CL118

Programming Fundamentals

Lab 05

Iterative Control statements in C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

LAB 05

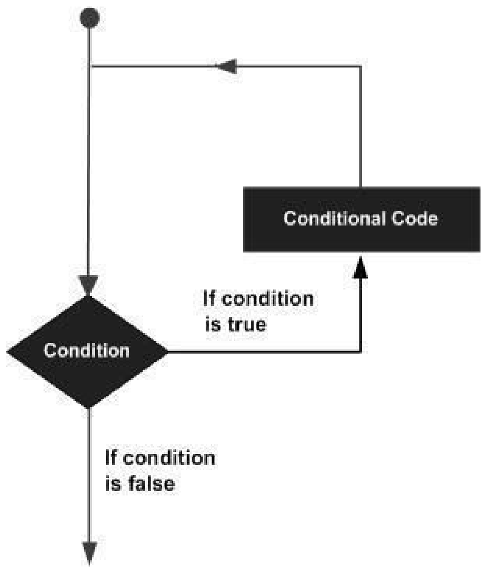
maham.mobin@nu.edu.pk | atiya.jokhio@nu.edu.pk

Learning Objectives

- Introduction to iterative statements
- While loop
- Do while loop
- Nested loops
- Break and continue statements

Iterative control flow is also referred to as repetition logic or loop. It executes the instructions repetitively multiple number of times. It allows us to execute statements or set of statements multiple number of times on the basis of the condition and just by writing down them once.

Iteration is the act of repeating the statements and each of the repetition is also called as iteration. Given below is the general form of a loop statement in most of the programming languages



C programming language provides the following types of loops to handle looping requirements:

Loop Type in C

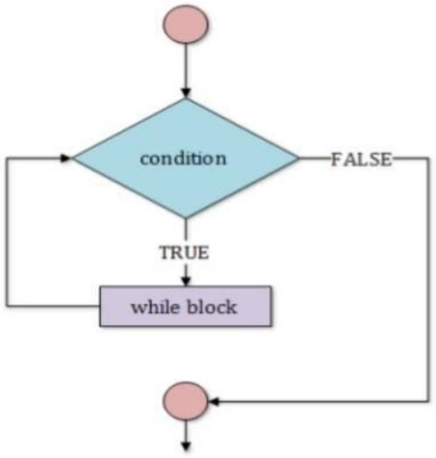
1. While loop

2. Do-while loop
3. For loop

1. **While 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.

In while loop, if the condition is not true, then the body of a loop will not be executed, not even once.



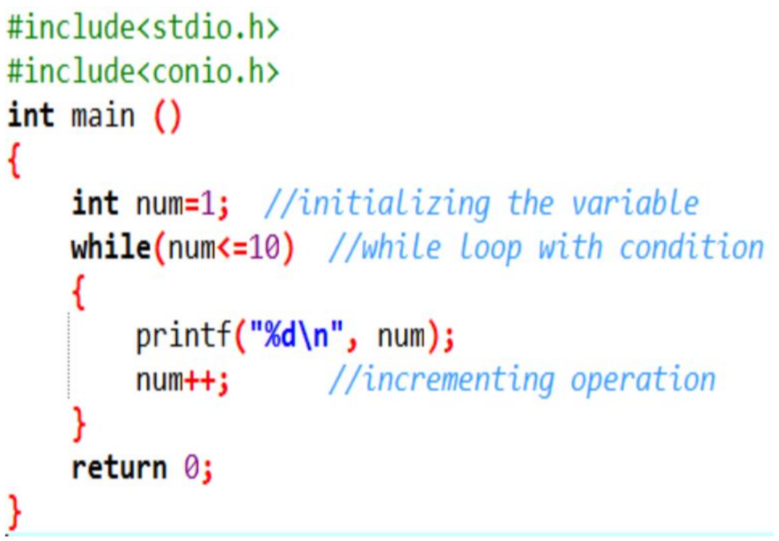
Syntax

The syntax of a while loop is –

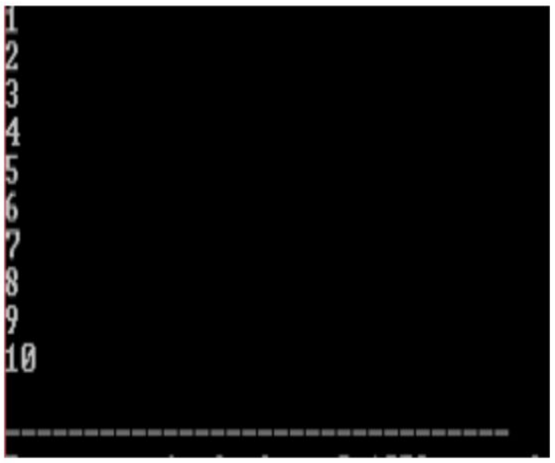
initialize loop counter;

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

- Here, statement(s) may be a single statement or a block of statements.
- The condition may be any expression, and true is any nonzero value.



