

Unit 2

Simple statements

- Every statement in C ends with a semicolon.

- For example:

```
int a=20;  
sum=sum+1;
```

- Even simple semicolon is considered as null statement in C programming.

Decision making Statements

C program statements are executed **sequentially**.

Decision Making statements are used to **control the flow** of program.

It allows us to control whether a program segment is executed or not.

It evaluates condition or logical expression first and based on its result (either true or false), the control is transferred to particular statement.

If result is true then it takes one path else it takes another path.

Decision making statements in C

Decision Making Statements are

One way Decision:	<code>if</code>	(Also known as simple if)
Two way Decision:	<code>if...else</code>	
Multi way Decision:	<code>if...else if...else if...else</code>	
Two way Decision:	<code>?:</code>	(Conditional Operator)
n-way Decision:	<code>switch...case</code>	

if Statement

Simple if-statement is used to control the execution sequence in the program.

If expression/condition is true then only the body of the if statement will be executed.

The structure or syntax of if statement is as follows

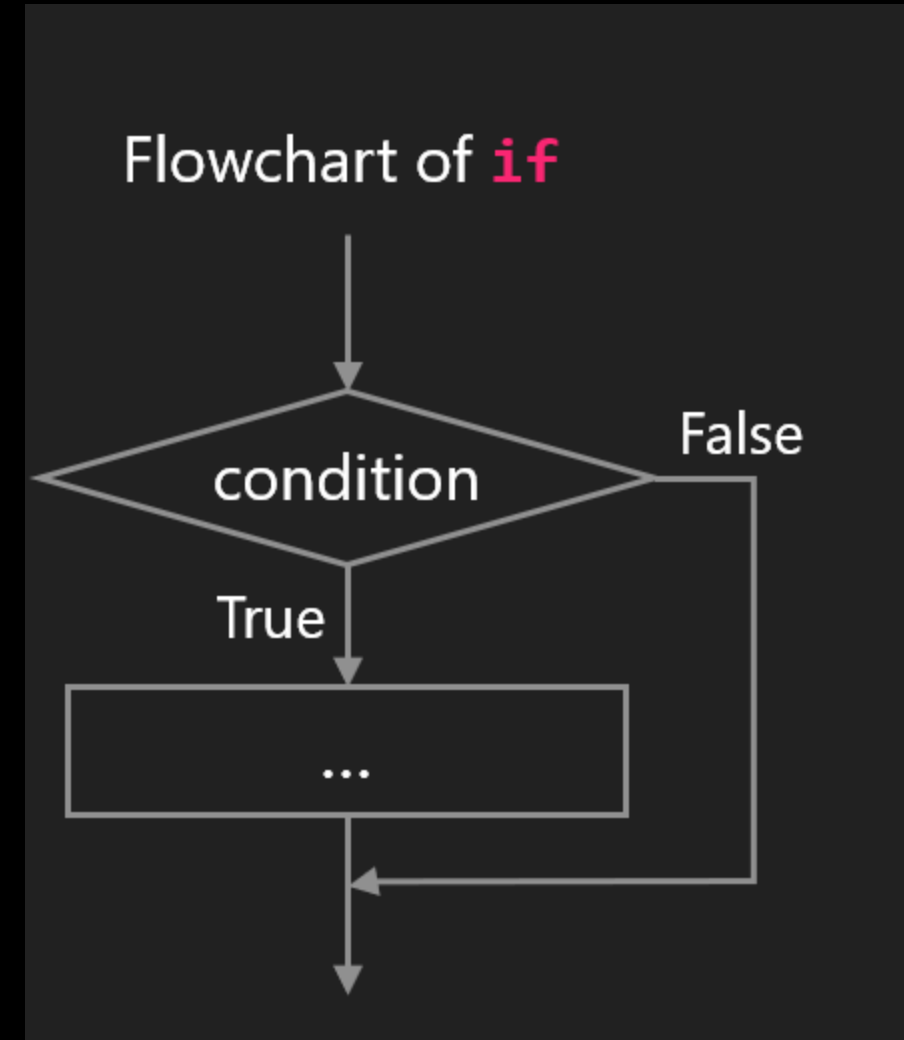
```
if(expression/condition)
```

```
{
```

```
//Body of if statement
```

```
//true block
```

```
}
```



Write a C program to print 'nine' if entered number is 9.

Write a C program to print odd or even number using simple if statement.

