NIELIT GOPRAKHPUR

Course Name: O Level (2nd Sem B1 Batch) **Subject: C Language**
Topic: C Loops **Date: 19-03-2020**

C Loops

Introduction:

Looping is one of the key concepts on any programming language. It executes a block of statements number of times until the condition becomes false. **Loops** are of 2 types: entry-controlled and exit-controlled. '**C**' programming provides us 1) while 2) do-while and 3) for **loop**.

While Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

```
while (condition) {

        statements;
}
```

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

After exiting the loop, the control goes to the statements which are immediately after the loop.

The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory. It is a good practice though to use the curly braces even we have a single statement in the body.

In while loop, if the condition is not true, then the body of a loop will not be executed, not even once. It is different in do while loop which we will see shortly.

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;          //initializing the variable
        while(num<=10)   //while loop with condition
        {
                printf("%d\n",num);
                num++;              //incrementing operation
        }
        return 0;
}
```

Output:

```
1
2
3
4
```

```
5
6
7
8
9
10
```

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.
The basic format of while loop is as follows:

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

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.
In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.
Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.
The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)
The following program illustrates the working of a do-while loop:
We are going to print a table of number 2 using do while loop.

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int num=1;		//initializing the variable
	do		//do-while loop
	{
		printf("%d\n",2*num);
		num++;			//incrementing operation
	}while(num<=10);
	return 0;
}
```

Output:

```
2
4
6
8
10
12
14
16
```

```
18
20
```

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the use of a simple for loop:

```
#include<stdio.h>
int main()
{
        int number;
        for(number=1;number<=10;number++)        //for loop to print 1-10 numbers
        {
                printf("%d\n",number);           //to print the number
        }
        return 0;
}
```

Output:

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