C

UNIT 2

Decision Making in C
In C, programs can choose which part of the code to execute based on some condition. This ability is called **decision making** and the statements used for it are called **conditional statements.**
These statements evaluate one or more conditions and make the decision whether to execute a block of code or not.

C - if Statement
The **if in C** is the simplest decision-making statement. It consists of the test condition and a block of code that is executed if and only if the given condition is true. Otherwise, it is skipped from execution.

Syntax :-
```
if (condition) {
// if body
// Statements to execute if condition is true
}
```

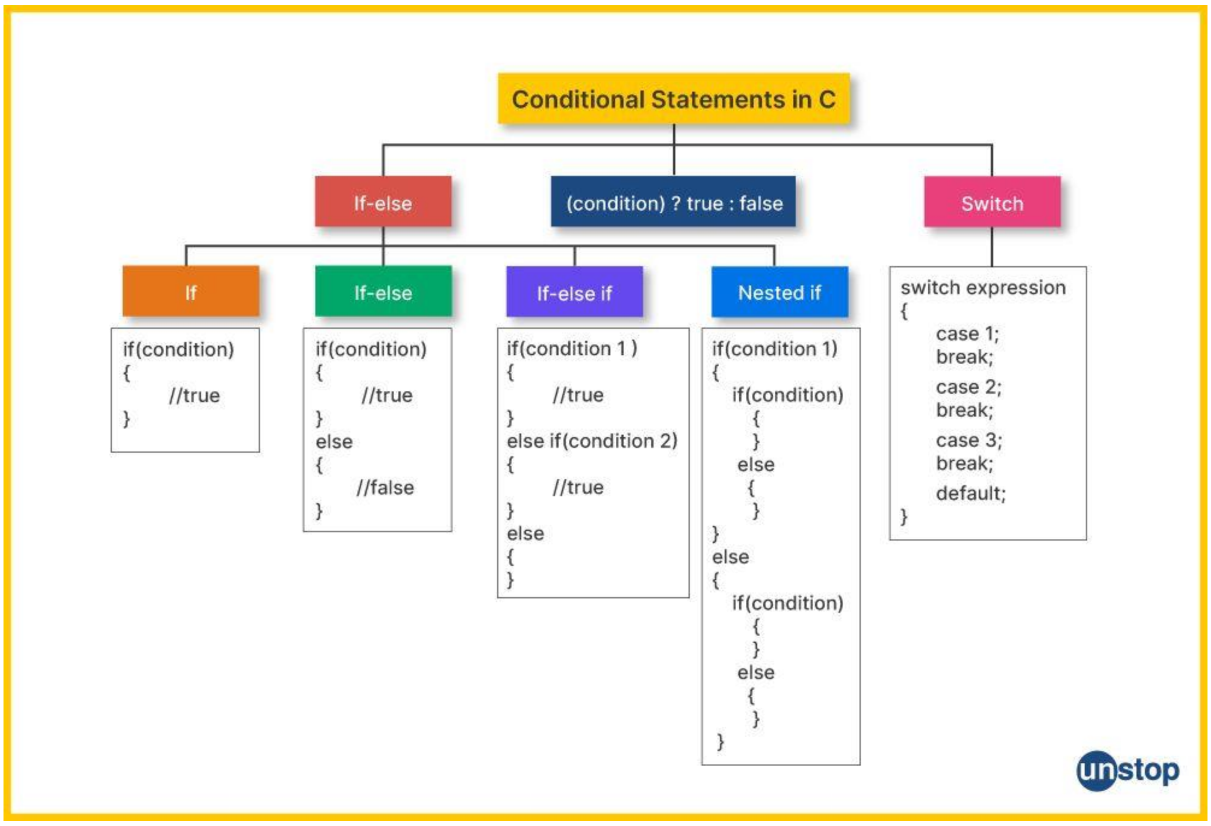
2. if-else in C
The if-else statement consists of two blocks, one for false expression and one for true expression.