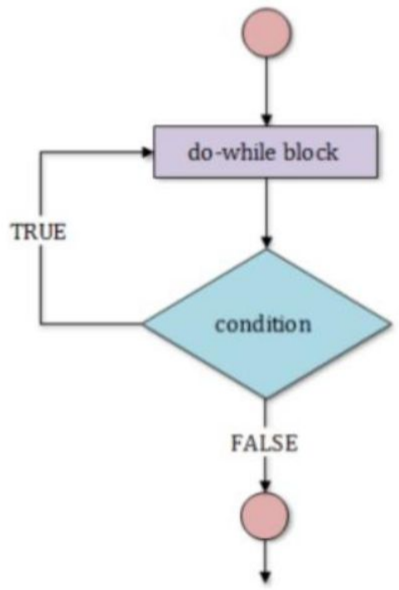
```
#include<stdio.h>
#include<conio.h>
int main ()
{
    int num=1;  //initializing the variable
    while(num<=10)  //while loop with condition
    {
        printf("%d\n", num);
        num++;       //incrementing operation
    }
    return 0;
}
```



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

2. 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.



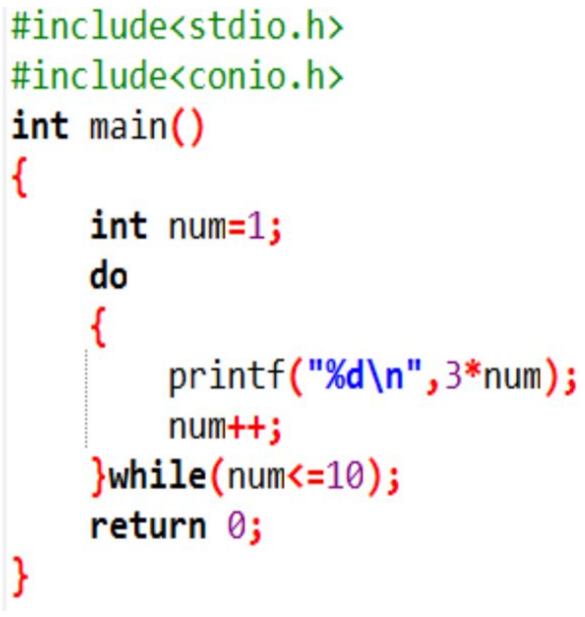
The syntax of while loop is as follows:

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

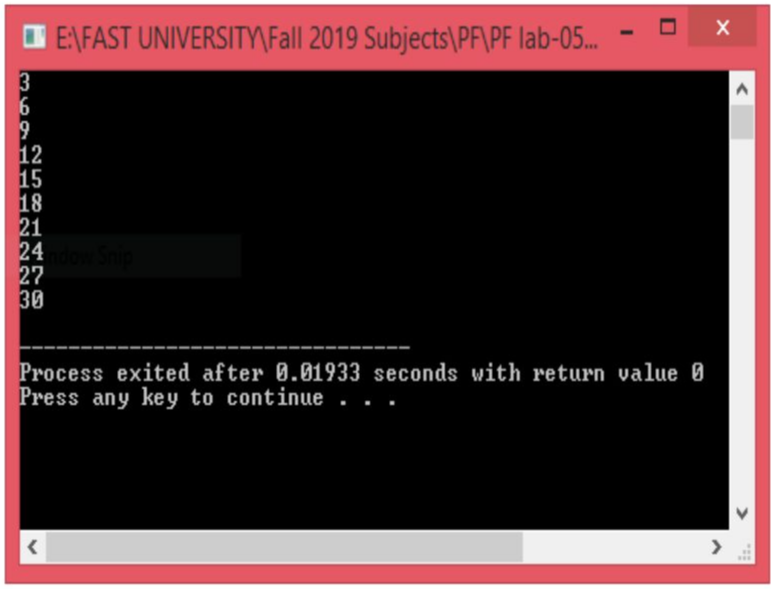
Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Example:



