-Decision Making and Branching-

Programming Fundamentals

Background

- We have a number of situations where:

1. we may have to change the order of execution of statements based on certain conditions,
2. or repeat a group of statements until certain specified conditions are met(in case of iterations).

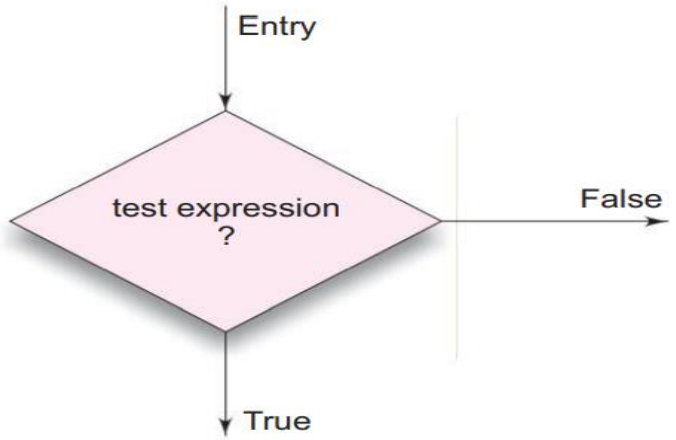
It involves decision-making.

Types of Decision-making/control capabilities:

1. If/ if-else statement
2. switch statement
3. Conditional/ternary operator statement (cond?expr1:expr2) //discussed in operators
4. goto statement (discouraged in programming)

A. THE IF STATEMENT

- It is basically a **two-way decision statement** and is used in conjunction with an expression.



- It takes the following form if:

```
if (test expression)
```

- The expression (relation or condition) is ‘true’ (or non-zero) or ‘false’ (zero), it transfers the control to a particular statement.

- This point of program has two paths to follow, one for the true condition and the other for the false condition.
- Some examples of decision making, using if statements are:

1. if (bank_balance ==0) borrow money
2. if (room is dark) put on lights
3. if (age > 55) person is retired

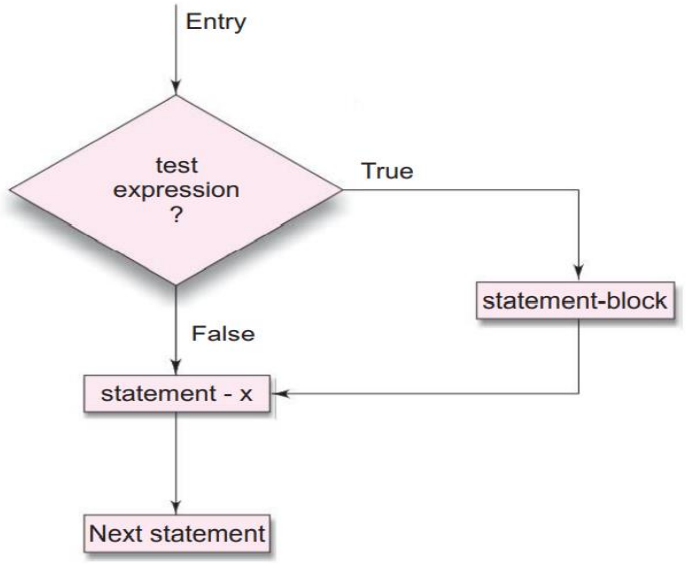
- The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. **Simple if statement**
2. **if.....else statement**
3. **Nested if....else statement**
4. **else if ladder**

1. SIMPLE IF STATEMENT

- Format:

```
if (test expression)
 {
      statement-block;
 }
statement-x;
```



- The ‘statement-block’ may be a single statement or a group of statements.
- If the test expression is true, the statement-block will be executed;
- otherwise the statement-block will be skipped and the execution will jump to the statement-x.
- When the condition is true both the statement-block and the statement-x are executed in sequence.

- **De Morgan's Rule:**

De Morgan's Rule is applied to decision statements, if **logical NOT** operator is applied to a compound logical expression, :

e.g.1: !(x&&y||!z) ➔ !x || !y && z

This rule is as follows:

"Remove the parentheses by applying the NOT operator to every logical expression component, while complementing the relational operators"

That is, x becomes !x

!x becomes x

&& becomes ||

|| becomes &&

! and ! will cancel each other.

