United International University

Dept. of Electrical and Electronic Engineering (EEE)

Course No. : **EEE 122**
Course Title: **Structured Programming Laboratory**

Lab Sheet 4
Iteration Statements in C

Outcomes

After finishing this lab students should be able to ...

1. use the **`while`** repetition statement to execute statements in a program repeatedly.
2. use the **`do...while`** repetition statements to execute statements repeatedly.
3. use the **`for`** statements to execute statements repeatedly
4. use the **`break`** and **`continue`** statements to alter the flow of control.

Contents

1 C - Iteration

In C programming generally statements are executed sequentially. But often programmer needs to execute a block of code several number of times. A iteration/loop statement allows us to execute a statement or group of statements multiple times.

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

1. **`while loop`** : Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2. **`for loop`** : Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3. **`do...while loop`** : It is more like a while statement, except that it tests the condition at the end of the loop body.
4. **`nested loops`** : You can use one or more loops inside any other while, for, or do..while loop.

1.1 `while` loop in C

A **`while`** loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

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

Flow Diagram

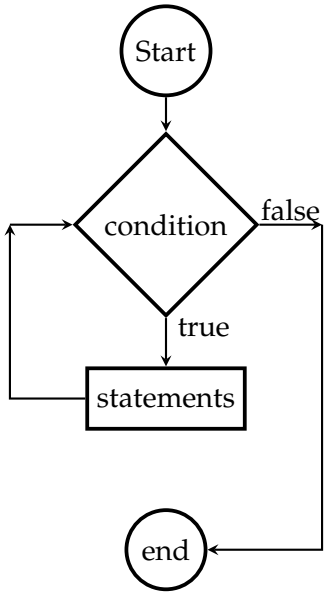

Fig: **`while`** loop flow chart

Example

```
#include <stdio.h>

int main (void) {
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 ) {
        printf("value of a: %d\n", a);
        a++;
    }
    return 0;
}
```

1.2 `for` loop in C

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.

Syntax

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

Flow Diagram

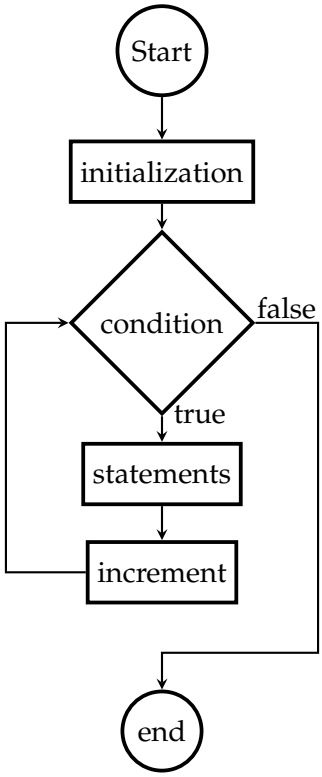

Fig: **for** loop flow chart

Example

```
#include <stdio.h>

int main (void) {
    int a;

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

1.3 do...while loop in C

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

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

Flow Diagram

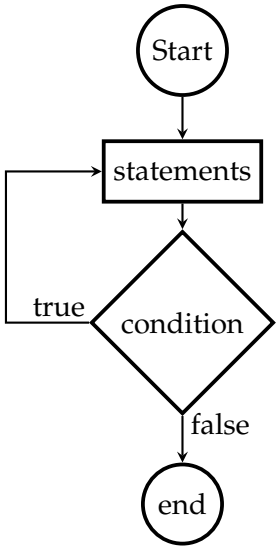

Fig: **do...while** loop flow chart

Example

```c
#include <stdio.h>

int main (void) {
    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do {
        printf("value of a: %d\n", a);
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

1.4 nested loop in C

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax

```c
for ( init; condition; increment ) {
    for ( init; condition; increment ) {
        statement(s);
    }
    statement(s);
}

or,
while(condition) {
    while(condition) {
        statement(s);
    }
    statement(s);
}

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

Example

```c
#include <stdio.h>

int main (void) {
    /* local variable definition */
    int i, j;

    for(i = 2; i<100; 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", i);
        }
    }

    return 0;
}
```

2 Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
C supports the following control statements.

1. **`break`** statement
2. **`continue`** statement
3. **`goto`** statement

2.1 `break` statements

The break statement in C programming has the following two usages

1. When a **`break`** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
2. It can be used to terminate a case in the **`switch`** statement (covered in the next chapter).

Syntax

