

Looping and Control Breaks

- Control and Nested Loops
- Common Mistakes and Advantages of Looping
- For, Do While and Do Until Loops
- Single-Level Break and Multiple-Level Control Break

Control

- **Branching**

- Branching is deciding what actions to take
- Branching is so called because the program chooses to follow one branch or another.

- **Looping**

- Looping is deciding how many times to take a certain action.
- Loops provide a way to repeat commands and control how many times they are repeated.

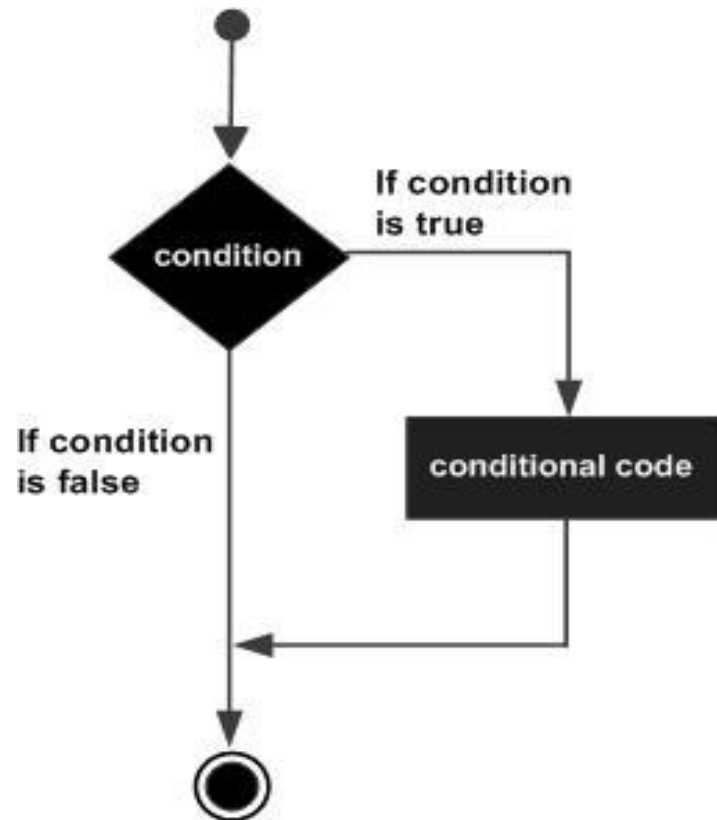
if statement

- This is the most simple form of the branching statements.
- It takes an expression in parenthesis and an statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

Syntax

if (expression)
 Statement;

```
if (expression)
{
    Block of statements;
}
```

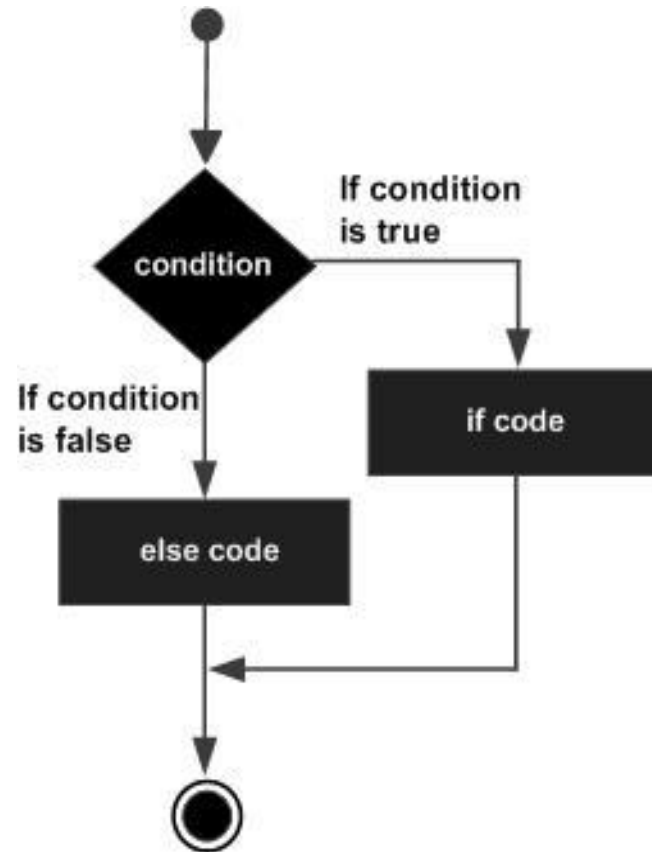


if Example

```
#include <stdio.h>
int main ()
{
    int a = 10;
    if( a < 20 )
    {
        /* if condition is true then print the following */
        printf("a is less than 20\n" );
    }
    printf("value of a is : %d\n", a);
    return 0;
}
```

Syntax

```
if (expression)
{
    Block of statements;
}
else
{
    Block of statements;
}
```

