Learning Objectives: By the end of the lab work, students should have following skills;

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Learning Objective** | **LAB No.** |
| **1** | Just Introduction about the tool(visual studio)  And the basics of the c++ | 1 |
| **2** | Static Array Examples in C++ | 2 |
| **3** | Dynamic Array Example in C++ | 3 |
| **4** | Admission Form Case Study | 4 |
| **5** | Linked List | 5 |
| **6** | Queue via Static Array | 6 |
| **7** | Queue via Dynamic Array | 7 |
| **8** | Queue via Linked List | 8 |
| **9** | Circular Queue via Array: | 9 |
| **10** | Input Restricted Queue via Array | 10 |
| **11** | Input Restricted Queue via Linked List | 11 |
| **12** | Output Restricted Queue via Array | 12 |
| **13** | Output Restricted Queue via Linked List | 13 |
| **14** | Stack | 14 |
| **15** | Graph | 15 |
| **16** | Binary Search Tree | 16 |

**Table of Contents**

[LAB # 1](#_Toc496515713) 4

Introduction about Tool And Syntax of C++

[LAB # 2](#_Toc496515718) 5

Static Array Examples in C++

[LAB # 3](#_Toc496515723) 9

Dynamic Array Example in C++

[LAB # 4](#_Toc496515727) 14

Admission Form Case Study

[LAB # 5](#_Toc496515731) 16

Linked List

[LAB # 6](#_Toc496515736) 24

Queue via Static Array

[LAB # 7](#_Toc496515740) 27

Queue via Dynamic Array

[LAB # 8](#_Toc496515744) 30

Queue via Linked List

[LAB # 9](#_Toc496515749) 33

Circular Queue via Array:

[LAB # 10](#_Toc496515753) 36

Input Restricted Queue via Array

[LAB # 11](#_Toc496515759) 41

Input Restricted Queue via Linked List

[LAB # 12](#_Toc496515765) 46

Output Restricted Queue via Array

[LAB # 13](#_Toc496515778) 50

Output Restricted Queue via Linked List

[LAB # 14](#_Toc496515782) 53

Stack

Task I: Counting Brackets

Task II: InFix To PostFix Converter

[LAB # 15](#_Toc496515787) 62

Graph

Task I: Graph via Linked List

[LAB # 16](#_Toc496515792) 67

Binary Search Tree

Task I: Tree Traversals

Task II: Displaying Maximum and Minimum value in BST

# LAB # 1

# Introduction About Tool(Visual Studio)

Visual studio is tool by Micro soft that provide different platform for programming purpose’s++,C++,asp.net are major platform in it and many others.

# Introduction About: C++(Syntax)

C++ syntax explained for lab work and develop a compiler.

Data Types: int, float, char, string

Input: cin>>variable name.

Out: cout<<”pakistan”; or cout<<variable;

Terminator:

Loops:for(int i=0;i<=length;i++){

Some lines

}

While(condition){

somelines

}

do{

somelines

}while(condition);

By considering above syntax our aim is to develop a compiler that is capable of detecting errors and understanding above syntax.

# LAB # 2

# Static Array Example in C++

**Based on:** .NET (2018)

**C++ program that uses Static Array**

**Lab Task:**

Implement following

* insert value in array
* delete value from array
* view element of array
* insert value at start of array
* insert a value at particular index

**Solution**

#include "iostream"

using namespace std;

void menu();

void main()

{

int index = -1, data[10], userchoice;

do

{

menu();

cin >> userchoice;

if (userchoice == 1)//insertvalueinarray

{

if (index == 9)

cout << "Array is Full " << endl;

else

{

index++;

cout << "Enter Value in array " << endl;

cin >> data[index];

}

}

else if (userchoice == 2)//delete value from array

{

int valuetodelete;

cout << "Enter Value you want to delete " << endl;

cin >> valuetodelete;

bool valuefound = false;

int i = 0;

while (i <= index)

{

if (data[i] == valuetodelete)

{

valuefound = true;

break;

}

i++;

}

if (valuefound)

{

while (i<index)

{

data[i] = data[i + 1];

i++;

}

index--;

}

else

{

cout << "Value not exsist " << endl;

}

}

else if (userchoice == 3)//view element of array

{

for (int i = 0; i <= index; i++)

{

cout << " value = " << data[i] << endl;

}

}

else if (userchoice == 4)//insert value at start of array

{

if (index == 9)

cout << "Arra is Full " << endl;

else

{

index++;

for (int i = index; i>0; i--)

{

data[i] = data[i - 1];

}

cout << "Enter value to be inserted " << endl;

cin >> data[0];

}

}

else if (userchoice == 5)

{

}

} while (userchoice != 6);

}

void menu()

{

cout << "Press 1 to insert value in array" << endl;

cout << "Press 2 to delete value from array" << endl;

cout << "Press 3 to view element of array" << endl;

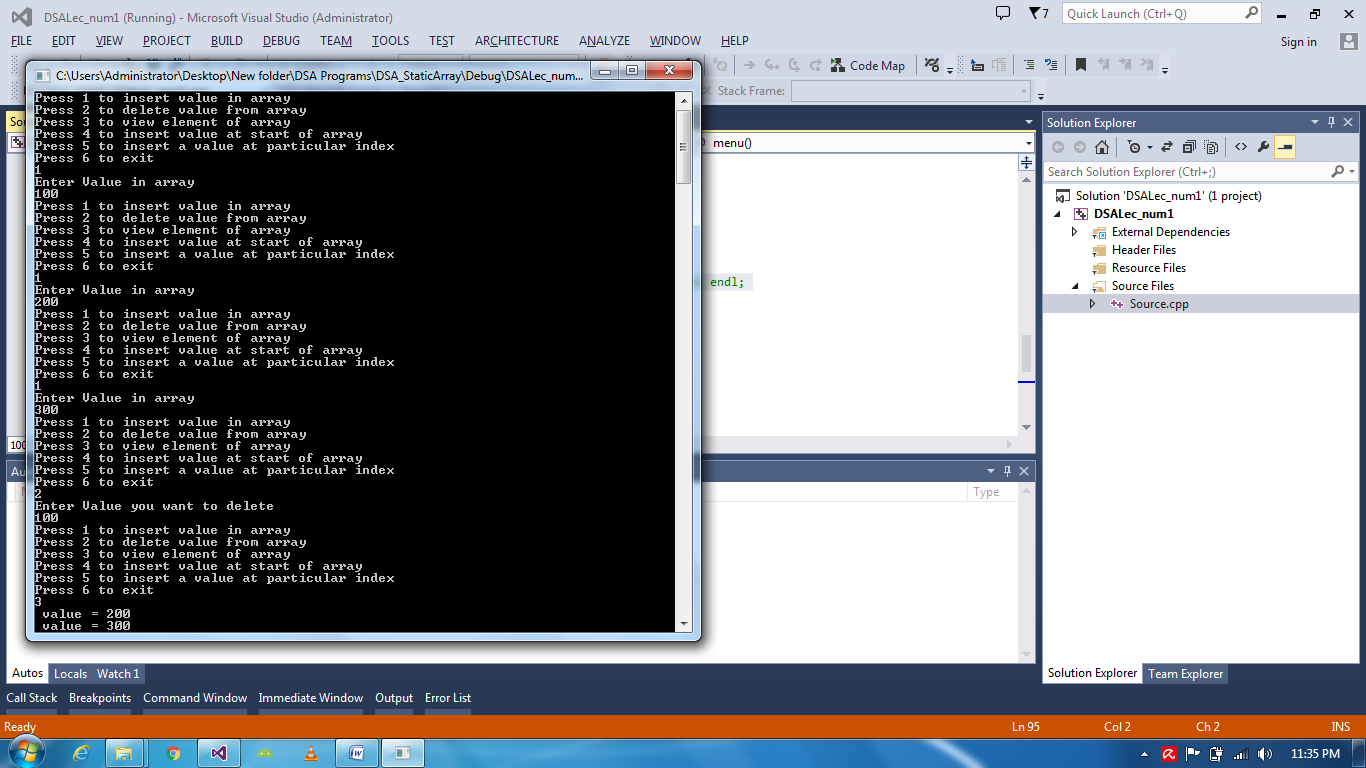
cout << "Press 4 to insert value at start of array " << endl;

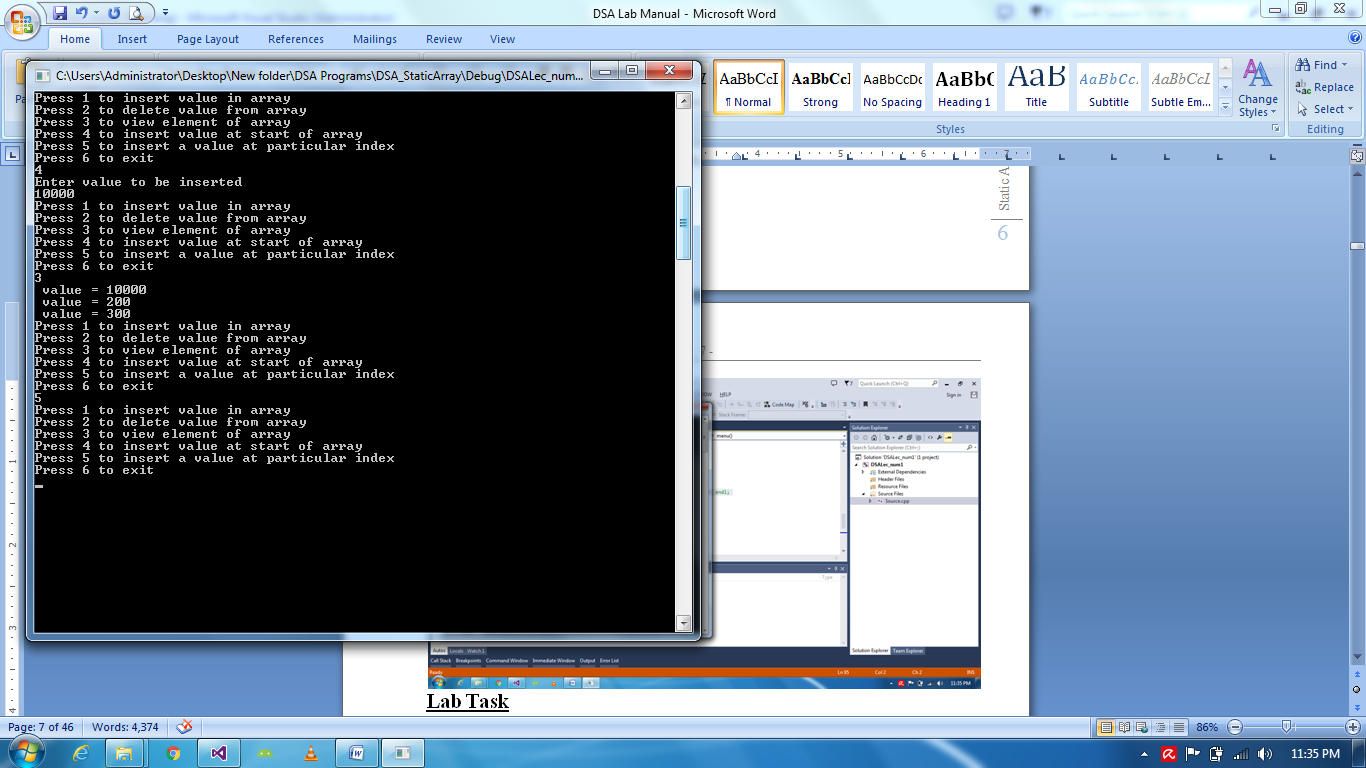
cout << "Press 5 to insert a value at particular index " << endl;

cout << "Press 6 to exit " << endl;

}

**Output**





# LAB # 3

# Dynamic Array Example in C++

**Code:**

**Lab Task**

* Create Dynamic Array of user defined size
* Grow and shrink size of array

**Solution**

#include "iostream"

# include <string>

using namespace std;

void menu();

void main()

{

int index = -1, size, userchoice;

cout << "Enter size of Array: "; cin >> size; cout << endl;

int \*data = new int[size];

do

{

menu();

cin >> userchoice;

if (userchoice == 1)

{

if (index == size - 1)

{

int \*b = new int[size];

for (int i = 0; i < size; i++)

{

b[i] = data[i];

}

size = size + 4;

data = new int[size];

for (int i = 0; i <= index; i++)

{

data[i] = b[i];

}

index++;

cin >> data[index];

}

else

{

index++;

cout << "Enter Value in array " << endl;

cin >> data[index];

}

}

else if (userchoice == 2)

{

int valuetodelete;

cout << "Enter Value you want to delete " << endl;

cin >> valuetodelete;

bool valuefound = false;

int i = 0;

while (i <= index)

