Control and Looping

if-else Statement:

if Statement: if statement is a decision making statement. The general from of if statement is as follows.

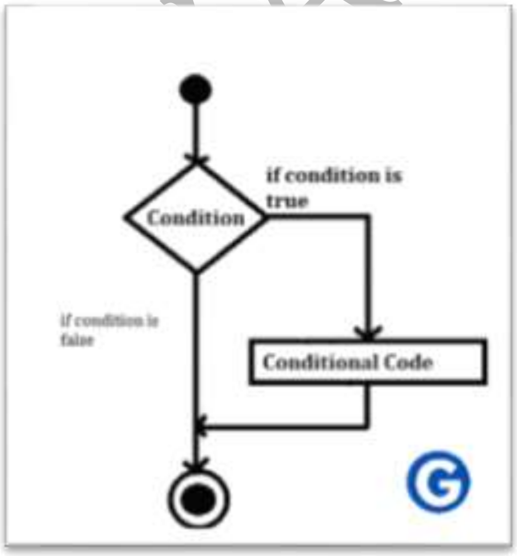
syntax:

```
if(expression)
{
  //code to be executed.
}
```

How to Works:

- The expression is true, then the statement are executed.
- The expression is false, then the statement are not executed.

Flow Diagram



If there is only one statement to be executed with if statement. Curly brackets ({}) are not required. When multiple statements are to be executed, all the statements are enclosed within a pair of brackets.

Example:

```
#include<stdio.h>
{
        int a;
        clrscr();
        printf("Enter Number: ");
        scanf("%d", &a);
        if(a>40)
        {
                printf("This Number greater then 20.\n");
        }
        getch();
}
```

Output:

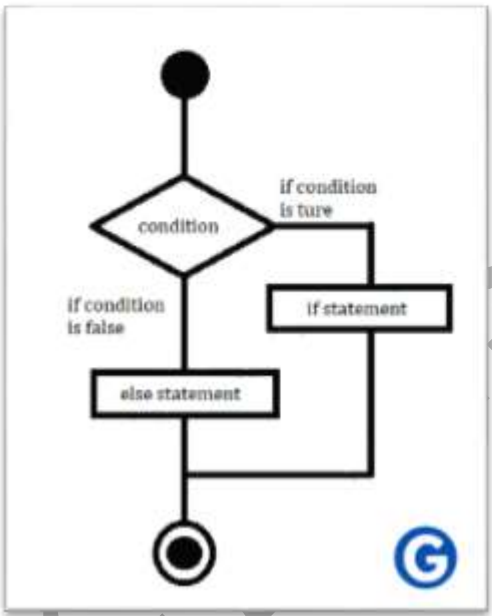
```
Enter Number: 30
This Number greater than 20.
```

if-else Statement: if-else statement is more general as compared to if statement. In if-else statement of the boolean expression return true then the statements in "if block" are executed but boolean expression not return true then the statements in "else block" are executed.

Syntax

```
if(boolean_expression)
{
        //run code if test expression is true.
}
else
{
        //run code if test expression is false.
}
```

Flow Diagram:



Example:

```
#include<stdio.h>
{
        int a;
        clrscr();
        printf(“Enter Number: ”);
        scanf(“%d”, &a);
        if(a>40)
        {
                printf(“This Number greater than 20.\n”);
        }
        else
        {
                printf(“This Number not greater than 20.”);
        }
        getch();
}
```

Output:

```
Enter Number: 50
This Number greater than 20.
Enter Number: 4
This Number not greater than 20.
```

Nested if-else statement: When if else statement is present inside the body of another “if” or “else” then this called as nested if else.