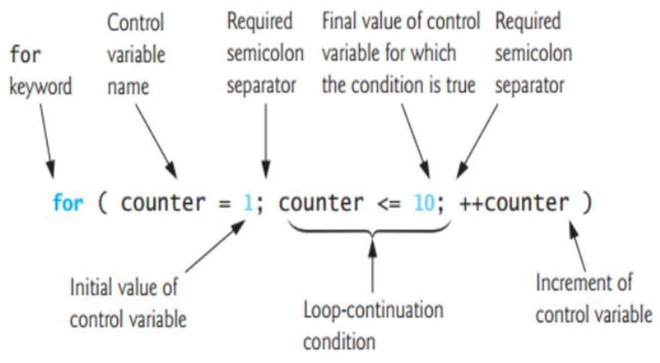
```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int num=1;
    do
    {
        printf("%d\n",3*num);
        num++;
    }while(num<=10);
    return 0;
}
```

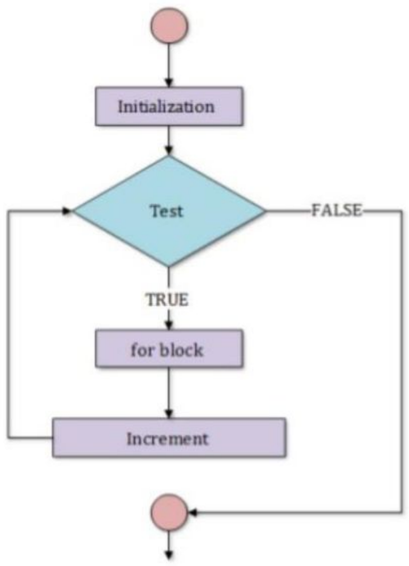


```
3
6
9
12
15
18
21
24
27
30

--------------------------------
Process exited after 0.01933 seconds with return value 0
Press any key to continue . . .
```

3. For loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.


for keyword
Control variable name
Required semicolon separator
Final value of control variable for which the condition is true
Required semicolon separator
for (counter = 1; counter <= 10; ++counter)
Initial value of control variable
Loop-continuation condition
Increment of control variable



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

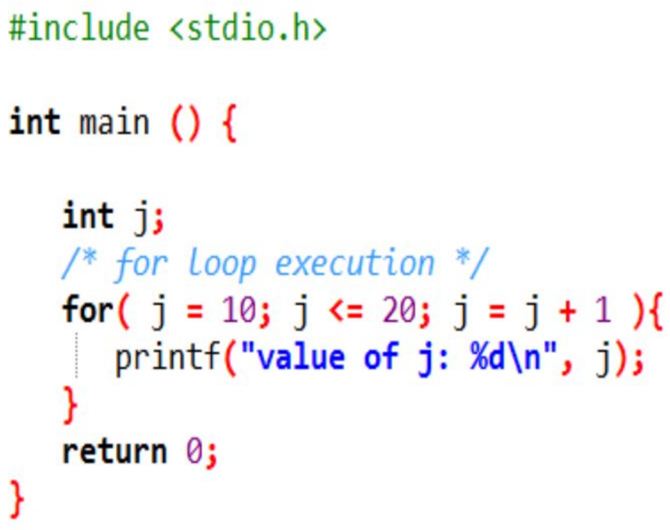
Syntax

```
for (initialization; condition; increment)
{
  statement(s);
}
```

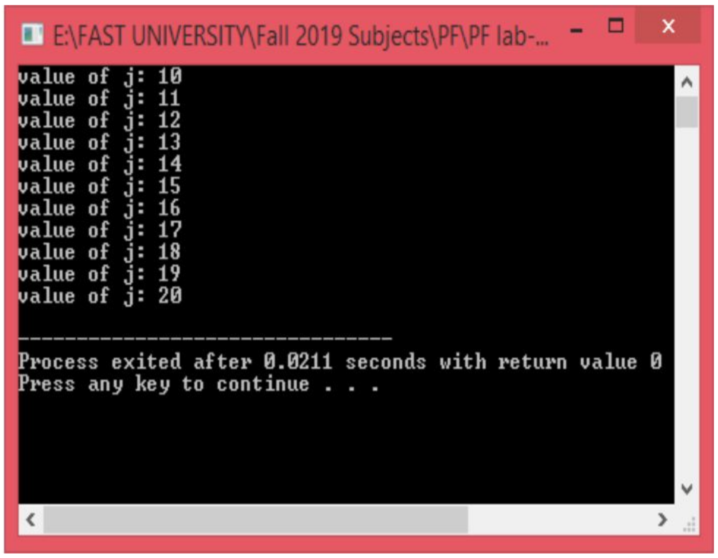
Here is the flow of control in a 'for' loop −