{

if (data[i] == valuetodelete)

{

valuefound = true;

break;

}

i++;

}

if (valuefound)

{

while (i<index)

{

data[i] = data[i + 1];

i++;

}

index--;

}

else

{

cout << "Value not exsist " << endl;

}

}

else if (userchoice == 3)

{

for (int i = 0; i <= index; i++)

{

cout << "Value #: " << i << " is: " << data[i] << endl;

}

}

else if (userchoice == 4)

{

if (index == size - 1)

{

int \*b = new int[size];

for (int i = 0; i < size; i++)

{

b[i] = data[i];

}

size = size + 4;

data = new int[size];

for (int i = 0; i <= index; i++)

{

data[i] = b[i];

}

index++;

cin >> data[index];

}

else

{

index++;

for (int i = index; i>0; i--)

{

data[i] = data[i - 1];

}

cout << "Enter value to be inserted " << endl;

cin >> data[0];

}

}

if (userchoice == 5)

{

int v, userindex;

if (index == size - 1)

{

int \*b = new int[size];

for (int i = 0; i < size; i++)

{

b[i] = data[i];

}

size = size + 4;

data = new int[size];

for (int i = 0; i <= index; i++)

{

data[i] = b[i];

}

index++;

cin >> data[index];

}

else

{

cout << "enter index for the value: "; cin >> userindex; cout << endl;

cout << "Enter value to insert: "; cin >> v; cout << endl;

index++;

for (int i = index; i > userindex; i--)

{

data[i] = data[i - 1];

}

data[userindex]=v;

cout << "Successfully Inserted" << endl;

}

}

} while (userchoice != 6);

}

void menu()

{

cout << "Press 1 to insert value in array" << endl;

cout << "Press 2 to delete value from array" << endl;

cout << "Press 3 to view element of array" << endl;

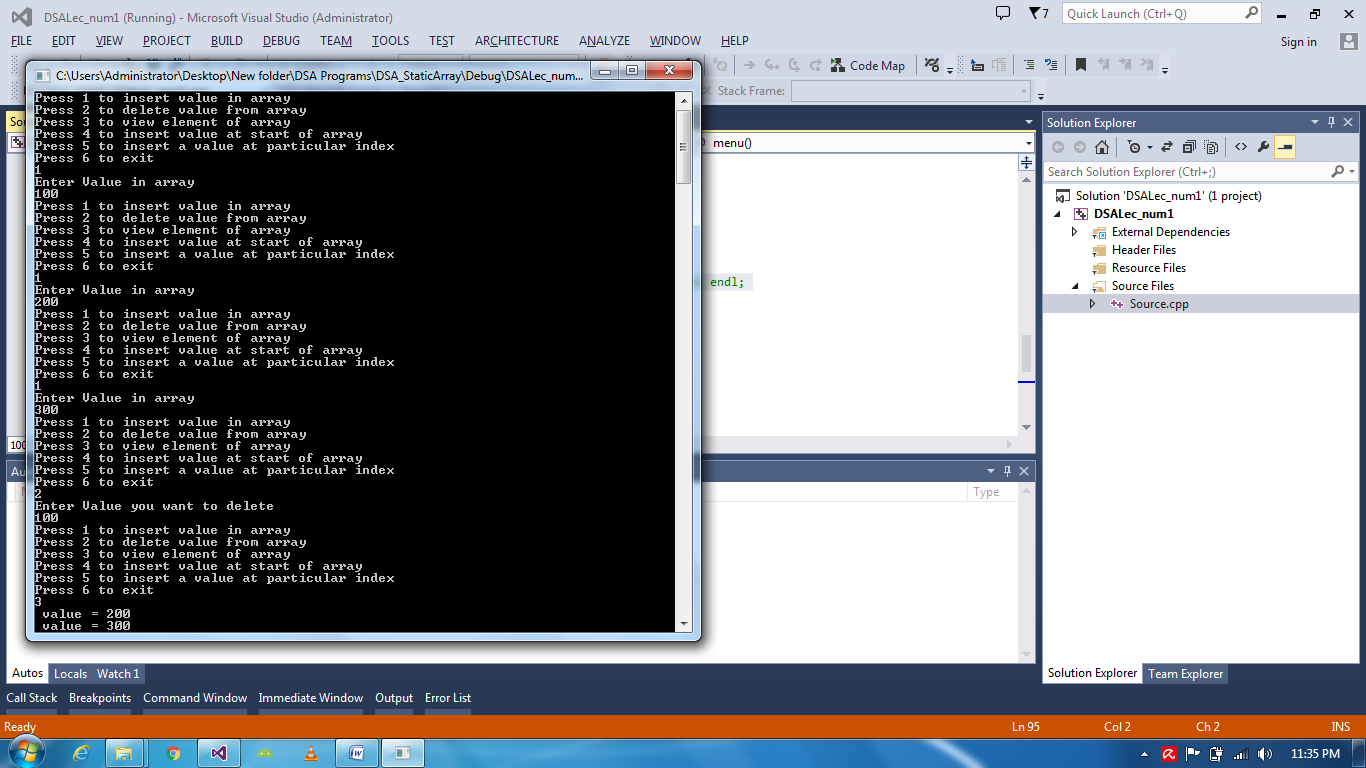
cout << "Press 4 to insert value at start of array " << endl;

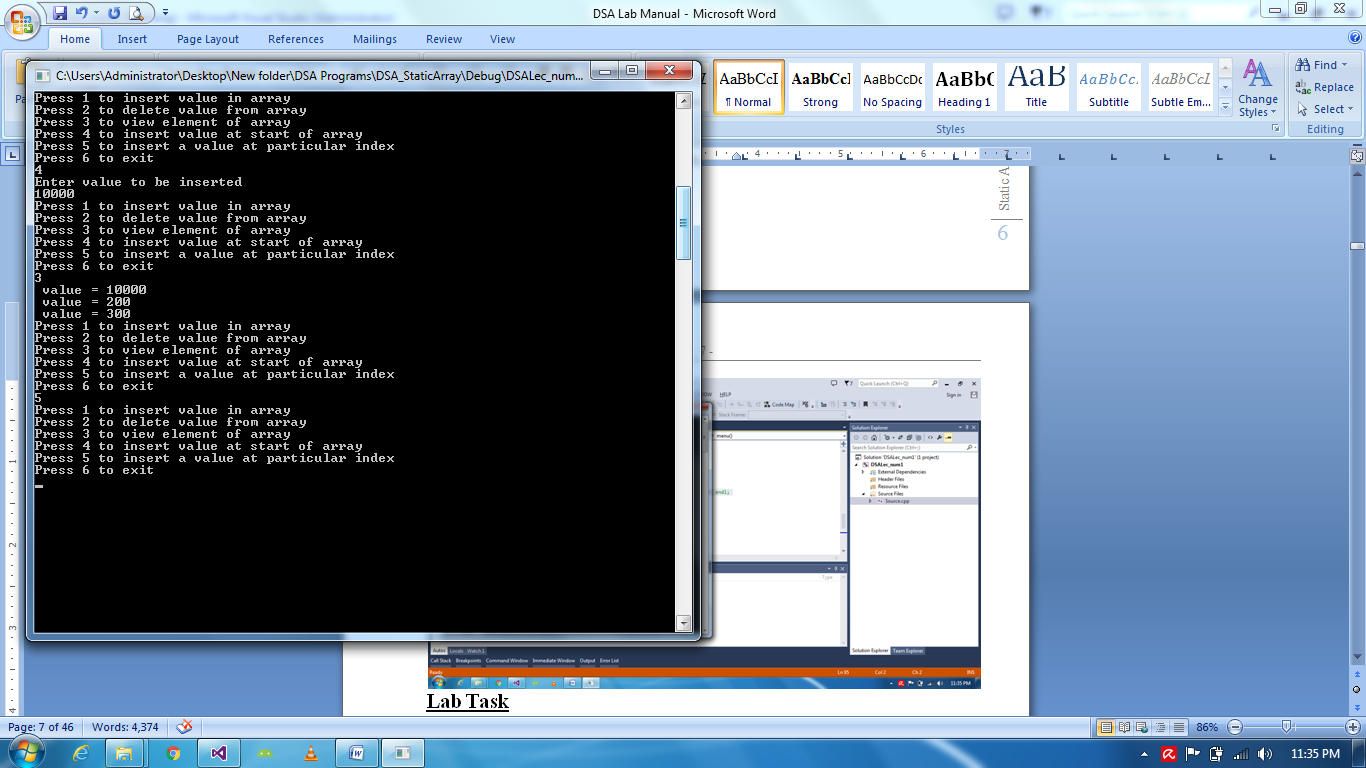
cout << "Press 5 to insert a value at particular index " << endl;

cout << "Press 6 to exit " << endl;

}

**Output**





# LAB # 4

# Admission Form Case Study

#include <iostream>

#include <string>

using namespace std;

struct AdmissionForm

{

int formnum;

string sname, fname;

float per;

};

void main()

{

int index, noofforms;

cout << "Enter initial value of form: ";

cin >> noofforms;

int index = 0;

AdmissionForm \*forms = new AdmissionForm[noofforms];

int opt;

do

{

// menu();

cin >> opt;

cout << endl;

if (opt == 1)

{

if (index == noofforms)

{

AdmissionForm \*copydata = new AdmissionForm[noofforms];

for (int i = 0; i < index; i++)

{

copydata[i] = forms[i];

}

noofforms += 3;

forms = new AdmissionForm[noofforms];

for (int i = 0; i < index; i++)

{

forms[i] = copydata[i];

}

cin >> forms[index].fname;

cin >> forms[index].sname;

cin >> forms[index].per;

forms[index].formnum = index + 1;

}

else

{

forms[index].formnum = index + 1;

index++;

}

}

}

}

# LAB # 5

# Linked List:

* Implement Single Way linked list.

**Solution**

Code:

# include <iostream>

# include <string>

using namespace std;

struct Student

{

int id;

Student \*nextnodeaddress;

};

class LinkedList

{

Student \*firstnode;

public:

LinkedList()

{

firstnode = NULL;

}

void Input()

{

Student \*newnode = new Student;

cout << "Enter id: " << endl;

cin >> newnode->id;

newnode->nextnodeaddress = NULL;

if (firstnode == NULL)

{

firstnode = newnode;

}

else

{

Student \*temp = firstnode;

while (temp->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = newnode;

}

}

void DisplayData()

{

Student \*temp = firstnode;

if (firstnode == NULL)

{

cout << "No data to show" << endl << endl;

}

else

{

while (temp->nextnodeaddress != NULL)

{

cout << "Id: " << temp->id << endl;

temp = temp->nextnodeaddress;

if (temp->nextnodeaddress == NULL)

{

cout << "Id: " << temp->id << endl;

}

}

}

}

void InsertAtFirstNode()

{

Student \*temp = new Student;

temp->nextnodeaddress = firstnode;

cout << "Enter ID: " << endl;

cin >> temp->id;

firstnode = temp;

}

void InsertAtLastNode()

{

Input();

}

void DeleteFirstNode()

{

if (firstnode == NULL)

{

cout << "Nothing to delete" << endl << endl;

}

else

{

Student \*temp = new Student;

temp = firstnode;

firstnode = firstnode->nextnodeaddress;

temp->nextnodeaddress = NULL;

delete temp;

}

}

void DeleteLastNode()

{

if (firstnode == NULL)

{

cout << "Nothing to delete" << endl << endl;

}

else

{

Student \*temp = firstnode;

while (temp->nextnodeaddress->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = NULL;

}

}

void InsertBeforeSpecificNode(int idtoinsert)

{

}

void InsertAfterSpecificNode(int idtoinsert)

{

if (firstnode == NULL)

{

cout << "No data. " << endl;

}

else if (firstnode->id == idtoinsert)

{

InsertAtFirstNode();

}

else

{

Student \*temp = firstnode;

while (temp != NULL)

{

if (temp->id == idtoinsert)

{

break;

}

temp = temp->nextnodeaddress;

}

if (temp == NULL)

{

InsertAtLastNode();

}

else

{

Student \*newnode = new Student;

cout << "Enter ID: " << endl;

cin >> newnode->id;

newnode->nextnodeaddress = temp->nextnodeaddress;

temp->nextnodeaddress = newnode;

}

}

}

};

void menu();

void main()

{

int userchoice=0;

LinkedList list;

int counter=0;

int userchoicenode = 0;

int idtoinsert = 0;

do {

menu();

cin >> userchoice;

if (userchoice == 1)//input

{

list.Input();

counter++;

}

else if (userchoice == 2)//display

{

list.DisplayData();

}

else if (userchoice == 3)//insert at first node

{

list.InsertAtFirstNode();

counter++;