Resolution of e.g.1: !x || !y && z !(x < = 0 || !condition) becomes x >0&& condition

2. THE IF.....ELSE STATEMENT

- Extension of the simple if statement. The general form is:

```
If (test expression)
      {
         True-block statement(s)
      }
else
      {
         False-block statement(s)
      }
statement-x
```

If the test expression is true,

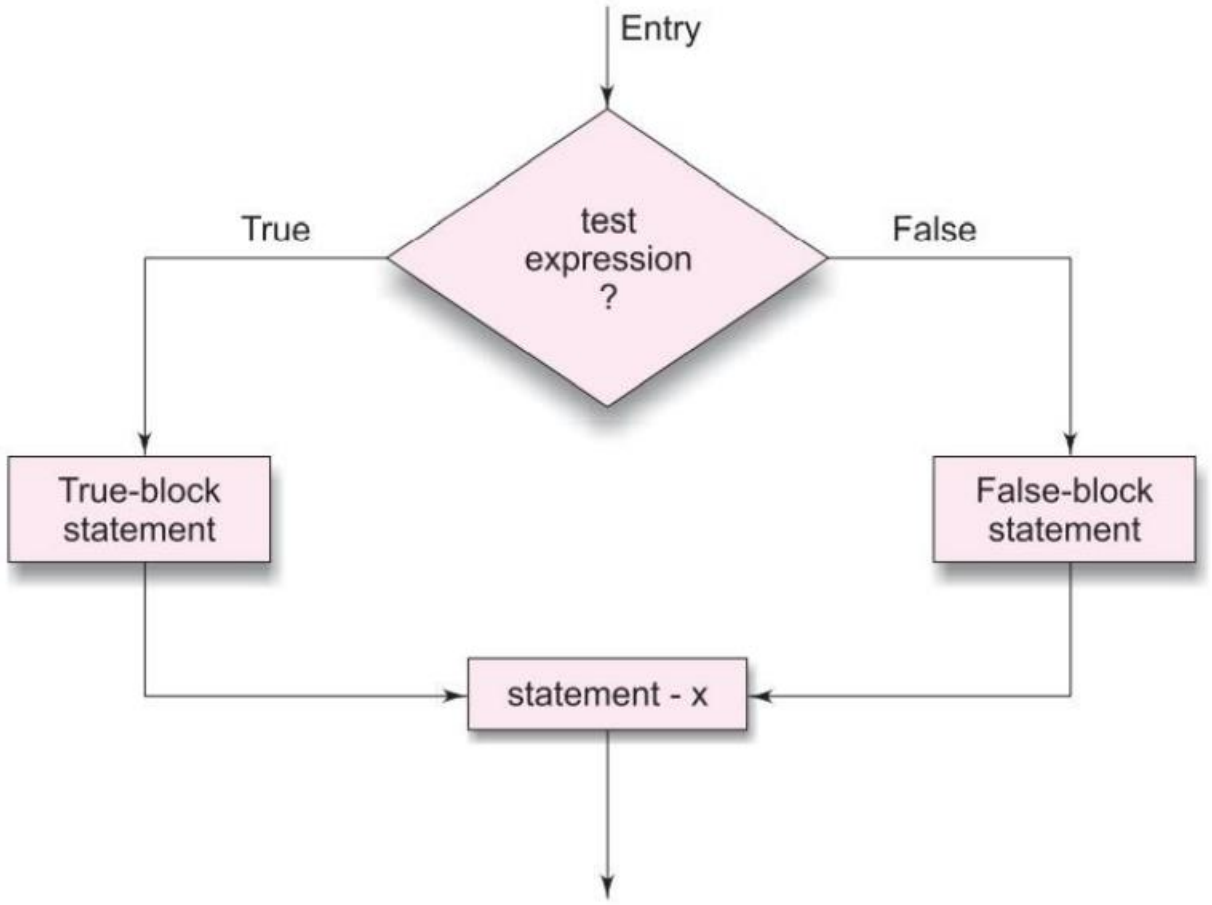
then the true-block statement(s), immediately following the if statements are executed;

otherwise,

the false-block statement(s) are executed.

