

UNIVERSITY OF BURGUNDY

SOFTWARE ENGINEERING

TUTORIAL 3

---

## Lab Report-3

---

*Author:*

Vamshi KODIPAKA

*Supervisor:*

Dr. Yohan FOUGEROLLE

October 28, 2018



# 1 Pointer as arrays

## 1.1 Declare and implement func DisplayPointerInfo()

```
#include <iostream>
#include "lab_3.h"

using namespace std;
//1.a> display pointers as variables—————

void DisplayPointerInfo (int *pntr, int length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        cout << *pntr++ << "_";
    }
    cout << endl;
}
```

## 1.2 Dynamically allocate n integers:: odd and even

```
// 1.b> pointers dynamic array to find odd and even—————

int n;
int *c, *d;

cout << "Enter no. of values for the array here:_" << endl;
cin >> n;
c = new int [n]           // dynamic arrays
d = new int [n];
    //reading arrays by pointers
    for (int i = 0; i < n ; i++)
    {
        *(c + i) = i * 2;
        *(d + i) = i * 2 + 1;
    }
```

# 2 Swapping arrays by Pointers

## 2.1 Swapping variables by Pointers

```
//2.a> swapping values of values in a variable by pointers ———
void swap (int *xx, int *yy)
```

```
{  
    int *t = xx;  
    xx = yy;  
    yy = t;  
}
```

## 2.2 Swapping arrays by pointers

```
//2.b> pointers as arrays :: SwapArray—————  
  
void SwapArray (int *xx, int *yy, int length)  
{  
    int temp, i;  
    for (i = 0; i < length; i++)  
    {  
        temp = *xx;  
        *xx = *yy;  
        *yy = temp;  
        *xx++;  
        *yy++;  
    }  
}
```

## 3 Allocation and Deallocation of 1D, 2D arrays by pointers

### 3.1 CreateArray() returns pointer of array

```
//3.1 Creating 1D array of n integers by pointers —————  
  
int* CreateArray (int n)  
{  
    int *pntr = new int[n];  
    return pntr;  
}
```

### 3.2 DeleteArray() of n integers

```
//3.2 Deleting 1D array of n integers by pointers—————  
  
void DeleteArray(int* p)  
{  
    delete p;  
}
```

### 3.3 CreateMatrix() of n X m floats

```
//3.3 Creating a matrix n X m integers by pointers————

int** CreateMatrix(int n, int m)
{
    int **a;
    a = new int* [n];
    for(int i = 0; i < n; ++i)
        a[i] = new int [m];
    return a;
}
```

### 3.4 DeleteMatrix() of n X m floats

```
//3.4 Deleting a matrix by pointers —————

void DeleteMatrix(int** p)
{
    delete p;
}
```

### 3.5 DisplayMatrix() of n X m floats with address

```
//3.5 Displaying a matrix n X m integers by pointers————

void DisplayMatrix(int** p, int q, int r)
{
    for (int i = 0; i < q; ++i)
    {
        for(int j = 0; j < r; ++j)
        {
            cout << *((p + i) + j) << " ";
        }
        cout << endl;
    }
}
```

## 4 A little bit of Geometry

### 4.1 Finding Dotproduct and inner product in 3D

```
//Finding Dot product and innerproduct of 3D
float dot_product ( float *a, float *b, int n)
```

```

{
    float product(0);
    for (int i = 0; i < n; i++)
        product += a[i]*b[i];
    return product;
}

float *inner_product(float *a, float *b, int n)
{
    if ( n!= 3) return NULL;
    float *product = new float [3];
    product[0] = a[1] * b[2] - a[2]*b[1];
    product[1] = a[2] * b[0] - a[0]*b[2];
    product[2] = a[0] * b[1] - a[1]*b[0];
}

```

## 4.2 main function for all the above programs

```

//main function for all the above programs

#include <iostream>
#include "lab_3.h"
using namespace std;

int main()
{
    int a = 66, b = 34;
    int *pa = &a, *pb = &b;           // inputs to array

    //pointers as variables -----
    //values before swapping
    cout << "Value_a_before_swap:" << a << endl;
    cout << "Value_b_before_swap:" << b << endl;
    swap(*pa, *pb);
    //values after swapping
    cout << "Value_a_after_swap:" << a << endl;
    cout << "Value_b_after_swap:" << b << endl;

    //1.a> pointers as arrays -----

    int arrA[] = {11, 22, 33, 44, 55};
    int arrB[] = {13, 15, 17, 19, 21};
}