}

else if (userchoice == 4)//insert at last node

{

list.InsertAtLastNode();

counter++;

}

else if (userchoice == 5)//delete first node

{

list.DeleteFirstNode();

counter--;

}

else if (userchoice == 6)//delete last node

{

list.DeleteLastNode();

counter--;

}

else if (userchoice == 7)//count nodes

{

cout << "Total Number of Nodes are: " << counter << endl;

}

else if (userchoice == 8)//insert BEFORE specific node

{

cout << "Please enter node number to insert before: " << endl;

cin >> userchoicenode;

if (userchoicenode > counter)

{

cout << "Id entered exceeds number of nodes, please try again. " << endl;

cout << "Please enter node number to insert: " << endl;

cin >> userchoicenode;

idtoinsert = counter - userchoicenode;

list.InsertBeforeSpecificNode(idtoinsert);

}

else

{

idtoinsert = counter - userchoicenode;

list.InsertBeforeSpecificNode(idtoinsert);

//coding to do for insertbeforespecificnode

}

}

else if (userchoice == 9)//insert AFTER specific node

{

//coding completed for insertafterspecificnode

}

} while (userchoice!=10);

}

void menu()

{

cout << "Press 1 to input data" << endl;

cout << "Press 2 to display data" << endl;

cout << "Press 3 to insert at first node" << endl;

cout << "Press 4 to insert at last node " << endl;

cout << "Press 5 to delete first node " << endl;

cout << "Press 6 to delete last node " << endl;

cout << "Press 7 to count nodes" << endl;

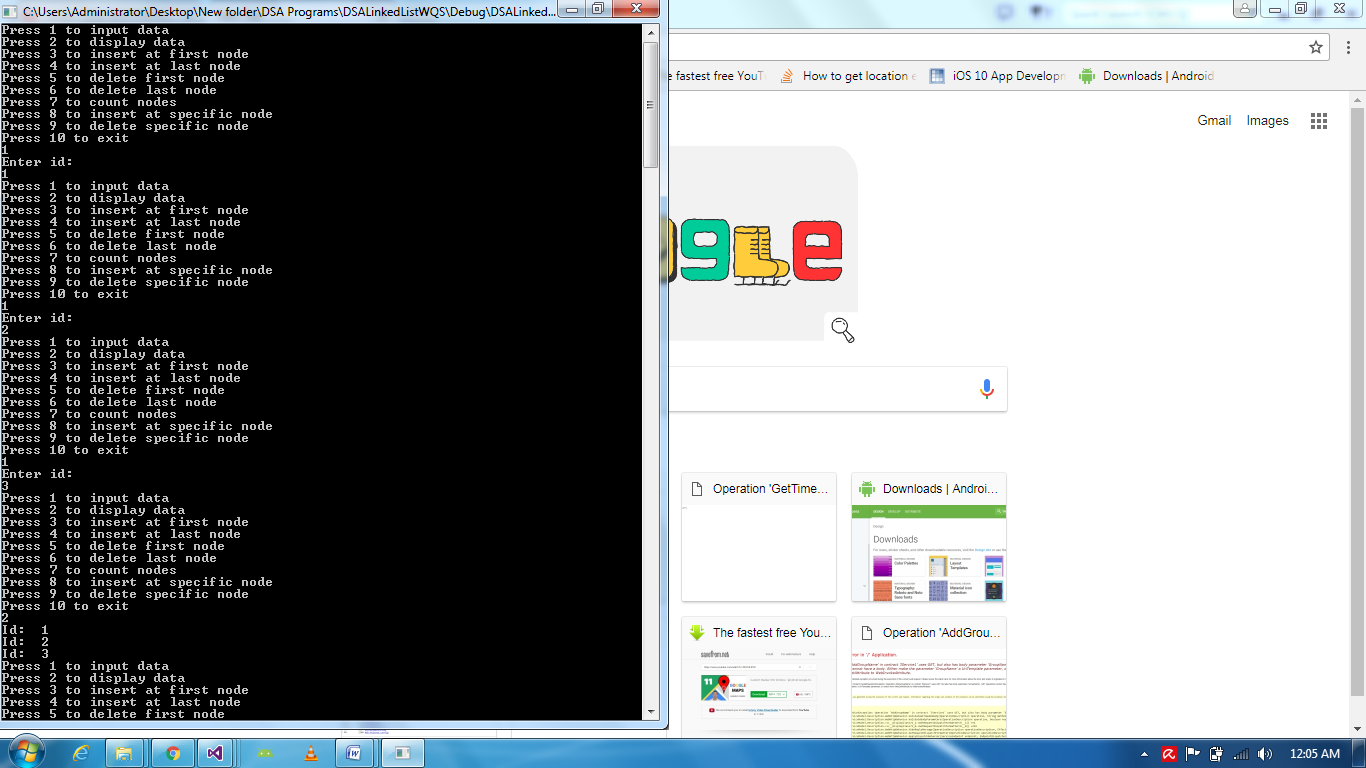
cout << "Press 8 to insert at specific node " << endl;

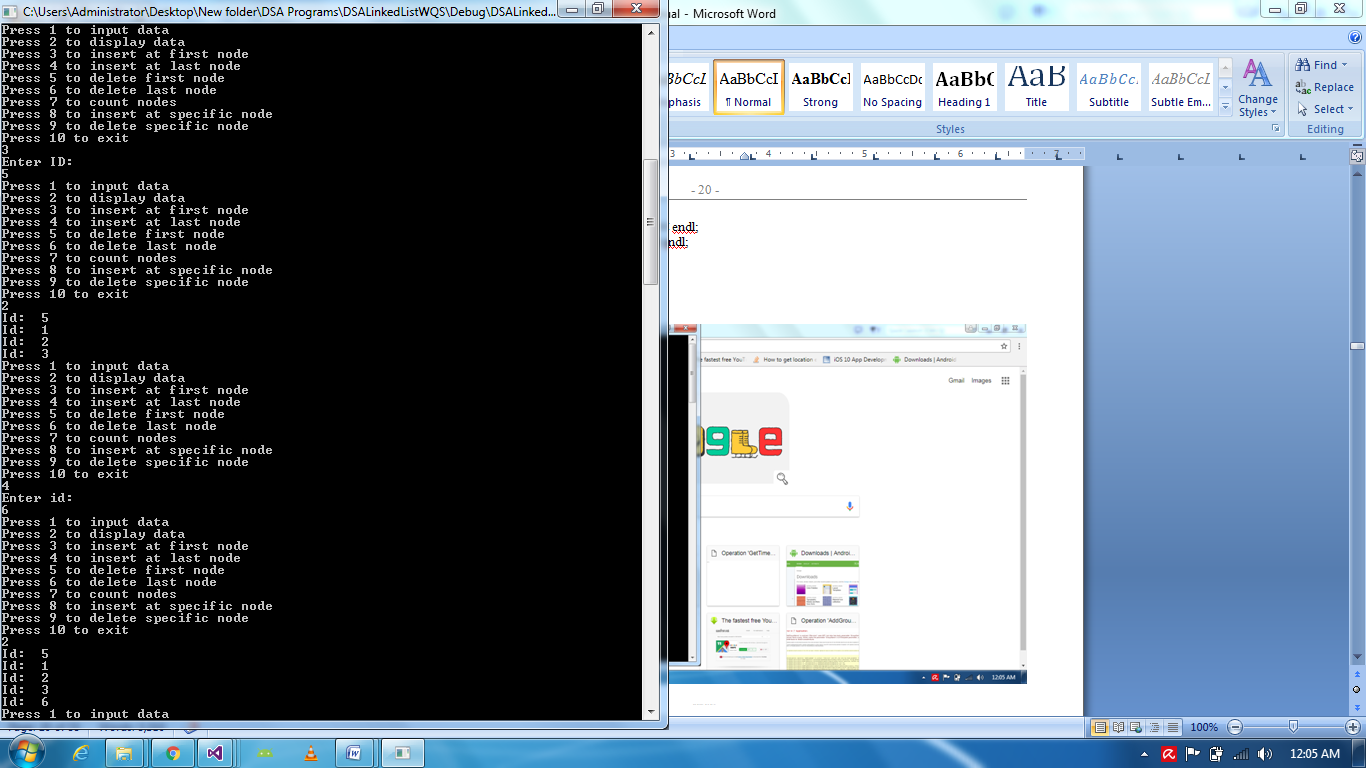
cout << "Press 9 to delete specific node " << endl;

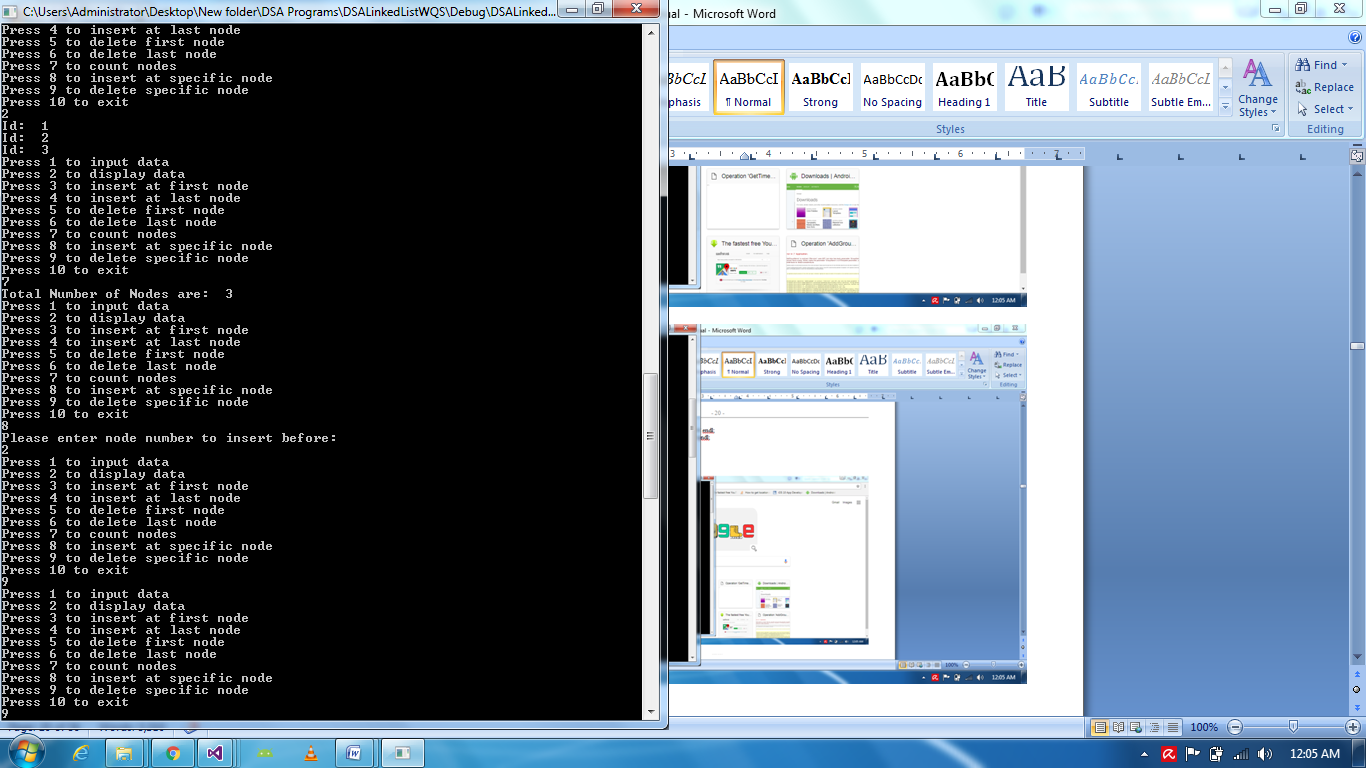
cout << "Press 10 to exit" << endl;

}

**Output**







# LAB # 6

# Queue via Static Array

Code:

#include<iostream>

#include<string>

using namespace std;

struct Data

{

int value;

};

class Queue

{

int Head;

int Tail;

Data data[10];

public:

Queue()

{

Head = -1;

Tail = -1;

}

void EnQueue(int value)

{

if (Tail == 9)

{

cout << "Queue is full" << endl;

}

else

{

if (Head==-1)//insertion first

{

Head=Tail=0;

data[Tail].value = value;

cout << "Successfully inserted " << data[Tail].value << endl;

}

else

{

Tail++;

data[Tail].value = value;

cout << "Successfully inserted " << data[Tail].value << endl;

}

}

}

void DeQueue()

{

if (Head = Tail ==-1)

{

cout << "No data" << endl;

}

else

{

if (Head = Tail == 0)

{

cout << "Deleting value...." << data[Head].value << endl;

Head = Tail = -1;