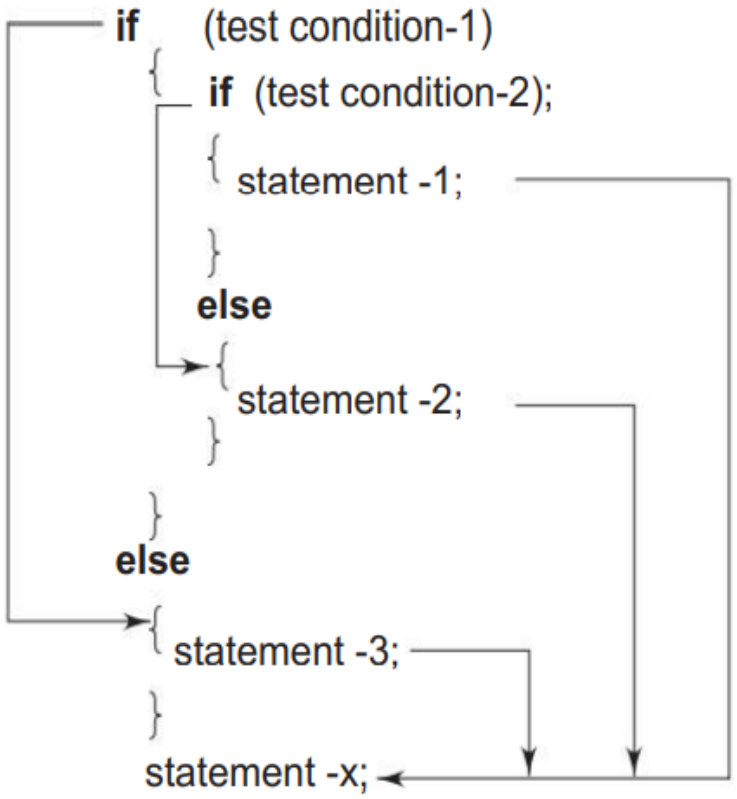
In either case, either true-block or false-block will be executed, not both.

Flowchart representation of the if-else statement:



3. NESTING OF IF....ELSE STATEMENTS

- When a series of decisions are involved, we may have to use more than one if...else statement in nested form as shown below:



If the condition-1 is false,

the statement-3 will be executed;

otherwise condition-2 is checked.

If the condition-2 is true,

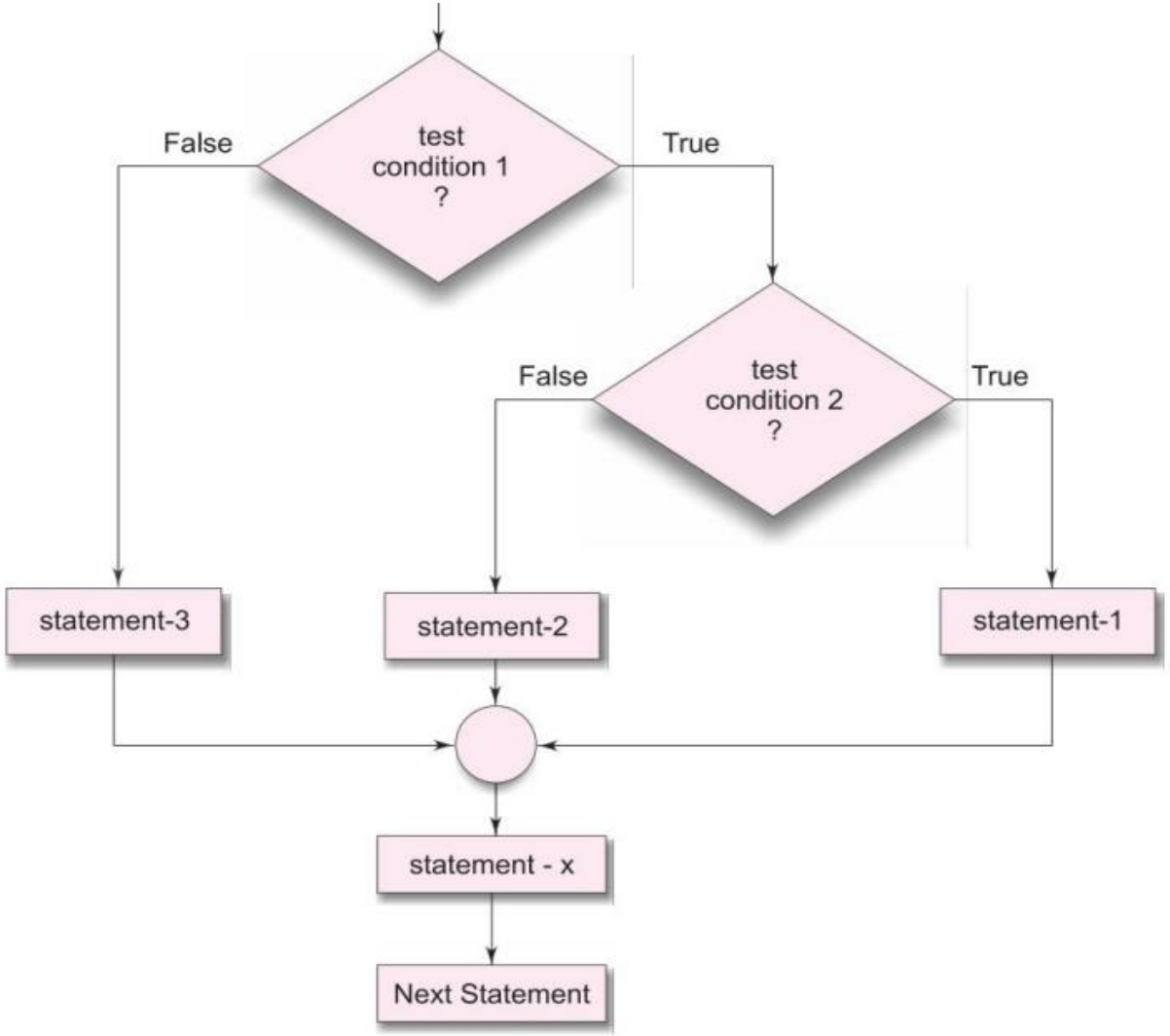
the statement-1 will be evaluated;