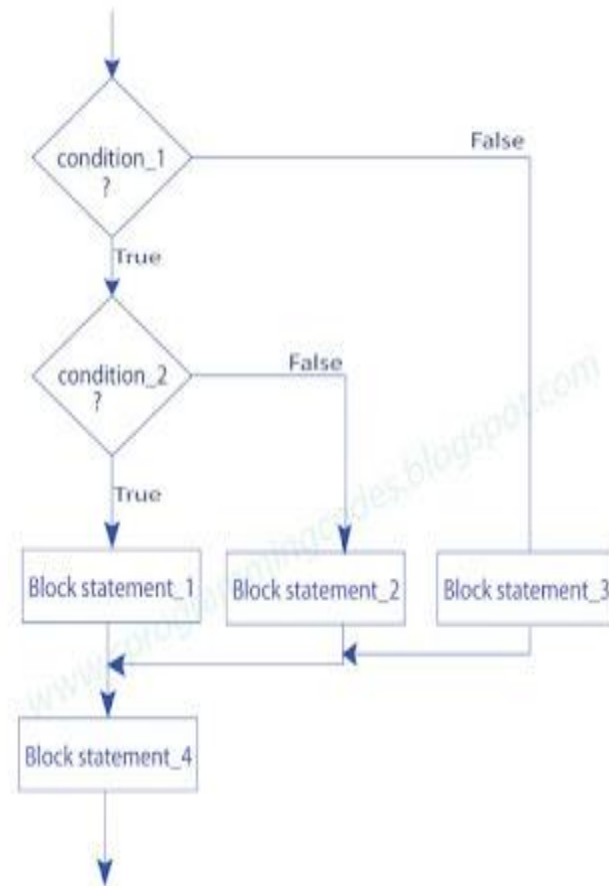


if.. else Example

```
#include <stdio.h>
int main ()
{
    int a = 100;
    if( a < 20 )
    {
        printf("a is less than 20\n" );
    }
    else
    {
        printf("a is not less than 20\n" );
    }
    printf("value of a is : %d\n", a);
    return 0;
}
```


Nested if statement

```
if(condition_1)
{
    if(condition_2)
    {
        block statement_1;
    }
    else
    {
        block statement_2;
    }
}
else
{
    block statement_3;
}
block statement_4;
```

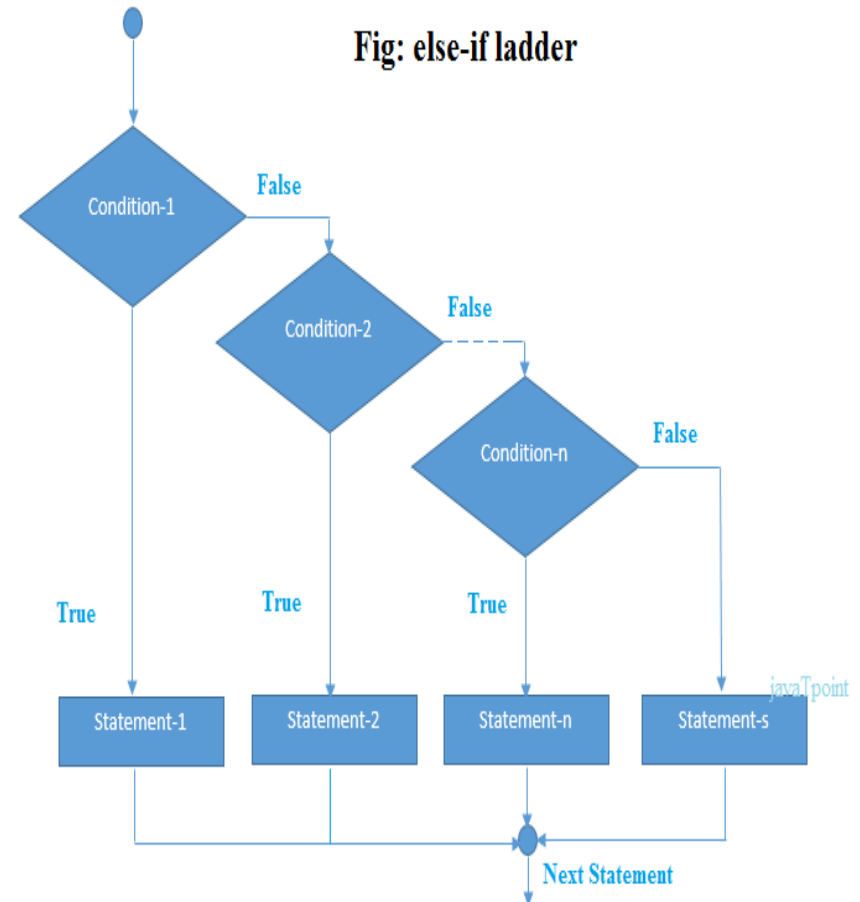


Nested if Example

```
if(code>10)
{
    if(code<20)
    {
        cost=100;
    }
    else
    {
        cost=150;
    }
}
else
{
    cost=50;
}
```