```

```
// 1.b> pointers dynamic array to find odd and even numbers——

int n;
int *c, *d;

cout << "Enter no. of values for the array here:_" << endl;
cin >> n;
c = new int [n];    // dynamic arrays
d = new int [n];    //reading arrays by pointers
for (int i = 0; i < n ; i++)
{
    *(c + i) = i * 2;
    *(d + i) = i * 2 + 1;
}

cout << "The even array are:_" << endl; //displaying pointers
DisplayPointerInfo(c, n);
cout << "The odd array are:_" << endl;
DisplayPointerInfo(d, n);

cout<<"_"<< endl;

//2. swapping values of arrays by pointers —————

//values in arrays before swapping

cout << "Array A before swap:_" << endl;
DisplayPointerInfo(arrA, 5);

cout << "Array B before swap:_" << endl;
DisplayPointerInfo(arrB, 5);

SwapArray(arrA, arrB, 5);
cout<<"_"<< endl;

//values in arrays after swapping

cout << "Array A after swap:_" << endl;
DisplayPointerInfo(arrA, 5);

cout << "Array B after swap:_" << endl;
DisplayPointerInfo(arrB, 5);
cout<<"_"<< endl;
```

```

//3.1 Creating 1D array of n integers by pointers —————

int x;
cout << "length_of_the_array_is:" << endl;
cin >> x;
int *pntr1 = CreateArray(x); //creating an array
for (int i = 0 ; i < x ; i++)
{
    *(pntr1+i)=i;
}
DisplayPointerInfo(pntr1 , x);

//3.2 Deleting 1D array of n integers by pointers ———

DeleteArray(pntr1);          //deleting an array

//3.3 Creating a matrix n X m integers by pointers————

int **p, h, w;
cout << "row_of_the_matrix_is:" << endl;
cin >> h;
cout << "column_of_the_matrix_is:" << endl;
cin >> w;

p = CreateMatrix(h, w);
DisplayMatrix(p, h, w);

//3.4 Deleting a matrix by pointers —————

DeleteMatrix(p);

//Finding Dot product and innerproduct of 3D
float *P = new float [3];
float *Q = new float [3];
P[0] = 1;
P[1] = 1;
P[2] = 0;
Q[0] = 1;
Q[1] = 1;
Q[2] = 1;

cout<< "Dot_Product_is:_____" << endl;
cout<< dot_product(P,Q,3) << endl;

```

```

        cout<< "Inner_Product_is:_" << endl;
        cout<< "_" << endl;

        float *R = inner_product(P,Q,3);
        for (int i=0; i<n; i++, R++)
        {
            cout<< "value_at_index" << i << "_is_"<< *R << endl;
        }
    }
}

```

## 5 Multiplication of matrices in general

```

#include <iostream>
using namespace std;

void enterData(int firMat[][10], int secMat[][10], int rowFir,
              int colFir, int rowSec, int colSec);
void multiplyMatrices(int firMat[][10], int secMat[][10], int
multRes[][10], int rowFir, int columnFir, int rowSec,
int colSec);
void display(int mult[][10], int rowFir, int colSec);

//main() for matrix
int main()
{
    int firMat[10][10], secMat[10][10], mult[10][10], rowFirst,
        colFir, rowSec, colSec, i, j, k;

    cout << "Enter_rows_and_column_for_first_matrix: ";
    cin >> rowFirst >> colFir;

    cout << "Enter_rows_and_column_for_second_matrix: ";
    cin >> rowSec >> colSec;

    // If colum of first matrix in not equal to row of second
    matrix, asking user to enter the size of matrix again.
    while (colFir != rowSec)
    {
        cout << "Error!_column_of_first_matrix_not_equal_to_row_of
        .....second." << endl;
        cout << "Enter_rows_and_column_for_first_matrix: ";
        cin >> rowFirst >> colFir;
        cout << "Enter_rows_and_column_for_second_matrix: ";
    }
}