- `Number % 2 == 0` then the number is even else number is odd

if..else statement

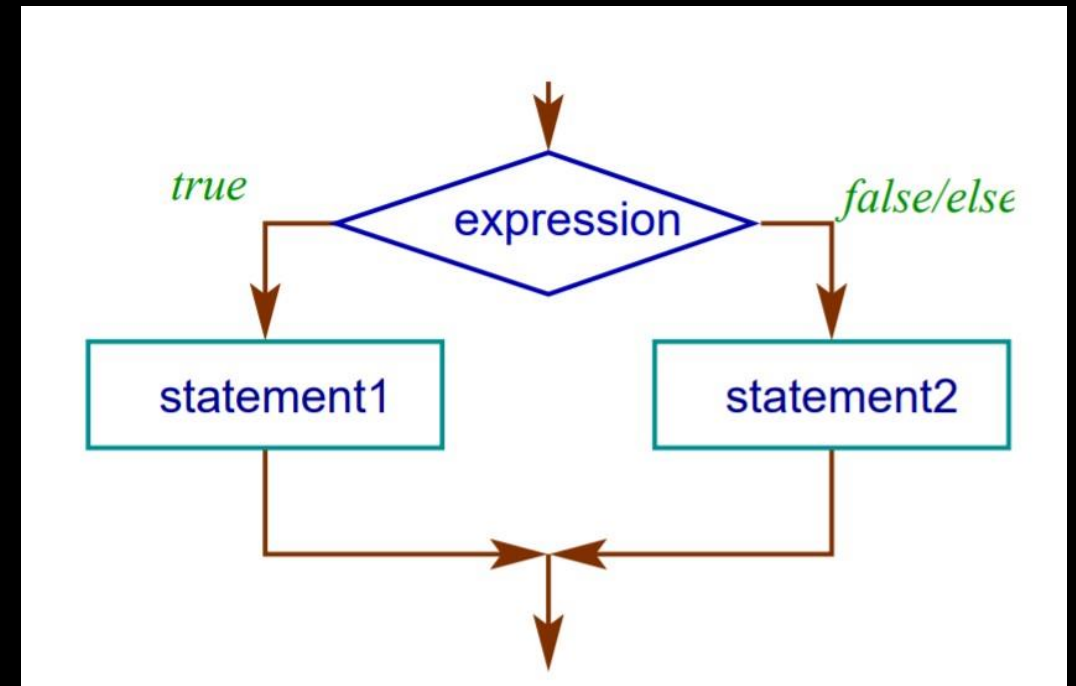
if...else is two branch decision making statement

If condition is true then true part will be executed else false part will be executed

else is keyword

Syntax

```
if(condition)
{
    // true part
}
else
{
    // false part
}
```



Write a C Program to check entered number is zero or non zero.

```
If(number == 0)
    printf("The entered number is zero.");
else
    printf("The entered number is nonzero");
```

Write a C program to check if number is divisible by 5 or not.

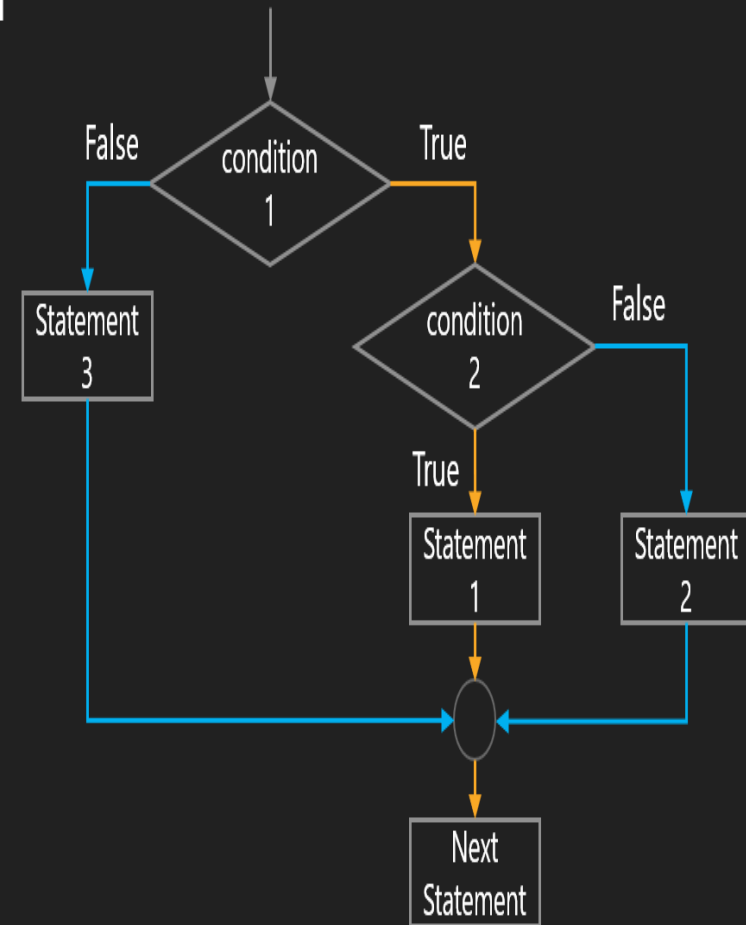
Nested if...else statement

If condition-1 is true then condition-2 is evaluated. If it is true then statement-1 will be executed.