Syntax :-
```
if (condition) {
// if body
// Statements to execute if condition is true
}
else{
        //false block of code
}
```

3. if-else-if in C

Syntax :-
```
if (condition) {
// if body Statements to execute if condition is true
}
else if (condition) {
        //true block of code
}
else{

}
```
3. Nested if-else in C:-
Nested if statements mean an if statement inside another if statement.


Conditional Statements in C
If-else
(condition) ? true : false
Switch
If
If-else
If-else if
Nested if
if(condition) { //true }
if(condition) { //true } else { //false }
if(condition 1) { //true } else if(condition 2) { //true } else { }
if(condition 1) { if(condition) { } else { } } else { if(condition) { } else { } }
switch expression { case 1; break; case 2; break; case 3; break; default; }
unstop

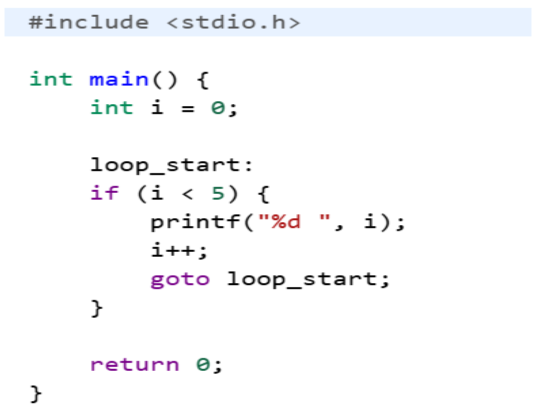
goto statement in C:-

The goto statement in C is a control flow statement that allows for an unconditional jump to a specified label within the same function.

Syntax:-

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

Example:-



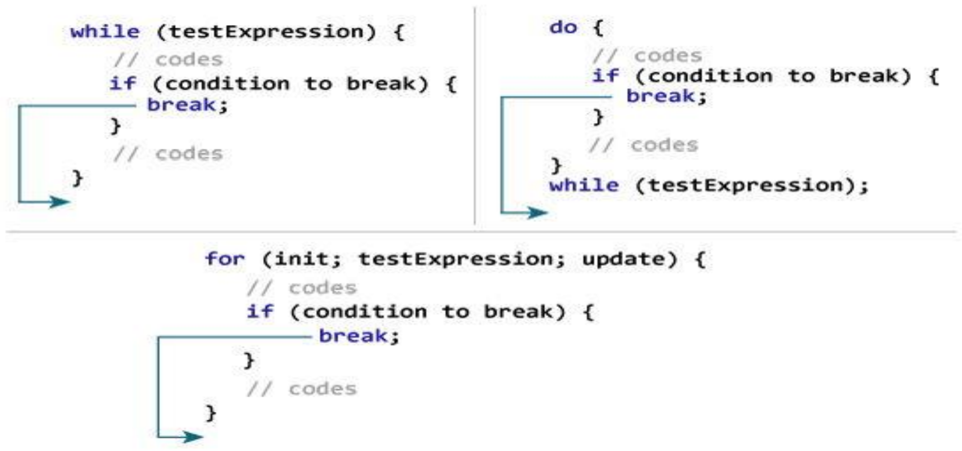
```
#include <stdio.h>

int main() {
    int i = 0;

    loop_start:
    if (i < 5) {
        printf("%d ", i);
        i++;
        goto loop_start;
    }

    return 0;
}
```

Break Statement in C
The break statement in C is a loop control statement that breaks out of the loop when encountered. It can be used inside loops or switch statements to bring the control out of the block. The break statement can only break out of a single loop at a time.



```
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
}
```

What are loops in C?
Loops in C programming are used to repeat a block of code until the specified condition is met. It allows programmers to execute a statement or group of statements multiple times without writing the code again and again.