}

else

{

cout << "Deleting value.." << data[Head].value << endl;

Tail--;

for (int i = 0; i < Tail; i++)

{

data[i].value = data[i + 1].value;

}

}

}

}

};

void main()

{

Queue MQ;

MQ.EnQueue(1);

MQ.EnQueue(2);

MQ.EnQueue(3);

MQ.EnQueue(4);

MQ.EnQueue(5);

MQ.EnQueue(6);

MQ.EnQueue(7);

MQ.EnQueue(8);

MQ.EnQueue(9);

MQ.EnQueue(10);

MQ.EnQueue(11);

MQ.DeQueue();

MQ.DeQueue();

MQ.DeQueue();

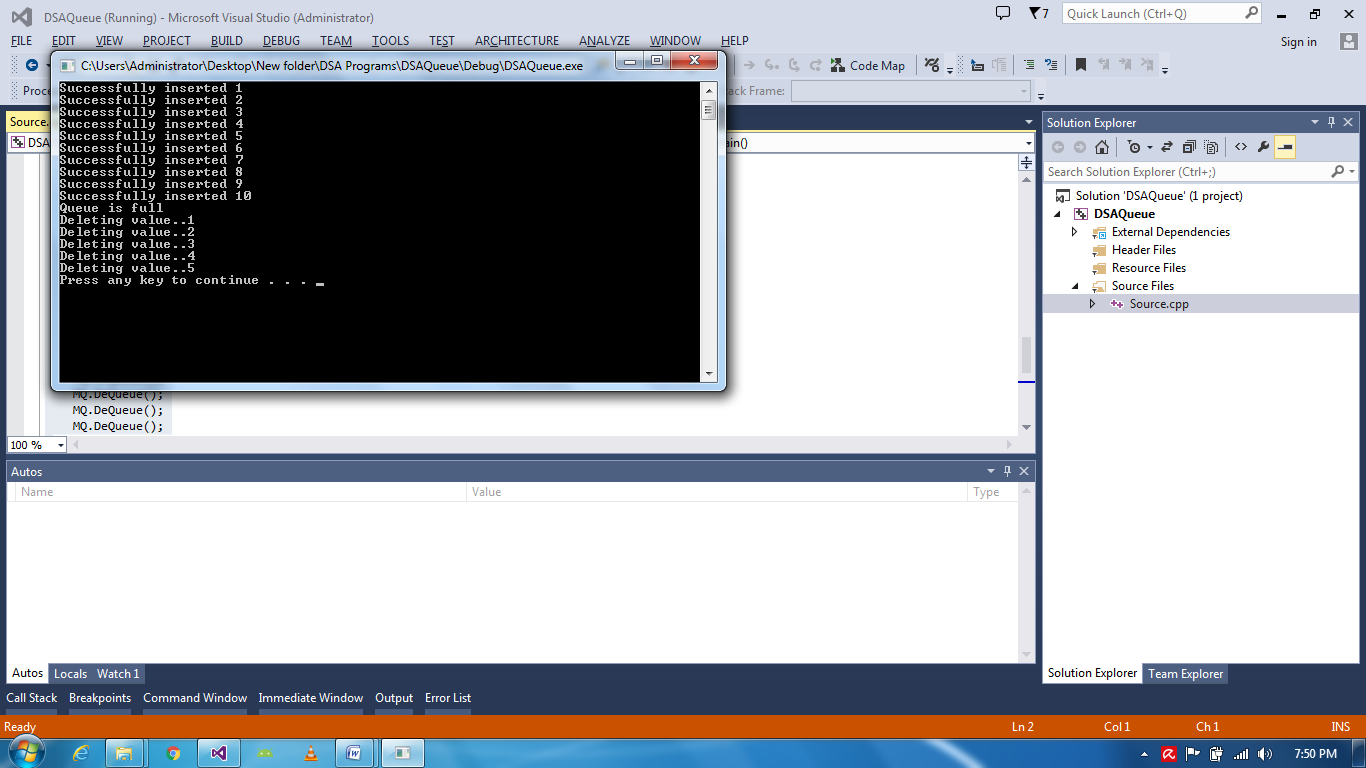
MQ.DeQueue();

MQ.DeQueue();

system("Pause");

}

**Output**



# LAB # 7

# Queue via Dynamic Array:

Code:

#include<iostream>

#include<string>

using namespace std;

struct Data

{

int value;

//via Linkedlist Enqueue InsertAtLast() . .. Dequeue DeleteAtFirst()

};

class Queue

{

public:

int Head;

int Tail;

int size = 0;

Data \*d;

Queue(int size)

{

Head = -1;

Tail = -1;

d = new Data[size];

}

void EnQueue(int value)

{

if (Head == -1)

{

Head = Tail = 0;

d[Tail].value = value;

cout << "Successfully inserted " << d[Tail].value << endl;

Tail++;

}

else if (Tail + 1 == size)

{

Data \*copyarray = new Data[size];

for (int i = 0; i < size; i++)

{

copyarray[i] = d[i];

}

Data \*d = new Data[size + 4];

for (int i = 0; i < size; i++)

{

d[i] = copyarray[i];

}

size = size + 4;

d[Tail].value = value;

cout << "Successfully inserted " << d[Tail].value << endl;

Tail++;

}

else

{

d[Tail].value = value;

cout << "Successfully inserted " << d[Tail].value << endl;

Tail++;

}

}

void DeQueue()

{

if (Head = Tail == -1)

{

cout << "No data" << endl;

}

else

{

if (Head = Tail == 0)

{

cout << "Deleting value...." << d[Head].value << endl;

Head = Tail = -1;

}

else

{

cout << "Deleting value.." << d[Head].value << endl;

Tail--;

for (int i = 0; i < Tail; i++)

{

d[i].value = d[i + 1].value;

}

}

}

}

};

void main()

{

cout << "Please enter size for queue" << endl;

int size;

cin >> size; cout << endl;

Queue MQ(size);

MQ.EnQueue(1);

MQ.EnQueue(2);

MQ.EnQueue(3);

MQ.EnQueue(4);

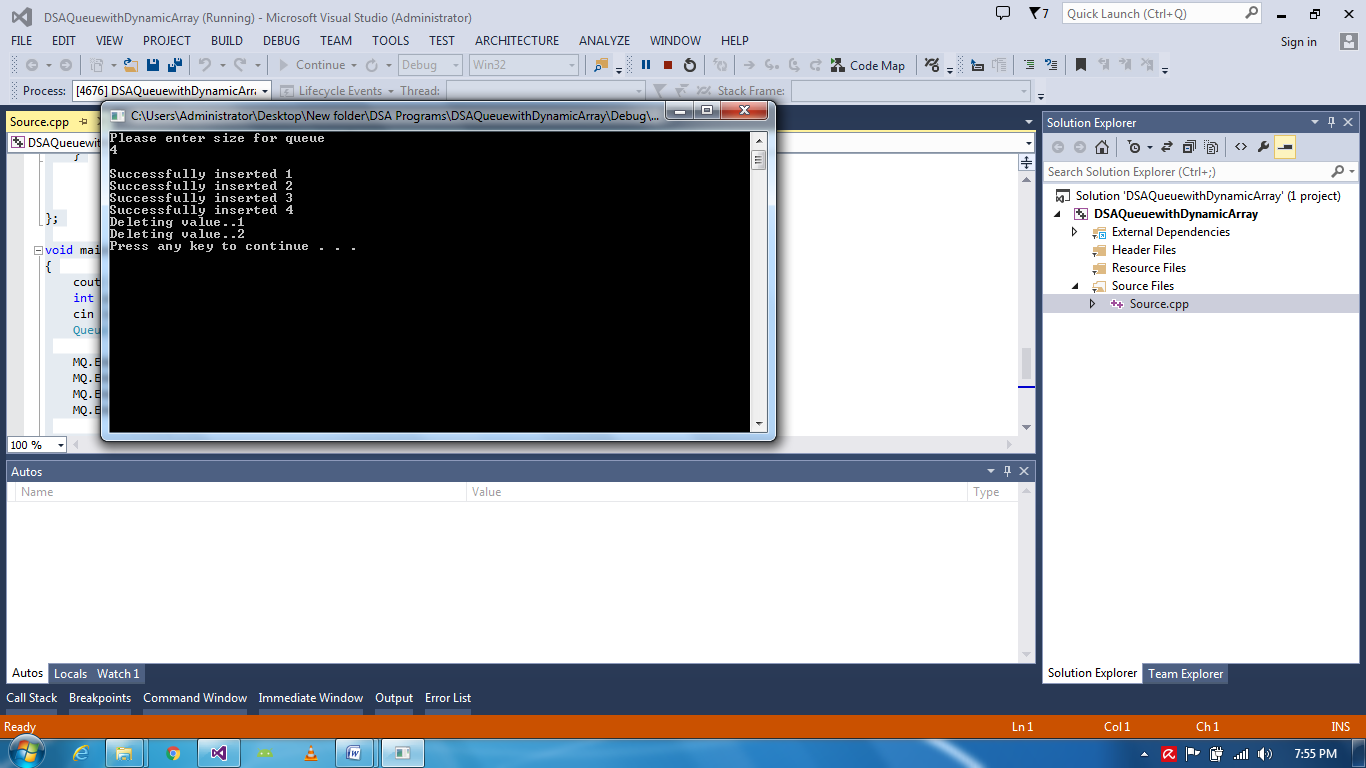
MQ.DeQueue();

MQ.DeQueue();

system("Pause");

}

**Output**



# LAB # 8

# Queue via LinkedList

# Code:

# include <iostream>

# include <string>

using namespace std;

struct Data

{

int id;

Data \*nextnodeaddress;

};

class Queue

{

Data d;

Data \*firstnode;

int Head, Tail;

public:

Queue()

{

firstnode = NULL;

Head = Tail = -1;

}

void EnQueue(int value)

{

if (firstnode == NULL)

{

Head = Tail = 0;

Data \*temp = new Data();

temp->id = value;

temp->nextnodeaddress = NULL;

firstnode=temp;

cout << "Successfully Entered: " << firstnode->id << endl;

Tail++;

}

else

{

Data \*newnode = new Data();

newnode->id = value;

newnode->nextnodeaddress = NULL;

Data \*temp = firstnode;

while (temp->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = newnode;

Tail++;

cout << "Successfully Entered: " << temp->id << endl;

}

}

void DeQueue()

{

if (Head = Tail == -1)

{

cout << "No Data" << endl;

}

else if (Tail == 0)

{

cout << "Deleting..." << firstnode->id << endl;

firstnode = NULL;

}

else

{

Data \*temp = firstnode;

firstnode = firstnode->nextnodeaddress;

cout << "Deleting .." << temp->id << endl;

temp->nextnodeaddress = NULL;

delete temp;

Tail--;

Data \*Forward = firstnode->nextnodeaddress;

Data \*Previous = firstnode;

while (Forward->nextnodeaddress != NULL)

{

while (Previous->nextnodeaddress == Forward)

{

Previous->id = Forward->id;

Previous = Previous->nextnodeaddress;

Forward = Forward->nextnodeaddress;

}

}

}

}

};

void main()

{

Queue kq;

kq.EnQueue(1);

kq.EnQueue(2);