if.. else if ..else Statement

```
if (expression)
{
    Block of statements;
}
else if(expression)
{
    Block of statements;
}
else
{
    Block of statements;
}
```



if.. else if ..else Example

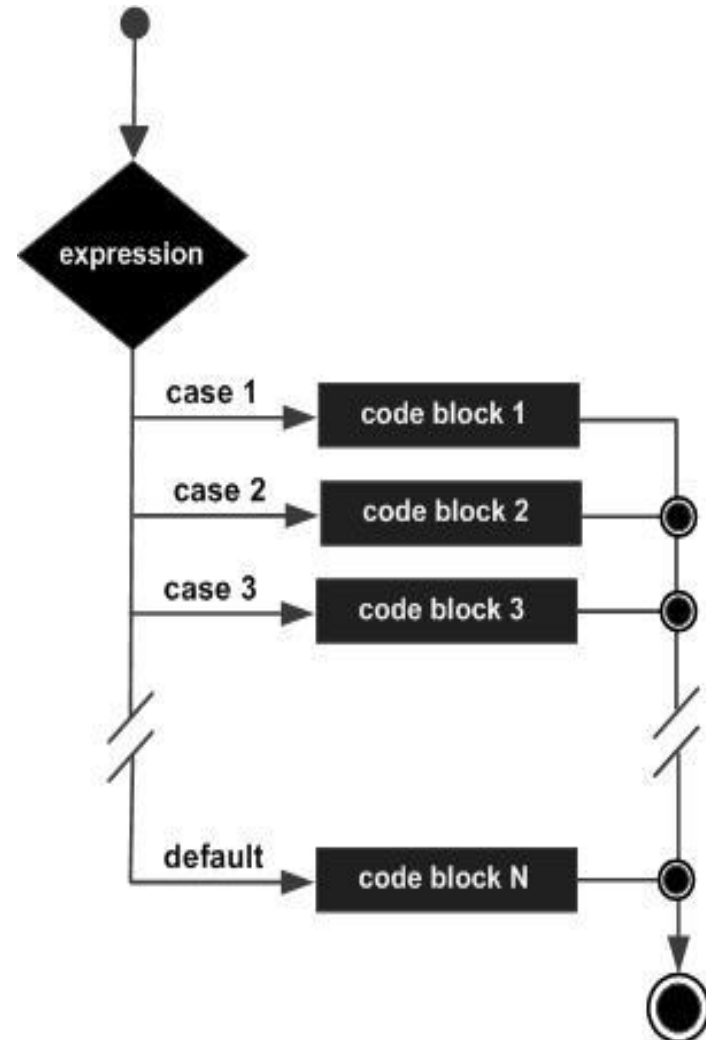
```
#include <stdio.h>
int main()
{
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);
    //checks if two integers are equal.
    if(number1 == number2)
    {
        printf("Result: %d = %d",number1,number2);
    }
    //checks if number1 is greater than number2.
    else if (number1 > number2)
    {
        printf("Result: %d > %d", number1, number2);
    }
    // if both test expression is false
    else
    {
        printf("Result: %d < %d",number1, number2);
    }
    return 0;
}
```

switch statement

- The switch statement is much like a nested if .. else statement.
- Its mostly a matter of preference which you use, switch statement can be slightly more efficient and easier to read.

Syntax

```
switch (integer_expression)  
{  
    case integer_constant_1:  
        statements;  
        break;  
    case integer_constant_2:  
        statements;  
        break;  
    ...  
    case integer_constant_N:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```



- The `integer_expression` must evaluate to one of the following types: `int`, `byte`, `short`, `long`, `char`.
- The `integer_constant_x` values must be either literal values (such as 5) or constants.
- The statements may be any sequence of zero or more statements. Notice that it is not necessary to use braces to form the statements (along with the following `break` statement) into a block, although this is sometimes done.
- The `break` statement is not required at the end of each case; however, if it is omitted, control will flow into the next set of statements. This is seldom what you want to happen. On the rare occasion that this is the desired behaviour, you should include a comment that the omission of `break` is intentional, otherwise you may "correct" this problem at some later date.
- The default case should come last. It is good style to always include a default case, even if you believe that all possibilities have been covered; in this case, you can put the statement `assert false`; in the default case to document your belief.
- The `break` statement in the last case is unnecessary.

break statement

- If a condition is met in switch case then execution continues on into the next case clause also if it is not explicitly specified that the execution should exit the switch statement. This is achieved by using *break* keyword.

default Condition

- If none of the listed conditions is met then default condition executed.

Example

