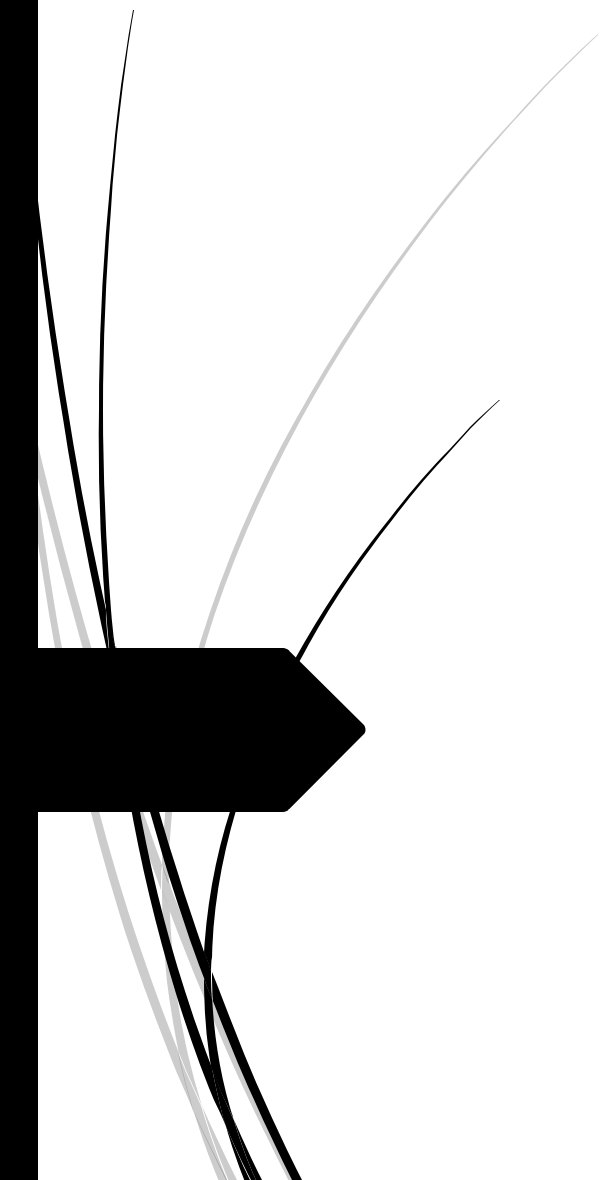


Control Statements

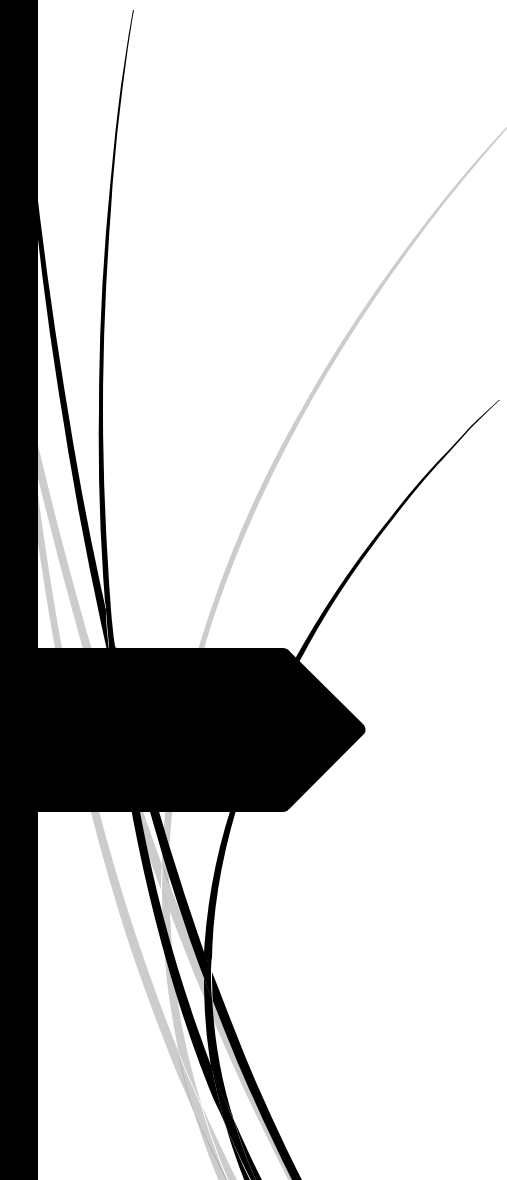
- Statements is an individual instruction of a program always end with a semicolon (;)
- Control statements are used for controlling the execution of the program
- They are classified into 3

1) Decision making statements
2) Iteration statement (Looping statements)
3) Jump statements

- **Decision making statement**
- **This statement execute at one time when a condition will be executed.**
  - **If**
  - **if else**
  - **Conditional operator**
  - **Nested if**
  - **else if ladder**
  - **switch**

- **Looping** statement / **Iteration** statement
- Used to execute a block of statement at multiple times
  - **while** statement
  - **do-while** statement
  - **for** statement
- 2 types loops
  - **Pre-test** : loop condition appear at beginning. (while, for)
  - **Post-test** : loop condition appear at the end of the loop (do-while)

- **Jump statement**
- **Used to transfer the program control with in a function unconditionally**
  - **goto**
  - **break**
  - **continue**



Decision making Statements

If statement

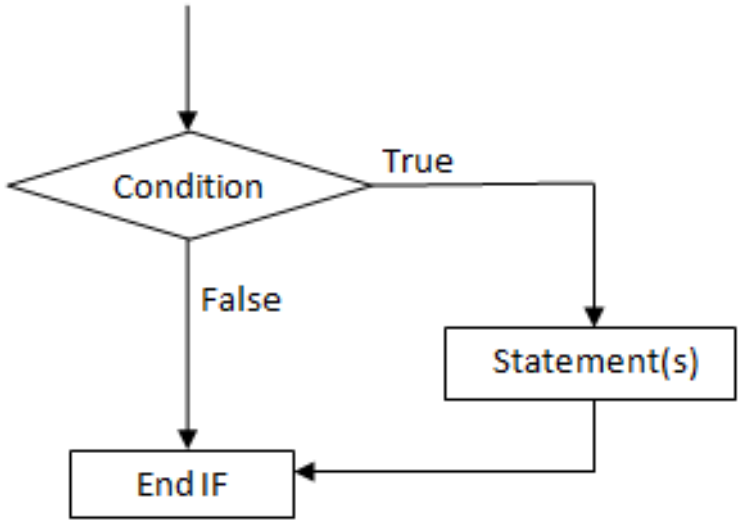
- Syntax :