```



```
        cin >> rowSec >> colSec;
    }

    // Function to take matrices data
    enterData(firMat, secMat, rowFirst, colFir, rowSec, colSec);

    // Function to multiply two matrices.
    multiplyMatrices(firMat, secMat, mult, rowFirst, colFir,
        rowSec, colSec);

    // Function to display resultant matrix after
    // multiplication.
    display(mult, rowFirst, colSec);

    return 0;
}

void enterData(int firMat[][10], int secMat[][10], int rowFir,
    int colFir, int rowSec, int colSec)
{
    int i, j;
    cout << endl << "Enter elements of matrix 1:" << endl;
    for(i = 0; i < rowFir; ++i)
    {
        for(j = 0; j < colFir; ++j)
        {
            cout << "Enter elements a" << i + 1 << j + 1 << ": ";
            cin >> firMat[i][j];
        }
    }

    cout << endl << "Enter elements of matrix 2:" << endl;
    for(i = 0; i < rowSec; ++i)
    {
        for(j = 0; j < colSec; ++j)
        {
            cout << "Enter elements b" << i + 1 << j + 1 << ": ";
            cin >> secMat[i][j];
        }
    }
}

void multiplyMatrices(int firMat[][10], int secMat[][10],
    int mult[][10], int rowFir, int colFir, int rowSec, int colSec)
{
```

```
int i, j, k;

// Initializing elements of matrix mult to 0.
for(i = 0; i < rowFir; ++i)
{
    for(j = 0; j < colSec; ++j)
    {
        mult[i][j] = 0;
    }
}

// Multiplying matrix firstMatrix and secondMatrix and
storing in array mult.

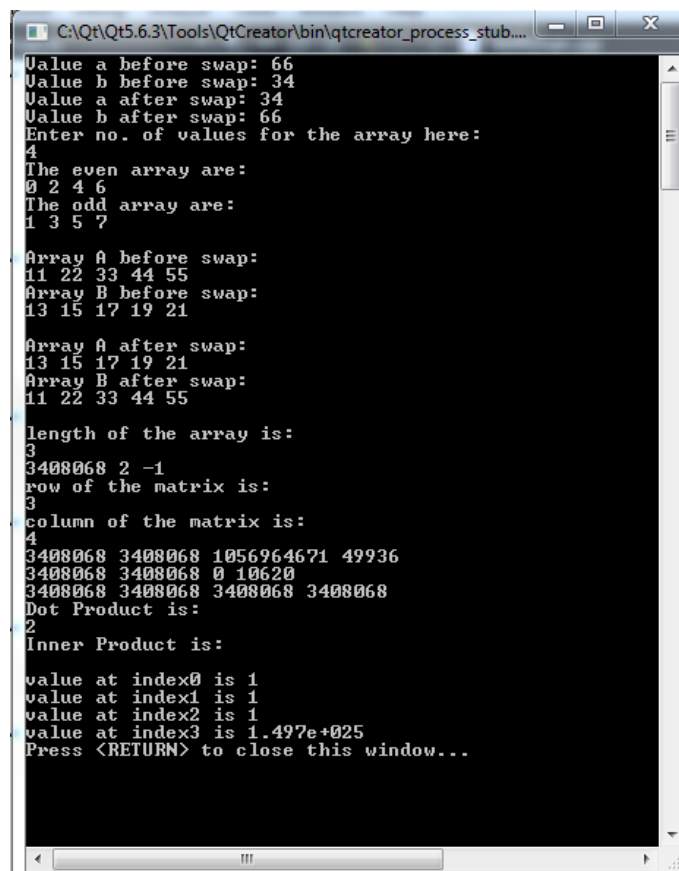
for(i = 0; i < rowFir; ++i)
{
    for(j = 0; j < colSec; ++j)
    {
        for(k=0; k<colFir; ++k)
        {
            mult[i][j] += firMat[i][k] * secMat[k][j];
        }
    }
}

}

void display(int mult[][10], int rowFirst, int columnSecond)
{
    int i, j;

    cout << "Output_Matrix:" << endl;
    for(i = 0; i < rowFirst; ++i)
    {
        for(j = 0; j < columnSecond; ++j)
        {
            cout << mult[i][j] << " ";
            if(j == columnSecond - 1)
                cout << endl << endl;
        }
    }
}
```

## 6 Outputs of all the programs

A screenshot of a Qt Creator console window. The window title is "C:\Qt\Qt5.6.3\Tools\QtCreator\bin\qtcreator\_process\_stub...". The console output is as follows:

```
Value a before swap: 66
Value b before swap: 34
Value a after swap: 34
Value b after swap: 66
Enter no. of values for the array here:
4
The even array are:
0 2 4 6
The odd array are:
1 3 5 7

Array A before swap:
11 22 33 44 55
Array B before swap:
13 15 17 19 21

Array A after swap:
13 15 17 19 21
Array B after swap:
11 22 33 44 55

length of the array is:
3
3408068 2 -1
row of the matrix is:
3
column of the matrix is:
4
3408068 3408068 1056964671 49936
3408068 3408068 0 10620
3408068 3408068 3408068 3408068
Dot Product is:
2
Inner Product is:

value at index0 is 1
value at index1 is 1
value at index2 is 1
value at index3 is 1.497e+025
Press <RETURN> to close this window...
```

Figure 1: All outputs



```
C:\Qt\Qt5.6.3\Tools\QtCreator\bin\qtcreator_process_stub.exe
Enter rows and column for first matrix: 3
3
Enter rows and column for second matrix: 3
3
Enter elements of matrix 1:
Enter elements a11: 1
Enter elements a12: 2
Enter elements a13: 3
Enter elements a21: 4
Enter elements a22: 5
Enter elements a23: 6
Enter elements a31: 7
Enter elements a32: 8
Enter elements a33: 9
Enter elements of matrix 2:
Enter elements b11: 0
Enter elements b12: 1
Enter elements b13: 2
Enter elements b21: 3
Enter elements b22: 4
Enter elements b23: 5
Enter elements b31: 6
Enter elements b32: 7
Enter elements b33: 8
Output Matrix:
24 30 36
51 66 81
78 102 126
Press <RETURN> to close this window...
```

Figure 2: output of the MatMul Program