- The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Example:



```c
#include <stdio.h>

int main () {

   int j;
   /* for loop execution */
   for( j = 10; j <= 20; j = j + 1 ){
      printf("value of j: %d\n", j);
   }
   return 0;
}
```



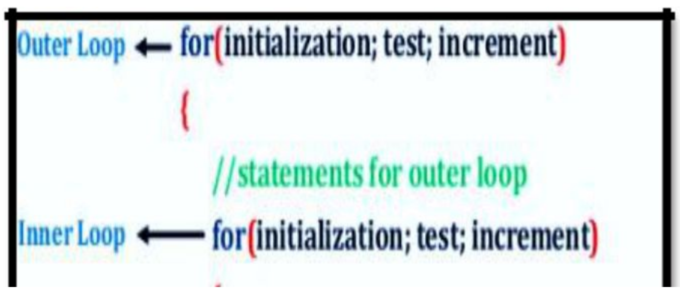
```
value of j: 10
value of j: 11
value of j: 12
value of j: 13
value of j: 14
value of j: 15
value of j: 16
value of j: 17
value of j: 18
value of j: 19
value of j: 20

--------------------------------
Process exited after 0.0211 seconds with return value 0
Press any key to continue . . .
```

Nested loops

The way if statements can be nested, similarly whiles and fors can also be nested, C programming allows to use one loop inside another loop.

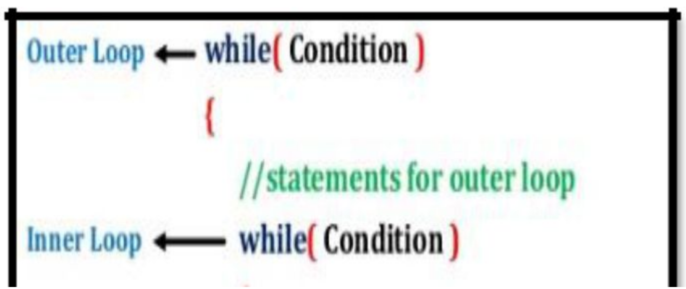
There can be mixed type of nested loop i.e. a for loop inside a while loop, or a while loop inside a do-while loop.

Syntax of Nested for:



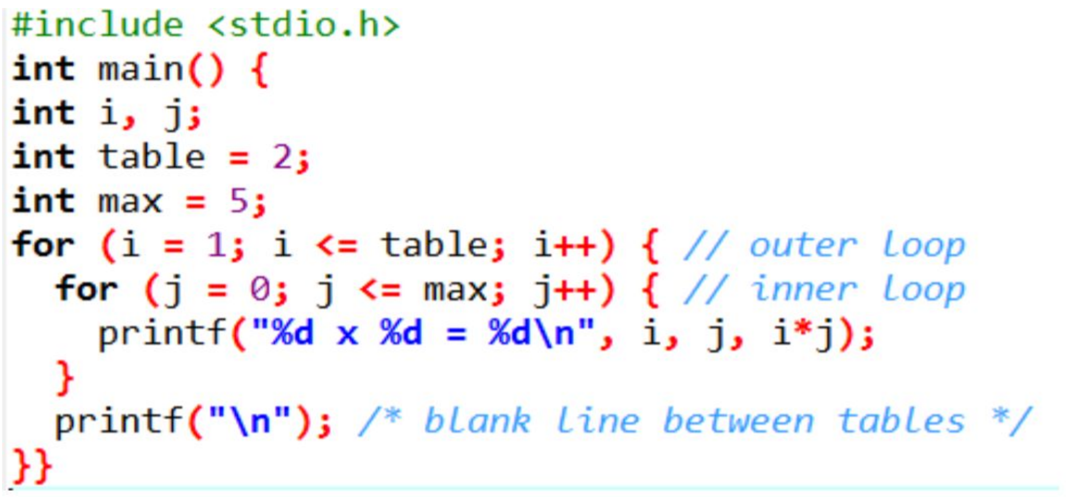
```
Outer Loop ←  for(initialization; test; increment)
                {
                   //statements for outer loop
Inner Loop ←──   for(initialization; test; increment)
```

Syntax of Nested while:

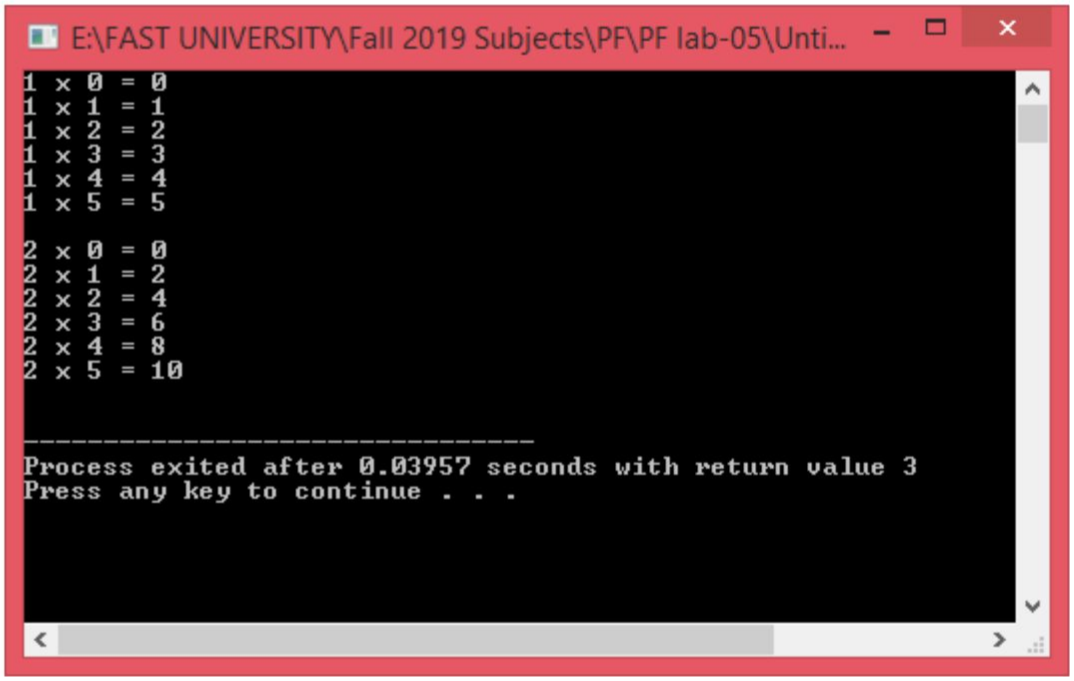


```
Outer Loop ←  while( Condition )
                {
                   //statements for outer loop
Inner Loop ←──   while( Condition )
```

<u>Example:</u>



```c
#include <stdio.h>
int main() {
int i, j;
int table = 2;
int max = 5;
for (i = 1; i <= table; i++) { // outer loop
  for (j = 0; j <= max; j++) { // inner loop
    printf("%d x %d = %d\n", i, j, i*j);
  }
  printf("\n"); /* blank line between tables */
}}
```



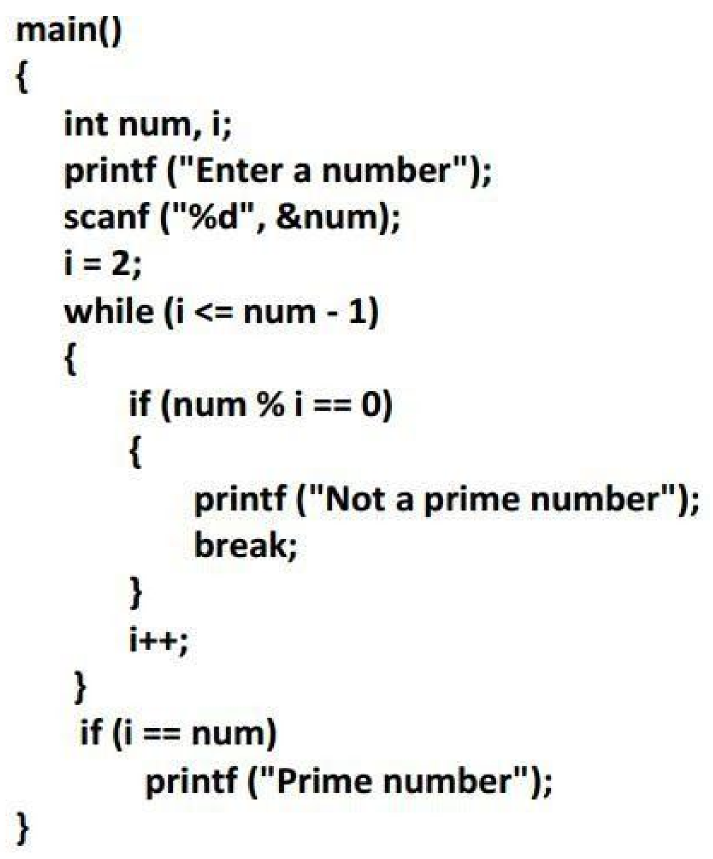
```
1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5

2 x 0 = 0
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10


--------------------------------
Process exited after 0.03957 seconds with return value 3
Press any key to continue . . .
```

- **The BREAK Statement**

We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword break allows us to do this. When break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an if. As an example, let’s consider the following example.