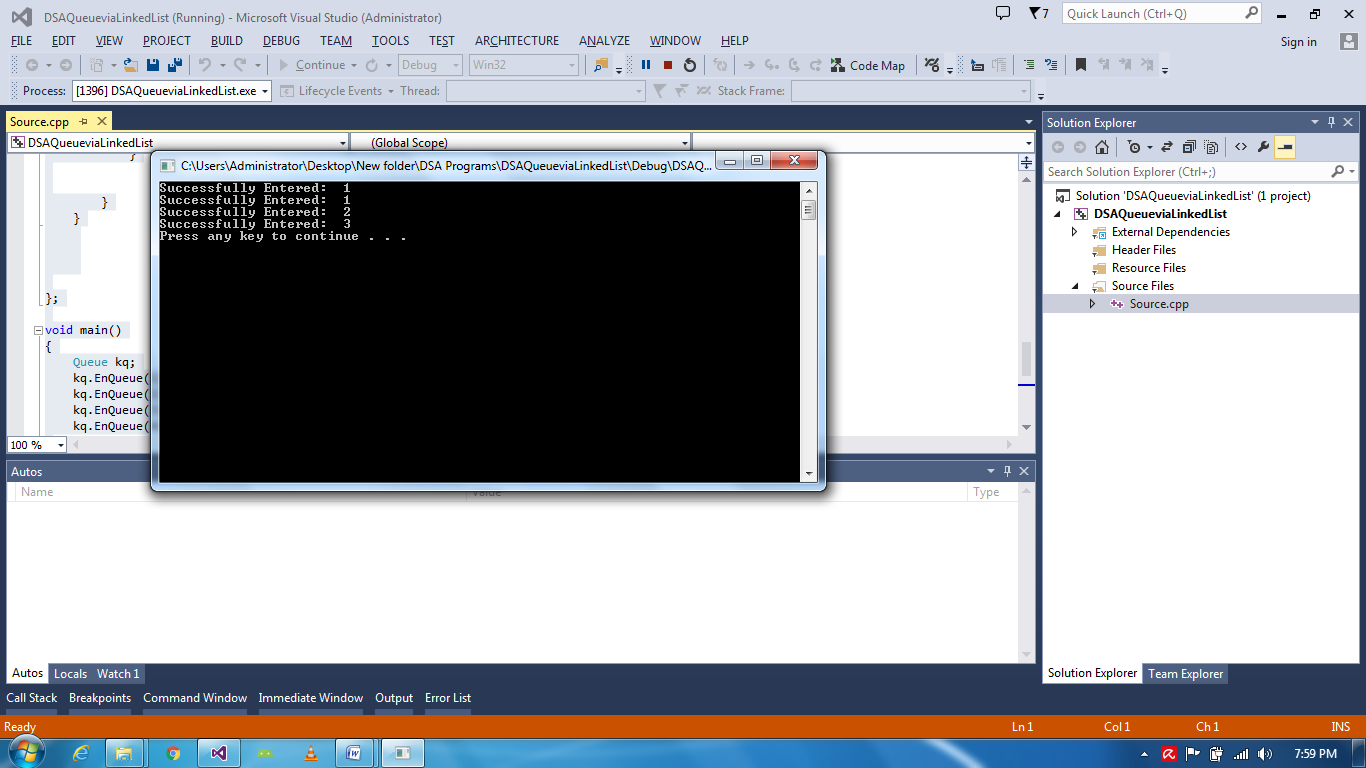
kq.EnQueue(3);

kq.EnQueue(4);

system("pause");

}

**Output**



# LAB # 9

# Circular Queue via Array:

Code;

# include <iostream>

# include <string>

using namespace std;

struct MData

{

int value;

};

class Queue

{

MData data[5];

int Head = -1;

int Tail = -1;

public:

void EnQueue(int value)

{

if (Tail == 4 && Head == 0)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail==4 && Head==4)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail + 1 == Head)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail == 4 && Head != 0)

{

if (Tail == 4)

{

Tail = 0;

}

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

Tail++;

cout << endl;

PrintData();

}

else

{

if (Head == -1)

{

Head++;

}

Tail++;

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

}

void DeQueue()

{

if (Head == Tail == -1)

{

cout << "No Data" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

}

else

{

Head++;

cout << "Deleted successfully Head=" << Head << " and Tail= " << Tail << endl;

cout << endl;

PrintData();

}

}

void PrintData()

{

for (int i = 0; i < Tail; i++)

{

cout << "Data[" << i << "]= " << data[i].value << endl;

}

cout << endl;

}

};

void main()

{

Queue mQ;

mQ.EnQueue(0);

mQ.EnQueue(1);

mQ.EnQueue(2);

mQ.EnQueue(3);

mQ.EnQueue(4);

mQ.EnQueue(5);

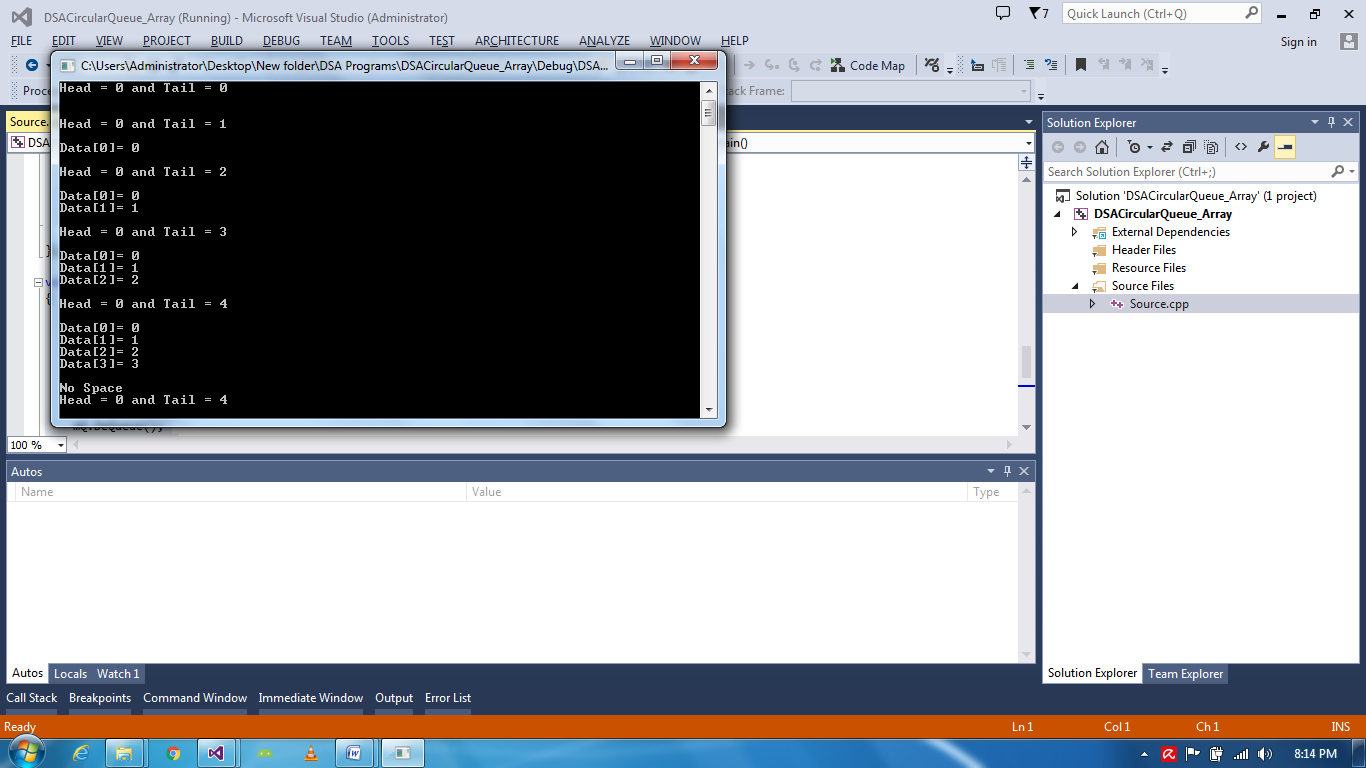
mQ.DeQueue();

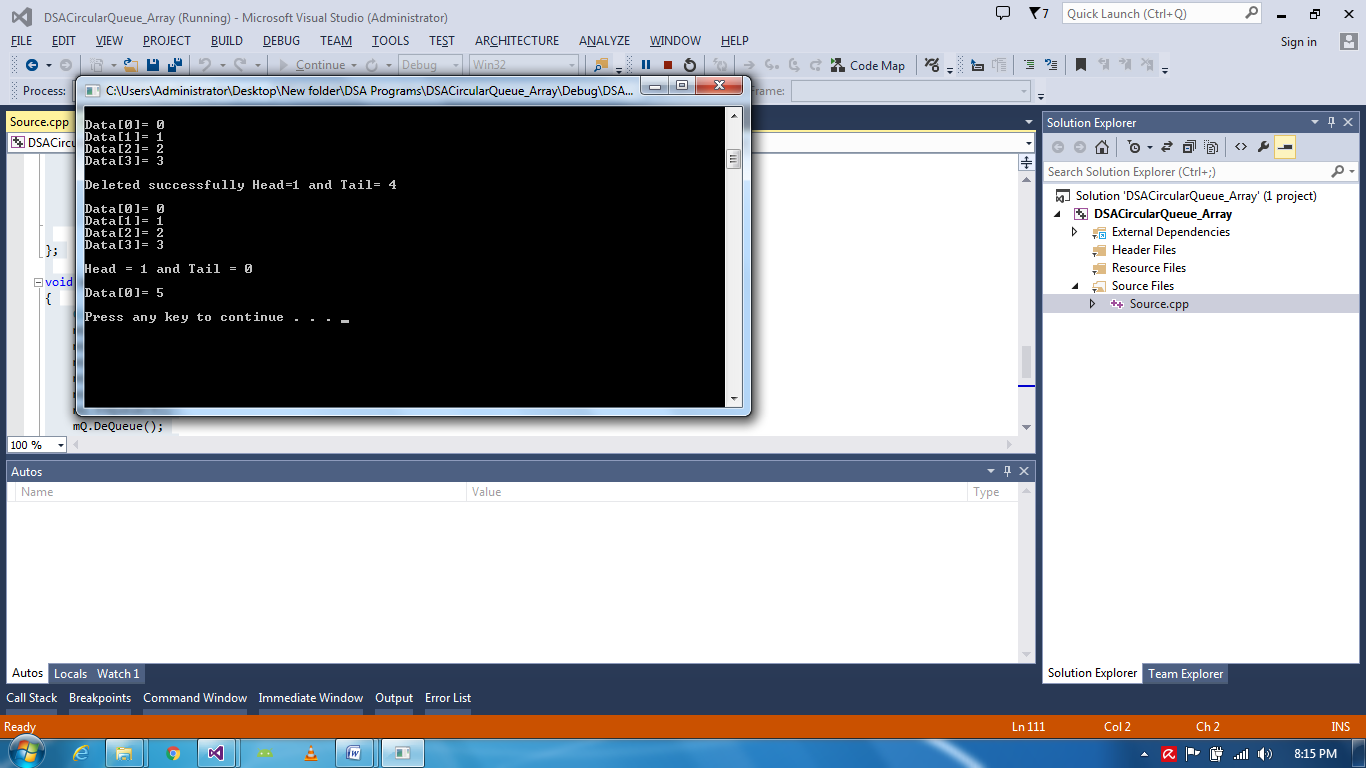
mQ.EnQueue(5);

system("pause");

}

**Output**

****



# LAB # 10

**Input Restricted Queue via Array:**

**Code:**

# include <iostream>

# include <string>

using namespace std;

struct MData

{

int value;

};

class Queue

{

MData data[5];

int Head = -1;

int Tail = -1;

char deletingposition;

public:

void EnQueue(int value)

{

if (Tail == 4 && Head == 0)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail == 4 && Head == 4)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail + 1 == Head)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail == 4 && Head != 0)

{

if (Tail == 4)

{

Tail = 0;

}

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

Tail++;

cout << endl;

PrintData();

}

else

{

if (Head == -1)

{

Head++;

}

Tail++;

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

}

void DeQueue()

{

cout << "Please enter position you want to delete(H for Head, T for Tail): " << endl;

cin >> deletingposition; cout << endl;

if (Head == Tail == -1)

{

cout << "No Data" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

}

else

{

if (deletingposition == 'H' || deletingposition=='h')

{

Head++;

cout << "Deleted successfully Head=" << Head << " and Tail= " << Tail << endl;

cout << endl;

PrintData();

}

else if (deletingposition == 'T' || deletingposition == 't')

{

Tail--;

cout << "Deleted successfully Head=" << Head << " and Tail= " << Tail << endl;

cout << endl;

PrintData();

}

}

}

void PrintData()

{

for (int i = 0; i < Tail; i++)

{

cout << "Data[" << i << "]= " << data[i].value << endl;

}

cout << endl;

}

};

void main()

{

Queue mQ;

mQ.EnQueue(0);

mQ.EnQueue(1);

mQ.EnQueue(2);

mQ.EnQueue(3);

mQ.EnQueue(4);