```
#include <stdio.h> main()
{
int Grade = 'A';
switch( Grade )
{
case 'A' :
    printf( "Excellent\n" );
case 'B' :
    printf( "Good\n" );
case 'C' :
    printf( "OK\n" );
case 'D' :
    printf( "Average....\n" );
case 'F' :
    printf( "You must do better than this\n" );
default :
    printf( "What is your grade anyway?\n" );
}
}
```

Using break statement

```
#include <stdio.h> main()
{
int Grade = 'B';
switch( Grade )
{
case 'A' :
    printf( "Excellent\n" );
    break;
case 'B' :
    printf( "Good\n" );
    break;
case 'C' :
    printf( "OK\n" ); break;
case 'D' :
    printf( "Average....\n" );
    break;
case 'F' :
    printf( "You must do better than this\n" );
    break;
default :
    printf( "What is your grade anyway?\n" );
    break;
}
}
```

Use of default condition

```
#include <stdio.h>
main()
{
    int Grade = 'L';
    switch( Grade )
    {
        case 'A' :
            printf( "Excellent\n" );
            break;
        case 'B' :
            printf( "Good\n" );
            break;
        case 'C' :
            printf( "OK\n" );
            break;
        case 'D' :
            printf( "Mmmmm....\n" );
            break;
        case 'F' :
            printf( "You must do better than this\n" );
            break;
        default :
            printf( "What is your grade anyway?\n" );
            break;
    }
}
```

while loop

- A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

- **Syntax**

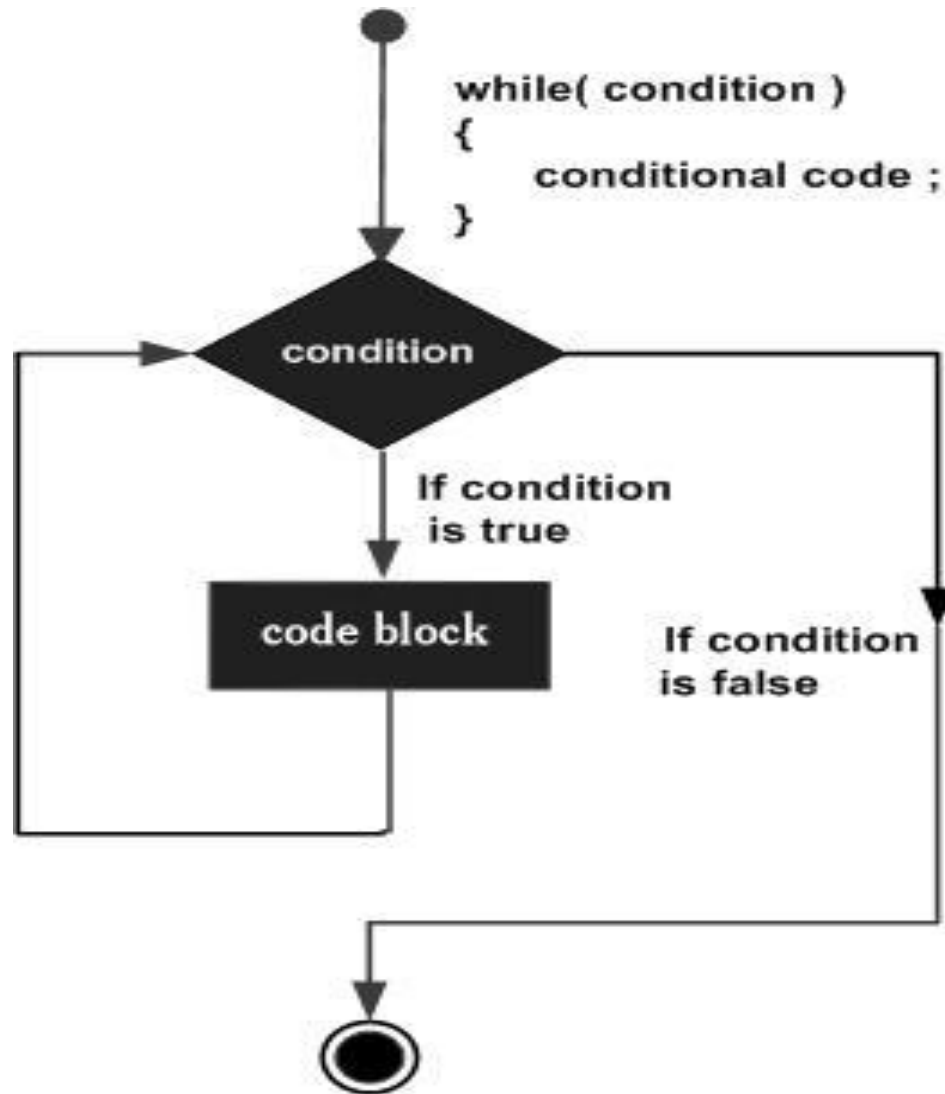
```
while(condition)
```