Syntax:

```
if(condition)
{
        //Nested if else inside the body of “if”
        if(conditionsecond)
        {
                //statement inside the body of nested "if"
        }
        else
        {
                //statement inside the body of nested "else"
        }
        else
        {
                //statements inside the body of "else"
        }
}
```

Example:

```
#include<stdio.h>
int main()
{
   int num=5;
   printf("Enter the value: ");
   scanf("%d", &num);
   if(num<100)
   {
      if(num==5)
      {
         printf("The Value is Equal to the 5");
      }
      else
```

```
    {
       printf("The value is Not Equal  to the 5");
    }
  }
  else
  {
     printf("The value is greater than 10");
  }
 return 0;
}
```

Output:

```
Enter the value: 20

The value is greater than 10

Enter the value: 3

The value is Not Equal to the 5

Enter the value: 5

The value is Equal to the 5
```

else-if ladder Statement: It is used when more number of condition are involved in the given problem. The expressions are evaluated from top to bottom. When expression is found to be true, the related statement are executed and remaining part is skipped. If none of the expression returns true, no action takes place.

Syntax:

```
if(condition_expression_one)

{
        statement1;
}
else if(condition_expression_two)
{
        statement2;
}
else if(condition_expression_three)
{
        statement3;
}
else
{
        statement4;
}
```

Example:

```
#include<stdio.h>
int main()
{
        int number=0;
        printf("Enter a Number:");
        scanf("%d", &number);
        if(number<=10)
        {
                printf("number is less than equal to 10");
        }
        else if(number<=50)
        {
                printf("number is less than equal to 50");
        }
        else if(number<=50)
        {
                printf("number is greater than equal to 50");
        }
        else
        {
                printf("Sorry sir/Mam");
        }
        return 0;
}
```

Output:

```
Enter a Number: 4
number is less than equal to 10
Enter a Number: 30
number is less than equal to 50
Enter a Number: 60
number is greater than equal to 50
```

For Loop

Loop Control Structure:

In c language, loop control structure is also called as iteration statements or decision making statements.

In programming, a loop is used to repeat a block of code until the specified condition is met.

They are three types of loops:

1. for loop
2. while loop
3. do-while loop

1. for Loop

A control flow statement for specifying iteration. In the loop initialization, test expression and update take place in one statement.

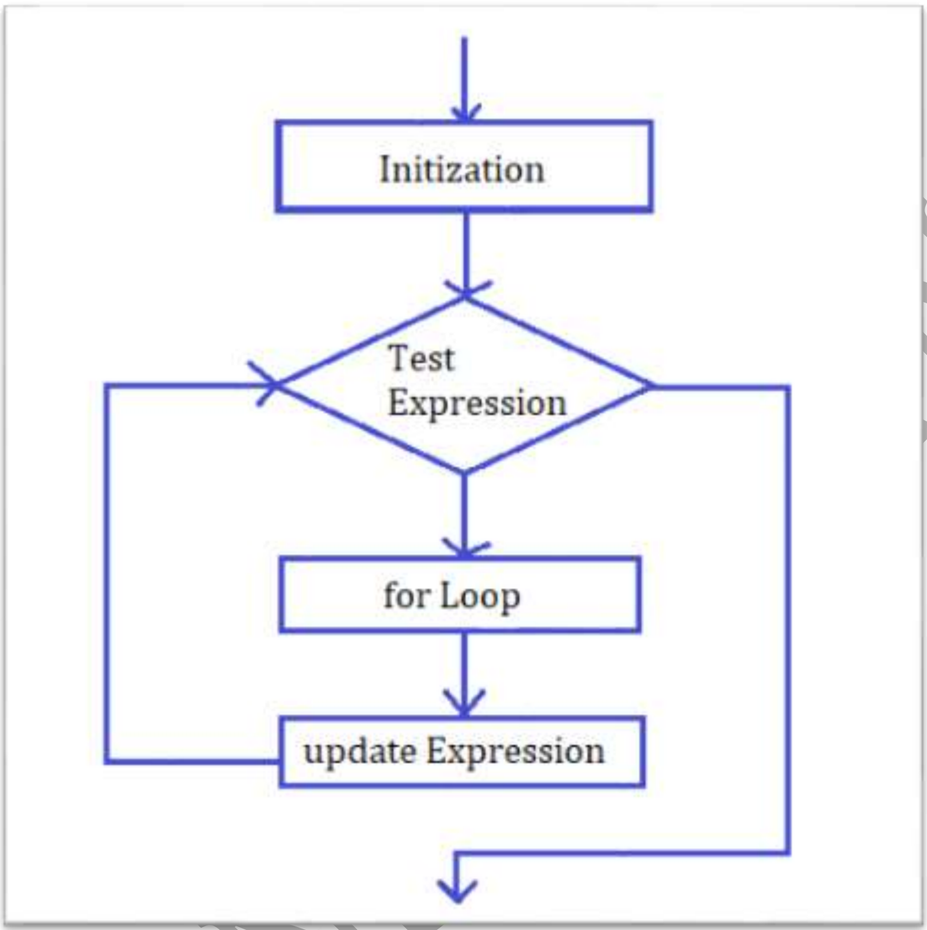
Syntax

```
for (initialization; testexpression; update)
{
   // statements inside the body of loop
}
```