Example: Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

All we have to do to test whether a number is prime or not, is to divide it successively by all numbers from 2 to one less than itself. If remainder of any of these divisions is zero, the number is not a prime. If no division yields a zero, then the number is a prime number. Following program implements this logic.



```
main()
{
    int num, i;
    printf ("Enter a number");
    scanf ("%d", &num);
    i = 2;
    while (i <= num - 1)
    {
        if (num % i == 0)
        {
            printf ("Not a prime number");
            break;
        }
        i++;
    }
     if (i == num)
         printf ("Prime number");
}
```

The CONTINUE statement

In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword continue allows us to do this. When continue is encountered inside any loop, control automatically passes to the beginning of the loop. A continue is usually associated with an if. As an example, let's consider the following program.

```c
main()
{
    int i, j;
    for (i = 1; i <= 2; i++)
    {
        for (j = 1; j <= 2; j++)
        {
            if (i == j)
            continue;
            printf ("\n%d %d\n", i, j);
        }
    }
}
```

The output of the above program would be...

```
1 2
2 1
```

Note that when the value of i equals that of j, the continue statement takes the control to the for loop (inner) bypassing rest of the statements pending execution in the for loop (inner).

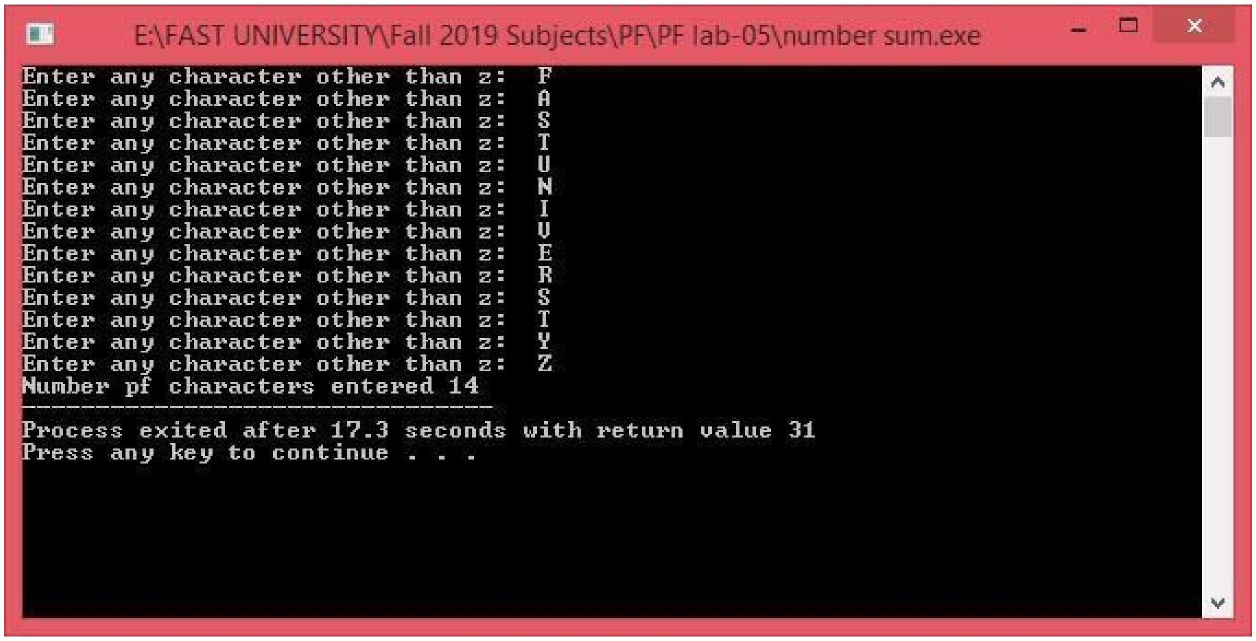
Lab Activity

Task-01

Write a program in which user continuously enters integer numbers, the program stops when the user enters any number other than the numbers between 200 and 400. Finally, it displays the sum of all the numbers entered by the user.