```
{
```

```
Statements;
```

```
}
```

- The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.
- When the condition becomes false, the program control passes to the line immediately following the loop.



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

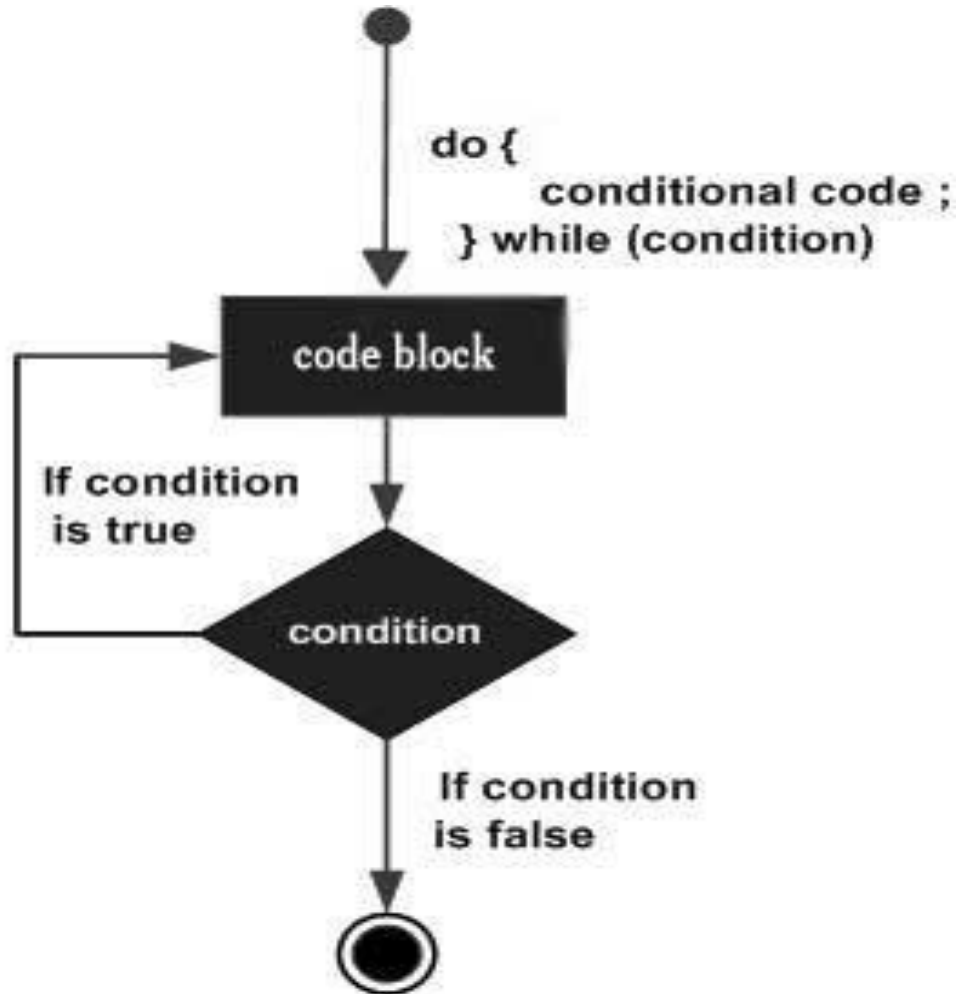
do while loop

- Unlike **while** loop, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

- Syntax

```
do  
{  
    statements;  
} while( condition );
```

- Notice that the conditional expression appears at the end of the loop, so the statements in the loop executes once before the condition is tested.
- If the condition is true, the flow of control jumps back up to do, and the statements in the loop executes again. This process repeats until the given condition becomes false.