If condition-1 is false then statement-3 will be executed.

Syntax

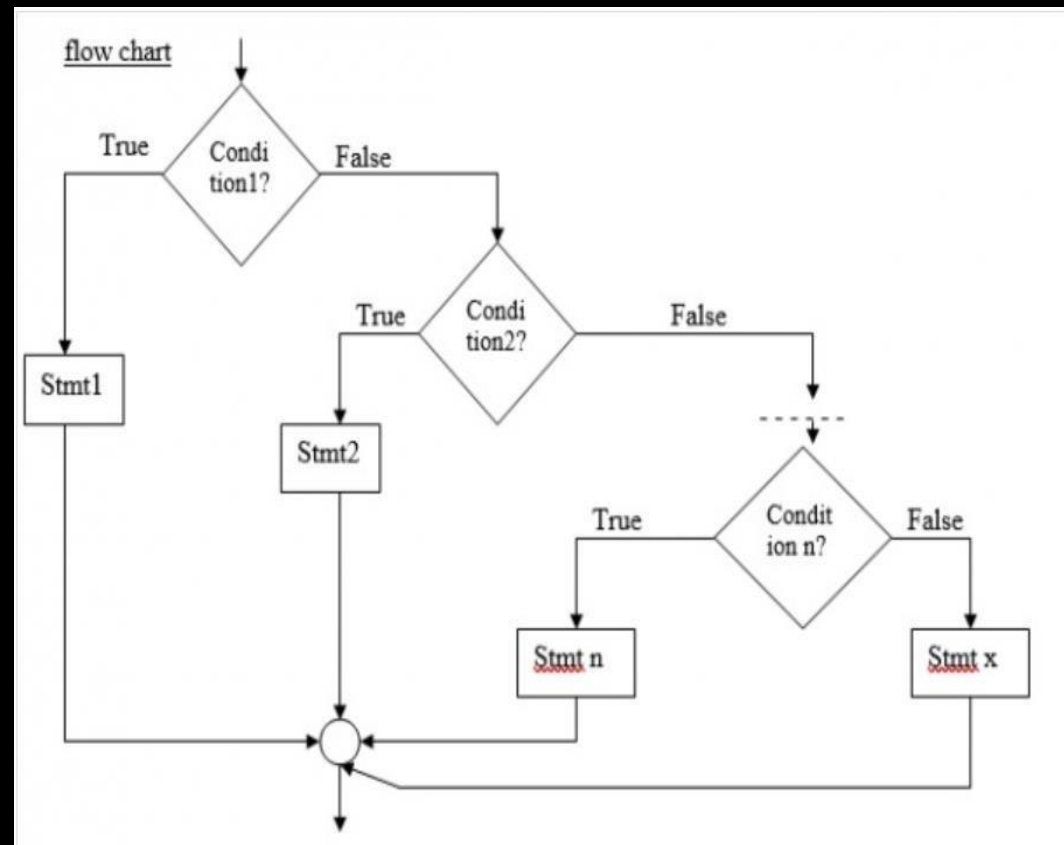
```
if(condition-1)
{
    if(condition-2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
```



If...else if...elseif...else() ladder

- A multi-way decision making statement
- Syntax

```
if (condition1)
stmt1;
else if (condition2)
stmt2;
- - - - -
- - - - -
else if (condition n)
stmtn;
else
stmt x;
```



Else if ladder example

Take customer Number and units consumed and the bill amount to be paid by the customer.

An electric power distribution company charges its domestic consumers as follows:

<i>Consumption Units</i>	<i>Rate of Charge</i>
0 - 200	Rs. 0.50 per unit
201 - 400	Rs. 100 plus Rs.0.65 per unit excess of 200
401 - 600	Rs. 230 plus Rs.0.80 per unit excess of 400
601 and above	Rs. 390 plus Rs.1.00 per unit excess of 600