```c
break;
```

Flow Diagram

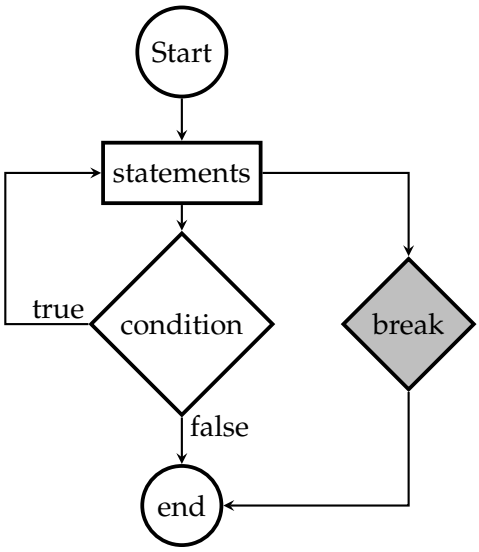

Fig: **break** statement flow chart

Example

```c
#include <stdio.h>

int main (void) {
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 ) {
        printf("value of a: %d\n", a);
        a++;
        if( a > 15) {
            /* terminate the loop using break statement */
            break;
        }
    }
    return 0;
}
```

2.2 continue statements

The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, continue statement causes the program control to pass to the conditional tests.

Syntax

```c
continue;
```

Flow Diagram

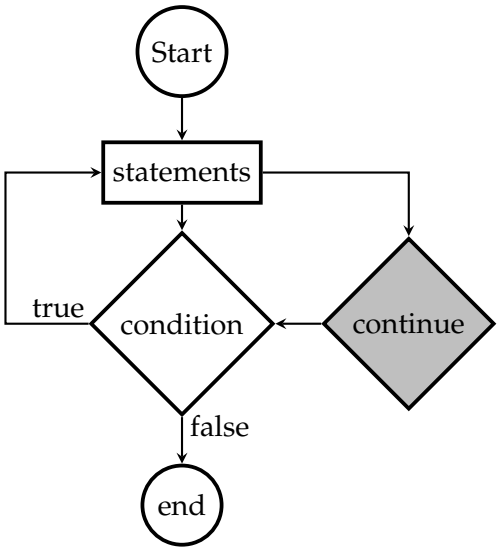

Fig: **continue** statement flow chart

Example

```c
#include <stdio.h>

int main (void) {
    /* local variable definition */
    int a = 10;
    /* do loop execution */
    do {
```

```
        if( a == 15) {
            /* skip the iteration */
            a = a + 1;
            continue;
        }
        printf("value of a: %d\n", a);
        a++;
    } while( a < 20 );

return 0;
}
```

2.3 `goto` statements

A **`goto`** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

Syntax

```
goto label;
..
.
label: statement;
```

Flow Diagram

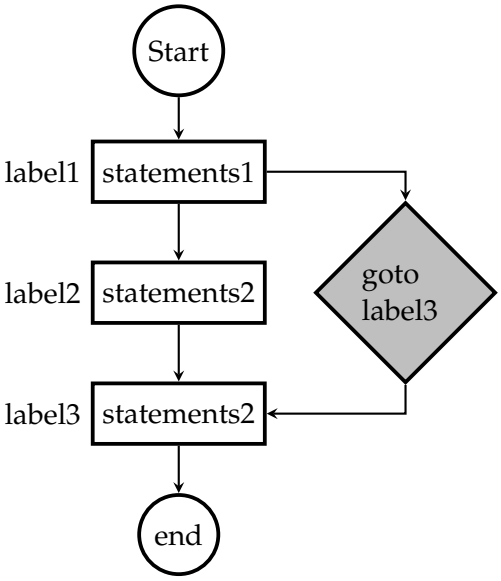

Fig: **`goto`** statement flow chart

Example

```
#include <stdio.h>

int main (void) {
/* local variable definition */
    int a = 10;
    /* do loop execution */
    LOOP:do {
        if( a == 15) {
            /* skip the iteration */
            a = a + 1;
            goto LOOP;
        }
        printf("value of a: %d\n", a);
        a++;
    }while( a < 20 );
return 0;
}
```

3 The Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

Example