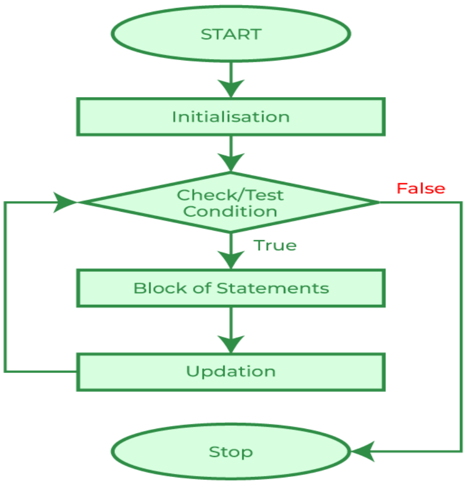
for Loop
for loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times. It enables programmers to perform n number of repetitions in a single line. for loop is **entry-controlled** loop, which means that the condition is checked before the loop's body executes.

Syntax:-
for (initialization; condition; increment/decrement) {

    // body of for loop

}

Flowchart:-

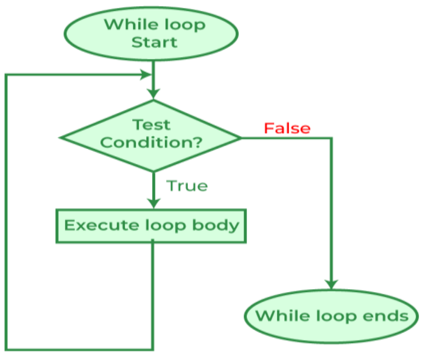


while Loop:-
A **while loop** is a block of code to perform repeated task till given condition is true. When condition become false it terminates. while loop is also an **entry-controlled loop** in which the condition is checked before entering the body.

Example:-

```
while (condition) {
   // Body of the loop
}
```

Flowchart:-

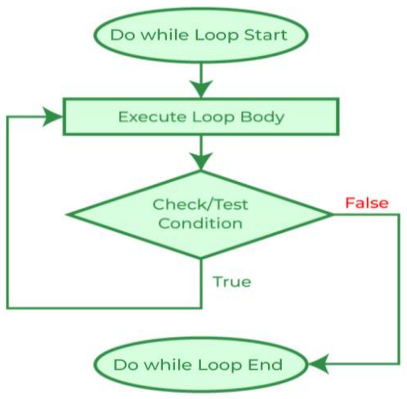


do-while Loop
The do-while loop is similar to a while loop, but the difference is that do-while loop tests condition after entering the body at the end. do-while loop is **exit-controlled loop**, which means that the condition is checked after executing the loop body. Due to this, the loop body will **execute at least once** irrespective of the test condition.