```
#include <stdio.h>
int main ()
{
int a = 10;
do
{
    printf("value of a: %d\n", a);
    a = a + 1;
}while( a < 20 );
return 0;
}
```

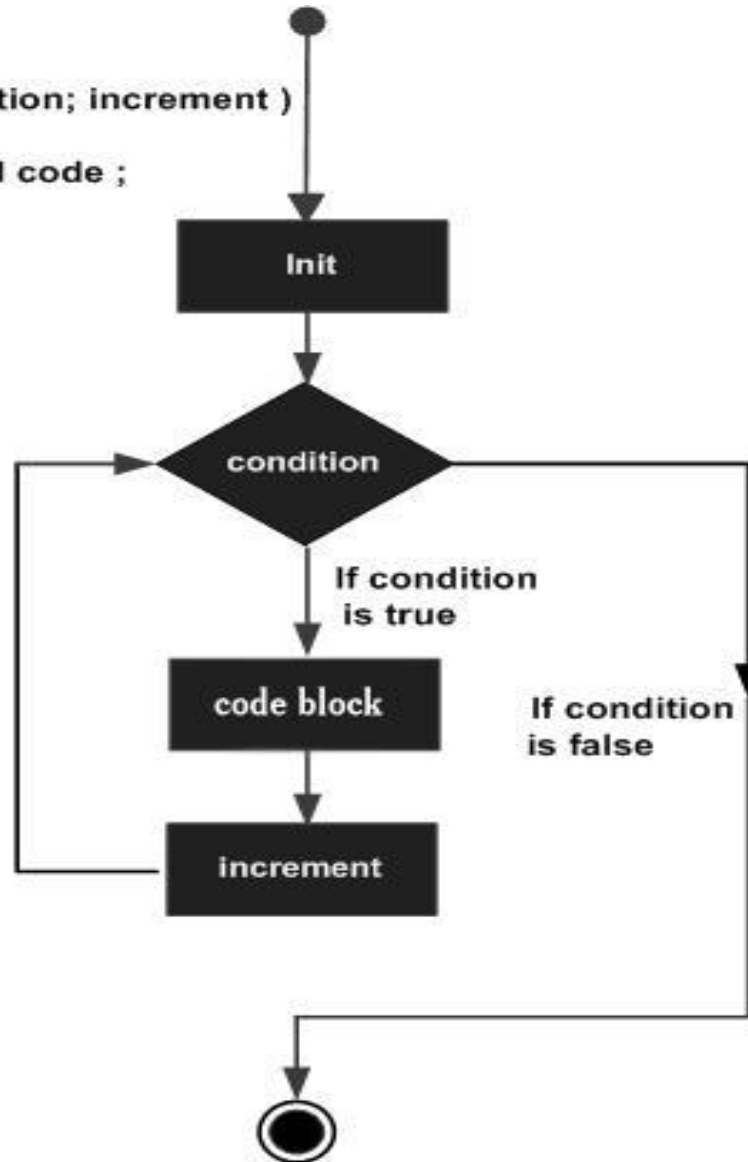
for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- Syntax

```
for ( initialization; condition; increment )  
{  
    statements;  
}
```

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

```
for( init; condition; increment )  
{  
    conditional code ;  
}
```



```
#include <stdio.h>
int main ()
{
int a;
for( a = 10; a < 20; a = a + 1 )
{
    printf("value of a: %d\n", a);
}
return 0;
}
```

Nested while Loops

```
while(condition)
{
    while(condition)
    {
        statements;
    }
    statements;
}
```


Nested do while Loops

```
do
{
    statements;
    do
    {
        statements;
    }while( condition );
}while( condition );
```

Nested for Loops

```
for ( initialization; condition; increment )  
{  
    for ( initialization; condition; increment )  
    {  
        statements;  
    }  
    statements;  
}
```

Infinite Loop

- A loop becomes an infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#include <stdio.h>
int main ()
{
    for( ; ; )
    {
        printf("This loop will run forever.\n");
    }
    return 0;
}
```

Common Mistakes

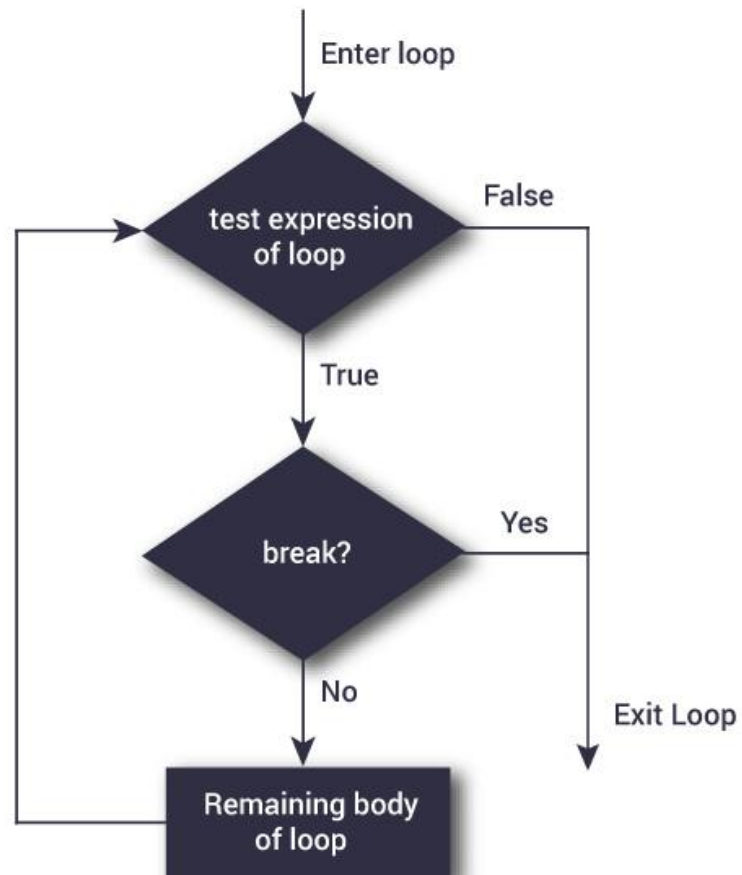
- Neglecting to initialize the loop control variable.
- Neglecting to alter the loop control variable
- Using the wrong comparison with the loop control variable.
- Including statements inside the loop that belong outside the loop.

