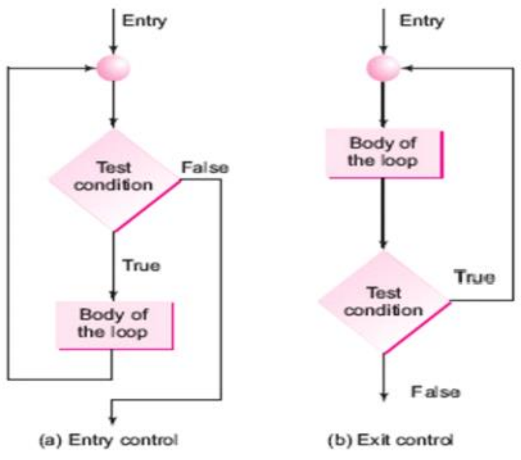
DECISION MAKING AND LOOPING

A looping process, in general, would include the following four steps:

1.Setting and initialization of a counter.
2.Execution of the statements in the loop.
3.Test for a specified condition for execution of the loop.
4.Incrementing the counter.



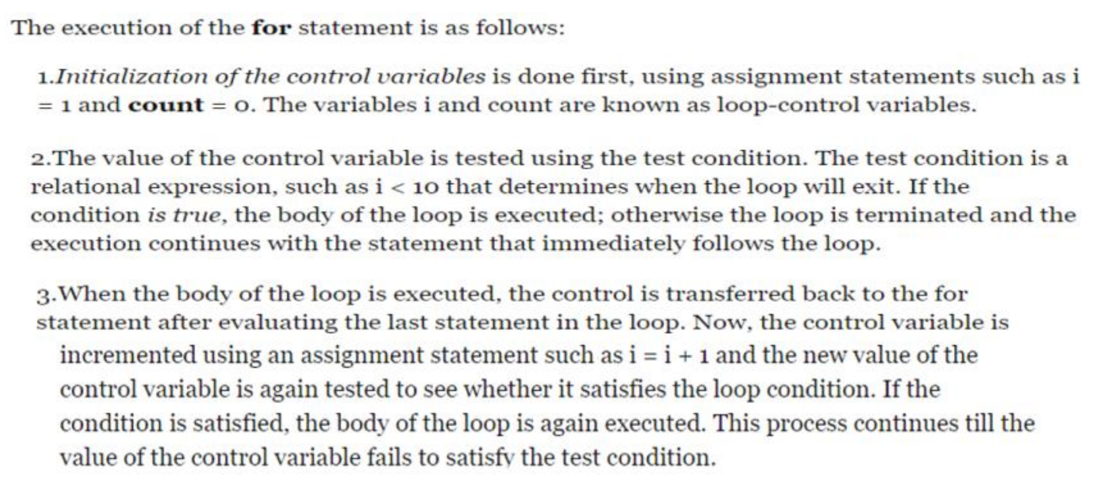
Loop control structures

.**while**
.**do**
.**for**

FOR STATEMENT

The **for** loop is another *entry-controlled* loop that provides a more concise loop control structure. The general form of the **for** loop is

```
for (initialization ; test condition ; increment)
{
    Body of the loop
}
```


The execution of the **for** statement is as follows:

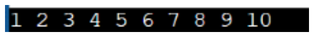
1.*Initialization of the control variables* is done first, using assignment statements such as i = 1 and **count** = 0. The variables i and count are known as loop-control variables.

2.The value of the control variable is tested using the test condition. The test condition is a relational expression, such as i < 10 that determines when the loop will exit. If the condition *is true*, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.

3.When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement such as i = i + 1 and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

Program to display numbers from 1 to 10

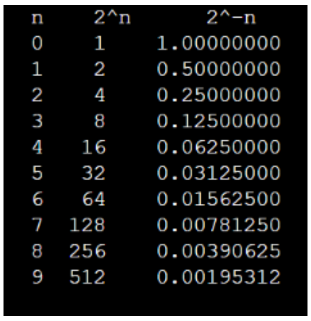
```
#include<stdio.h>
int main() {
  int i;

 for (i = 1; i < 11; i++)
  {
    printf("%d ", i);
  }
  return 0;
}
```



```
1 2 3 4 5 6 7 8 9 10
```