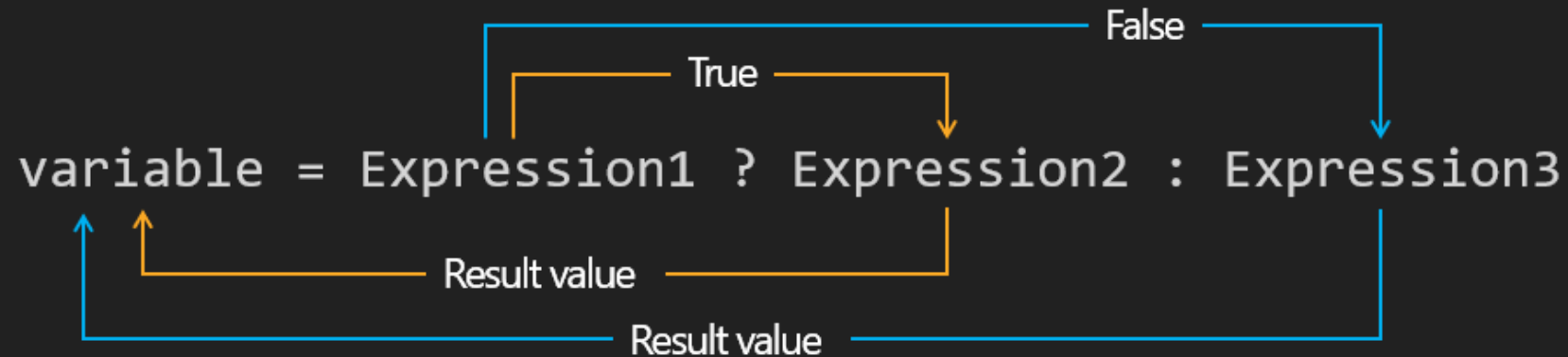
```
void main()
{
    int units, custnum;
    float charges;
    printf("Enter CUSTOMER NO. and UNITS consumed\n");
    scanf("%d %d", &custnum, &units);
    if (units <= 200)
        charges = 0.5 * units;
    else if (units <= 400)
        charges = 100 + 0.65 * (units - 200);
    else if (units <= 600)
        charges = 230 + 0.8 * (units - 400);
    else
        charges = 390 + (units - 600);
    printf("\n\nCustomer No: %d: Charges = %.2f\n", custnum, charges);
}
```

Conditional Operator(? :)

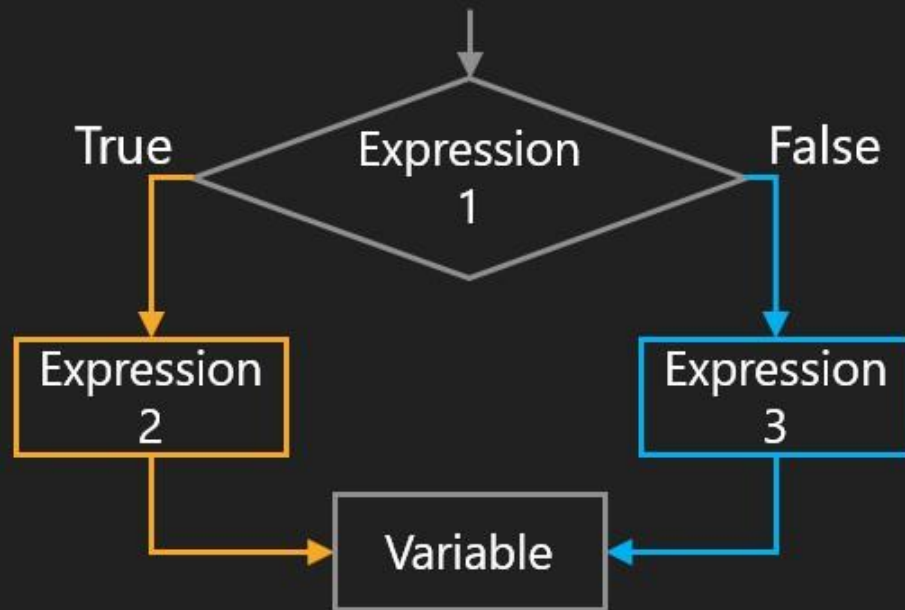
The conditional works operator is similar to the if-else.

It is also known as a **ternary operator**.

It returns first value of expression (before colon(:)) if expression is true and second value of expression if expression is false.



Conditional operator(? :) flowchart



- ▶ Here, **Expression1** is the condition to be evaluated.
- ▶ If the condition(Expression1) is **True** then Expression2 will be executed and the result will be returned.
- ▶ Otherwise, if condition(Expression1) is **false** then Expression3 will be executed and the result will be returned.

Conditional Operator Example

- An employee can apply for a loan at the beginning of every six months, but he will be sanctioned the amount according to the following company rules:
- *Rule 1* : An employee cannot enjoy more than two loans at any point of time.
- *Rule 2* : Maximum permissible total loan is limited(5000)and depends upon the category of the employee.