```
#include <stdio.h>

int main (void) {
    for( ; ; ) {
        printf("This loop will run forever.\n");
    }
    return 0;
}
```

4 Programming Examples

Example: 1

Description: Write a program in C which takes a number from user and test it whether it is a prime number or not.

Source Code

```
#include <stdio.h>
int main(void){
    int i,n,flag;
    printf("\nEnter the number:");
    scanf("%d",&n);
    for(i=2;i<=n-1;i++){
        if(n%i==0)
        flag=1;
    }
    if(flag==1)
        printf("\n%d is not a prime number.",n);
    else
        printf("\n%d is a prime number.",n);
    return 0;
}
```

Output

```
Enter the number:11
11 is a prime number.
```

Example: 2

Description: Write a program in C which finds the factorial of any number.

Source Code

```
#include <stdio.h>
int main(void){
    int number=1;
    long int answer;
    printf("\nEnter a number: ");
    scanf("%d", &number);
    if(number<0)
        printf("\nNo value of factorial for negative value.");
    else if(number==0)
        printf("\nFactorial is:1");
    else if( number >= 1 ) {
        answer=1;
        while(number>0){
            answer = answer * number--;
        }
        printf("Factorial is: %ld\n", answer);
    }
    return 0;
}
```

Output

```
Enter a number: 10
Factorial is: 3628800
```

Example: 3

Description:Write a program in C which converts a binary number to a decimal number.

Source Code

```
#include <stdio.h>
int main(void){
    long int n,k,rem,d,a=1,dec=0;
    printf("\nEnter the number in binary:");
    scanf("%ld",&n);
    for(k=n;k>0;k/=10){
        rem=k%10;
        d=rem*a;
        dec+=d;
        a*=2;
    }
    printf("\nThe decimal equivalent is:%d",dec);
    return 0;
}
```

Output

```
Enter the number in binary: 10010101101
The decimal equivalent is:1997
```

Example: 4

Description: Write a program in C which finds the gcf of two given integers.

Source Code

```
#include <stdio.h>
int main(void){
    int a,b,c;
    printf("\nEnter the numbers:\n");
    scanf("%d %d",&a,&b);
    while(b!=0){
        c=a%b;
        a=b;
        b=c;
    }
    printf("\ngcf is:%d",a);
    return 0;
}
```

Output

```
Enter the numbers:16 8
gcf is:8
```

Example: 5

Description: Write a program in C which finds the sum of the series: $1^2 + 2^2 + 3^2 + ... + n^2$, n taking from keyboard. You are not allowed to use algebraic summation low.

Source Code

```
#include <stdio.h>
int main(void){
    long int n,sum=0,i;
    printf("\nHow many numbers?");
    scanf("%ld",&n);
    for(i=0;i<=n;i++){
        sum+=i*i;
    }
    printf("\nSum=%ld",sum);
    return 0;
}
```

Output

```
How many numbers?5
Sum=55
```

5 Practice session

Sl	Source Code
Practice 1	

```c
#include <stdio.h>
int main(void){
    int i=10;
    do {
        printf("i=%d\n",i);
        i=i-3;
    }while(i) ;
    return 0;
}
```

Practice 2	

```c
#include <stdio.h>
int main(void){
    int i;
    for(i=1;i<10;i++){
        if(i==3)
        continue;
        printf("%d ",i);
    }
    return 0;
}
```

Practice 3	

```c
#include <stdio.h>
int main(void){
    do {
        printf ( "Hello there \n") ;
    } while ( 4 < 1 ) ;
    return 0;
}
```

Practice 4	

```c
#include <stdio.h>
int main(void){
    int i,j=10;
    for(;i=j;j-=2)
        printf("%d\n",j);
    return 0;
}
```

6 Lab Assignments

1. Write a program to print all prime numbers from 1 to 300.
2. Write a program to generate all combinations of 1, 2 and 3 using for loop.
3. Write a C program to print the following number pyramid:

```
1
12
123
1234
12345
.
.
.
```

   Number of lines are taken from keyboard.

4. Write a C program to find the value of ${}^{n}P_{r}$. n and r are taken from the keyboard .

Acknowledgment

First OBE version, prepared by:
B.K.M. Mizanur Rahman,
Assistant Professor,
Department of EEE, UIU

Second Update , prepared by:
Nazmul Alam,
Part Time Faculty,
Department of EEE, UIU