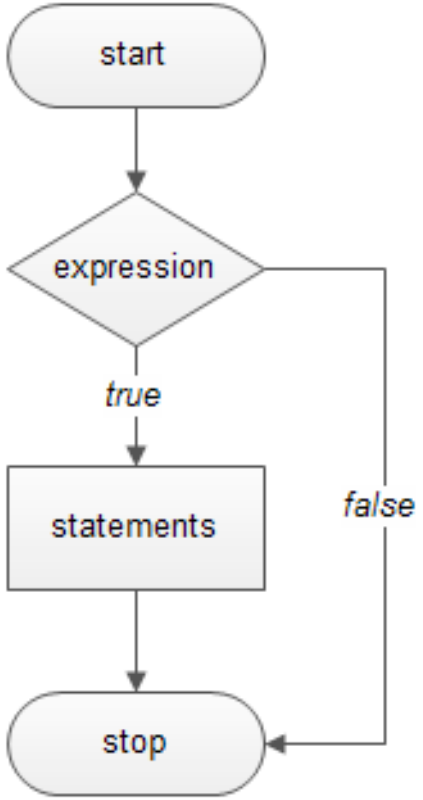
```
if (condition)
{
    // block of code to be executed if the condition is true
}
```

Decision making condition

Start

End


Condition
True
False
Statement(s)
End IF


start
expression
true
statements
false
stop

```
if (condition)
 {

      // statement part

}
```

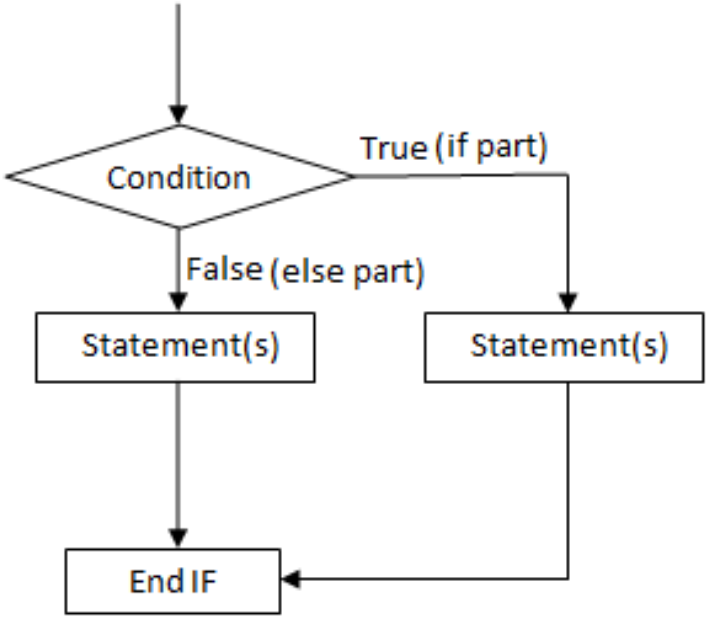
Steps

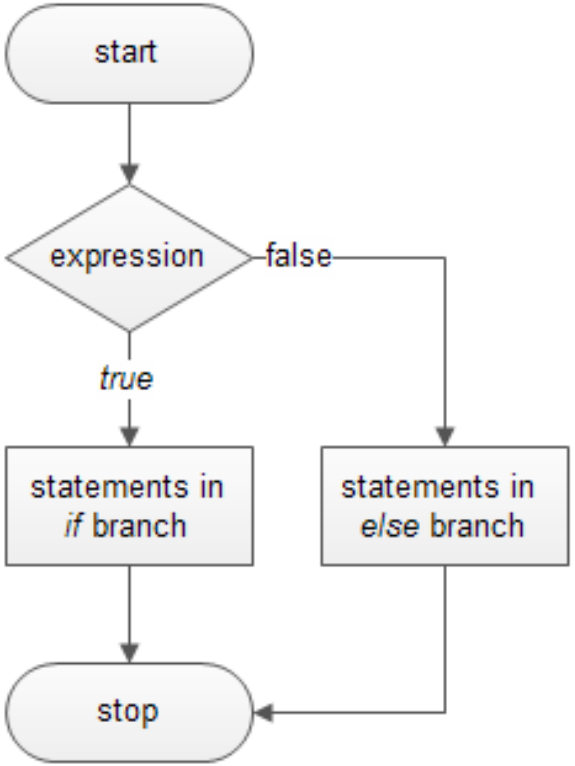
1. Condition check
2. If it is true, the statement part executed.
3. If it is false the statement part does not executed

If-else statement

- **Syntax :**

```
        if (condition)   <-- Decision making condition
Start   {

            // block of code to be executed if the condition is true

End     }
        else
Start   {

            // block of code to be executed if the condition is false

End     }
```


Condition
True (if part)
False (else part)
Statement(s)
Statement(s)
End IF


start
expression
false
true
statements in
if branch
statements in
else branch
stop

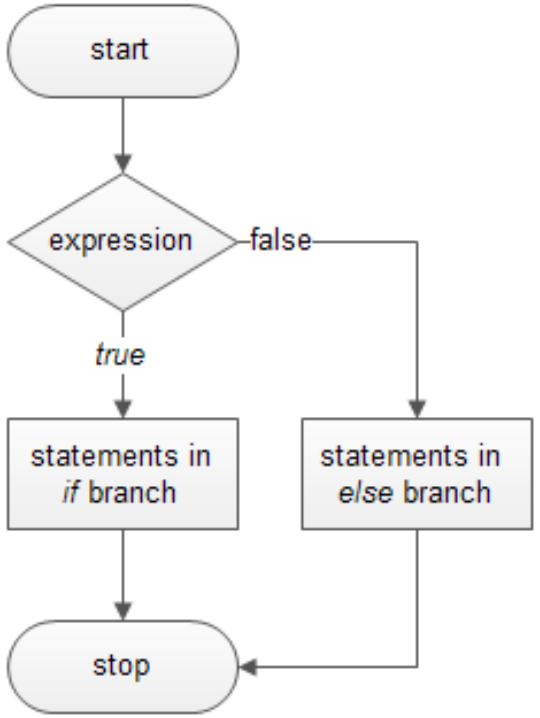
```
if (condition)
 {
      // statement 1
}
else
{
      // statement 2
}
```

Steps

1. Condition check
2. If it is true, the statement 1 will be executed.
3. If it is false the statement 2 will be executed.