otherwise

the statement-2 will be evaluated and then

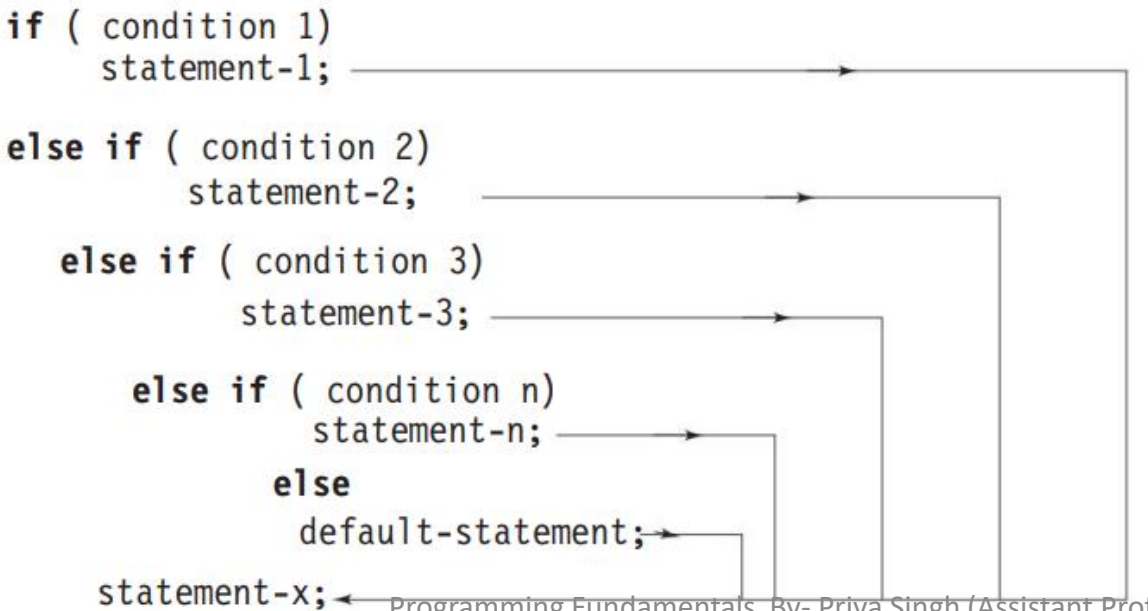
the control is transferred to the statemet-x.

