ITERATIVE STATEMENTS

while statement

The *while* statement is used when the program needs to perform repetitive tasks. The general form of a *while* statement is:

```
while ( condition)
        statement ;
```

The program will repeatedly execute the statement inside the *while* until the condition becomes false(0). (If the condition is initially false, the statement will not be executed.) Consider the following program:

```
main( )
{  int p, n, count;
float r, si;
count = 1;
while ( count <= 3 )
        {
                printf ( "\nEnter values of p, n and r " ) ;
                scanf("%d %d %f", &p, &n, &r ) ;
                si=p * n * r / 100 ;
                printf ( "Simple interest = Rs. %f", si ) ;
                count = count+1;
        }
}
```

Some outputs of this program:

```
Enter values of p, n and r 1000 5 13.5
Simple Interest = Rs. 675.000000
Enter values of p, n and r 2000 5 13.5
Simple Interest = Rs. 1350.000000
Enter values of p, n and r 3500 5 13.5
Simple Interest = Rs. 612.000000
```

The program executes all statements after the *while* 3 times. These statements form what is called the 'body' of the *while* loop. The parentheses after the *while* contain a condition. As long as this condition remains true all statements within the body of the *while* loop keep getting executed repeatedly.

Consider the following program;

```
 /* This program checks whether a given number is a palindrome or not */

#include <stdio.h>
int main()
{
   int n, reverse = 0, temp;
   printf("Enter a number to check if it is a palindrome or not\n");
   scanf("%d",&n);
    temp = n;
   while( temp != 0 )
   {
         reverse = reverse * 10;
        reverse = reverse +temp%10;
       temp = temp/10;
   }
   if ( n == reverse )
      printf("%d is a palindrome number.\n", n);
else
      printf("%d is not a palindrome number.\n", n);

   return 0;
}
```

Output:

```
Enter a number to check if it is a palindrome or not
12321
12321 is a palindrome

Enter a number to check if it is a palindrome or not
12000
12000 is not a palindrome
```

do-while Loop

The body of the *do-while* executes at least once. The *do-while* structure is similar to the *while* loop except the relational test occurs at the bottom (rather than top) of the loop. This ensures that the body of the loop executes at least once. The *do-while* tests for a positive relational test; that is, as long as the test is True, the body of the loop continues to execute. The format of the do-while is

do
{ block of one or more C statements; }
while (test expression)

The test expression must be enclosed within parentheses, just as it does with a while statement.

Consider the following program

```
// C program to add all the numbers entered by a user until user enters 0.

#include <stdio.h>
int main()
{    int sum=0,num;
  do            /* Codes inside the body of do...while loops are at least executed once. */
  {
      printf("Enter a number\n");
     scanf("%d",&num);
      sum+=num;
  }
  while(num!=0);
  printf("sum=%d",sum);
return 0;
}
```

Output:

```
Enter a number
3
Enter a number
-2
Enter a number
```

0
sum=1

Consider the following program:

```
#include <stdio.h>
main()
{
    int i = 10;
   do
  {
       printf("Hello %d\n", i );
       i = i -1;
  }while ( i > 0 );
}
```

Output

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

Program

8. Program to count the no of digits in a number

```
#include <stdio.h>
int main()
{
  int n,count=0;
  printf("Enter an integer: ");
  scanf("%d", &n);
```

```
do
  {
        n/=10;          /* n=n/10 */
        count++;
 } while(n!=0);

 printf("Number of digits: %d",count);
}
```

Output

Enter an integer: 34523

Number of digits: 5

for Loop

The *for* is the most popular looping instruction. The general form of *for* statement is as under:

```
for ( initialise counter ; test counter ; Updating counter )
{
        do this;
        and this;
        and this;
}
```

The *for* allows us to specify three things about a loop in a single line:

(a) Setting a loop counter to an initial value.

(b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.

(c) Updating the value of loop counter either increment or decrement.

Consider the following program

```
int main(void)
{
int num;
 printf("    n    n cubed\n");
for (num = 1; num <= 6; num++)
```

```
        printf("%5d %5d\n", num, num*num*num);
return 0;
}
```

The program prints the integers 1 through 6 and their cubes.

```
n   n cubed
1    1
2    8
3    27
4    64
5    125
6    216
```

The first line of the *for* loop tells us immediately all the information about the loop parameters: the starting value of num, the final value of num, and the amount that num increases on each looping [5].

Grammatically, the three components of a *for* loop are expressions. Any of the three parts can be omitted, although the semicolons must remain.

Consider the following program:

```
main( )
{
        int i ;
        for ( i = 1 ; i <= 10 ; )
        {
         printf ( "%d\n", i ) ;
         i = i + 1 ;
        }
}
```

Here, the increment is done within the body of the *for* loop and not in the *for* statement. Note that in spite of this the semicolon after the condition is necessary.

Programs:

9. Program to print the sum of 1st N natural numbers.

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

```
{
 int n,i,sum=0;
 printf("Enter the limit: ");
scanf("%d", &n);
for(i=1;i<=n;i++)
 {
            sum = sum +i;
 }
printf("Sum of N natural numbers is: %d",sum);
}
```

Output

Enter the limit: 5
Sum of N natural numbers is 15.

10. Program to find the reverse of a number

```
#include<stdio.h>
int main()
{
int num,r,reverse=0;
printf("Enter any number: ");
scanf("%d",&num);
for(;num!=0;num=num/10)
{
        r=num%10;
        reverse=reverse*10+r;
}
 printf("Reversed of number: %d",reverse);
return 0;
 }
```

Output:

Enter any number: 123
Reversed of number: 321

NESTING OF LOOPS

C programming language allows using one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

The syntax for a nested for loop statement in C is as follows:

```
for ( init; condition; increment )
{

 for ( init; condition; increment)

   {
            statement(s);
   }
   statement(s);
}
```

The syntax for a nested while loop statement in C programming language is as follows:

```
while(condition)
{
 while(condition)
{
        statement(s);
}
statement(s);
}
```

The syntax for a nested do...while loop statement in C programming language is as follows:

```
do
{
  statement(s);
  do
  {
    statement(s);
}while( condition );
```

}while(condition);

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

Programs:

11. program using a nested for loop to find the prime numbers from 2 to 20:

```
#include <stdio.h>
int main ()
{
  /* local variable definition */
int i, j;
  for(i=2; i<20; i++)
 {
     for(j=2; j <= (i/j); j++)
       if(!(i%j))
            break; // if factor found, not prime
      if(j > (i/j)) printf("%d is prime\n", i);
  }

  return 0;
}
```

Output

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
```

12.
```
    *
   ***
  *****
 *******
*********
```

```
#include <stdio.h>
int main()
{
  int row, c, n,I, temp;
 printf("Enter the number of rows in pyramid of stars you wish to see ");
 scanf("%d",&n);
 temp = n;
 for ( row = 1 ; row <= n ; row++ )
 {
    for ( i= 1 ; i < temp ; i++ )
    {
          printf(" ");
          temp--;
          for ( c = 1 ; c <= 2*row - 1 ; c++ )
          {
                printf("*");
                printf("\n");
          }
    }
 }
  return 0;
}
```

13. Program to print series from 10 to 1 using nested loops.

```
#include<stdio.h>
void main ()
{
          int a;
          a=10;
        for (k=1;k=10;k++)
       {
          while (a>=1)
```

```
        {
                printf ("%d",a);
                a--;
        }
        printf("\n");
        a= 10;
        }
}
```

Output:

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