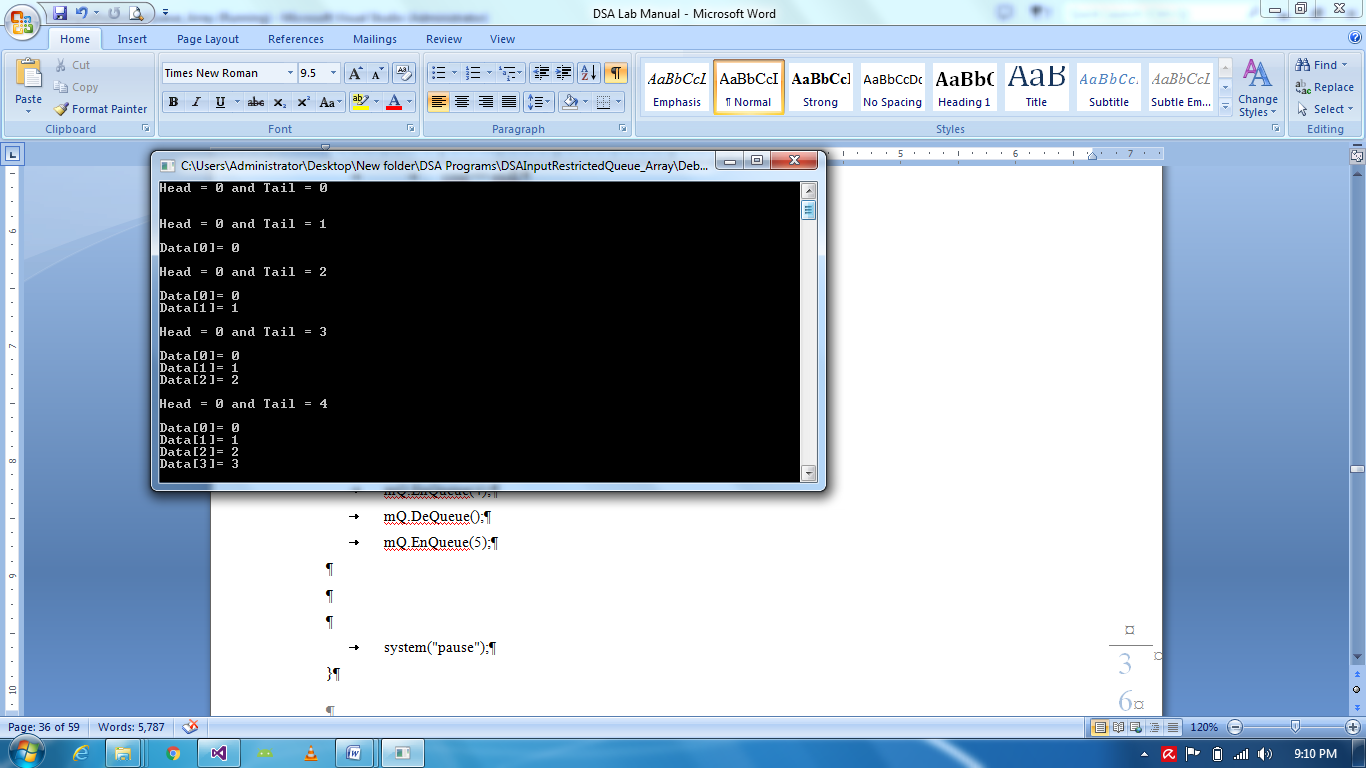
mQ.DeQueue();

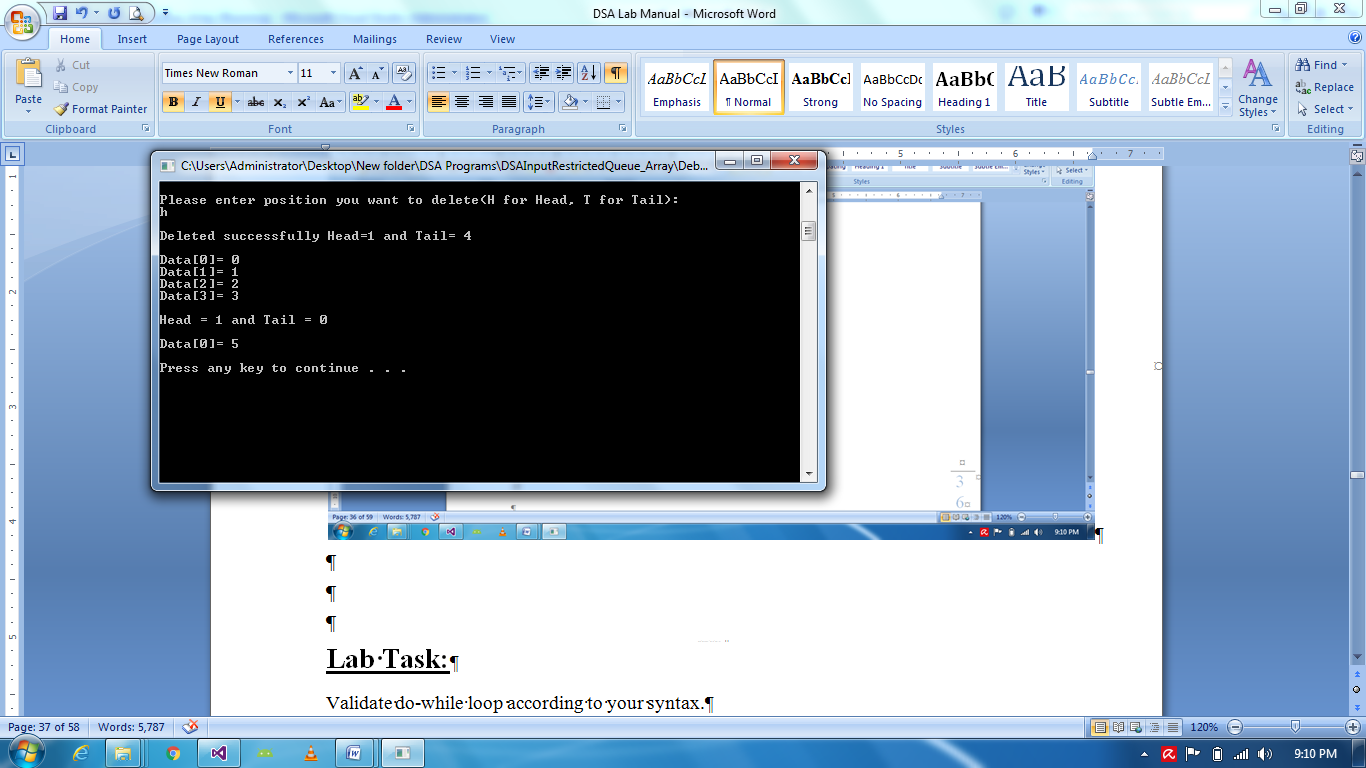
mQ.EnQueue(5);

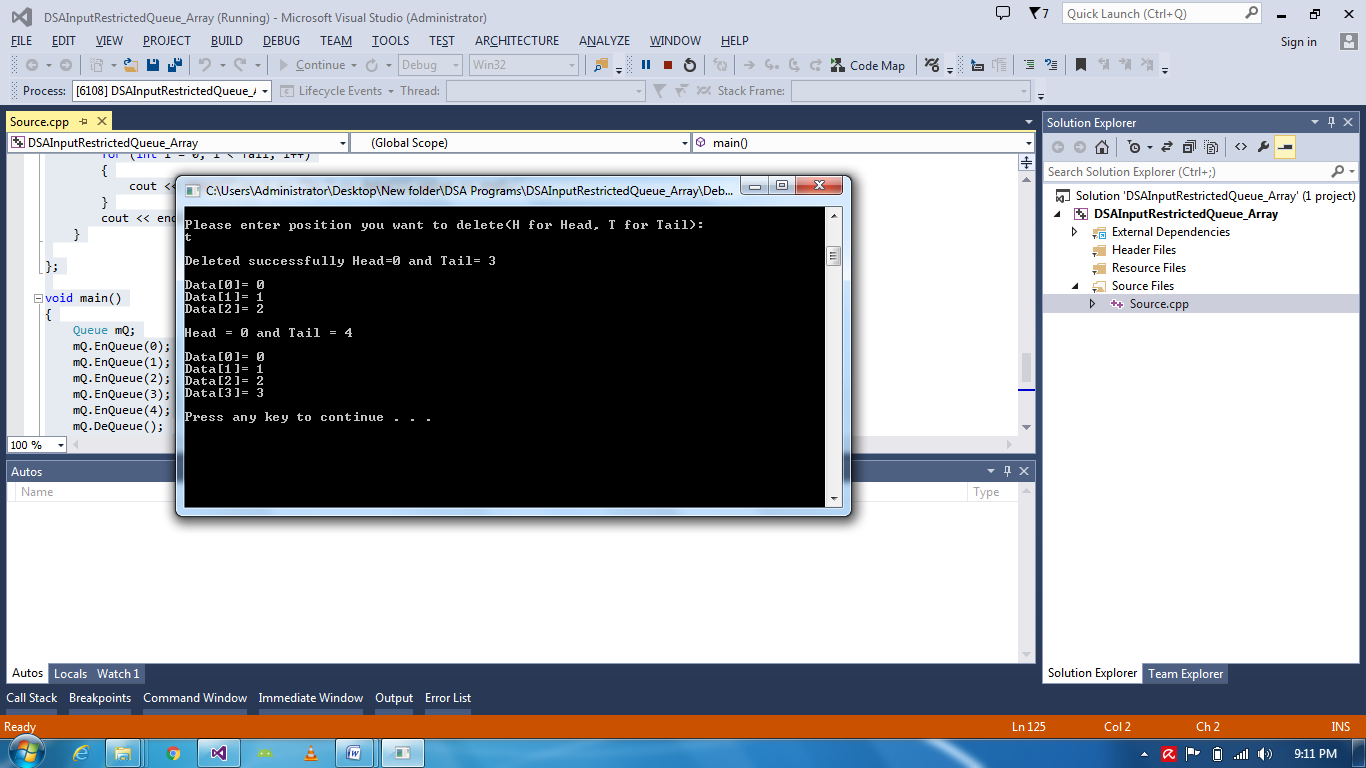
system("pause");

}

**Output**:



****

****

# LAB # 11

# Input Restricted Queue via Linked List:

Code:

# include <iostream>

# include <string>

using namespace std;

struct Data

{

int id;

Data \*nextnodeaddress;

};

class Queue

{

Data d;

Data \*firstnode;

int Head, Tail;

char deletingposition;

public:

Queue()

{

firstnode = NULL;

Head = Tail = -1;

}

void EnQueue(int value)

{

if (firstnode == NULL)

{

Head = Tail = 0;

Data \*temp = new Data();

temp->id = value;

temp->nextnodeaddress = NULL;

firstnode = temp;

cout << "Successfully Entered: " << firstnode->id << endl;

Tail++;

}

else

{

Data \*newnode = new Data();

newnode->id = value;

newnode->nextnodeaddress = NULL;

Data \*temp = firstnode;

while (temp->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = newnode;

Tail++;

cout << "Successfully Entered: " << temp->id << endl;

}

}

void DeQueue()

{

cout << "Enter deleting position(H for Head, T for Tail): "; cin >> deletingposition; cout << endl;

if (deletingposition == 'T' || deletingposition == 't')

{

if (Head = Tail == -1)

{

cout << "No Data" << endl;

}

if (Head = Tail == -1)

{

cout << "No Data" << endl;

}

else if (Tail == 0)

{

cout << "Deleting..." << firstnode->id << endl;

firstnode = NULL;

}

else

{

Data \*temp = firstnode;

firstnode = firstnode->nextnodeaddress;

cout << "Deleting .." << temp->id << endl;

temp->nextnodeaddress = NULL;

delete temp;

Tail--;

Data \*Forward = firstnode->nextnodeaddress;

Data \*Previous = firstnode;

while (Forward->nextnodeaddress != NULL)

{

while (Previous->nextnodeaddress == Forward)

{

Previous->id = Forward->id;

Previous = Previous->nextnodeaddress;

Forward = Forward->nextnodeaddress;

}

}

}

}

else if (deletingposition == 'H' || deletingposition == 'h')

{

if (Head = Tail == -1)

{

cout << "No Data" << endl;

}

else if (Head == 0)

{

cout << "Deleting..." << firstnode->id << endl;

firstnode = NULL;

}

else

{

Data \*temp = firstnode;

firstnode = firstnode->nextnodeaddress;

cout << "Deleting .." << temp->id << endl;

temp->nextnodeaddress = NULL;

delete temp;

Head--;

}

}

}

};

void main()

{

Queue kq;

kq.EnQueue(1);

kq.EnQueue(2);

