Loop

Session 6

Objectives

- Understand ‘for’ loop in ‘C’
- Work with comma operator
- Understand nested loops
- Understand the ‘while’ loop and the ‘do-while’ loop
- Work with break and continue statements
- Understand the exit() function

What is a Loop?

Section of code in a program which is executed repeatedly, until a specific condition is satisfied

3 types of Loop Structures

The for loop

The while loop

The do….while loop

The **for** loop-1

Syntax

```
for (initialize counter; conditional test; re-evaluation parameter)
{
        statement
}
```

- The initialize counter is an assignment statement that sets the loop control variable, before entering the loop
- The conditional test is a relational expression, which determines, when the loop will exit
- The evaluation parameter defines how the loop control variable changes, each time the loop is executed

The **for** loop-2

- The three sections of the **for** loop must be separated by a semicolon(;)

- The statement, which forms the body of the loop, can either be a single statement or a compound statement

- The **for** loop continues to execute as long as the conditional test evaluates to true. When the condition becomes false, the program resumes on the statement following the **for** loop

The for loop-3

```
/*This program demonstrates the for loop in a C program */
  #include <stdio.h>

  main()
  {
            int count;
            printf("\tThis is a \n");

            for(count = 1; count <=6 ; count++)
                  printf("\n\t\t nice");

            printf("\n\t\t world. \n");
  }
```

Example

The Comma Operator

The scope of the **for** loop can be extended by including more than one initializations or increment expressions in the for loop specification

The format is : exprn1 , exprn2 ;

Example

```
#include <stdio.h>
main()
{
          int i, j , max;
          printf("Please enter the maximum value \n");
          printf("for which a table can be printed: ");
          scanf("%d", &max);

          for(i = 0 , j = max ; i <=max ; i++, j--)
                printf("\n%d  +  %d  =  %d",i, j, i + j);
}
```

Nested for Loops-1

The **for** loop will be termed as a **nested for** loop when it is written as follows

```
for(i = 1; i<max1; i++)
        {
                        .
                        .
                for(j = 0; j < = max2; j++)
                {
                        .
                        .
                }
                .
                .
        }
```

Nested for Loops-2

Example

```
#include <stdio.h>
  main()
  {
    int i, j, k;
    i = 0;
    printf("Enter no. of rows :");
    scanf("%d", &i);
    printf("\n");
    for (j = 0; j < i ; j++)
    {
      printf("\n");
      for (k = 0; k <= j; k++) /*inner for loop*/
      printf("*");
    }
  }
```

The while Loop-1

Syntax

```
while (condition is true)
  statement ;
```

The while loop repeats statements while a certain specified condition is True

The while Loop-2

```
/* A simple program using the while loop */

  #include <stdio.h>
  main()
     {
      int count = 1;
      while( count <= 10)
      {
            printf("\n This is iteration %d\n",count);
            count++;
      }

      printf("\n The loop is completed. \n");
         }
```

Example

do…while Loop-1

Syntax

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

- In the do while loop the body of the code is executed once before the test is performed

- When the condition becomes False in a **do while** the loop will be terminated, and the control goes to the statement that appears immediately after the while statement

do...while Loop-2

Example

```
#include <stdio.h>
   main ()
    {
   int num1, num2;
              num2 = 0;
   do
   {
        printf( "\nEnter a number : ");
        scanf("%d",&num1);
        printf( " No. is %d",num1);
        num2++;
   } while (num1 != 0);
   printf ("\nThe total numbers entered were %d",--num2);

/*num2 is decremented before printing because count for last 
integer (0) is not to be considered */
   }
```

Jump Statements-1

return expression

- The return statement is used to return from a function
- It causes execution to return to the point at which the call to the function was made
- The return statement can have a value with it, which it returns to the program

Jump Statements-2

goto label

- The goto statement transfers control to any other statement within the same function in a C program
- It actually violates the rules of a strictly structured programming language
- They reduce program reliability and make program difficult to maintain

Jump Statements-3

break statement

- The break statement is used to terminate a case in a switch statement
- It can also be used for abrupt termination of a loop
- When the break statement is encountered in a loop, the loop is terminated immediately and control is passed to the statement following the loop

break statement

Example

```
#include <stdio.h>
  main ()
  {
     int count1, count2;
     for(count1 = 1, count2 = 0;count1 <=100; count1++)
     {
        printf("Enter %d count2 : ",count1);
        scanf("%d", &count2);
        if(j==100) break;
     }
```

Jump Statements-4

continue statement

- The continue statement causes the next iteration of the enclosing loop to begin
- When this statement is encountered, the remaining statements in the body of the loop are skipped and the control is passed on to the re-initialization step

continue statement

Example

```
#include <stdio.h>
   main ()
   {
      int num;
      for(num = 1; num <=100; num++)
      {
         if(num % 9 == 0)
                continue;
         printf("%d\t",num);
      }
   }
```

Jump Statements-5

exit() function

- The exit() is used to break out of the program
- The use of this function causes immediate termination of the program and control rests in the hands of the operating system