Program to display numbers from 0 to 9   along 2^the number and 1/2^the number

```
#include<stdio.h>

int main()
   {
             long p;
              int n;
              double q;
              p=1;

             printf("    n    2^n      2^-n \n");
              for(n=0;n<10;++n)
              {
                        if(n==0)
                           p=1;
                        else
                           p=p*2;

                        q=1.0/p;
                        printf("%5d  %3ld    %5.8lf \n",n,p,q);

                            }


      return 0;
   }
```



```
 n    2^n       2^-n
 0     1     1.00000000
 1     2     0.50000000
 2     4     0.25000000
 3     8     0.12500000
 4    16     0.06250000
 5    32     0.03125000
 6    64     0.01562500
 7   128     0.00781250
 8   256     0.00390625
 9   512     0.00195312
```

Additional Features of for Loop

- More than one variable can be initialized at a time in the **for** statement
  for(p=1,n=0; n<17; n++)

- Increment section also can have more than one part
  for(n=1,m=50; n<=m; n=n+1, m=m-50)

- The test condition may have any compound relation and the testing need not be limited only to the loop control variable.
  sum=0;
  for(i=0; 1<20 && sum<100;++i)

- It is also possible to use expressions in the assignment statements of initialization and increment sections.
  for(x=(m+n)/2;x>0;x=x/2)

- One or more sections may be omitted

```
m=5;
for( ; m<50; )
{
   printf("%d",m);
   m++;
}
```

- A delay loop using null statement can be set up as follows

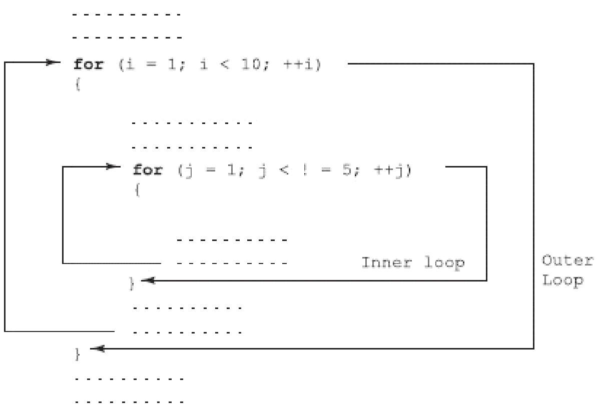
```
for(j=1000; j>0; j--)
     ;
```

Or

```
for(j=1000; j>0; j--)  ;
```

Nesting of for Loops

One for statement within another for statement



```
- - - - - - - - - -
- - - - - - - - - -
for (i = 1; i < 10; ++i)
{
    . . . . . . . . . . .
    . . . . . . . . . . .
    for (j = 1; j < ! = 5; ++j)
    {
        - - - - - - - - - -
        - - - - - - - - - -
    }
    . . . . . . . . . .
    . . . . . . . . . .
}
. . . . . . . . . .
. . . . . . . . . .
```

Program to display the pattern given below

#include<stdio.h>

```
int main()
   {
      for (int i = 1; i <= 5; i++)
       {
           for (int j = 1; j <= i; j++)
            {
                printf("*");
            }
         printf("\n");
       }
     return 0;
   }
```

OUTPUT

```
*
**
***
****
*****
```

while STATEMENT

The basic format of the while statement is :

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

- The while is an entry-controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop
- On exit, the program continues with the statement immediately after the body of the loop.
- Braces are needed if the body contains more than one statement.

Program to display numbers from 1 to 5

```
#include <stdio.h>
int main() {
        int i = 1;

        while (i <= 5)
        {
                 printf("%d ", i);
                  i++;
        }

        return 0;
}
```

```
1 2 3 4 5
```

Program to calculate 1^2+2^2+3^2…….10^2

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

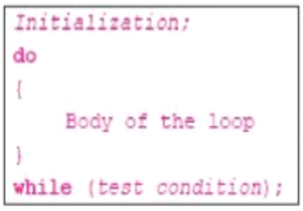
```
int main()
    {
        int sum =0;
        int n=1;

            while( n<=10 )
                {
                    sum=sum + n*n;
                    n=n+1;
                }

         printf("Sum of the squares of the numbers is: %d",sum);
        return 0;
    }
```

OUTPUT

```
Sum of the squares of the numbers is: 385
```

do STATEMENT

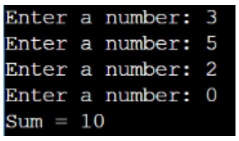
- The while loop makes a test condition before the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.
- On some occasions, it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the **do statement.**
- This takes the following form:



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

Program to accept numbers from the user till the user enters 0. the sum of the numbers is then displayed

```
#include <stdio.h>
int main()
 {
        int num, sum = 0;

        // the body of the loop is executed at least once
        do
       {
                printf("Enter a number: ");
                scanf("%d", &num);
                sum += num;
         } while(num != 0);

        printf("Sum = %d",sum);

        return 0;
}
```



```
Enter a number: 3
Enter a number: 5
Enter a number: 2
Enter a number: 0
Sum = 10
```

Multiplication table for 8

```
#include<stdio.h>

int main()
   {
      int a=8;
      int i=1;
      int b;
       printf("Multiplication table for 8\n");
       do
        {
                 b=a*i;
                 i=i+1;
                 printf("%d ",b);
       }while(i<=10);

     return 0;
   }
```

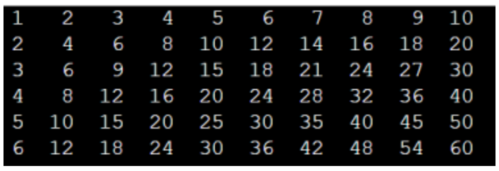
OUTPUT

```
Multiplication table for 8
8 16 24 32 40 48 56 64 72 80
```

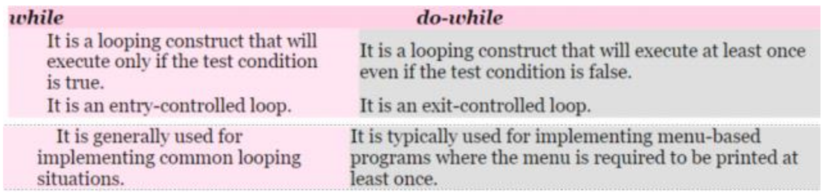
Multiplication table for 1 to 6 using nested do

```
#include<stdio.h>

int main()
   {
      int row, col, y;
      row=1;
             do{
                               col=1;
                                do{
                                         y=row*col;
                                         printf("%4d",y);
                                         col=col+1;

                                }while(col<=10);

                                printf("\n");
                                row=row+1;
               } while(row<=6);

     return 0;
   }
```

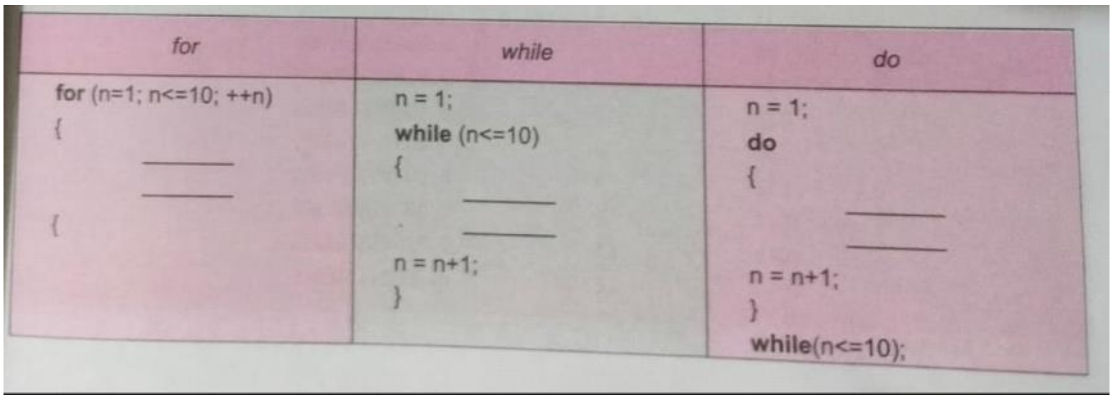
OUTPUT



```
1   2   3   4   5   6   7   8   9  10
2   4   6   8  10  12  14  16  18  20
3   6   9  12  15  18  21  24  27  30
4   8  12  16  20  24  28  32  36  40
5  10  15  20  25  30  35  40  45  50
6  12  18  24  30  36  42  48  54  60
```