How to work:

- The initialization is executes only once.
- test expression are specifies the condition required for the loop to continue. More than one condition can be used. When condition fails, loop stops.
- Update is evaluated after every iteration of the loop.
- Again the test expression is evaluated.

for Loop Flowchart:



Example:

```
#include <stdio.h>

int main()

{

  int i, n;

  printf("Enter a number: ");

  scanf("%d", &n);
  for (i = 1; i <=n; i++)
  {
    printf("%d ", i);
  }
  return 0;
}
```

Output:

```
Enter a number: 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

2. while Loop:

It is used for executing a block of statement repeatedly until a given condition return false.

Syntax:

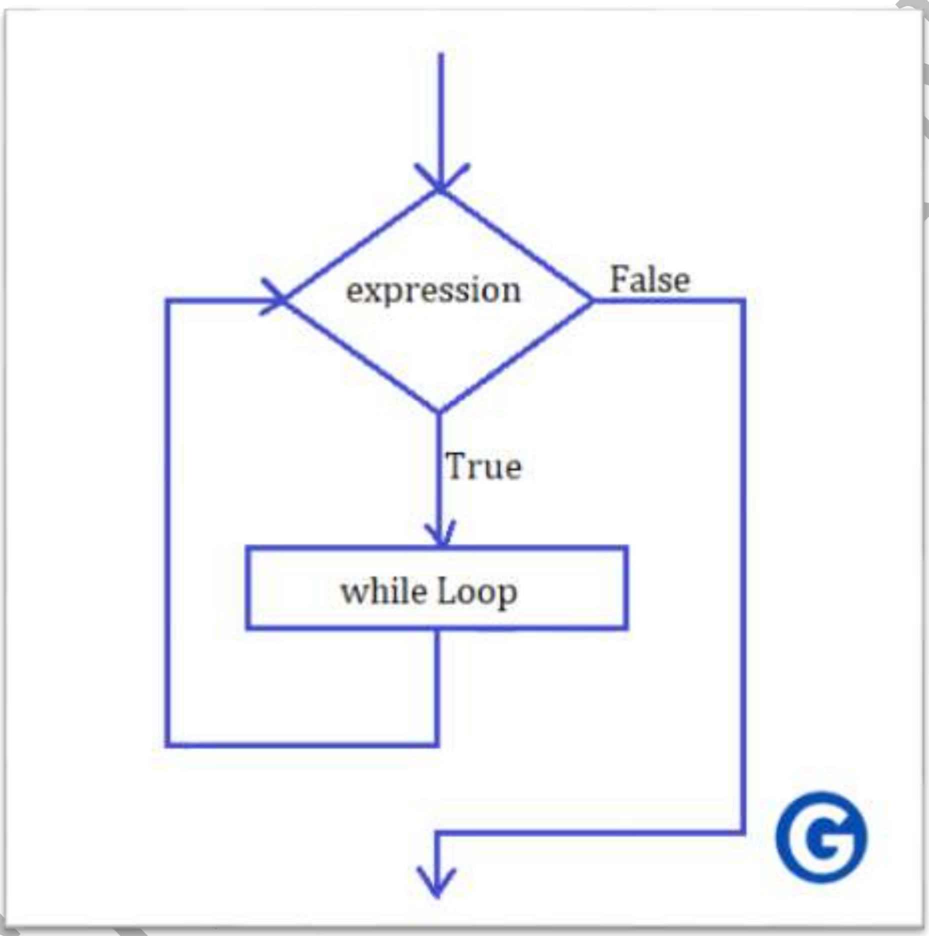
```
while (expression) {
  // the body of the loop
}
```

How to Work:

- The while loop evaluates the expression.
- If expression is true then statements inside the body of while loop are executed. Then expression is evaluates again.

- Again the expression is evaluates. This procedure repeats until the expression returns false.
- If expression is false, the loop ends

Flowchart of while loop:



Example:

```
#include <stdio.h>

int main()

{
    int i=1, n=0;
    printf("Enter the Number: ");
    scanf("%d", &n);
    while (i <= n)
    {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Output:

```
Enter the Number: 10
1
2
3
4
5
6
7
8
9
10
```

Note: The body of loop must contain a statement which alters the value of the control variable. Otherwise loop becomes infinite and is not terminated.

3. do-while Loop

The body of do-while loop is executed at least once. Only then, the expression is evaluated.

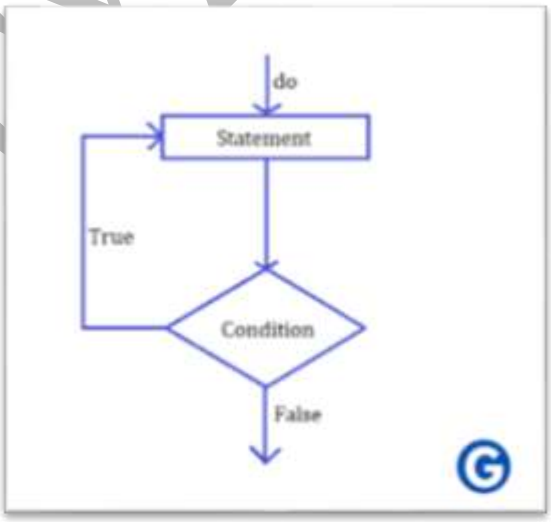
Syntax:

```
do {
  // the body of the loop
}
while (expression);
```

How to Works:

- If expression is true then program loop is executed again and expression is evaluated once more.
- The process goes on until expression is evaluated to false. if once expression is false, the loop end (terminates).

Flowchart of do-while Loop



Example:

```
#include<stdio.h>

int main()

{

   int i=1;

   do

   {

      printf("%d \n",i);

      i++;

   }

   while(i<=3);

return 0;

}
```

