Lesson Number : 21

Jump Statements :

1. break Statement :

→ The break statement is used inside loops and switch statements.

→ Sometimes, it becomes necessary to come out of the loop even before the loop condition become ~~false~~ True.

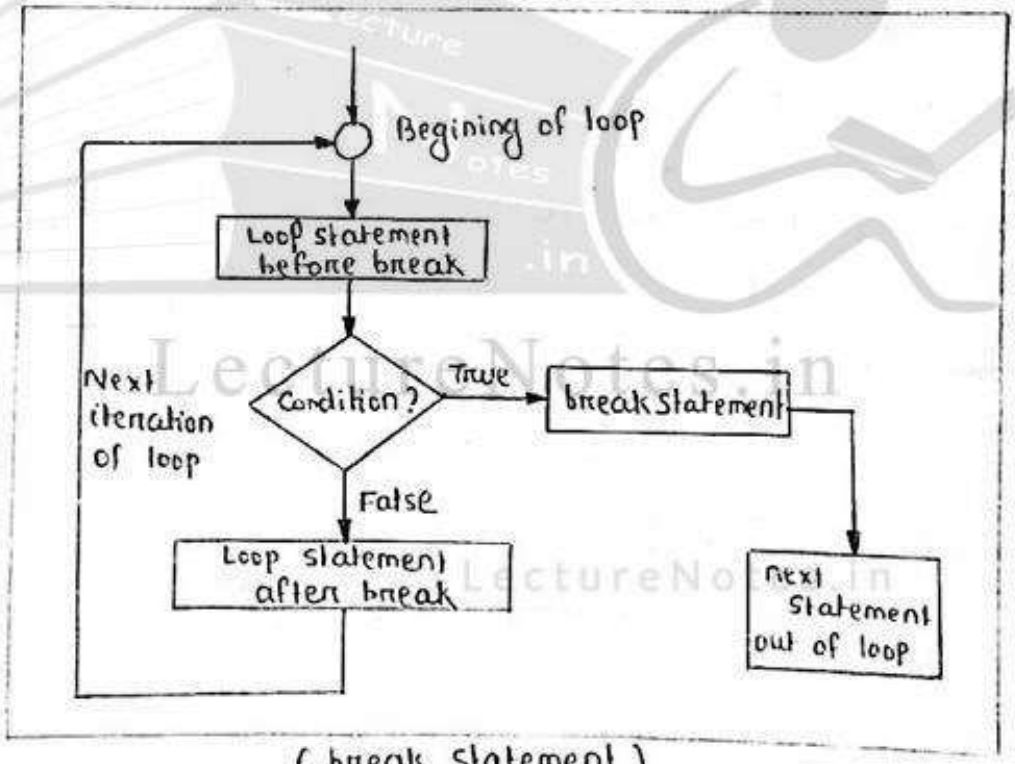
→ In such a situation, break statement is used to terminate the loop.

→ This statement causes an immediate exit from that loop in which this statement appears.

→ It can be written as ;

```
break ;
```

→ Flowchart :



(break statement)

→ When break statement is encountered, loop is terminated and the control is transferred to the statement immediately after the loop. The break statement is generally written along with a condition.

Example : /* program to understand the use of break */

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int n ;
   clrscr();
  for ( n=1 ; n<=5 ; n++)
    {
       if (n==3)
          { printf ("I understand the use of break\n");
            break ;
          }
       printf (" Number = %d\n", n) ;
    }
  printf (" Out of for loop\n");
  getch();
}
```

output : Number = 1
Number = 2
I understand the use of break
Out of for loop.

Example : /* program to find whether a number is prime or not */

```
#include <stdio.h>
#include <conio.h>
# include <math.h>
void main()
 {
    int num, i, flag = 1 ;
    clrscr();
    printf (" Enter a number :\n");
    scanf ("%d", &num) ;
```

```
for( i=2; i<=sqrt(num); i++)
 {
      if (num%i==0)
       {
          printf (" %d is not prime\n", num);
          flag=0;
          break;
       }
 }
 if ( flag==1)
     printf (" %d is prime\n",.num);
 getch();
}
```

Output : Enter a number : 5
5 is prime .

2) Continue Statement :

→ The continue statement is used when we want to go to the next iteration of the loop after skipping some statements of the loop.

→ The continue statement can be written as -