- Flowchart of if-else statements:



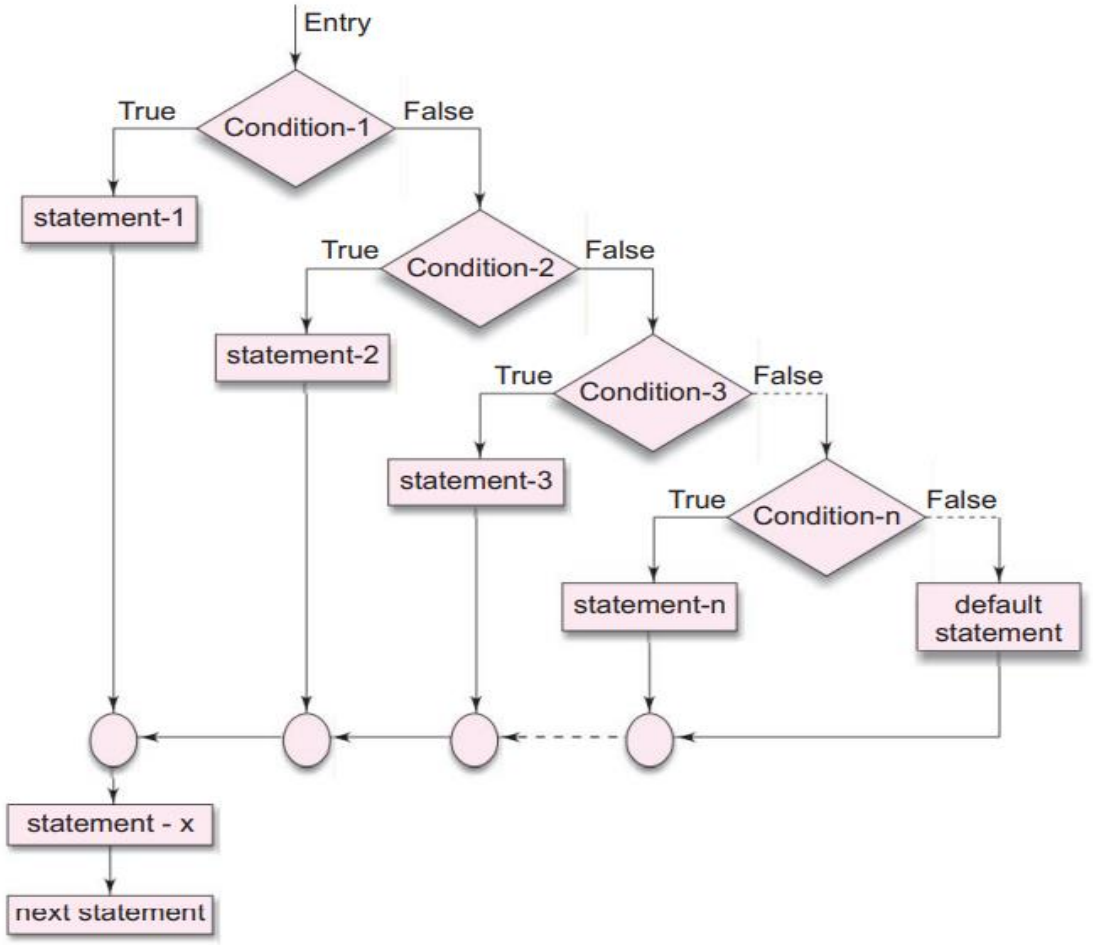
4. THE ELSE IF LADDER

- There is another way of putting ifs together when multipath decisions are involved.
- A **multipath decision** is a **chain of ifs** in which the statement associated with each else is an if.
- It takes the following general form:



- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of the ladder), downwards.
- As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder).
- When all the conditions become false, then final else containing the default-statement will be executed. (optional)

- Flow chart representation of if-else ladder:



- **Indentation rules to be followed in if-else:**

1. Indent statements that are dependent on the previous statements; provide at least three spaces of indentation.
2. Align vertically else clause with their matching if clause.
3. Use braces on separate lines to identify a block of statements.
4. Indent the statements in the block by at least three spaces to the right of the braces.
5. Align the opening and closing braces.
6. Use appropriate comments to signify the beginning and end of blocks.
7. Indent the nested statements as per the above rules.
8. Code only one clause or statement on each line.