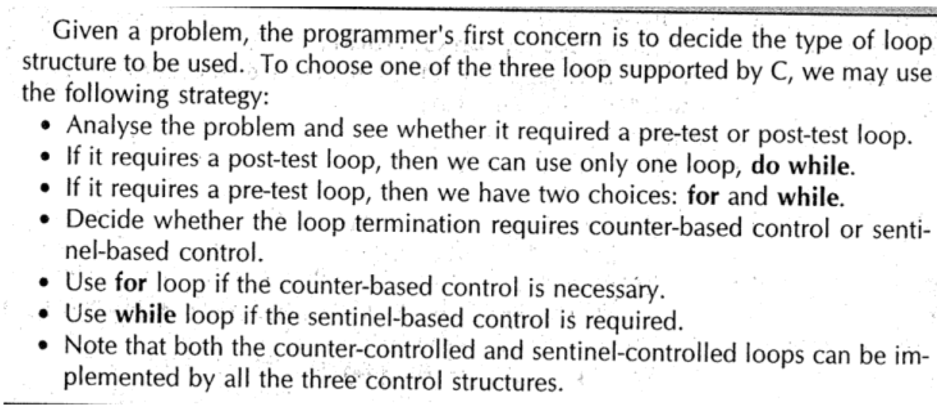


while	*do-while*
It is a looping construct that will execute only if the test condition is true.	It is a looping construct that will execute at least once even if the test condition is false.
It is an entry-controlled loop.	It is an exit-controlled loop.
It is generally used for implementing common looping situations.	It is typically used for implementing menu-based programs where the menu is required to be printed at least once.



for	while	do

for (n=1; n<=10; ++n)
{
______
______
{
	n = 1;
	while (n<=10)
	{
	______
	______
	n = n+1;
	}
		n = 1;
		do
		{
		______
		______
		n = n+1;
		}
		while(n<=10);



Given a problem, the programmer's first concern is to decide the type of loop structure to be used. To choose one of the three loop supported by C, we may use the following strategy:

- Analyse the problem and see whether it required a pre-test or post-test loop.
- If it requires a post-test loop, then we can use only one loop, **do while**.
- If it requires a pre-test loop, then we have two choices: **for** and **while**.
- Decide whether the loop termination requires counter-based control or sentinel-based control.
- Use **for** loop if the counter-based control is necessary.
- Use **while** loop if the sentinel-based control is required.
- Note that both the counter-controlled and sentinel-controlled loops can be implemented by all the three control structures.

JUMPS in LOOPS

Loops perform a set of operations repeatedly until the control variable fails to satisfy the test condition. The number of times a loop is repeated is decided in advance and the test condition is written to achieve this. Sometimes, when executing a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs.

Jumping Out of a Loop

- An early exit from a loop can be accomplished by using the **break** statement or the **goto** statement.
- We have already seen the use of the break in the switch statement and the goto in the if..else construct. These statements can also be used within while, do or for loops
- When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

Fig: Exiting loop with break statement

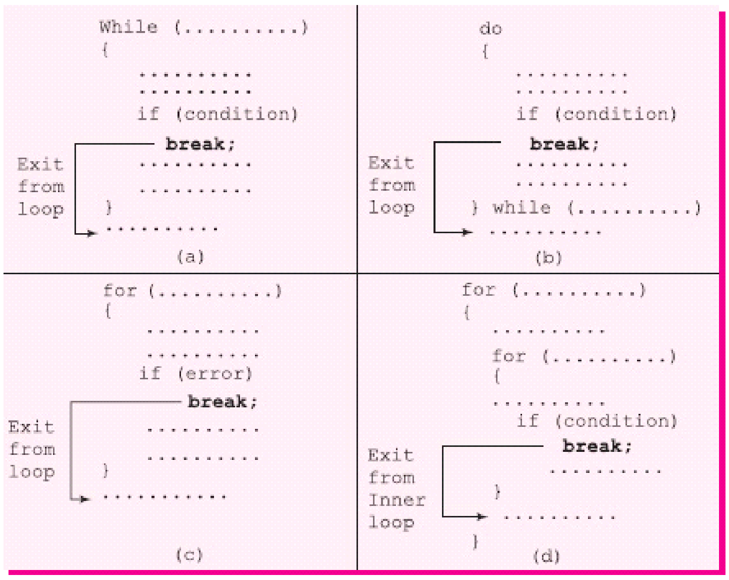
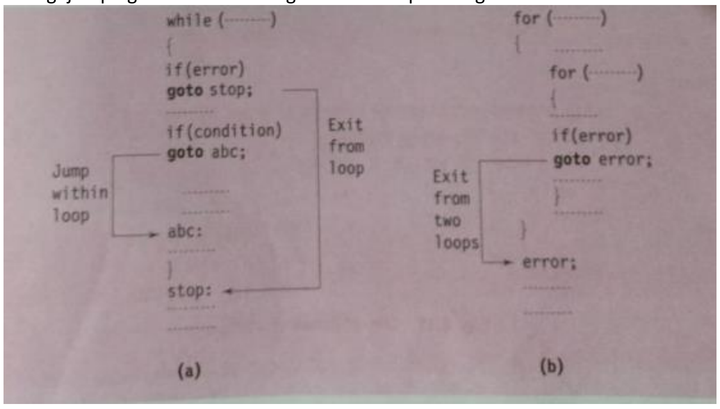