Conditional operator (? :)

- It is a ternary operator
- Same as if – else statement



- Syntax :

Condition ? // *true statement* : // *false statement*

(*condition*) ? // statement 1 : // statement 2

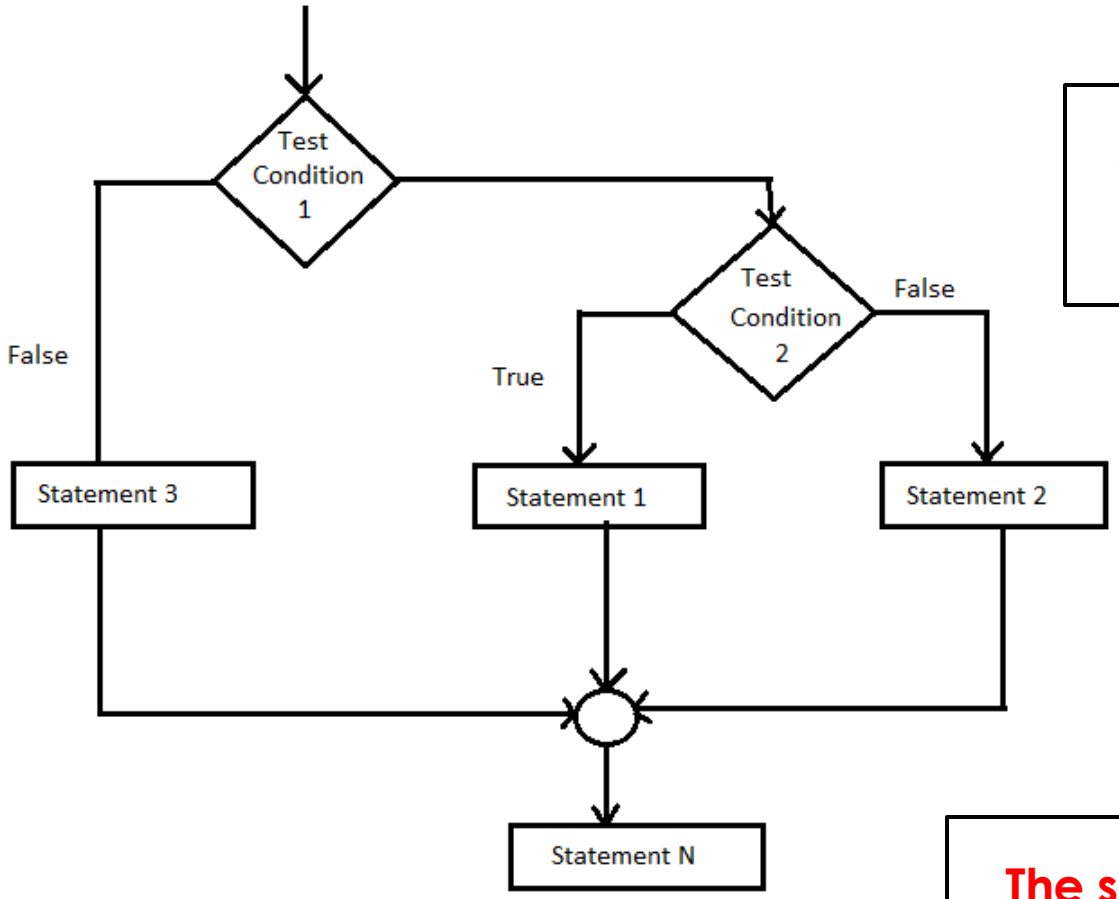
Steps

1. Condition check
2. If it is true, the statement 1 will be executed.
3. If it is false the statement 2 will be executed.

Nested If statement

- Syntax :

```
if (condition 1)
 {
        if (condition 2)
            {
                // block of code to be executed if the condition 1
                   and condition 2 are true
            }
}
```