```
continue;
```

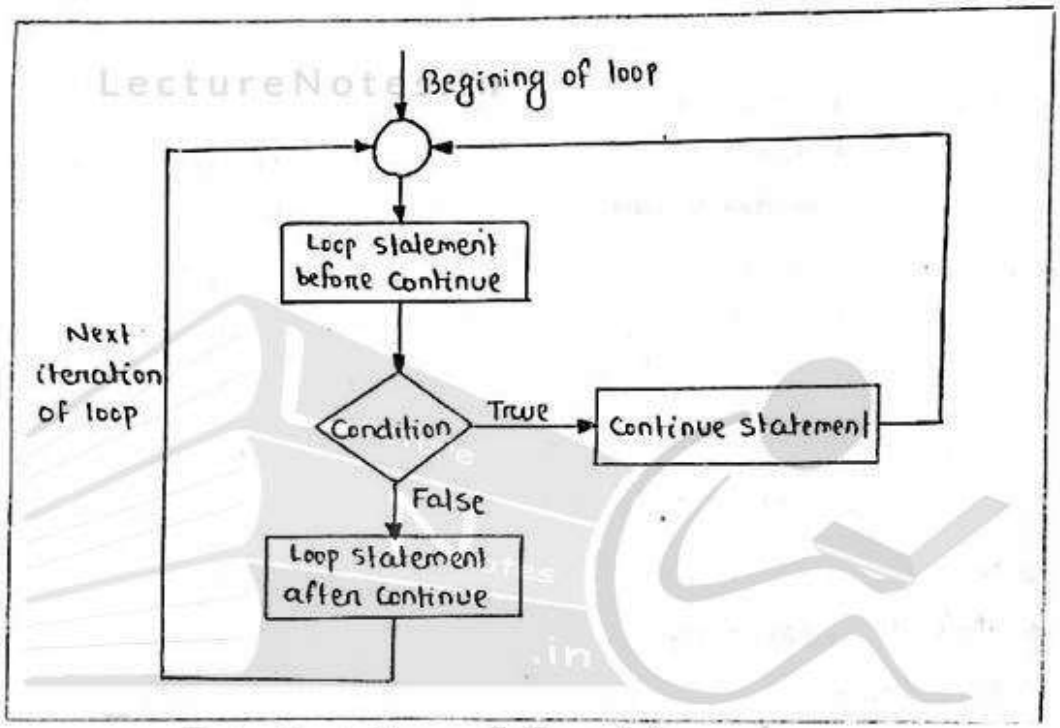
→ It is generally used with a condition.

→ When the continue statement is encountered, all the remaining statements (statements after continue) in the current iteration are not executed and the loop continues with the next iteration.

→ The difference between break and continue is that when break is encountered the loop terminates and the control is transfered to the next statement following the loop, but when a continue statement is encountered, the loop is not terminated and the control is transferred to the begining of the loop

→ In while and do-while loops, after continue statement the control is transferred to the test condition and then the loop continues, whereas in for loop after continue statement the control is transferred to update expression and then the condition is tested.

→ Flowchart :



(continue statement)

Example : /* program to understand the use of continue statement */

```
#include <stdio.h>
# include <conio.h>
void main( )
{
   int n ;
   clrscr( ) ;
   for ( n=1 ; n<=5 ; n++ )
       {
          if ( n == 3 )
           {
              printf (" I understand the use of continue \n") ;
```

```
    continue;
  }
    printf("Number = %d\n", n);
}
  printf("out of for loop\n");
  getch();
}
```

output :
```
Number = 1
Number = 2
I understand the use of continue
Number = 4
Number = 5
Out of for loop.
```

Example : /* write a program to find the sum and average of 10 positive integers */

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i=1, n, sum=0;
    float avg;
    clrscr();
    printf("Enter 10 positive numbers\n");
      while(i<=10)
    {
      printf("Enter number %d :", n);
      scanf("%d", &n);
      if(n<0)
        {
          printf("Enter only positive numbers\n");
          continue;
        }
```

```
    sum = sum + n ;
     i++;
  }
    avg = sum / 10.0;
    printf ( " Sum = %d  Avg = %f \n ", sum, avg ) ;
    getch();
}
```


Output : Enter 10 positive numbers

Enter number : 12.
Enter number : 05
Enter number : 02
Enter number : 07
Enter number : 11
Enter number : 8
Enter number : 3
Enter number : 20
Enter number : 21
Enter number : 01

Sum = 80
Avg = 8.0

3) goto statement :

→ The goto statement is an unconditional control statement that transfers the flow of control to another part of the program.

→ Syntax :

```
goto label ;
-----
-----
label :
    statement ;
    -----
    -----
```

→ Here label is any valid C identifier and it is followed by a Colon

→ whenever the statement goto label ; is encountered, the control is transferred to the statement that is immediately after the label.