Output:

```
1

2

3
```

GYANSABHA

break and continue

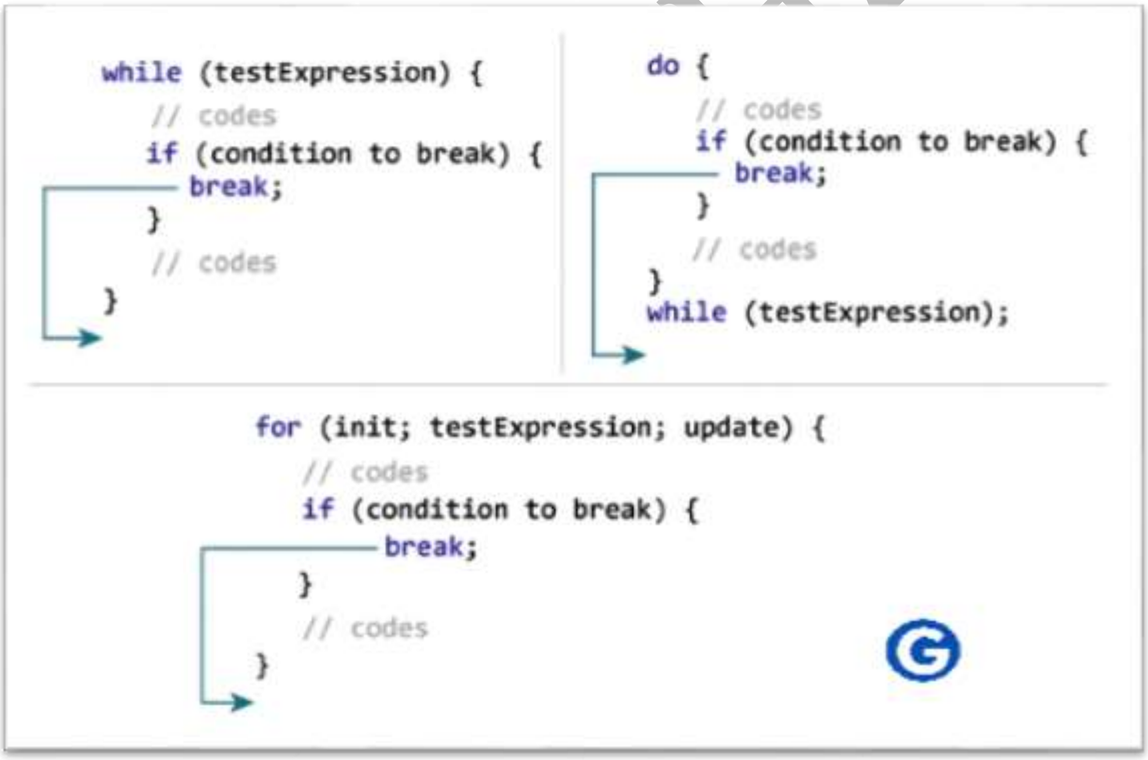
Break:

The **break** statement ends the loop immediately when it is encountered. (i.e., while, do-while, for loop or switch statement).

Syntax:

```
break;
```

How to Works:



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

Example:

```
#include<stdio.h>

void main ()

{

   int i;

   for(i = 0; i<10; i++)

   {

      printf("%d ",i);

      if(i == 5)

      break;

   }

   printf("\n");

   printf("Came outside of loop i = %d", i);

}
```

Output:

