# Department of Computing

**[CS212]: [Object Oriented Programming]**

**Class: [BSCS 5C]**

# Lab [1]: [Re-visiting Structured Programming]

**Date: [Feb 1, 2016]**

**Time: [10:00-1:00]**

# Instructor: [Um-e-Ghazia]

# 

# Lab [1]: Re-visiting Structured Programming

**Introduction**

A procedural program is written as a list of instructions, telling the computer, step-by-step, what to do: Open a file, read a number, multiply by 4, display something. Program units include the main or program block, subroutines, functions, procedures; file scoping; includes/modules; libraries.

Procedural programming is fine for small projects. It is the most natural way to tell a computer what to do, and the computer processor's own language, machine code, is procedural, so the translation of the procedural high-level language into machine code is straightforward and efficient. What is more, procedural programming has a built-in way of splitting big lists of instructions into smaller lists: the function. This lab is about revising these concepts.

**Objectives**

The objective of this lab is review of procedural programming C.

**Tools/Software Requirement**

Microsoft Visual Studio 2010/2012

**Description**

You are required to complete the given tasks using procedural language C. The idea is to review the procedural language C that you have already understood. Remember in structured approach functionality is identified and functions are created.

**Lab Tasks**

**Lab Task 1**

Follow the given details of each topic with the lab practices step by step to understand the structured programming concepts.

# Variables

A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer.

## Naming Variables

The name of variable can be called identifier or variable name in a friendly way. It has to follow these rules:

* The name can contain letters, digits and the underscore but the first letter has to be a letter or the underscore. Be avoided underscore as the first letter because it can be clashed with standard system variables.
* The length of name can be up to 247 characters long in Visual C++ but 31 characters are usually adequate. Keywords cannot be used as a variable name.

Of course, the variable name should be meaningful to the programming context.

## Declaring Variables

To declare a variable you specify its name and kind of data type it can store. The variable declaration always ends with a semicolon, for example:

|  |  |
| --- | --- |
| int counter; |  |
| char ch; |  |

You can declare variables at any point of your program before using it. The best practice suggests that you should declare your variables closest to their first point of use so the source code is easier to maintain. In C programming language, declaring a variable is also defining a variable.

## Initializing Variables

You can also initialize a variable when you declare it, for example:

|  |  |
| --- | --- |
| **int x = 10;** |  |
| **char ch = 'a';** |  |

|  |
| --- |
| Lab Practice 1 |
| Write a program to input the marks of a student of five different Projects through a keyboard, Find out the aggregate marks and percentage marks obtained by a student. Assume that’s marks obtained by a student are out of 100. |
| #include<stdio.h>  #include<conio.h>  void main()  {  float phy,chem,maths,comp,bio,mobtained,Percentage;  printf("Enter the marks of Physics:");  scanf("%f",&phy);  printf("\nEnter the marks of Chemistry:");  scanf("%f",&chem);  printf("Enter the marks of mathematics:");  scanf("%f",&maths);  printf("Enter the marks of Computer:");  scanf("%f",&comp);  printf("Enter the marks of BIO:");  scanf("%f",&bio);  mobtained=phy+chem+maths+comp+bio;  Percentage=(mobtained/500)\*100;  printf("The aggregated marks are %f\n",mobtained);  printf("The aggregated marks are %f",Percentage);  getch();  } |

# Conditional structure in C

# A control structure is an instruction, statement or group of statements which determines the sequence of execution of other statements.

## If statement

**If** statement is a type of control structure. The basic operation of if statement is that a statement or group of statements is executed under if, if the value of expression is true .

 if (expression)

{

statement;

}

### Lab Practice 2

#include<stdio.h>

#include<conio.h>

void main()

{

//\*\*Example: Take match score from user.\*\*

int score;

Score=275;

if(score>=300)

{

printf("\n\*\*\*Pakistan won \t: %d",score);

}

printf("\n\*\*--\*\*");

getch();

}

The statement if (scores>=300)

 now evaluates to be false because the value of variable score is 275.  The statement under if is not executed. The statement

 printf("\n\*\*\*Pakistan won \t: %d",score);

## If- else statement

**If-else** statement is a type of control structure. The basic operation of if else statement is that a statement or group of statements is executed under if, if the value of expression is true and if the expression is false, statements under **else** are evaluated.

In C language, Statement associated either with if or else are executed not both group of statements are executed. The else clause is optional. The general form of if else statement is:-

if (expression)

{

statement;

}

else

{

            statement;

}

**Note: The curly braces mark the beginning and the end statements under if and else.**

Here is a program which illustrates the functioning of if else statement.

### Lab Practice 3

#include<stdio.h>

#include<conio.h>

void main()

{

//\*\*Example: Take match score from user.\*\*

int score;

Score=275;

if(score>=300)

{

printf("\n\*\*\*Pakistan won and scores are \t: %d",score);

}

else

{

printf("\n\*\*\*England won and scores are \t:%d",score);

}

printf("\n\*\*--\*\*");

getch();

}

The statement

 if(score>=300)

Now evaluates to be false because the value of variable score is 275.  The statement under if is not executed. The statement

 printf("\n\*\*\*Pakistan won and scores are \t: %d",score);

 Under else is executed as the expression evaluates to be false.