Switch() statement

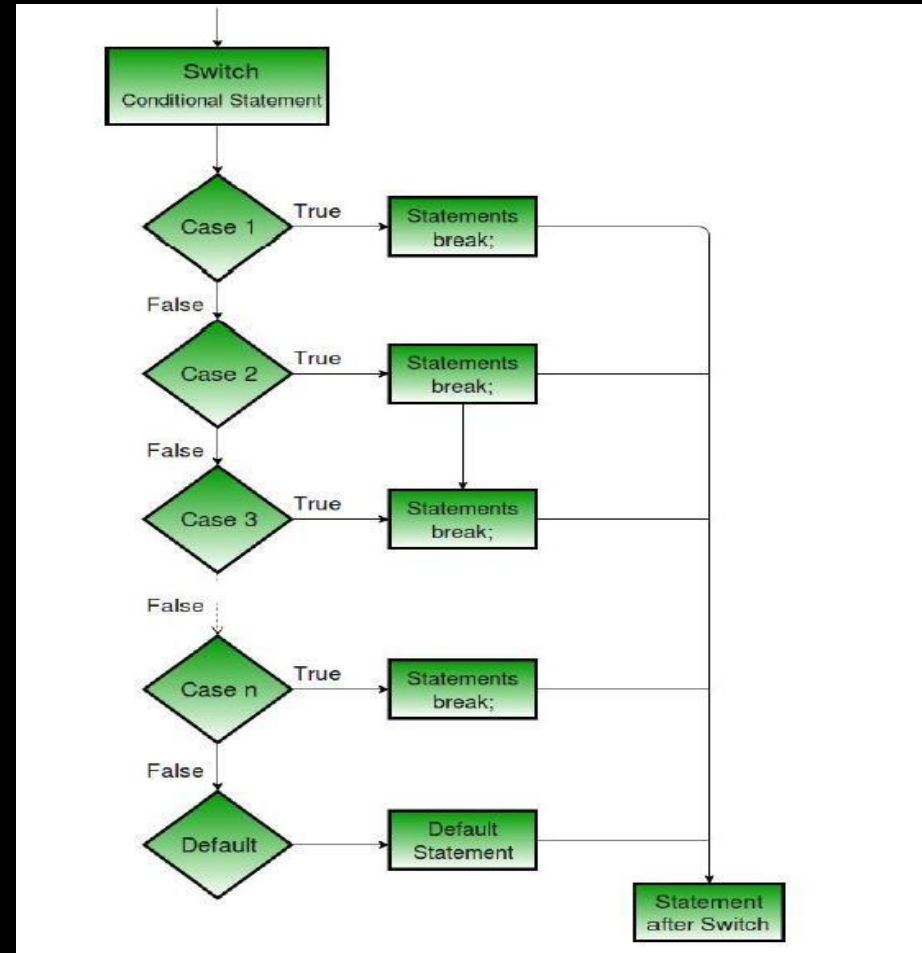
The switch statement allows to execute one code block among many alternatives. It works similar to if...else..if ladder.

Syntax

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

- ▶ The expression is evaluated once and compared with the values of each **case**.
- ▶ If there is a match, the corresponding statements after the matching **case** are executed.
- ▶ If there is no match, the **default** statements are executed.
- ▶ If we do not use **break**, all statements after the matching label are executed.
- ▶ The **default** clause inside the **switch** statement is optional.

Switch() Flowchart



Switch() Example

```
#include<stdio.h>
void main()
{
char city;
printf("Enter the city code \n");
scanf("%c", &city );
switch(city)
{
case 'V' : printf("Vadodara ");
break ;
case 'C' : printf("Calcutta ");
break ;
case 'D' : printf("Delhi ");
break ;
case 'M' : printf("Madras ");
break ;
default:printf("Invalid Option");
}
}
```

Loops in C

- When a block of code needs to be executed multiple times.
- A loop statement allows us to execute block of code multiple times.
- A **Loop** executes the sequence of statements many times until the stated condition becomes false.
- A loop consists of two parts
 - a body of a loop
 - a control statement.
- The control statement is a combination of conditions that direct the body of the loop to execute until the specified condition becomes false.
- The purpose of the loop is to repeat the same code a number of times.