Advantages of Looping

- Reduce length of Code
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

break statement

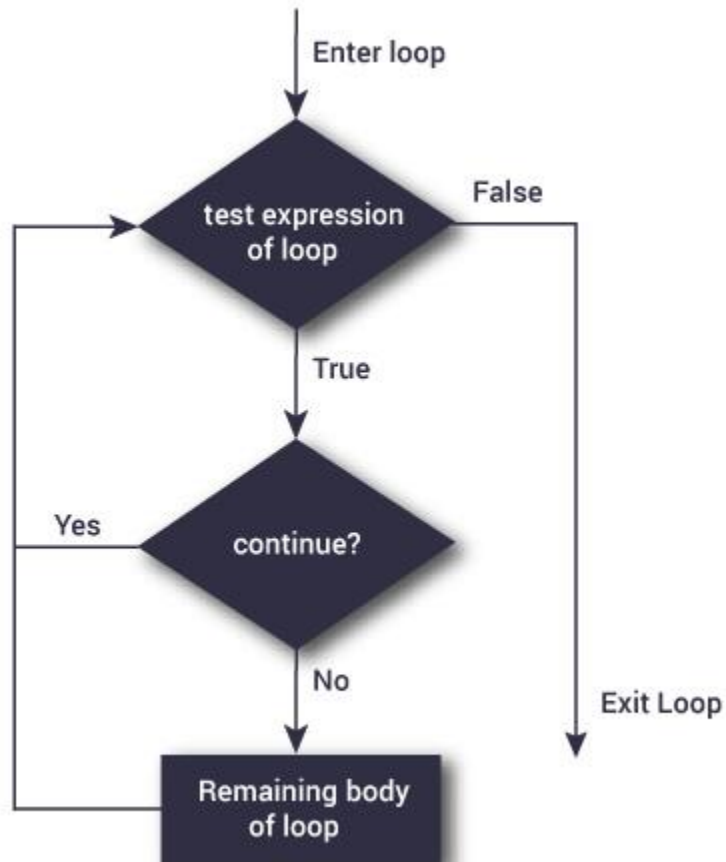
- The **break** statement in C programming has the following two usages –
 1. When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
 2. It can be used to terminate a case in the **switch** statement.




```
int a = 10;
while( a < 20 )
{
    printf("value of a: %d\n", a);
    a++;
    if( a > 15)
    {
        break;
    }
}
```

continue statement

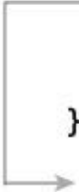
- The **continue** statement in C programming works somewhat like the **break** statement.
- Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute.
- For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.



```
int a = 10;
do
{
    if( a == 15)
    {
        a = a + 1;
        continue;
    }
    printf("value of a: %d\n", a);
    a++;
} while( a < 20 );
```


break	continue
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for , while, do) statements.
A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered.	A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue.
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops. You will get an error if this appears in switch statement.
When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.

```
while (test Expression)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```



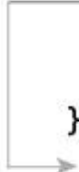
A diagram showing a line from the `break;` statement extending to the left and then turning downwards to point at the closing curly brace of the `while` loop, indicating an exit from the loop.

```
while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```




A diagram showing a line from the `continue;` statement extending to the left and then turning upwards to point at the opening curly brace of the `while` loop, indicating a jump to the start of the loop iteration.

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



A diagram showing a line from the `break;` statement extending to the left and then turning downwards to point at the closing curly brace of the `for` loop, indicating an exit from the loop.

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



A diagram showing a line from the `continue;` statement extending to the left and then turning upwards to point at the opening curly brace of the `for` loop, indicating a jump to the start of the loop iteration.

Exercise

- To find larger of two numbers.
- To find maximum of three numbers.
- Write a program that accepts an age value and checks whether the person is eligible for voting or not.
- To find number is negative, positive or zero.
- To find number is of 2digit, 3digit, 4digit or above.

- An electric distribution company charges its domestic consumers as follows.