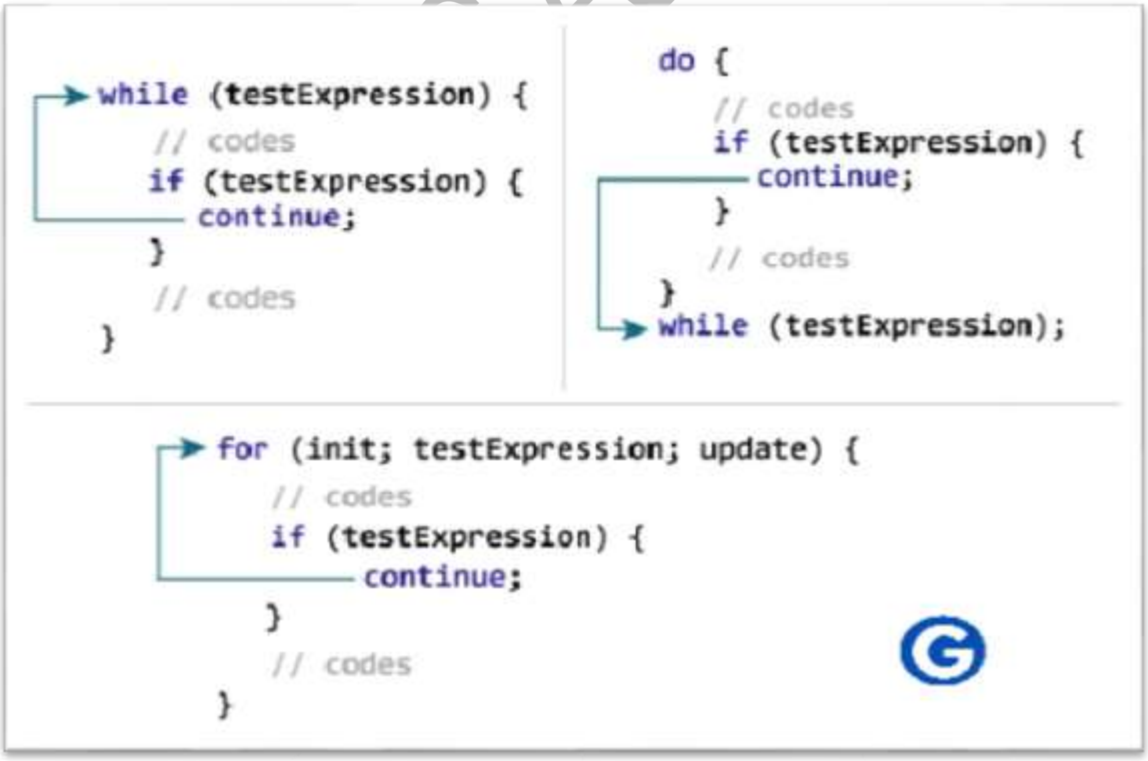
```
0 1 2 3 4 5

Came outside of loop i = 5
```

Continue:

The **continue** statement request the complier to skip the following statement and continue from the first statement of loop. It causes the control to go directly to the test condition and then continue the lop process. On encountering continue, cursor leave the current cycle of loop and start with the next cycle. Continue statement is usually associated with **if statement.**

When continue statement is encountered in a looping statement, the execution control skips the rest of the statement in the looping block and directly jumps to the beginning of the loop. The continue statement can be used with looping statements like, while, do-while and for loop.

How to Works:



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Example:

```
#include<stdio.h>
int main()
{
   int i=0;
   for(i=0;i<=10;i++)
   {
      if(i==5)
      {
         continue;
      }
      printf("%d \n", i);
   }
}
```

Output:

```
0
1
2
3
4
6
7
8
9
10
```

Switch Case

A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifier. The value of provided by the user is compared with all the cases inside the switch block until the match is found. If a case match is not found, then the default statement is executed.

You can do the same thing with the if...else..if ladder.