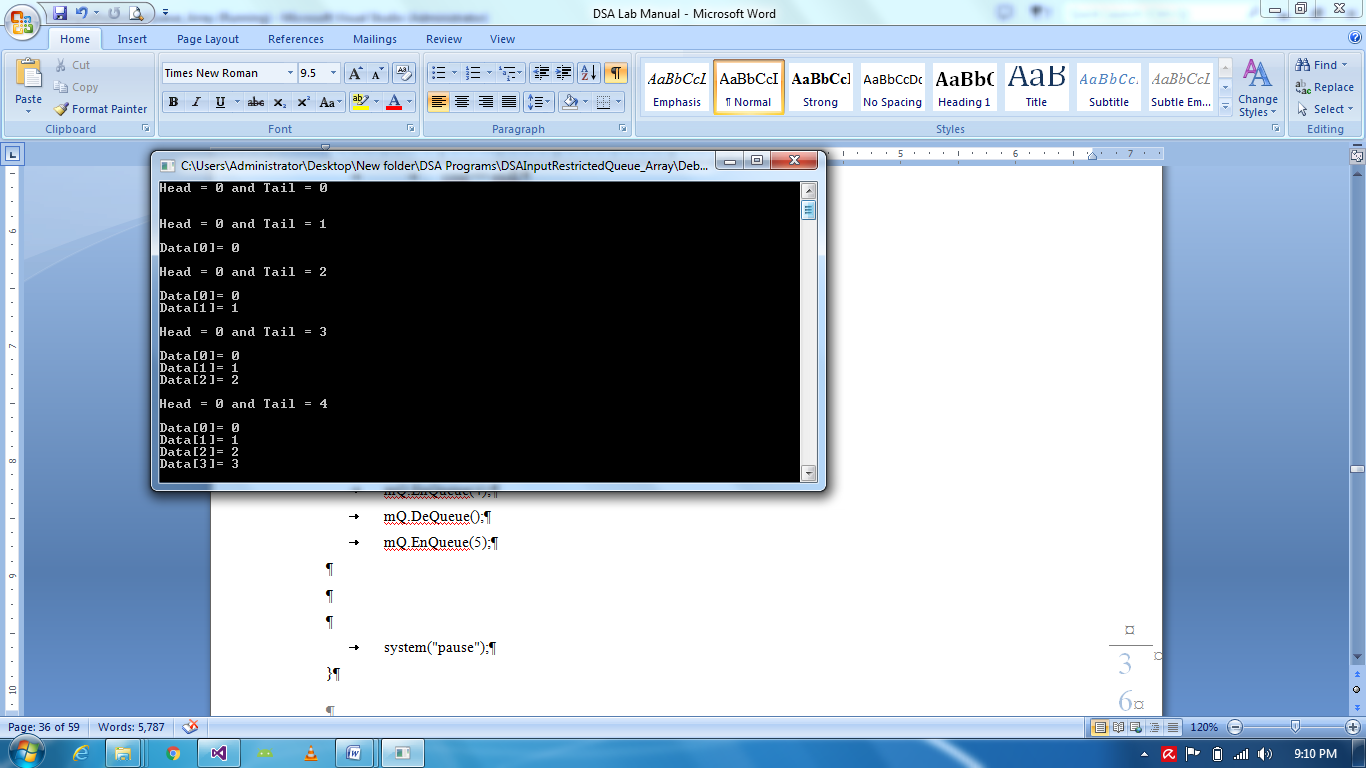
kq.EnQueue(3);

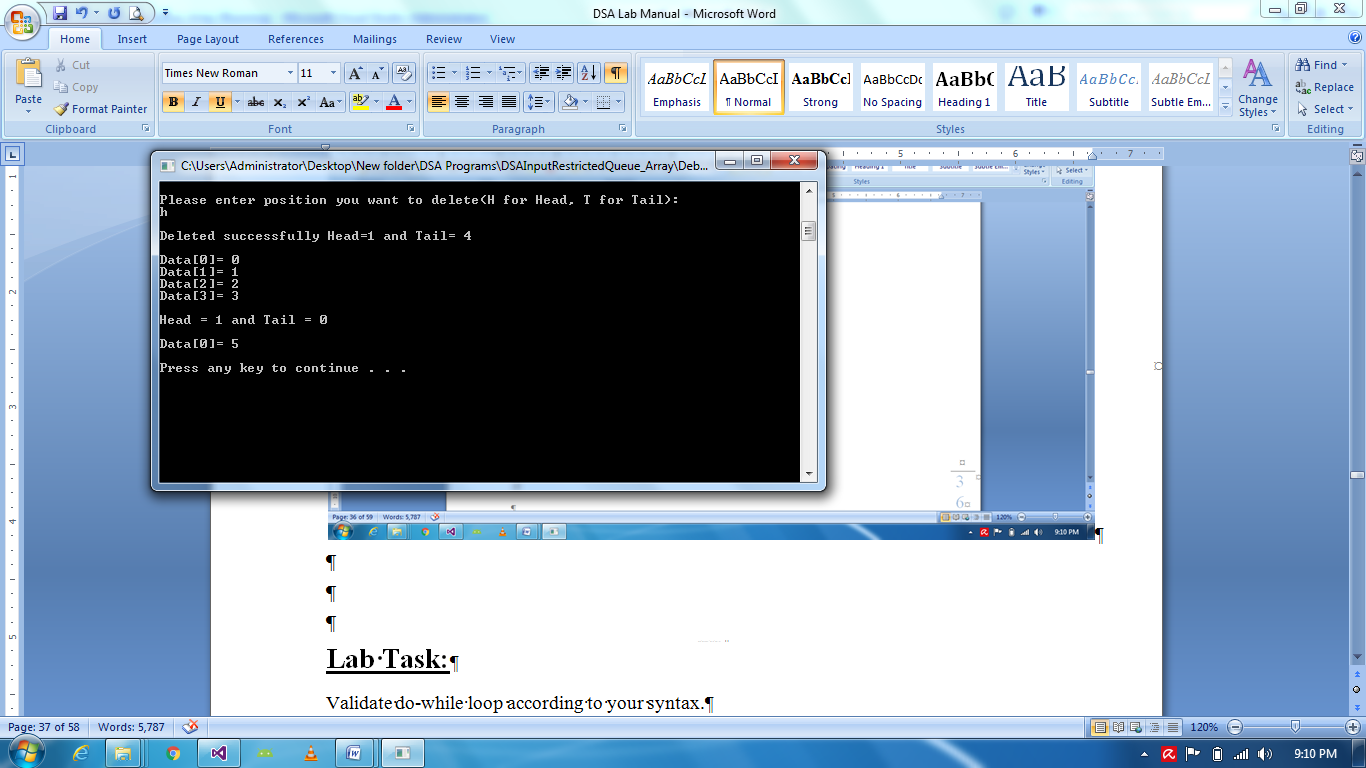
kq.EnQueue(4);

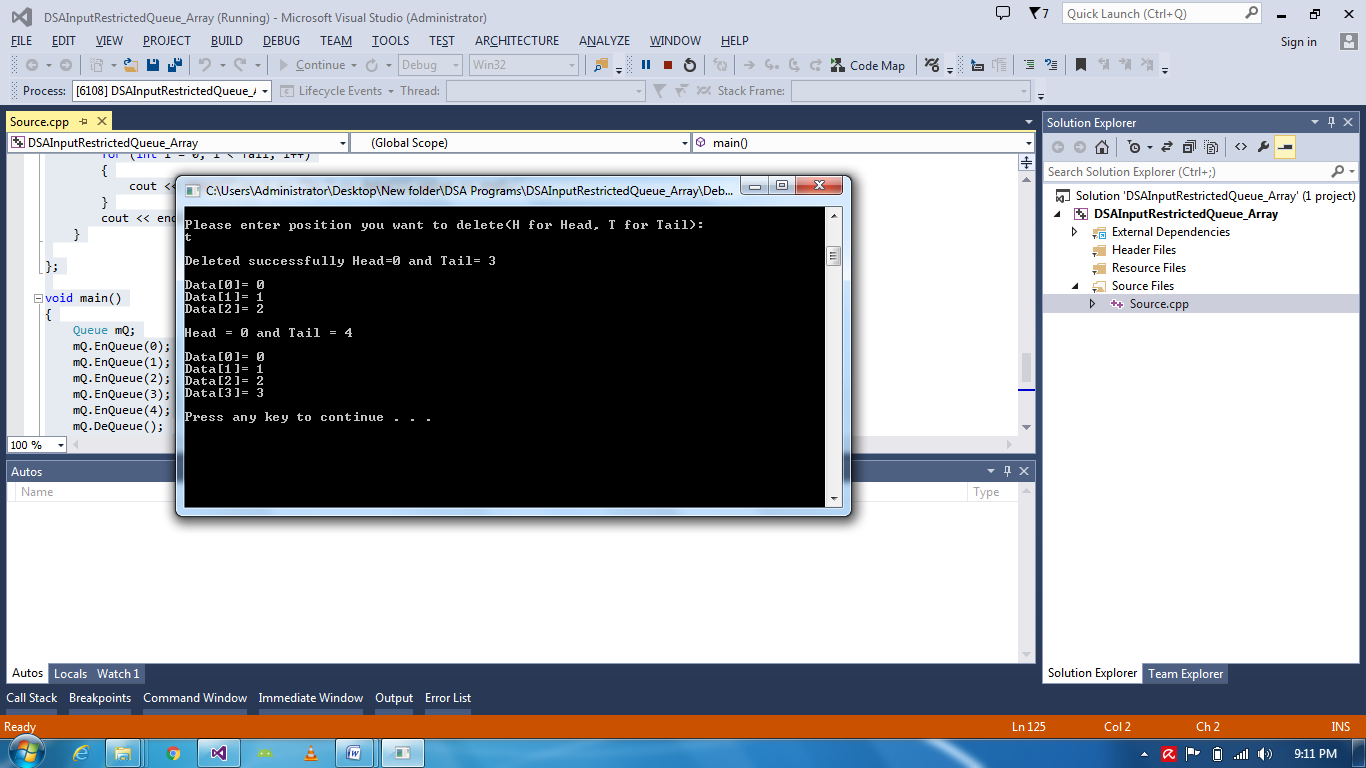
system("pause");

}

**Output**:



****

****

# LAB # 12

**Output Restricted Queue via Array:**

Code:

# include <iostream>

# include <string>

using namespace std;

struct MData

{

int value;

};

class Queue

{

MData data[5];

int Head = -1;

int Tail = -1;

char insertingposition;

public:

void EnQueue(int value)

{

cout << "Please enter position you want to insert from(H for Head, T for Tail): " << endl;

cin >> insertingposition; cout << endl;

if (Tail == 4 && Head == 0)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail == 4 && Head == 4)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail + 1 == Head)

{

cout << "No Space" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else if (Tail == 4 && Head != 0)

{

if (insertingposition == 'T' || insertingposition == 't')

{

if (Tail == 4)

{

Tail = 0;

}

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

Tail++;

cout << endl;

PrintData();

}

else

{

if (Head == -1)

{

Head++;

}

Tail++;

data[Tail].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

}

else if (insertingposition == 'H' || insertingposition == 'h')

{

if (Tail == 4)

{

Tail = 0;

}

Tail++;

for (int i = Tail; i >= 0; i--)

{

data[i] = data[i - 1];

}

data[Head].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

else

{

if (Head == -1)

{

Head++;

data[Head].value = value;

}

Tail++;

for (int i = Tail; i >= 0; i--)

{

data[i] = data[i - 1];

}

data[Head].value = value;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

PrintData();

}

}

void DeQueue()

{

if (Head == Tail == -1)

{

cout << "No Data" << endl;

cout << "Head = " << Head << " and Tail = " << Tail << endl;

cout << endl;

}

else

{

Head++;

cout << "Deleted successfully Head=" << Head << " and Tail= " << Tail << endl;

cout << endl;

PrintData();

}

}

void PrintData()

{

for (int i = 0; i <= Tail; i++)

{

cout << "Data[" << i << "]= " << data[i].value << endl;

}

cout << endl;

}

};

void main()

{

Queue mQ;

mQ.EnQueue(0);

mQ.EnQueue(1);

mQ.EnQueue(2);

mQ.EnQueue(3);

mQ.EnQueue(4);

mQ.EnQueue(5);

mQ.DeQueue();

mQ.EnQueue(5);

system("pause");

}

**Output**

//has errors

**Lab 13**

**Output Restricted Queue via Linked List**

**Code**:

# include <iostream>

# include <string>

using namespace std;

struct Data

{

int id;

Data \*nextnodeaddress;

};

class Queue

{

Data d;

Data \*firstnode;

int Head, Tail;

char insertingposition;

public:

Queue()

{

firstnode = NULL;

Head = Tail = -1;

}

void EnQueue(int value)

{

cout << "Please enter inserting position (H for Head, T for Tail): ";

cin >> insertingposition; cout << endl;

if (insertingposition == 'H' || insertingposition == 'h')

{

if (firstnode == NULL)

{

Head = Tail = 0;

Data \*temp = new Data();

temp->id = value;

temp->nextnodeaddress = NULL;

firstnode = temp;

cout << "Successfully Entered: " << firstnode->id << endl;

Tail++;

}

else

{

Data \*newnode = new Data();

newnode->id = value;

newnode->nextnodeaddress = firstnode;

firstnode = newnode;

Data \*temp = firstnode;

while (temp->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = newnode;

Tail++;

cout << "Successfully Entered: " << temp->id << endl;

}

}

if (insertingposition == 'T' || insertingposition == 't')

{

if (firstnode == NULL)

{

Head = Tail = 0;