An if statement contain another if statement is called nested if

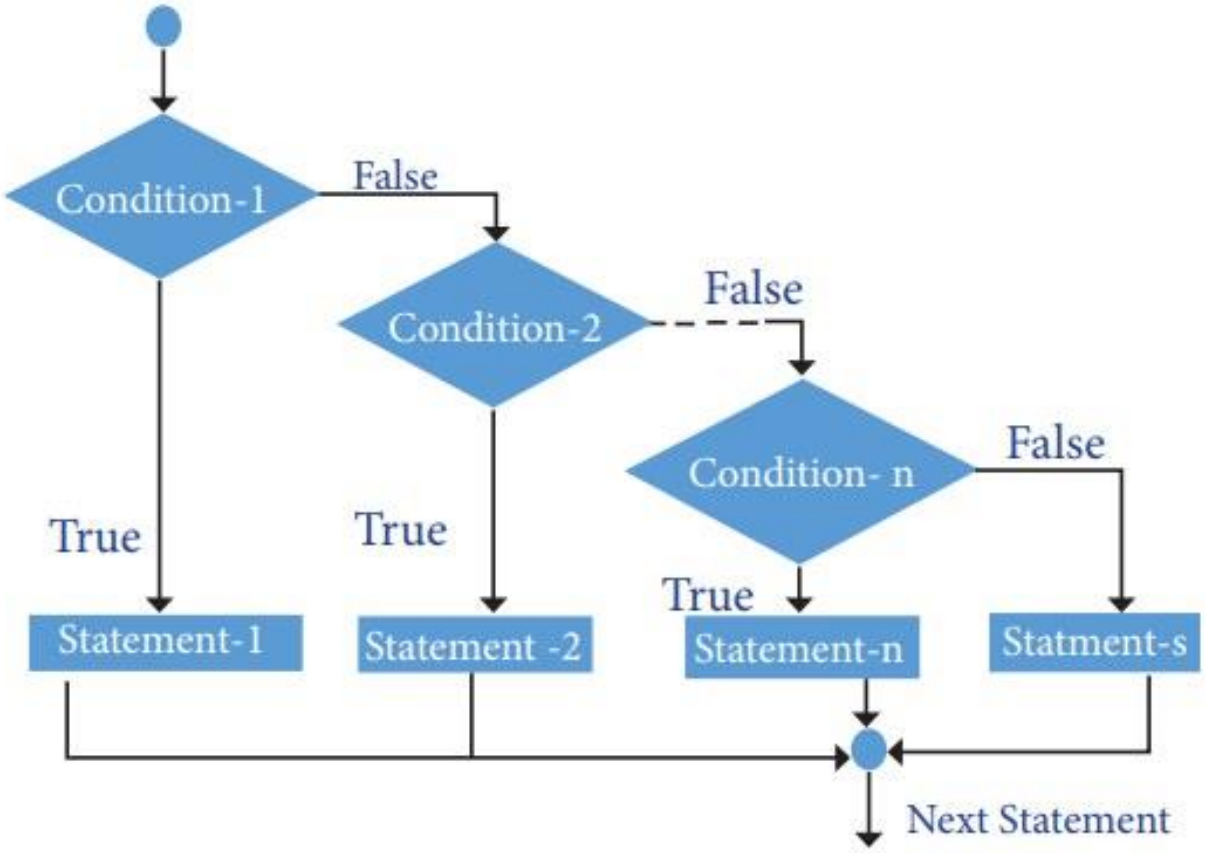
The statement is executed only when both the codes are true

else if ladder

- A variable which has more than
- two possibilities this statements are used
- Act as a multiple branch statement

- Syntax :

```
if (condition 1)
{
    statements 1;
}
else if (condition 2)
{
    statements 2;
}
else if (condition 3)
{
    statements 3;
}
else
{
statements n;
}
```

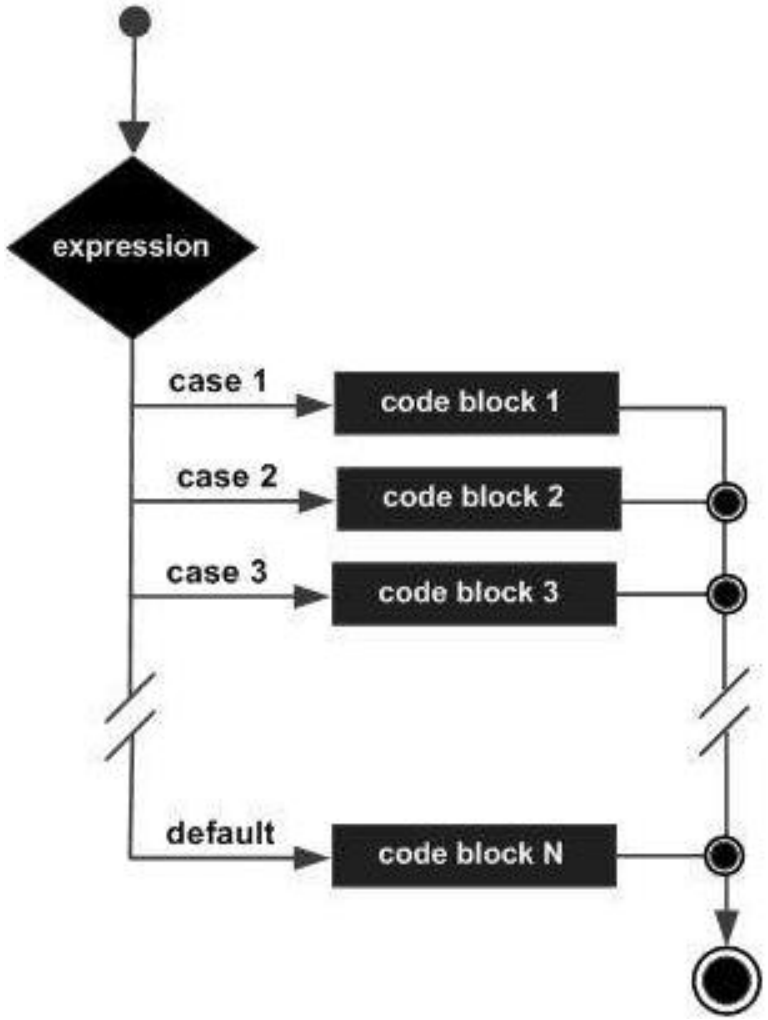

Condition-1
False
Condition-2
False
Condition- n
False
True
True
True
Statement-1
Statement -2
Statement-n
Statment-s
Next Statement