Example : /* program to print whether the no. is even or odd */

```
# include <stdio.h>
# include <conio.h>
void main()
{
   int n;
   clrscr();
   printf ("Enter the number :");
   scanf ("%d", &n);
   if (n%2 == 0)
        goto even;
   else
        goto odd;
  even :
      printf ("Number is even\n");
       goto end;
  odd :
       printf ("Number is odd\n");
        goto end;

  end :
        printf ("\n");

  getch();
}
```

output : Enter the number : 14
Number is even.

→ The label can be placed anywhere.

→ If the label is after the goto then the control is transferred forword and it is known as forword jump or forword goto, and if the lebel is before the goto then the control is transferred backwords and it is known as backword jump or backword goto.

→ In forword goto, the statements between goto and label will not be executed and in backword goto, statements between goto and label will be executed repeatedly.

→ There should always be a statement after any label.

→ If label is at the end of the program, and no statements are to be written after it, we can write the null statement (single semicolon) after the label, because a program cannot end with a label.

4) The return statement :

→ The return statement is used to return from a function.

→ It is a jump statement because it causes execution to return (jump back) to the point at which the call to the function was made.

→ If return has a value associated with it, that value is the return value of the function.

→ If no value is specified, assumed that a garbage value is returned.

→ The general form of the return statement is ;

```
return expression;
```

Example: In void main(), the empty pair of parentheses indicates that the function has no argument. The keyword void means that the function does not return any information to the operating system.

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int n;
   clrscr();
   printf("Enter a number:\n");
   scanf("%d", &n);
   if (n%2==0)
        printf("Even");
   else
        printf("Odd");
   return;  /* return statement with no value */
}
```

Example: In int main(), the keyword int means that the function returns an integer value to the operating system. when int is specified, the last statement in the program must be "return 0".

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int n;
   clrscr();
   printf("Enter a no:");
   scanf("%d", &n);
```

```
if (n %.2 == 0)
    printf ("Even");
else
    printf ("odd");
return 0 ;   /* return statement with an integer value */
}
```

Example: In main(), bydefault the keyword is int, so it also returns an integer value to the operating syetem. when main() is written, the last statement in the program must be "return 0".

Example:

```
#include <stdio.h>
#include <conio.h>
main()
{
  int n;
  clrscr();
  printf ("Enter a number :\n");
  scanf ("%d", &n);
 if (n %.2 == 0)
      printf ("Even");
  else
      printf ("odd");
  return 0 ;  /* return with an integer value 0*/
}
```

The exit() Function :

→ Just as we can break out of a loop, we can break out of a pgm by using standard library function exit();

→ This function causes immediate termination of the program (entire program) forcing a return to the operating system.

→ The General form of exit() is :

```
exit (return-code);
```

→ The value of return code is return to the calling process, which is usually the OS. zero is generally used as a return code to indicate normal program termination. Other arguments are used to indicate some sort of error.

```
/* program for testing a number for prime */
#include <stdio.h>
#include <conio.h>
void main()
{
   int n, i, p, q;
    clrscr();
  printf("Enter a number :\n");
   scanf("%d", &n);
  if(n==2)
      { printf("The no. %d is an even prime", n);
        exit(0);
      }
  p = sqrt(n);
  for(i=2; i<=p; i++)
        { q = n%i;

  }   if(q==0) break;
   if(q==0)
        printf("The no %d is not a prime no.\n", n);
   else  printf("The no %d is a prime no.\n", n);

    getch();
}
```

Example: /* A menu driven progrnam */

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int choice;
   clrscr();
while(1)
    {  printf("1. Creat Database\n");
       printf("2. Insert new record\n");
       printf("3. Modify a record\n");
       printf("4. Display all records\n");
       printf("5. Exit\n");
     printf("Enter your choice :");
     scanf("%d", &choice);
 switch(choice)
  {
     case 1 :
          printf(" Database created ....\n\n");
          break;
     case 2 :
          printf(" Record inserted ....\n\n");
          break;
     case 3 :
          printf(" Record modified ....\n\n");
          break;
     case 4 :
          printf(" Record displayed ....\n\n");
          break;
```

```
    Case 5 :
        exit(1);
    default :
        printf (" Wrong choice \n");
    }
}
  getch();
}
```

Output :

```
1. Create Database
2. Insert new record
3. Modify a record
4. Display all records
5. Exit
Enter your choice : 3
Record modified .....
```

The statement while(1) puts the entire logic in an infinite loop. This is necessary since the menu must keep reappearing on the screen once an item is selected and an appropriate action taken.