Lab 06: Loops

Familiarization with Iteration control statements and the While loop in specific.

# 7.1 Repetition Structure

In our daily life, many activities are repetitive in nature. For example in data processing, we have operations such as read students’ records one at a time, until the end of the file; add a list of fifty numbers; or repeatedly request for a data until it is valid.

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

In programming, such repeated actions are known as loops. A loop statement allows us to execute a statement or group of statements multiple times. Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming.

.Basically, a Loop (iteration) control statement allows you to repeat one or more statements until some condition becomes true. This type of control statement is what makes computers so valuable. A computer can repeatedly execute the same instructions over-and-over again without getting bored with the repetition.

A infinite loop is one that goes on forever, and that must be avoided.

There are basically two types of loop control: one is known as counter-controlled repetition where the number of times of repetition is known beforehand, and the other is the sentinel-controlled repetition where the loop stops when a special sentinel value is encountered, or when the terminating condition is met.

C provides 3 loop constructs:

## ‘while’

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

## ‘do-while’

It is more like a while statement, except that it tests the condition at the end of the loop body.

## ‘for’

It is more like a while statement, except that it tests the condition at the end of the loop body.

# 7.2 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

Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

### 2. continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

### 3. goto statement

Transfers control to the labeled statement.

# 7.3 The ‘while’ loop

The while loop checks whether the test expression is true or not. If it is non-zero (true), the *statement* in the loop body is executed and control is then passed back to the beginning of the loop, and the condition is tested again i.e. the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false. If it is zero (false), then the loop terminates and control is passed to the next statement after the ‘while’ construct.

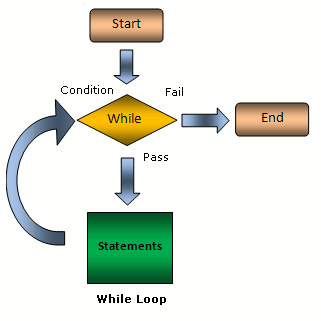
It is possible that the loop body is not executed at all, if the loop condition is false at entry.

## 7.3.1 Syntax of the ‘while’ loop

while (test expression) {

statement/s to be executed.

}



## EXAMPLE 7.1

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* while loop execution \*/

while( a < 20 ) {

printf("value of a: %d\n", a);

a++;

}

return 0;

}

|  |
| --- |
| Output: |

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

## EXAMPLE 7.2

#include <stdio.h>

int main(void)

{

      int j;

      j = -5;

      // while loop

      while(j <= 0)

      {

            printf("%d ", j);

            j = j + 1;

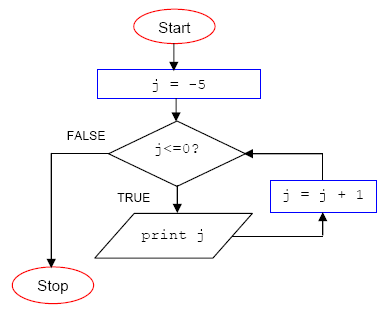
      }

      return 0;

}

|  |
| --- |
| Flow chart: |

.



# Student Tasks:

## Task 1:

Write a program to print the table (till 10) of the integer taken input from the user.

## Task 2:

Write a program that prints squares of the integer taken as input, until the square is less than 1000.

## Task 3:

Write a program that takes character input. Terminate when ‘X’ is pressed.

## Task 4:

Write a program that takes character input unless Enter is pressed and print total number of inputs by the user.

## Task 5:

Write a program to generate a series of

1. first 50 even numbers
2. first 50 odd numbers

## Task 6:

Print the following series:

1. 1,2,3,…………….30
2. 1,2,2,3,3,4,4,5,…..50
3. 0,1,1,2,3,5,8,…….100