### Lab Practice 4

#include <stdio.h>

#include <conio.h>

void main()

{

int x ;

printf("Please Enter the Vale");

scanf("%d",&x);

if ((x>20) && (x<100)) printf("x is inside open interval 20-100");

if ((x<5) || (x>20)) printf("x is not inside closed interval 5-20");

if (!(x>20)) printf("x is smaller or equal to 20");

getch();

}

Note: **AND** operation is true only when both its operands are true. An **OR** operation is true if either one of its operands is true.

**The for loop**

The “for loop” loops from one number to another number and increases by a specified value each time.  
The “for loop” uses the following structure:

**for (Start value; end condition; increase value)**

**Statement;**

## The do while loop

The “do while loop” is almost the same as the while loop. The “do while loop” has the following form:

**do**

**{**

**do statement;**

**}**

**while(expression);**

# Decisions using switch

C switch statement is a muti-way decisions that tests whether a variable or expression matches one of a number of constant integer values, and branches accordingly.

Here is an example of using C switch statement:

**Lab Practice 5**

|  |
| --- |
| #include <stdio.h> |

|  |
| --- |
| int main() { |

|  |
| --- |
| int color = 1; |
| printf("Please choose a color(1: red,2: green,3: blue):\n"); |

|  |
| --- |
| scanf("%d", &color); |
|  |

|  |
| --- |
| case 1: |

|  |
| --- |
| printf("you chose red color\n"); |
| break; |

|  |
| --- |
| case 2: |
| printf("you chose green color\n"); |

|  |
| --- |
| break; |
| case 3: |

|  |
| --- |
| printf("you chose blue color\n"); |
| break; |

|  |
| --- |
| default: |
| printf("you did not choose any color\n"); |

|  |
| --- |
| } |
| return 0; |

|  |
| --- |
| } |
|  |

## Array:

Array is a collection of homogenous data stored under unique name. The values in an array are called as 'elements of an array.' These elements are accessed by numbers called as 'subscripts or index numbers.' Arrays may be of any variable type.

Array is also called as 'subscripted variable.'

## Single / One Dimensional Array:

The array which is used to represent and store data in a linear form is called as 'single or one dimensional array.'

**Syntax:**

<data-type> <array\_name> [size];

### Example:

int a[3] = {2, 3, 5};

char ch[20] = "TechnoExam" ;

float stax[3] = {5003.23, 1940.32, 123.20} ;

**Total Size (in Bytes):**

total size = length of array \* size of data type

In above example, a is an array of type integer which has storage size of 3 elements. The total size would be 3 \* 2 = 6 bytes.

## Memory Allocation:

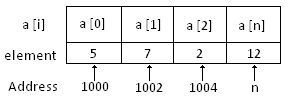


Fig: Memory allocation for one dimensional array

## Lab Practice 6

#include <stdio.h>

#include <conio.h>

void main()

{

int a[3], i;;

clrscr();

printf("\n\t Enter three numbers : ");

for(i=0; i<3; i++)

{

scanf("%d", &a[i]); // read array

}

printf("\n\n\t Numbers are : ");

for(i=0; i<3; i++)

{

printf("\t %d", a[i]); // print array

}

getch();

}

#### Output :

Enter three numbers: 9 4 6

Numbers are: 9 4 6

### Features:

* Array size should be positive number only.
* String array always terminates with null character ('\0').
* Array elements are countered from 0 to n-1.
* Useful for multiple reading of elements (numbers).

### Disadvantages:

* There is no easy method to initialize large number of array elements.
* It is difficult to initialize selected elements.

# Two Dimensional Arrays:

The array which is used to represent and store data in a tabular form is called as 'two dimensional array.' Such type of array specially used to represent data in a matrix form.

The following syntax is used to represent two dimensional arrays.

**Syntax:**

<Data-type> <array\_nm> [row\_subscript][column-subscript];

**Example:**

int a[3][3];

## Memory Allocation :

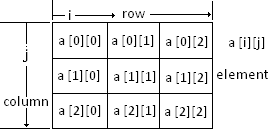


Fig : Memory allocation for two dimensional array

**Lab Practice 7**

Program to demonstrate two dimensional array.