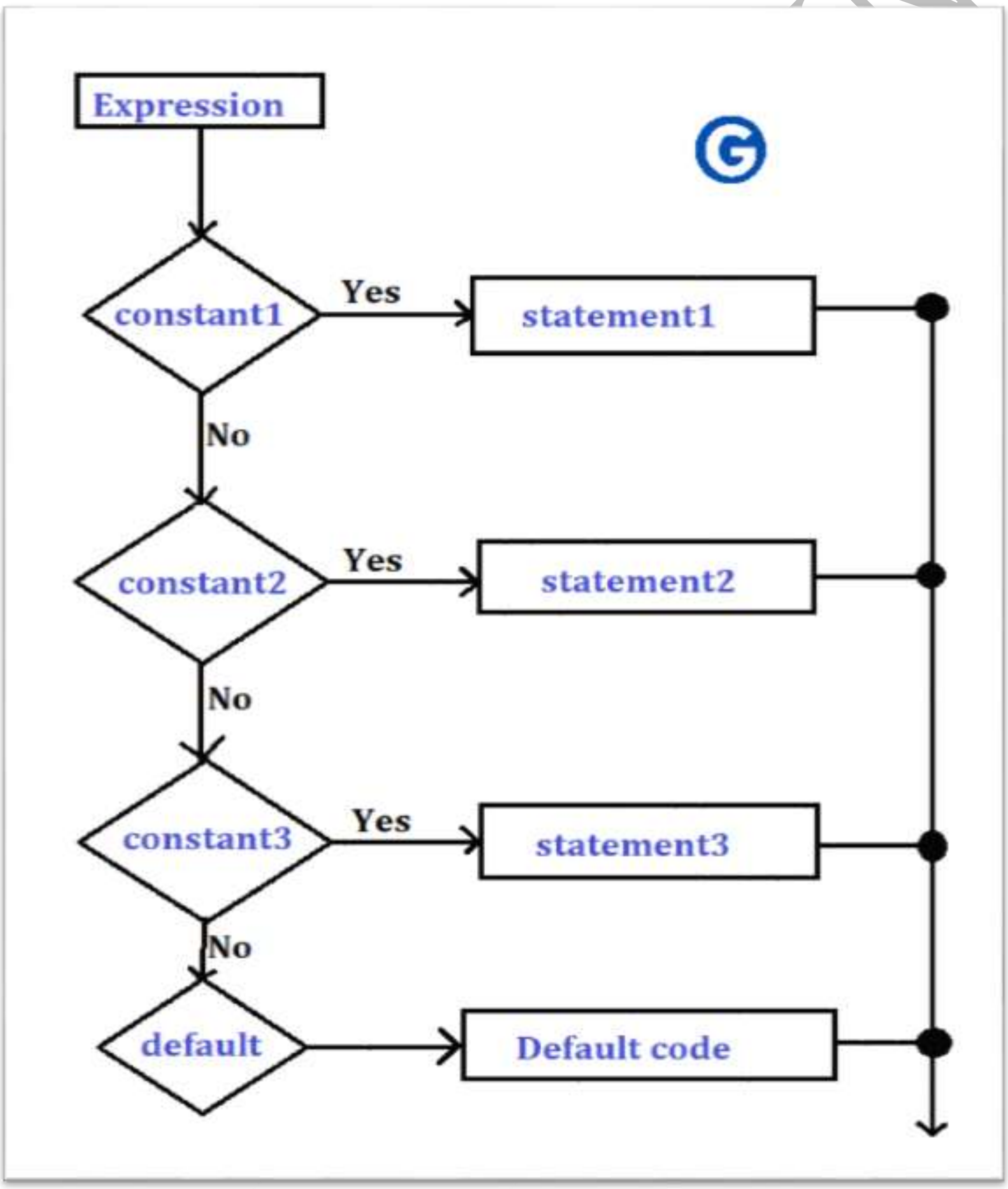
Syntax:

```
switch (expression)
{
   case constant1:
     // statements
     break;
   case constant2:
     // statements
     break;
   .
   .
   default:
     // default statements
}
```

How to Works:

- If there is a match, the corresponding statements after the matching label are executed.
- If a case match is not found, then the default statement is executed.

Flowchart:



Example:

```
#include <stdio.h>

void main()

{

   int num=0;

   printf("Gyansabha Employees:");

   scanf("%d", &num);

   switch(num)

   {

      case 1:

      printf("Founder and CEO Of Gyansabha");

      break;

      case 2:

      printf("CFO Of Gyansabha");

      break;

      case 3:

      printf("CTO Of Gyansabha");

      break;

      default:

      printf("For Better Future");
   }


}
```

Output:

```
Gyansabha Emplyoess:1

Founder and CEO Of Gyansabha
```

goto statement

goto statement is an unconditional branching statement. It is used to jump from one statement to another statement.

Syntax:

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

Example:

```
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Table you want to print?\n");
  printf("Enter the Number:");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

Output:

```
Table you want to print?

Enter the Number:10

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50

10 x 6 = 60

10 x 7 = 70

10 x 8 = 80

10 x 9 = 90

10 x 10 = 100
```