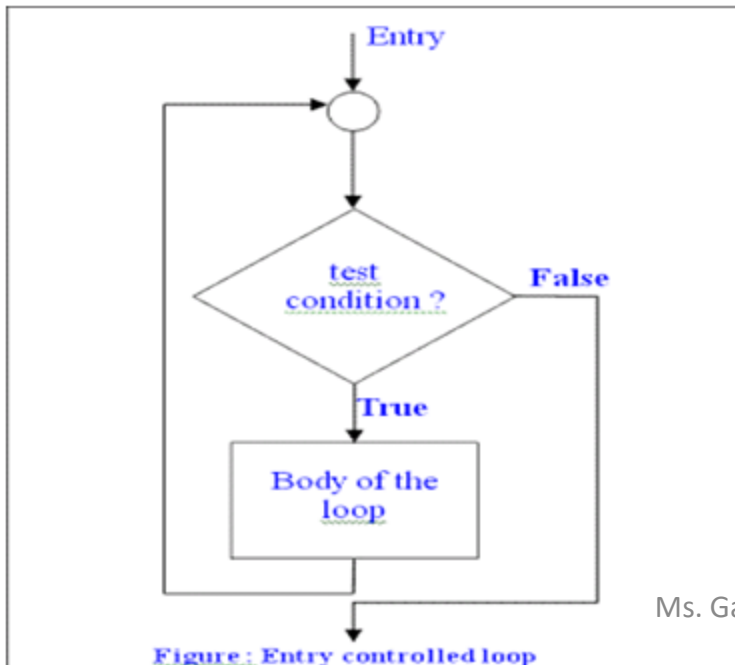
Consumption in units	Rate of charges
0-200	Rs 0.50 per unit
201-400 Rs. 100 Plus	Rs. 0.65 per unit excess of 200
401-600 Rs. 230 Plus	Rs 0.80 per unit excess of 400
Above 600 Rs. 425 Plus	Rs 1.25 per unit excess of 600

- Generate even and odd number between 1 to any number and their sum.
- Generate the multiplication table of any number.
- To check a given number is palindrome or not.
- Calculate the average of n numbers.

- Write difference between while, do..while and for loop.

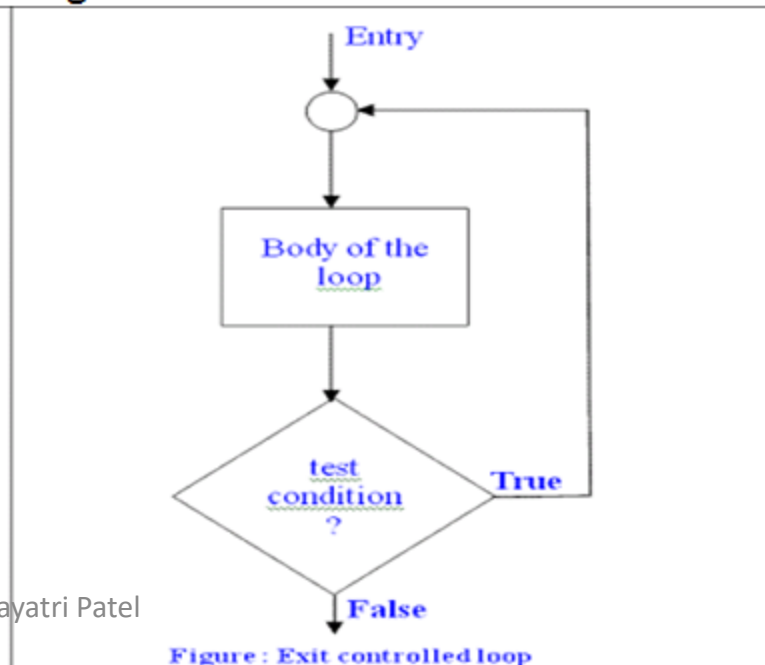
Entry controlled loop:

The types of loop where the test condition is stated before the body of the loop, are known as the entry controlled loop. So in the case of an entry controlled loop, the condition is tested before the execution of the loop. If the test condition is true, then the loop gets the execution, otherwise not. For example, the for loop is an entry controlled loop. In the given figure, the structure of an entry controlled loop is shown.



Exit controlled loop:

The types of loop where the test condition is stated at the end of the body of the loop, are known as the exit controlled loops. So, in the case of the exit controlled loops, the body of the loop gets execution without testing the given condition for the first time. Then the condition is tested. If it comes true, then the loop gets another execution and continues till the result of the test condition is not false. For example, the do statement or the do....while loop is an exit controlled loop. The structure of an exit controlled loop is given in the given figure.



No.	Topics	Entry controlled loops	Exit controlled loops
01	Test condition	Test condition appears at the beginning.	Test condition appears at the end.
02	Control variable	Control variable is counter variable.	Control variable is counter & sentinel variable.
03	Execution	Each execution occurs by testing condition.	Each execution except the first one occurs by testing condition.
04	Examples	<pre>===== sum = 0; n = 1; while (n <= 10) { sum = sum + n*n; n = n+ 1; } =====</pre> <p>Pre-test control loop</p>	<pre>===== do { printf("Input a number.\n"); scanf("%d", &num); } while(num>0); =====</pre> <p>Post-test control loop</p>

while / do...while / for

The main difference between for loop, while loop, and do while loop is

- While loop checks for the condition first. so it may not even enter into the loop, if the condition is false.

- do while loop, execute the statements in the loop first before checks for the condition. At least one iteration takes places, even if the condition is false.

- for loop is similar to while loop except that

 - initialization statement, usually the counter variable initialization

 - a statement that will be executed after each and every iteration in the loop, usually counter variable increment or decrement