#include <stdio.h>

#include <conio.h>

void main()

{

int a[3][3], i, j;

clrscr();

printf("\n\t Enter matrix of 3\*3 : ");

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

{

scanf("%d",&a[i][j]); //read 3\*3 array

}

}

printf("\n\t Matrix is : \n");

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

{

printf("\t %d",a[i][j]); //print 3\*3 array

}

printf("\n");

}

getch();

}

#### Output:

Enter matrix of 3\*3: 3 4 5 6 7 2 1 2 3

Matrix is:

3 4 5

6 7 2

1 2 3

# **Functions**

Functions are building blocks of the programs. They make the programs more modular and easy to read and manage. All C++ programs must contain the function main( ). The execution of the program starts from the function main ( ). A C program can contain any number of functions according to the needs. The general form of the function is: -

|  |
| --- |
| return\_type  function\_name(parameter list)  {              body of the function    } |

The function consists of two parts function header and function body. The function header is:

|  |
| --- |
| return\_type   function\_name(parameter list) |

|  |
| --- |
| Lab Practice 8 |
| #include<iostream>  using namespace std;  int factorial(int n);    int main ()  {              int n1,fact;              cout <<"Enter the number whose factorial has to be calculated" <<  endl;              cin >> n1;              fact=factorial(n1);              cout << "The factorial of " << n1 << "  is : " << fact << endl;              return(0);    }  int factorial(int n)  {              int i=0,fact=1;              if(n<=1)              {                          return(1);              }              else              {                          for(i=1;i<=n;i++)                          {                                      fact=fact\*i;                          }                          return(fact);              }  } |

# What is a pointer?

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The unary or monadic operator **&** gives the ``address of a variable''. The indirection or dereference operator **\*** gives the ``contents of an object pointed to by a pointer''. To declare a pointer to a variable does:

**int \*pointer;**

# Assigning a Variable's Memory Address to a Pointer

Before we can assign a memory address to a pointer, we need to declare one. Declaring a pointer in C++ is as simple as to declare any other variable with one single difference. Asterix symbol " \* " needs to be add and located after variable type and before a variable name. One rule has to be followed when assigning memory address to a pointer: pointer type has to match with variable type it will point to. One exception is a pointer to void, which can handle different types of variables it will point to. To declare a pointer pMark of type unsigned short int a following syntax is to be used:

## Lab Practice 9

#include <iostream>  
int main()  
{  
 **using** **namespace** std;  
  
// Declare and initialize a pointer.unsigned short int \* pPointer = 0;// Declare an integer variable and initialize it with 35698unsigned short int twoInt = 35698;// Declare an integer variable and initialize it with 77unsigned short int oneInt = 77;// Use address-of operator & to assign a memory address of twoInt to a pointerpPointer = &twoInt;// Pointer pPointer now holds a memory address of twoInt// Print out associated memory addresses and its valuescout << "pPointer's memory address:\t\t" << &pPointer << endl;  
 cout << "Integer's oneInt memory address:\t" << &oneInt << "\tInteger value:\t" << oneInt << endl;  
 cout << "Integer's twoInt memory address:\t" << &twoInt << "\tInteger value:\t" << twoInt << endl;  
 cout << "pPointer is pointing to memory address:\t" << pPointer << "\tInteger value:\t" << \*pPointer << endl;  
  
**return** 0;  
}

***OUTPUT:***

pPointer's memory address: 0xbff43314  
Integer's oneInt memory address: 0xbff43318 Integer value: 77  
Integer's twoInt memory address: 0xbff4331a Integer value: 35698  
pPointer is pointing to memory address: 0xbff4331a Integer value: 35698

# Passing Arrays to Functions

Array elements can be passed to a function by calling the function by value, or by reference. In the call by value we pass values of array elements to the function, whereas in the call by reference we pass addresses of array elements to the function. These two calls are illustrated below:

## Lab Practice 10

**Note: This Exercise will help you to understand about array passing to a function by reference.**

#include<stdio.h>  
   #include<conio.h>  
   void read(int \*,int);  
   void dis(int \*,int);  
    
   void main()  
   {  
    int a[5],b[5],c[5],i;  
  
    printf("Enter the elements of first list \n");  
    read(a,5);  
    printf("The elements of first list are \n");  
    dis(a,5);  
   }  
  
   void read(int c[],int i)  
   {  
    int j;  
    for(j=0;j<i;j++)  
      scanf("%d",&c[j]);  
    fflush(stdin);  
   }  
  
   void dis(int d[],int i)  
   {  
    int j;  
    for(j=0;j<i;j++)  
    printf("%d ",d[j]);  
    printf("\n");  
   }

### Output

Enter the elements of first list

1

2

3

4

5

The elements of first list are

1 2 3 4 5

**Lab Task 2**

Write a program that generates an arithmetic series of first 10 numbers and stores it in an array. Write a function that takes the starting value, the incremental value (that is added to every number to generate the series) and a pointer storing the starting address of the array. With the help of the pointer store the series in the array and print it in the main function.

For example:

Starting value: 2

Incremental value: 3

Arithmetic series: {2,5,8,11,14,17,20,23,26,29}

**Lab Task 3**

Write a program that will determine if string1 is a part of string2. Write a function named “substring” that returns 1 if this is true and false if it not. Use a pointer that will send the starting address of both the arrays. The result should be printed in the main function.

For example

String1: BE

String2: BECE

Output: Yes, String1 is a part of String2

**Deliverables**

Source Code of Task 2

Source Code of Task 3