Types of loops in C

1.Entry controlled loop

2. Exit controlled loop

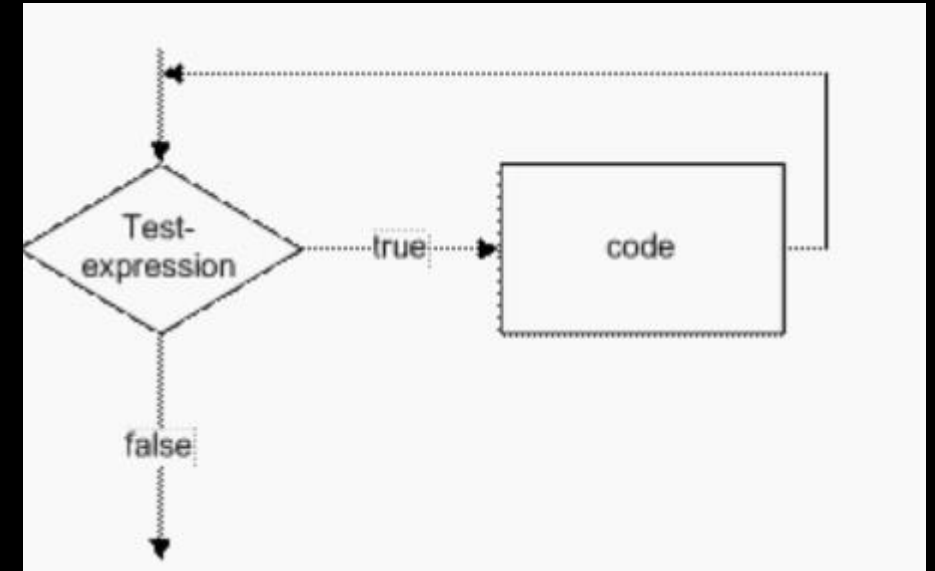
- **Entry controlled loop:** A condition is checked before executing the body of a loop. It is also called as a pre-checking loop. E.g. for(),while()
- **Exit controlled loop:** A condition is checked after executing the body of a loop. It is also called as a post-checking loop. E.g. do..while()

while() loop

- Syntax of while loop In C programming language is as follows:

```
    initialization;  
while(test expression){  
    statements/code;  
    update statements;  
}
```

- It is an entry-controlled loop.
- In while loop, a condition is evaluated before processing a body of the loop.
- If a condition is true then and only then the body of a loop is executed.
- After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false.
- Once the condition becomes false, the control goes out of the loop.



While() loop example

```
#include<stdio.h>
int main()
{
    int number=10;    //initializing the variable
    while(number<=20) //while loop with condition
    {
        printf("%d\n",number);
        number++;    //incrementing operation
    }
    return 0;
}
```


For loop

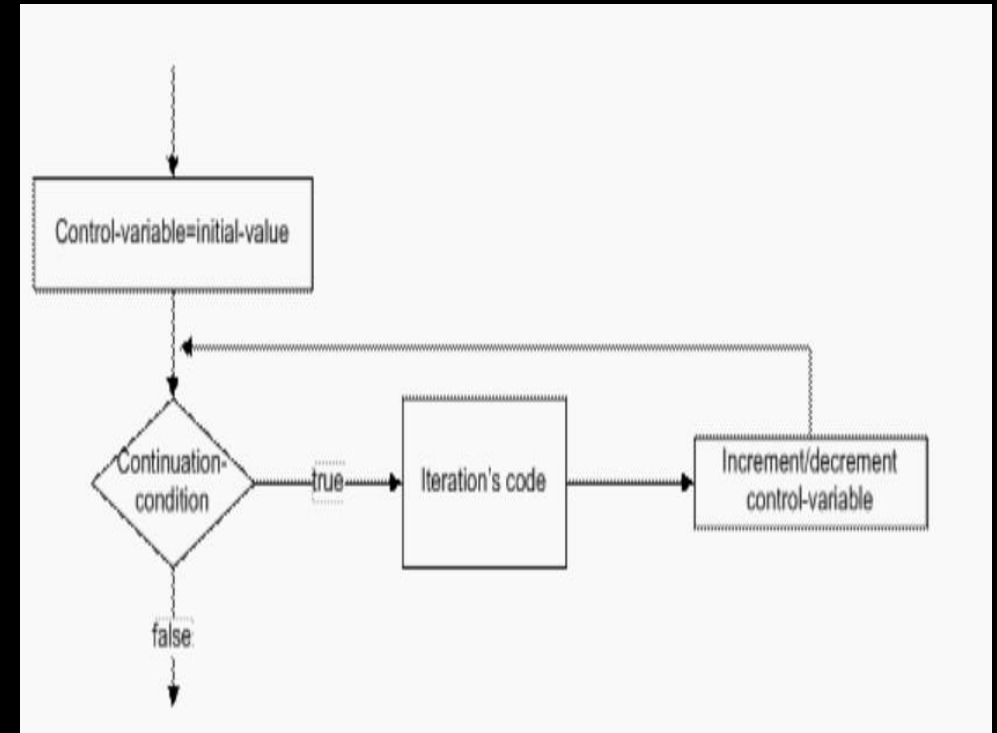
The general structure of for loop syntax in C is as follows:

```
for (control variable initialization; condition; increment or  
decrement )  
{  
    statements;  
}
```

The initialization in the for loop is performed only once.

The condition is a Boolean expression that tests and compares the counter variable to a fixed value after each iteration, stopping the for loop when false is returned.

The increment/decrement increases (or decreases) the counter by a set value.