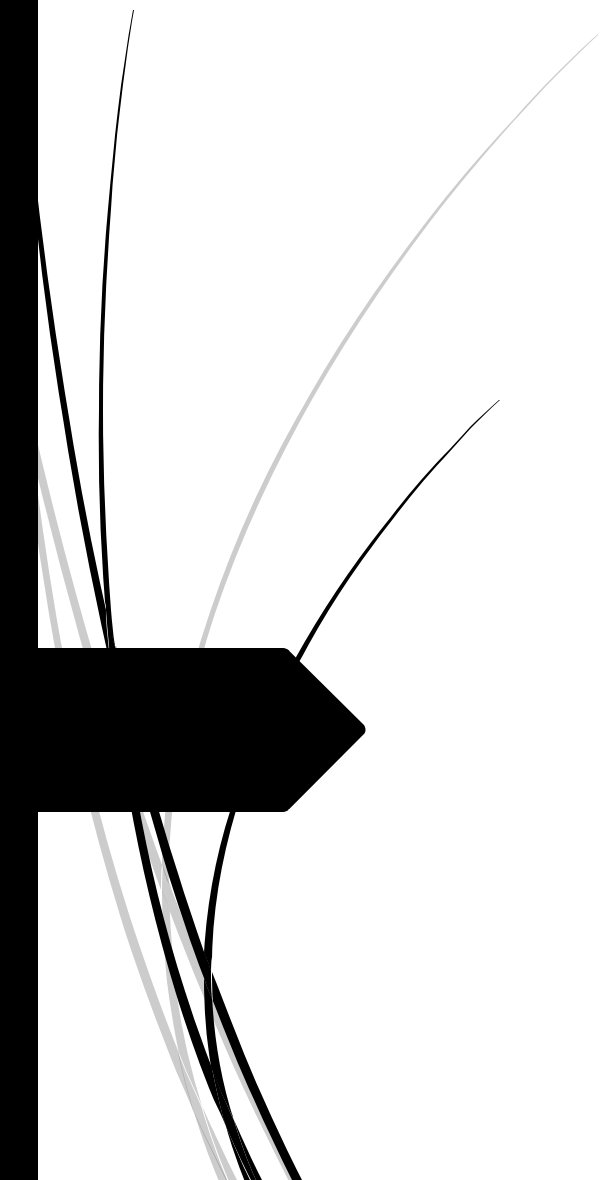
- **Switch statement**
- Re-arranged form of else-if ladder
- It consist of
  - switch variable
  - Constant case values
  - Case statements
  - Break statement
  - And a default statement

Switch statement

- Syntax :

```
switch (variable)
 {
        case value1 : statement 1; break;
        case value2 : statement 2; break;
        case value3 : statement 3; break;

        default : default statement

}
```


expression
case 1
code block 1
case 2
code block 2
case 3
code block 3
default
code block N



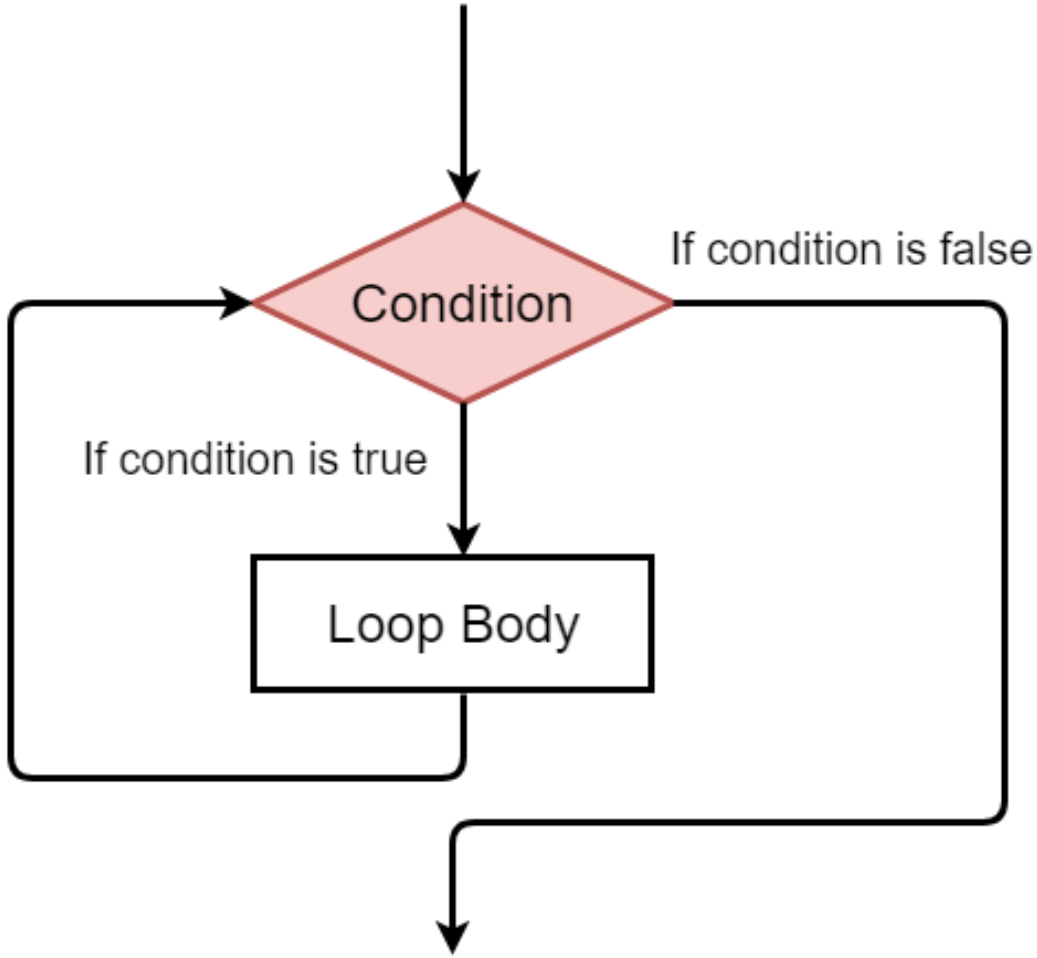
Iteration / Loop Statements

- **Looping** statement / **Iteration** statement
- Used to execute a block of statement at multiple times
  - **while** statement
  - **do-while** statement
  - **for** statement
- 2 types loops
  - **Pre-test** : loop condition appear at beginning. (while, for)
  - **Post-test** : loop condition appear at the end of the loop (do-while)

while loop

- It is an entry controlled loop (pre-test)
- Syntax :

```
        while (condition)   <-- Decision making condition
Start   {
            // block of code to be executed if the condition is true
End     }
```


Condition
If condition is false
If condition is true
Loop Body

```
while (condition)
{

    // statement part

}
```