B. THE SWITCH STATEMENT

- Switch: built-in **multiway decision statement**
- Tests the value of a given variable (or expression) against a list of values and when a match is found, a block of statements associated with that is executed.
- Format:

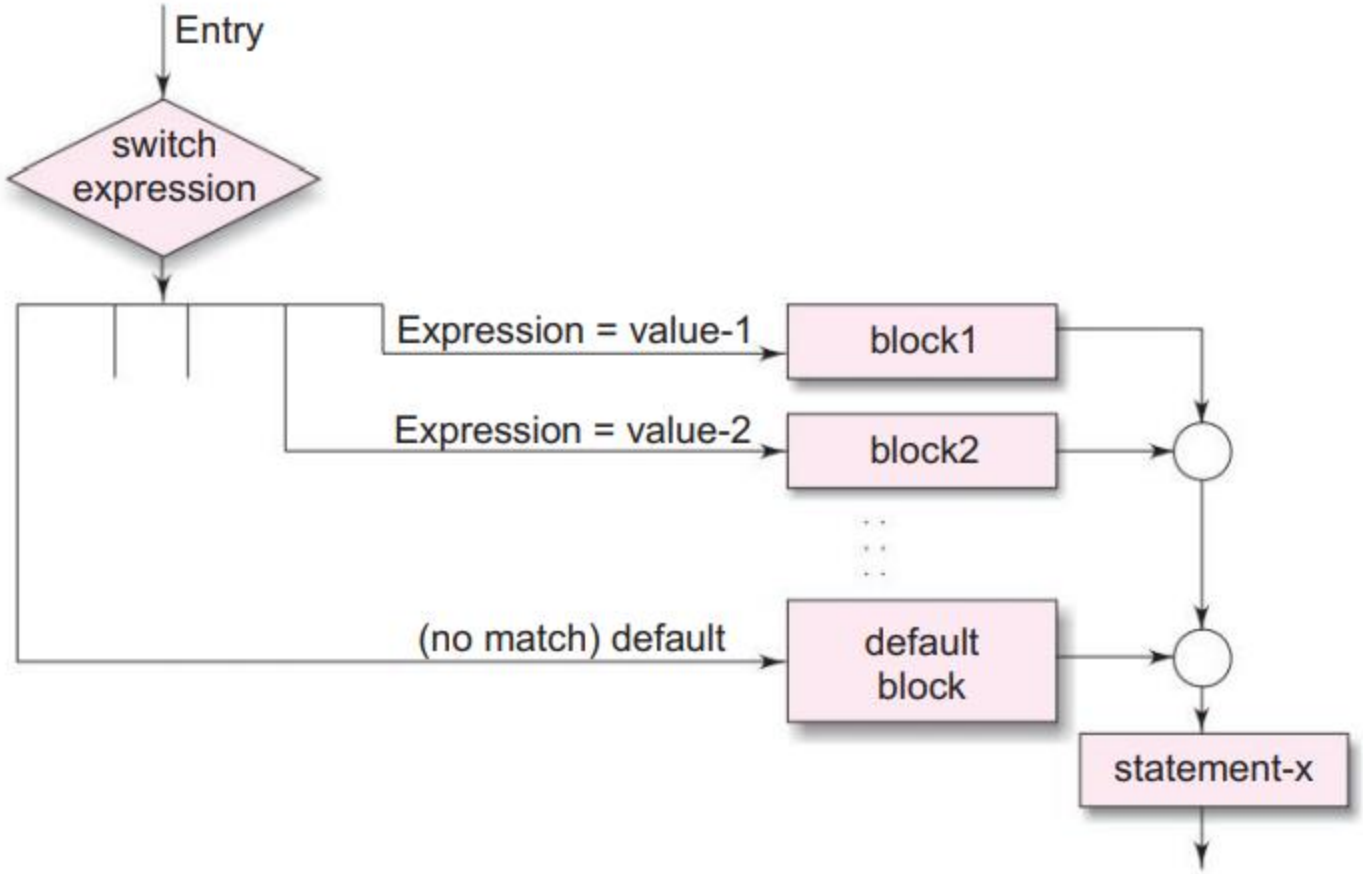
```
switch (expression)
{
   case value-1:
                block-1
                break;
   case value-2:
                block-2
                break;
   ......
   ......
   default:
                default-block
                break;
}
statement-x;
```

expression: integer expression or characters.

value-1, value-2 ..... (case labels): constants or constant expressions (evaluable to an integral constant)
Each of these values should be unique within a switch statement.
Labels end with a colon (:).
(ANSI C permits the use of as many as 257 case labels).
Block-1, block-2 .... :statement lists having 0 or more statements.
No need to put braces around these blocks.