For loop Example

```
#include<stdio.h>
int main()
{
    int number;
    for(number=10;number<=20;number++)
    {
        printf("%d\n",number);
    }
    return 0;
}
```

For loop Example

- In C, the for loop can have multiple expressions separated by commas in each part.

```
for (x = 0, y = num; x < y; i++, y--) {  
    statements;  
}
```

Example:

```
#include<stdio.h>  
int main()  
{  
    int number,y=2;  
    for(number=10;number<=20;number++,y++)  
    {  
        printf("Number=%d\t y=%d\n",number,y);  
    }  
    return 0;  
}
```

For loop

- Also, we can skip the initial value expression, condition and/or increment by adding a semicolon.

```
int i=0;  
int max = 10;  
for (; i < max; i++) {  
    printf("%d\n", i);  
}
```

Nested for loop

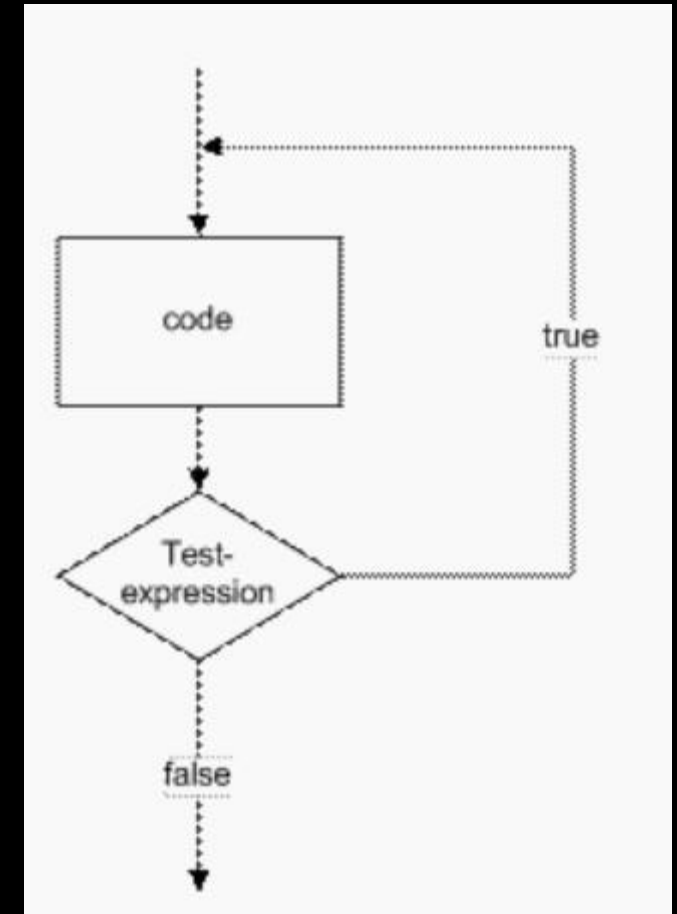
- Loops can also be nested where there is an outer loop and an inner loop.
- For each iteration of the outer loop, the inner loop repeats its entire cycle. `#include <stdio.h>`

```
int main() {  
    int i, j;  
    int table = 3;  
    int max = 10;  
    for (i = 1; i <= table; i++) { // outer loop  
        for (j = 0; j <= max; j++) { // inner loop  
            printf("%d x %d = %d\n", i, j, i*j);  
        }  
        printf("\n"); /* blank line between tables */  
    }  
}
```

do..while()

- A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop.
- It is also called an exit-controlled loop.
- Syntax of do...while loop in C programming language is as follows:

```
do {  
    statements/code;  
    update statement;  
} while (test expression);
```



do while()

- In a while loop, the body is executed if and only if the condition is true.
- In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.
- In the do-while loop, the body of a loop is always executed at least once.
- After the body is executed, then it checks the condition.
- If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

do while()

The main difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int main()
```

```
{
```

```
    int num=1;
```

```
    do      //do-while loop
```

```
    {
```

```
        printf("%d\n",2*num);
```

```
        num++;
```

```
    }while(num<=10);
```

```
    return 0;
```

```
}
```



break statement

The break statement terminates the loop immediately when it is encountered.

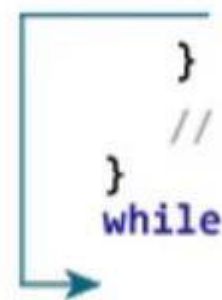
syntax :

break;

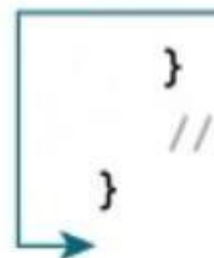
```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```