Task-02

Write a program in which user continuously enters a character, the program stops when the user enters 'z' character and finally displays the number of characters entered.

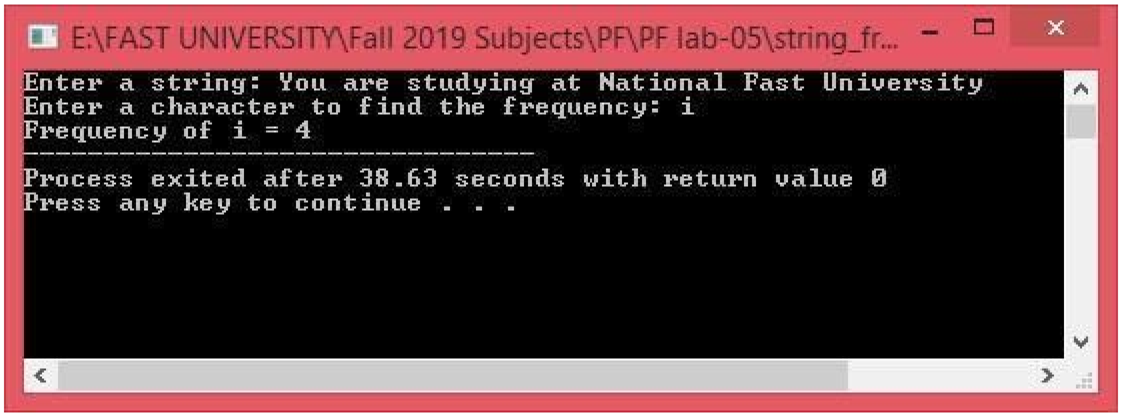

E:\FAST UNIVERSITY\Fall 2019 Subjects\PF\PF lab-05\number sum.exe

```
Enter any character other than z:  F
Enter any character other than z:  A
Enter any character other than z:  S
Enter any character other than z:  T
Enter any character other than z:  U
Enter any character other than z:  N
Enter any character other than z:  I
Enter any character other than z:  V
Enter any character other than z:  E
Enter any character other than z:  R
Enter any character other than z:  S
Enter any character other than z:  T
Enter any character other than z:  Y
Enter any character other than z:  Z
Number pf characters entered 14
--------------------------------
Process exited after 17.3 seconds with return value 31
Press any key to continue . . .
```

Task-03

Write a program that generates and displays the first N three-digit odd numbers. Whereas the number N is provided by the user.

Task-04


E:\FAST UNIVERSITY\Fall 2019 Subjects\PF\PF lab-05\string_fr...

```
Enter a string: You are studying at National Fast University
Enter a character to find the frequency: i
Frequency of  i = 4
--------------------------------
Process exited after 38.63 seconds with return value 0
Press any key to continue . . .
```

Task-05

Write a program that asks user to input the starting number and ending number of the range.

The program should display the number of multiples of 5 in between that range.