- When the switch is executed, the **value of the expression** is successfully compared against the values value-1, value-2,....
- If a **case** is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.
- The **break** statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement following the switch.
- The **default** is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place if all matches fail and the control goes to the statement-x

- Flowchart representation of switch-case to represent its selection process:



Rules for using 'switch statement':

- The switch expression must be an integral type.
- Case labels must be constants or constant expressions.
- Case labels must be unique.
- No two labels can have the same value.
- Case labels must end with colon.
- The break statement transfers the control out of the switch statement.
- The break statement is optional. That is, two or more case labels may belong to the same statements.
- The default label is optional and executed when an expression does not find matching case label.
- There can be at most one default label.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to nest switch statements.

C. THE ? : OPERATOR

Called conditional/ternary operator

Useful for making two-way decisions.

This operator is a combination of ? and :

It takes three operands.

This operator is popularly known as the conditional/ternary operator.

Format:

conditional expression ? expression1 : expression2

Covered in operators' lecture

D. GOTO

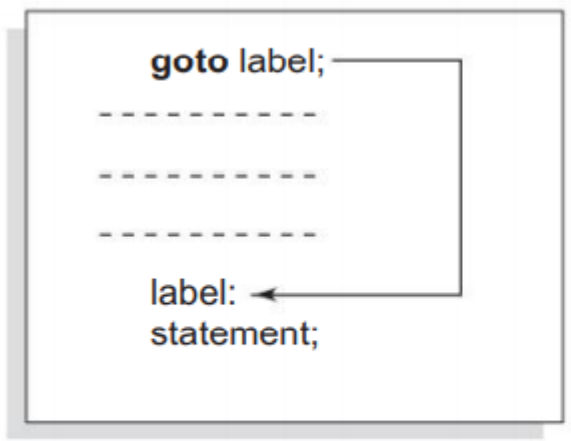
- Used to **branch unconditionally** from one point to another in the program.
- Not popular in a highly structured language like C.
- **Requires a label** in order to identify the place where the branch is to be made.
- **Label:** valid variable name, and must be followed by a colon.
- The label is placed immediately before the statement where the control is to be transferred.

- **The label:** can be anywhere in the program (before or after the *goto label;* statement).
- When a statement like ***goto begin;*** is encountered, control is transferred from current point to that mentioned in label **unconditionally**.
- A **goto** breaks the normal sequential execution of the program
- Example:

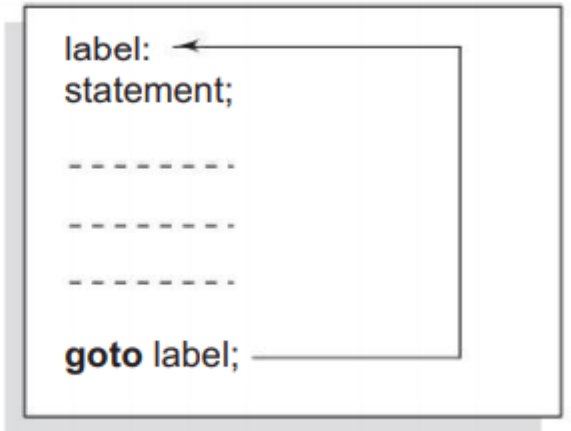
1. A goto is often used at the end of a program to direct the control to go to the input statement, to read further data.

2. Another use of the goto statement is to transfer the control out of a loop (or nested loops) when certain peculiar conditions are encountered.

- Backward jump: If the **label:** is before the statement **goto label;** a loop will be formed and some statements will be executed repeatedly.
- Forward Jump: If the **label:** is placed after the **goto label;** some statements will be skipped.



Forward jump



Backward jump