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



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



Break Example

```
void main()
{
    int num,counter,flag=0;
    printf("\nPlease enter a number");
    scanf("%d",&num);//num=5
    for(counter=2;counter<=num/2;counter++) //counter=2,3//counter=2
    {
        if(num%counter==0)
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
    {
        printf("%d is a prime number.",num);
    }
    else
    {
        printf("%d is not a prime number.",num);
    }
}
```

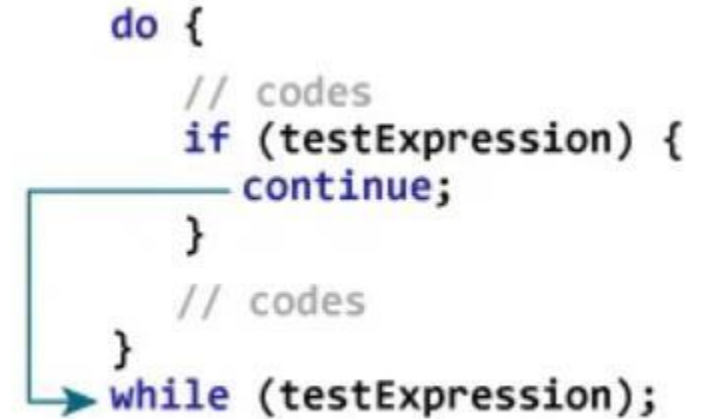
Continue statement

- The continue statement skips the current iteration of the loop and continues with the next iteration.
- Syntax: continue;

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



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



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

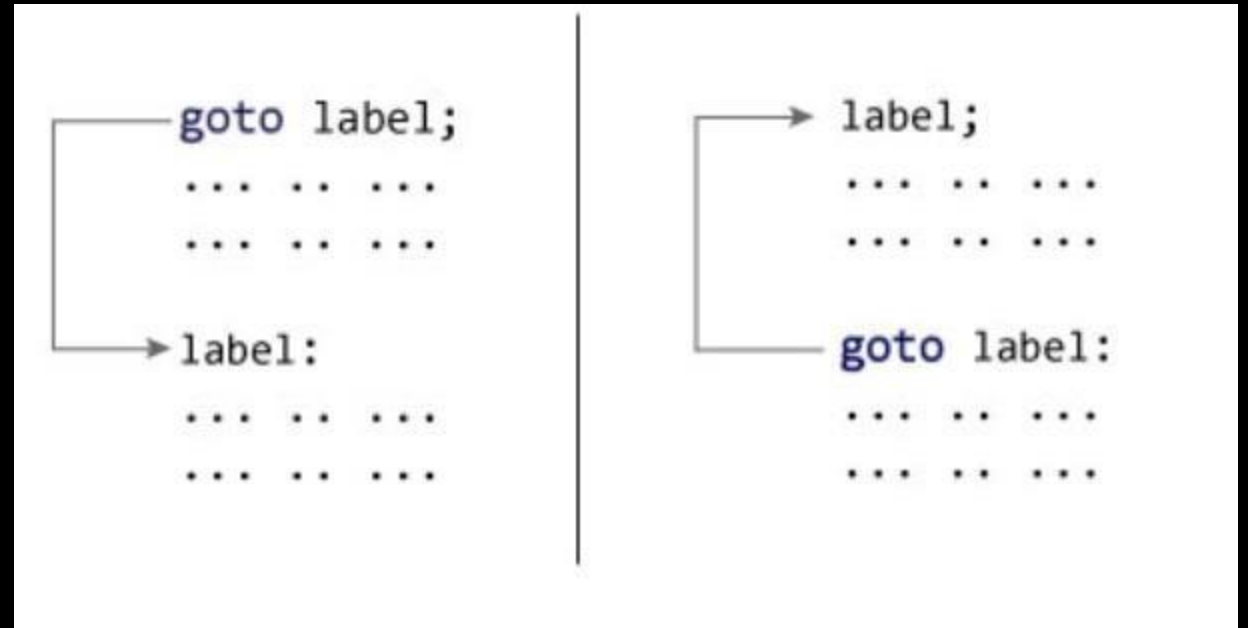


goto statement

- goto statement is used for changing the normal sequence of program execution by transferring control to some other part of the program.

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

In this syntax, label is an identifier.
When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.



//Write a C Program Which Print 1 To 10 Number Using goto statement.

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int i=1;
```

```
    count:        //This is Label
```

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

```
    i++;
```

```
    if(i<=10) {
```

```
        goto count;    //This jumps to label "count:"
```

```
    }
```

```
}
```

Reasons to avoid goto statement

- Use of goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled.
- In modern programming, goto statement is considered a harmful construct and a bad programming practice.
- The goto statement can be replaced in most of C program with the use of break and continue statements.
- In fact, any program in C programming can be perfectly written without the use of goto statement.
- All programmer should try to avoid goto statement as possible as they can.