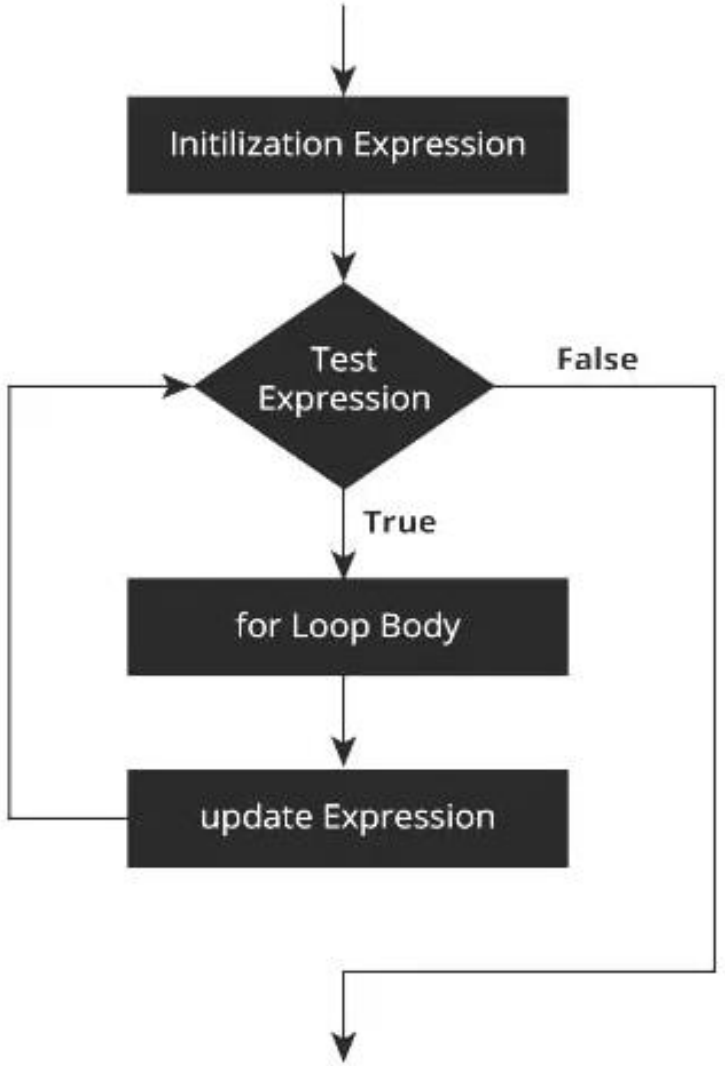
Steps

1. Condition check
2. If it is true, the statement part executed.
3. Repeat the statement part.
4. Exit from the loop whenever the condition goes to fault state

for loop

- It is an entry controlled loop (pre-test)
- For loop is a repetition control statement.
- Syntax :

```
        for (initialization ; condition ; update)
Start → {
            // body of the for loop
End →   }
```


Initilization Expression
Test Expression
False
True
for Loop Body
update Expression

```
for (initialization ; condition ; update)
 {

      // body of the for loop

 }
```

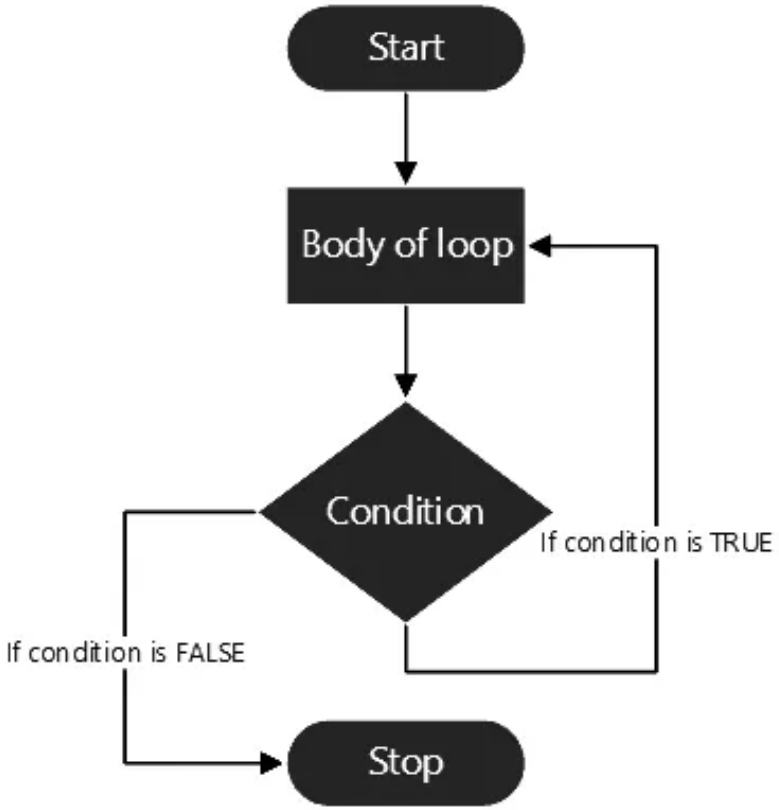
Steps

1. Initiate the variable
2. Check the condition
3. If it is true, execute the body part at onetime.
4. Update the variable by 1
5. Again check the condition, and repeat the step until the condition is false

do-while loop

- It is an exit controlled loop (post-test)
- The statement will be executed at ones without depending the condition
- Syntax :

