(s) to be executed.

IMPLEMENTING OR IN A SWITCH

We can however implement the logical OR(||) operator in a switch statement. The following if is equivalent to the switch given below

```
i f ( letter == ‘a’ || letter == ‘A’)
            cout << “Administration”;
else
if ( letter == ‘p’ || letter == ‘P’)
            cout << “Purchase”;
else
if ( letter == ‘s’ || letter == ‘S’)
            cout << “Sales”;
else
if ( letter == ‘w’ || letter == ‘W’)
            cout << “.Warehousing”;
else
cout <<”letter input is incorrect, Enter again”;
```

=

```
switch ( letter )
{
  case ‘a’ :            // empty case
  case ‘A’ : cout<<”Administration”;
                break;
  case ‘p’ :            // empty case
  case ‘P’ : cout<<”Purchase”;
                break;
  case ‘s’ :            // empty case
  case ‘S’ : cout<<”Sales”;
                break;
  case ‘w’ :            // empty case
  case ‘W’ : cout<<”Warehousing”;
                break;
default : cout<< “Enter  again”;}
```

EXPLANATION

Leaving a case label statement empty and without a break statement takes the control to the next case

Makes it fall through to the next case

Thus if letter is ‘a’ or ‘A’, in any case the output statement cout<<”Administration”;

After which the break is encountered and control comes out of the switch.

Therefore, we can see that leaving a case empty can be used to implement the logic OR (||) in a switch statement

DIFFERENCE BETWEEN IF AND SWITCH

If statement

Can be used to check a range of values using any of the relational and logical operators(>,<,>=,<=).

Variables of any type can be used in the condition eg float, int or char

Switch statement

Can be used th check only for exact match using equality operator(= =)

Only integer variables can be used in the switch bracket which determines the case to be executed