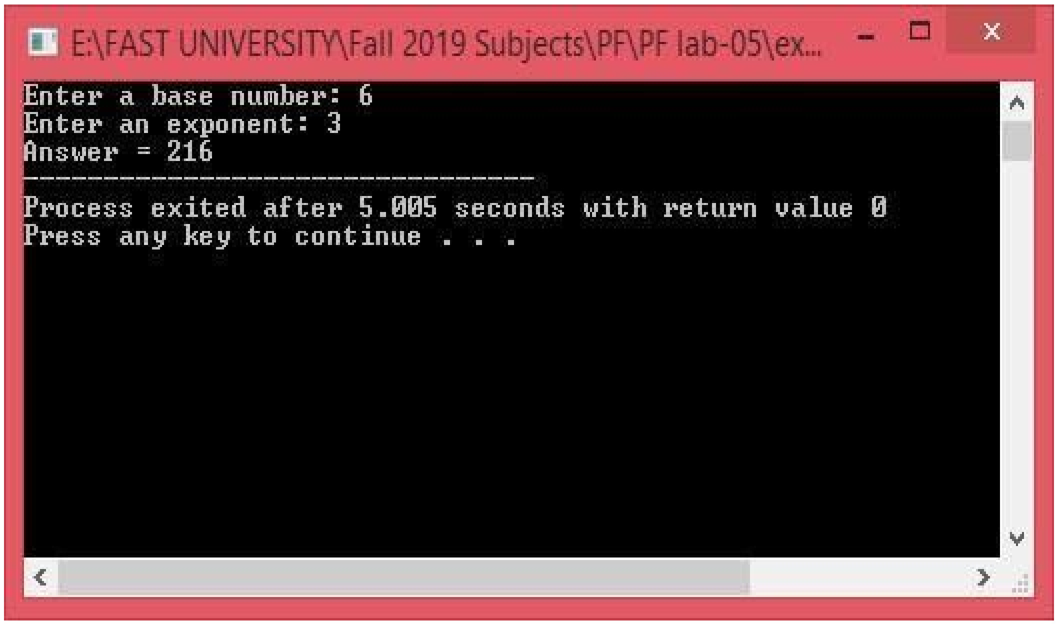
Task-06

Write a program to print half pyramid using numbers.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Task-07:

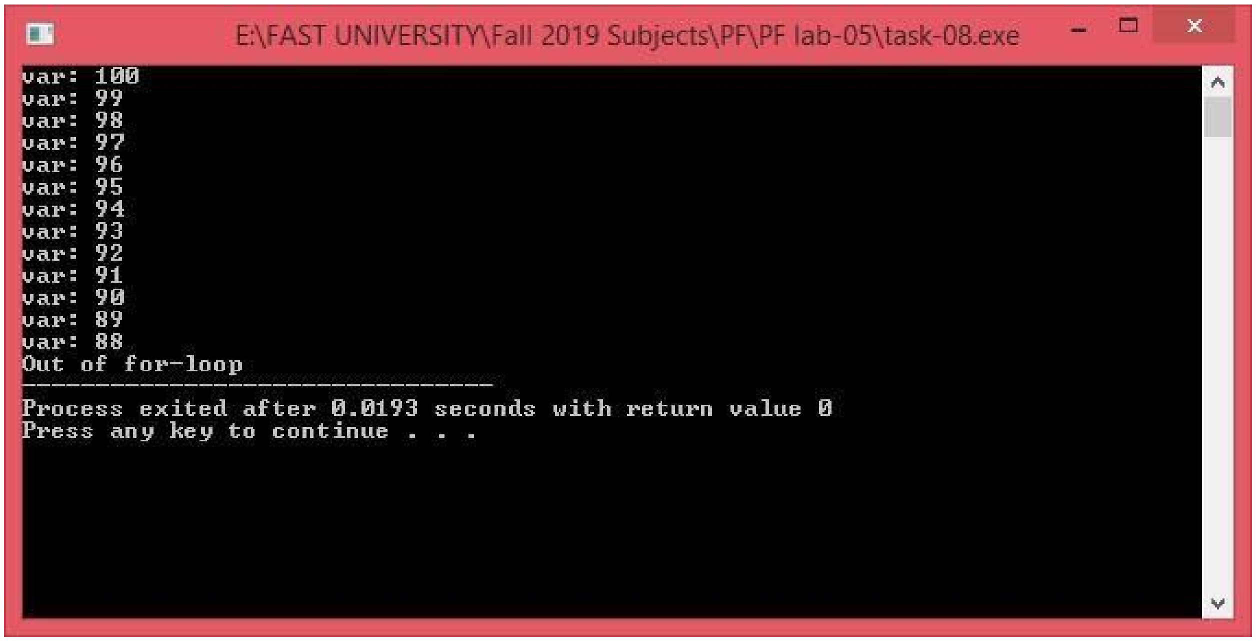
Write a program that takes two integers from the user (a base number and an exponent) and calculates the power.



```
Enter a base number: 6
Enter an exponent: 3
Answer = 216
--------------------------------
Process exited after 5.005 seconds with return value 0
Press any key to continue . . .
```

Task-08:

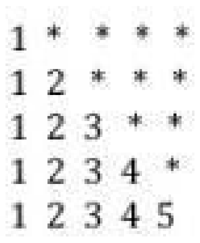
Write a program that uses break statement when value of variable becomes 88 and print the statement that “Out of for loop”. Out should be like the displayed below.


E:\FAST UNIVERSITY\Fall 2019 Subjects\PF\PF lab-05\task-08.exe

```
var: 100
var: 99
var: 98
var: 97
var: 96
var: 95
var: 94
var: 93
var: 92
var: 91
var: 90
var: 89
var: 88
Out of for-loop
--------------------------------
Process exited after 0.0193 seconds with return value 0
Press any key to continue . . .
```

Task-09

Write a program to print following pattern.



```
1 * * * *
1 2 * * *
1 2 3 * *
1 2 3 4 *
1 2 3 4 5
```