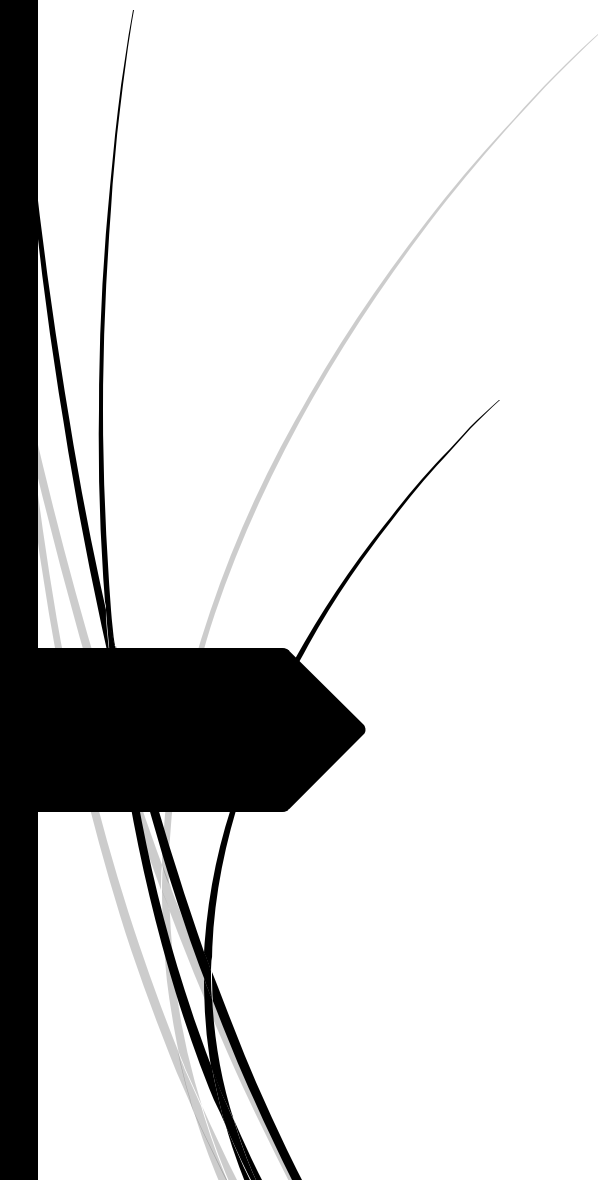
```
        do
Start → {

            // body of the for loop

End →   } (condition);
```


Start
Body of loop
Condition
If condition is TRUE
If condition is FALSE
Stop

```
do
 {

      // body of the for loop

 } (condition);
```

Steps

1. Execute the body part at ones
2. Check the condition
3. If it is true, repeat the loop
4. If it is false, end the loop

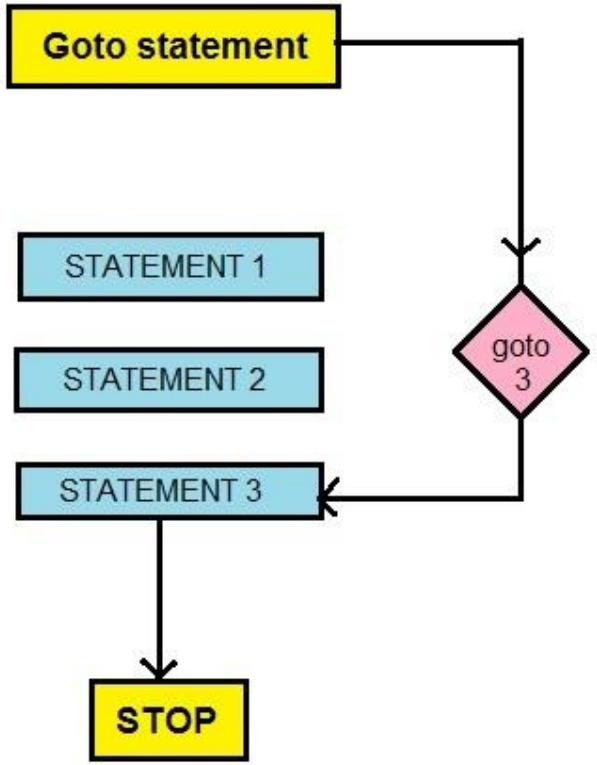


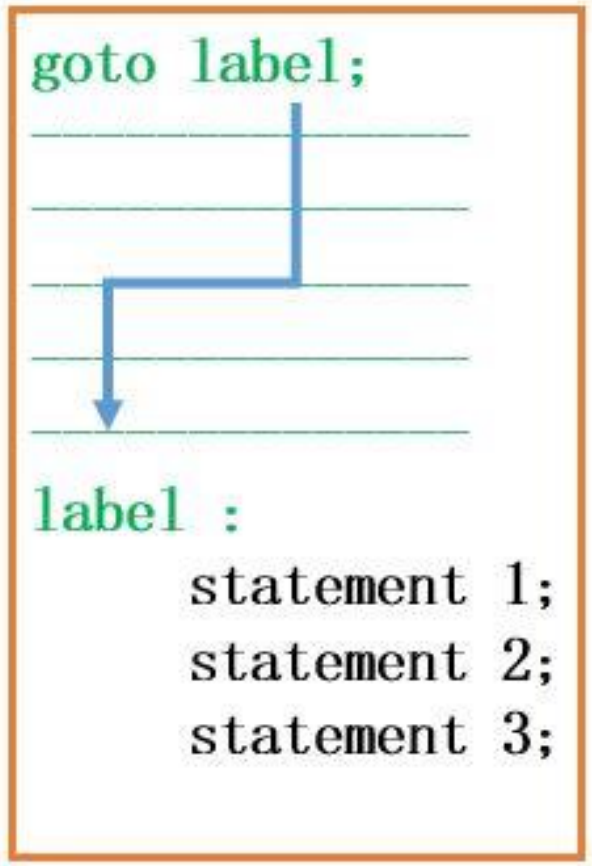
Jump Statements

- Jump statement
- Used to transfer the program control with in a function unconditionally
  - goto
  - break
  - continue