Fig: jumping within and exiting from the loops with goto statement



```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates
#include <stdio.h>

int main() {
  int i;
  double number, sum = 0.0;

        for (i = 1; i <= 10; ++i)
         {
                printf("Enter n%d: ", i);
                scanf("%lf", &number);

                // if the user enters a negative number, break the loop
                if (number < 0.0)
                 {
                   break;
                 }

                sum += number;
          }

        printf("Sum = %lf", sum);

  return 0;
}
```

```
Enter n1: 2.2
Enter n2: 3.5
Enter n3: 4.2
Enter n4: -1.2
Sum = 9.900000
```

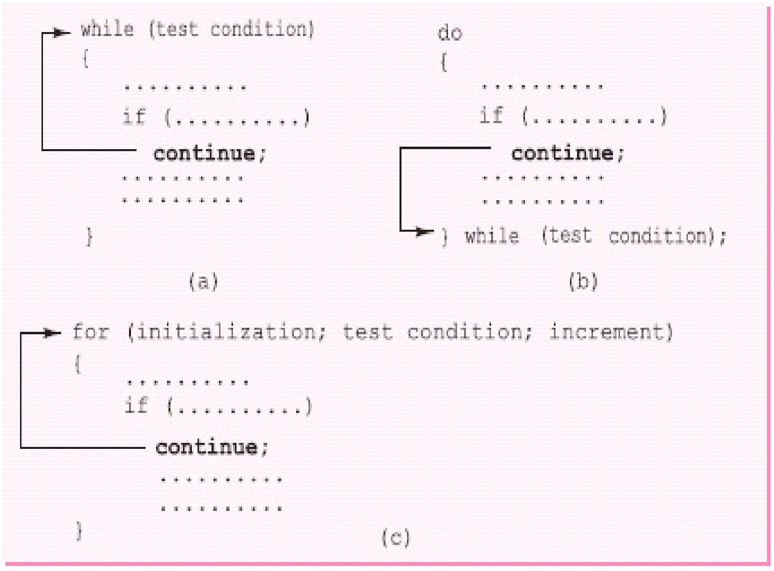
Skipping a Part of a Loop

- During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. For example, in processing of applications for some job, we might like to exclude the processing of data of applicants belonging to a certain category.

- Like the break statement, C supports another similar statement called the **continue** statement.

- However, unlike the break which causes the loop to be terminated, the continue, as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between.

- The continue statement tells the compiler. "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".



```
while (test condition)
{
    ..........
    if (..........)
      continue;
    ..........
    ..........
}
            (a)

do
{
    ..........
    if (..........)
      continue;
    ..........
    ..........
} while (test condition);
            (b)

for (initialization; test condition; increment)
{
    ..........
    if (..........)
      continue;
      ..........
      ..........
}
            (c)
```

```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the result

#include <stdio.h>
int main() {
        int i;
        double number, sum = 0.0;

        for (i = 1; i <= 10; ++i)
          {
                printf("Enter  n%d: ", i);
                scanf("%lf", &number);

                if (number < 0.0)
                 {
                   continue;
                }

                sum += number;
         }
        printf("Sum = %.2lf", sum);

    return 0;
  }
```

```
    Enter n1: 1.2
    Enter n2: 6.4
    Enter n3: -2.8
    Enter n4: 0.3
    Enter n5: 5.2
    Enter n6: -8.4
    Enter n7: -9.2
    Enter n8: 4.4
    Enter n9: 3.1
    Enter  n10: 6.2
    Sum = 26.80
```

Jumping out of the Program
exit() --this function is used to break out of a program.

```
if(test-condition)
   exit(0);
```

The exit() function takes an integer value as its argument. 0- normal termination, nonzero value-termination due to some error or abnormal condition .
exit() needs inclusion of header file<stdlib.h>