Syntax:-

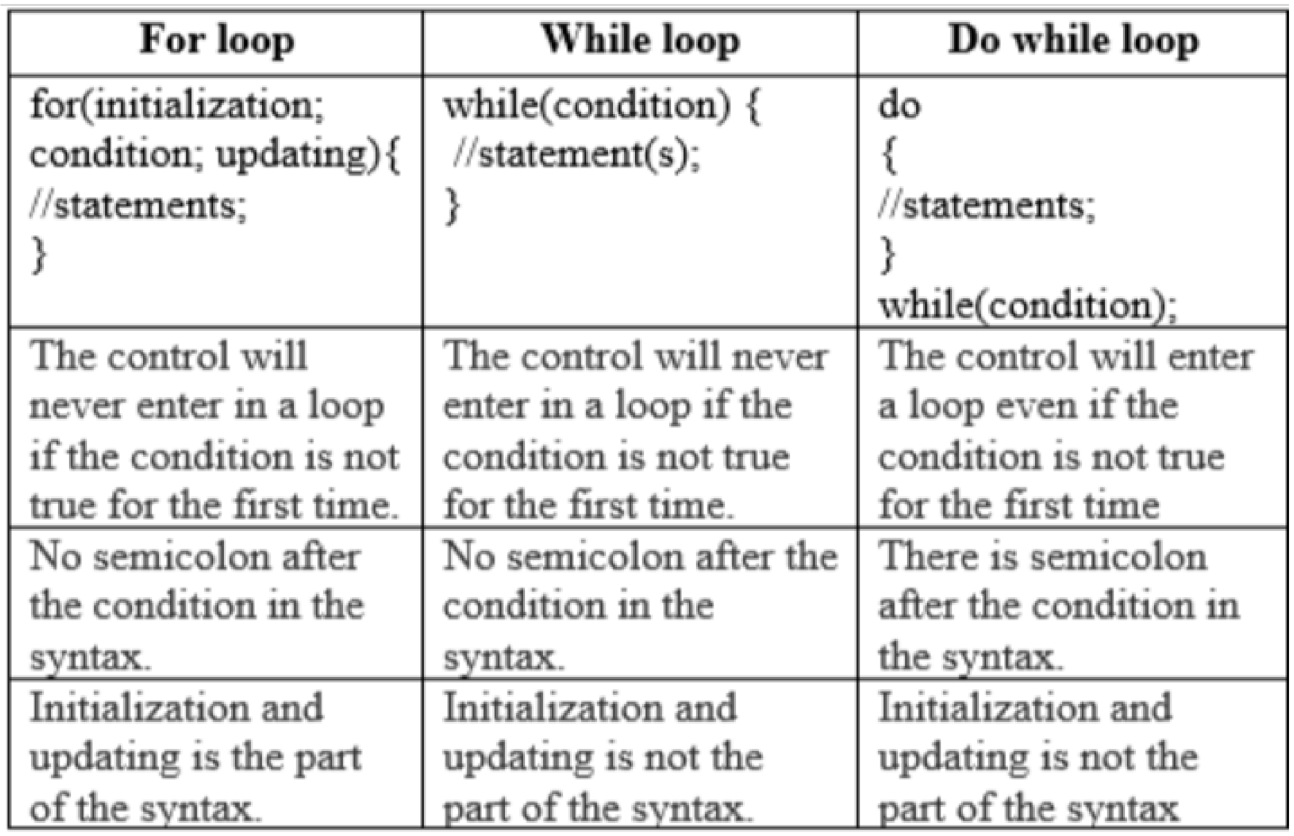
```
do {
    // Body of the loop
} while (condition);
```

Flowchart:-



Difference Between Break and Continue Statement in C

Break	Continue
Break statement stops the entire process of the loop.	Continue statement only stops the current iteration of the loop.
Break also terminates the remaining iterations.	Continue doesn’t terminate the next iterations; it resumes with the successive iterations.
Break statement can be used with switch statements and with loops	Continue statement can be used with loops but not switch statements.
In the break statement, the control exits from the loop.	In the continue statement, the control remains within the loop.
It is used to stop the execution of the loop at a specific condition.	It is used to skip a particular iteration of the loop.



For loop	While loop	Do while loop

for(initialization;
condition; updating){
//statements;
}
	while(condition) {
	//statement(s);
	}
		do
		{
		//statements;
		}
		while(condition);

The control will never enter in a loop if the condition is not true for the first time.	The control will never enter in a loop if the condition is not true for the first time.	The control will enter a loop even if the condition is not true for the first time
No semicolon after the condition in the syntax.	No semicolon after the condition in the syntax.	There is semicolon after the condition in the syntax.
Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax

Difference between goto and continue statement:-

Feature	continue	goto
Used in	Loops only	Anywhere in a function
Control flow	Skips to next loop iteration	Jumps to a labeled statement

Example	int i;
	
	for (i = 0; i < 10; i++) {
	  if (i == 4) {
	    continue;
	  }
	  printf("%d\n", i);
	}
	
	output: 0 1 2 3 5 6 7 8 9
		int n=0;
		if(n==0)
		    goto end;
		printf(“the number is: %d”,n);
		end:
		printf(“End of program”);
		
		return 0;
		output: End of program