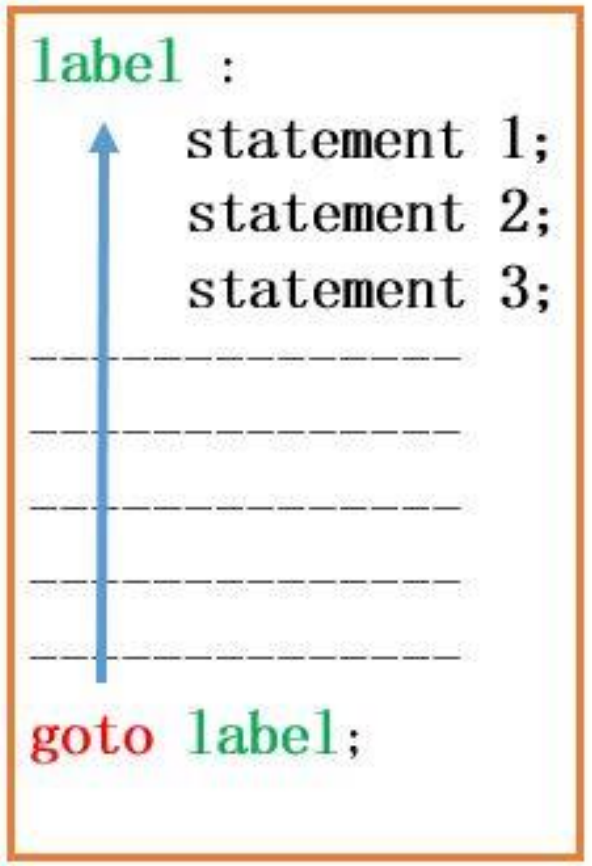
- **goto**
  - Can be used any ware in the program
  - It transfer the control from one part to another part of the program without any condition
  - A goto statement can be do both forward and backward reference
  - Syntax :

```
goto label ;
```


Goto statement
STATEMENT 1
STATEMENT 2
STATEMENT 3
goto
3
STOP



Forward Reference



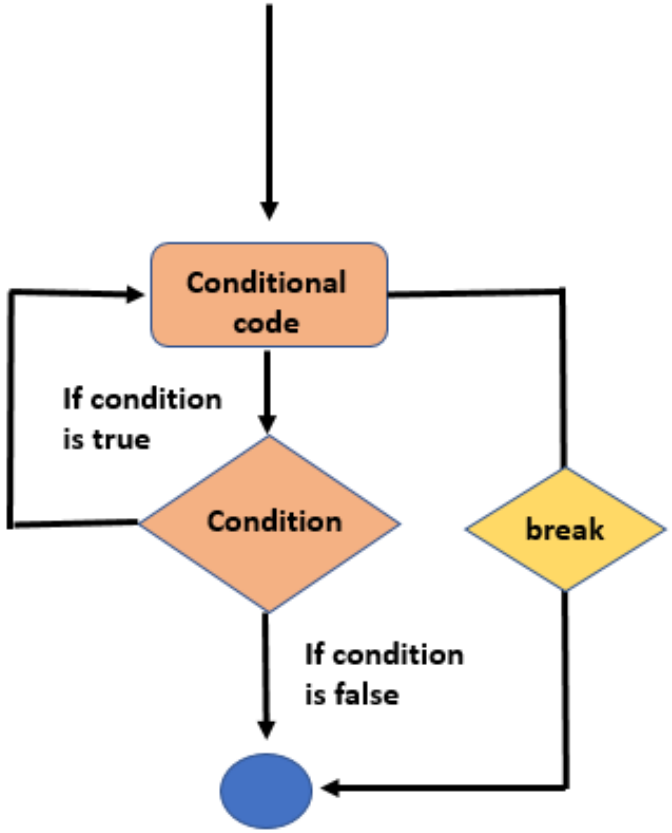
Backward Reference

Break

- used in loop and switch statements
- It is used to exit immediately from a loop or switch statement
- It can be used to end an infinite loop

- Syntax :

```
break ;
```

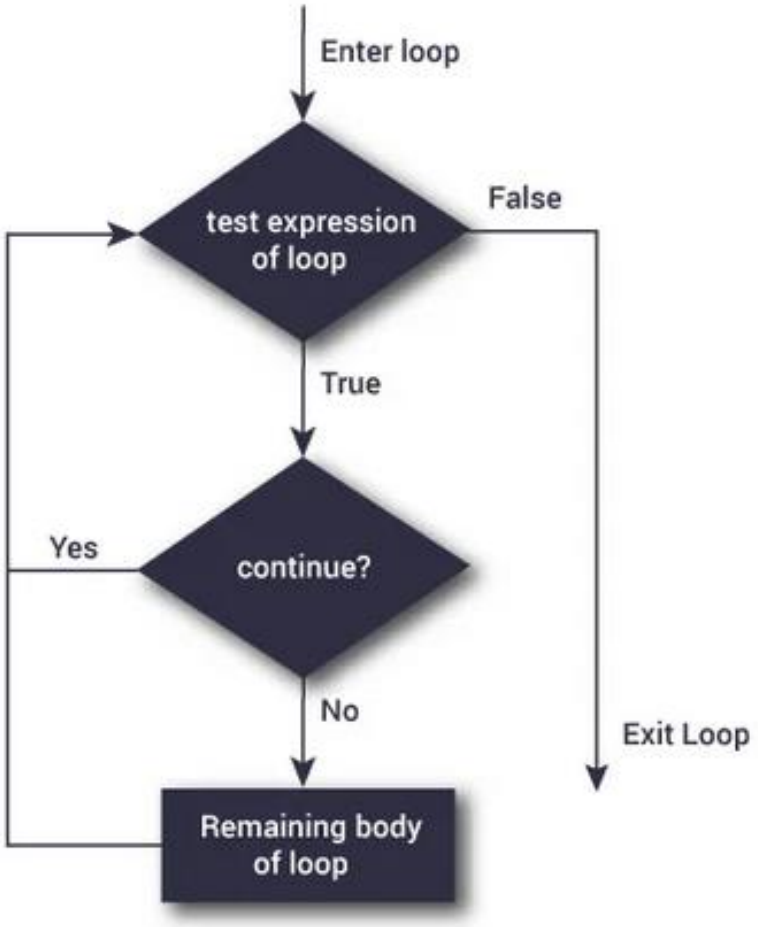

Conditional code
If condition is true
Condition
break
If condition is false

continue

- It is used to continue a program immediately
- Also used to skip the iteration of loop
- Opposite action of break statement

- Syntax :

```
continue ;
```


Enter loop
test expression of loop
False
True
Yes
continue?
No
Exit Loop
Remaining body of loop

Teaching materials Prepared by

RIDHUN DEV

Lecture IHRD College

+1 / +2 Computer Science

8089552581