Data \*temp = new Data();

temp->id = value;

temp->nextnodeaddress = NULL;

firstnode = temp;

cout << "Successfully Entered: " << firstnode->id << endl;

Tail++;

}

else

{

Data \*newnode = new Data();

newnode->id = value;

newnode->nextnodeaddress = NULL;

Data \*temp = firstnode;

while (temp->nextnodeaddress != NULL)

{

temp = temp->nextnodeaddress;

}

temp->nextnodeaddress = newnode;

Tail++;

cout << "Successfully Entered: " << temp->id << endl;

}

}

}

void DeQueue()

{

if (Head = Tail == -1)

{

cout << "No Data" << endl;

}

else if (Tail == 0)

{

cout << "Deleting..." << firstnode->id << endl;

firstnode = NULL;

}

else

{

Data \*temp = firstnode;

firstnode = firstnode->nextnodeaddress;

cout << "Deleting .." << temp->id << endl;

temp->nextnodeaddress = NULL;

delete temp;

Tail--;

Data \*Forward = firstnode->nextnodeaddress;

Data \*Previous = firstnode;

while (Forward->nextnodeaddress != NULL)

{

while (Previous->nextnodeaddress == Forward)

{

Previous->id = Forward->id;

Previous = Previous->nextnodeaddress;

Forward = Forward->nextnodeaddress;

}

}

}

}

};

void main()

{

Queue kq;

kq.EnQueue(1);

kq.EnQueue(2);

kq.EnQueue(3);

kq.EnQueue(4);

system("pause");

}

**Output:**

**//has errors**

**Lab 14**

**Stack:**

# include <iostream>

# include <string>

using namespace std;

struct MyData

{

int value;

};

class Stack

{

public:

static int SP;

MyData d[10];

void Push(int value)

{

if (SP == 10)

{

cout << "Stack is full" << endl;

}

else

{

d[SP].value = value;

SP++;

}

}

void Pop()

{

if (SP == 0)

{

cout << "No data" << endl;

}

else

{

cout << d[SP - 1].value << " is deleted" << endl;

SP--;

}

}

void Peek()

{

if (SP == 0)

{

cout << "No data" << endl;

}

else

cout << "Top value is: " << d[SP-1].value << endl;

}

};

int Stack::SP = 0;

void main()

{

Stack mystack;

mystack.Push(5);

mystack.Push(10);

mystack.Push(15);

mystack.Push(20);

mystack.Push(25);

mystack.Push(30);

mystack.Pop();

mystack.Pop();

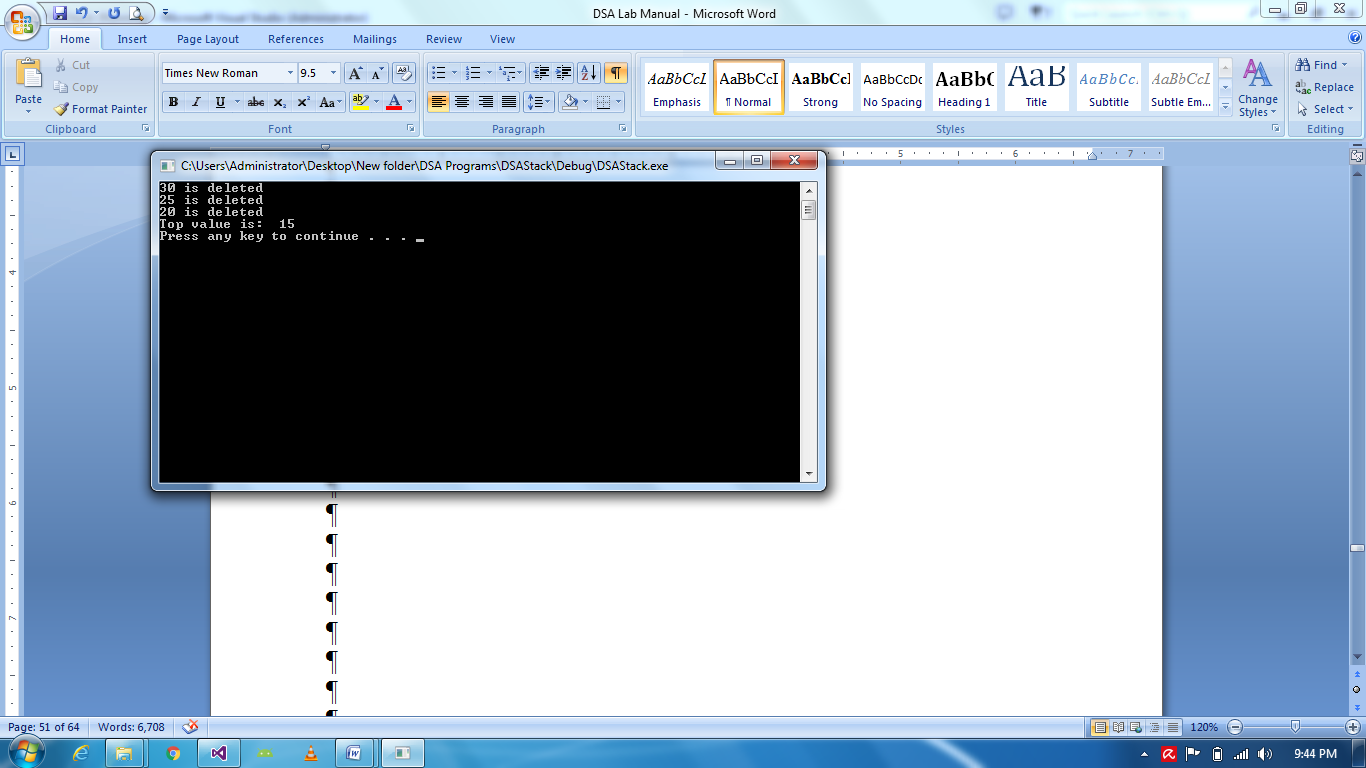
mystack.Pop();

mystack.Peek();

system("pause");

}

**Output**



**Task I: Counting Brackets (Stack):**

Code:

# include <iostream>

# include <string>

using namespace std;

class Stack

{

static int SP;

char \*ch;

public:

Stack(int s)

{

ch = new char[s];

}

void Push(char c)

{

++SP ;

ch[SP] = c;

}

void Pop()

{

SP--;

}

bool IsValid()

{

if (SP == -1)

{

return true;

}

return false;

}

};

int Stack::SP = -1;

int CountOpeningBracket(string s)

{

int c = 0;

for (int i = 0; i < s.length(); i++)

{

if (s[i] == '{')

{

c++;

}

}

return c;

}

void main()

{

string code;

cin >> code;

int c = CountOpeningBracket(code);

Stack s(c);

for (int i = 0; i < code.length(); i++)

{

if (code[i] == '{')

{

s.Push('{');

}

else if (code[i] == '}')

{

s.Pop();

}

}

if (s.IsValid())

{

cout << "Correct brackets" << endl;

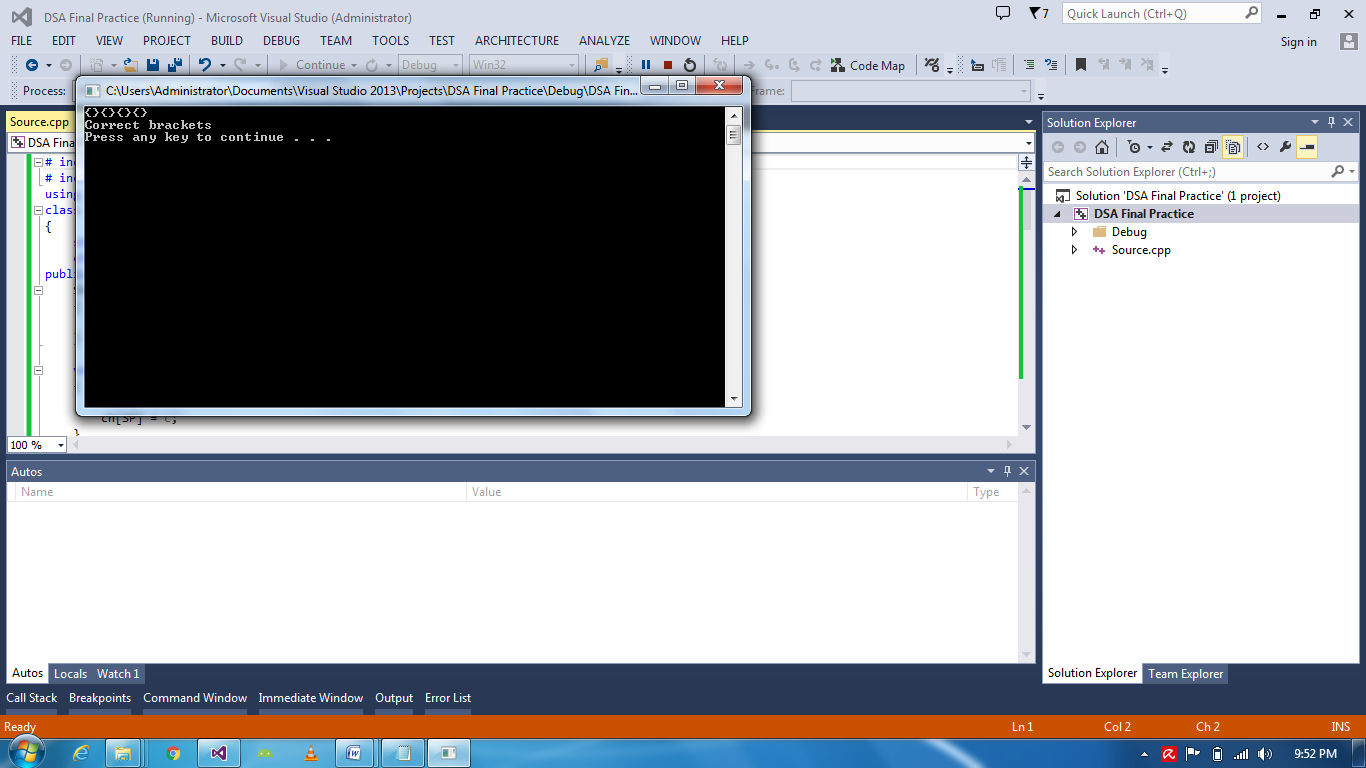
}

else cout << "Brackets mismatch" << endl;

system("pause");

}

**Output**



**Task II: InFix To PostFix Converter:**

Code:

# include <iostream>

# include <string>

using namespace std;

struct node

{

string ch;

node \* next;

};

class Stack

{

node \* first;

public:

void push(string c)

{

node \* newnode = new node;

newnode->ch = c;

newnode->next = first;

first = newnode;

}

string pop()

{

if (first != NULL)

{

node \* temp = new node;

temp = first;

first = first->next;

temp->next = NULL;

string ch = temp->ch;

delete temp;

return ch;

}

}

string peek()

{

if (first != NULL)

return first->ch;

else

return "x";

}

};

struct st

{

int num;

st \*next;

};

class myStack

{

st \*First;

public:

myStack()

{

First = NULL;

}

void push(int n)

{

st \*newNum = new st;

newNum->num = n;

newNum->next = First;

First = newNum;

}

void Pop()

{

if (First != NULL)

{

st \*temp = new st;

First = First->next;

temp->next = NULL;

delete temp;

}

else

{

cout << "Stack is empty" << endl;

}

}

int Peek()

{

return First->num;

}

};

float evalpostfix(string);

string InfixToPostfixConv(string);

int main()

{

string infix = "2+3\*4/2+3\*6\*(4/2\*(3+2)\*6+8/2)";

cout << "Answer is : " << InfixToPostfixConv(infix) << endl;

system("pause");

string postfixi = InfixToPostfixConv(infix);

float val = evalpostfix(postfixi);

cout << "\nAnswer is: " << val << endl;

system("pause");

return 0;

}

float evalpostfix(string postfix)

{

myStack s;

int i = 0;

char ch;

float val = 0;

int size = postfix.length();

for (int i = 0; i < size; i++)

{

ch = postfix[i];

if (isdigit(ch))

{

s.push(int(ch - '0'));

}

else

{

float num1 = s.Peek();

s.Pop();

float num2 = s.Peek();

s.Pop();

if (ch == '/')

{

val = num2 / num1;

}

else if (ch == '\*')

{

val = num2\*num1;

}

else if (ch == '-')

{

val = num2 - num1;

}

else if (ch == '+')

{

val = num2 + num1;

}

else if (ch == '^')

{

val = 1.0;

for (int i = 0; i<int(num1); i++)

{

val \*= num1;

}

val = num2 + num1;

}

s.push(int(val));

}

}

return val;

}

string InfixToPostfixConv(string infix)

{

Stack s;

string postfix = "";

infix = "(" + infix + ")";

for (int i = 0; i<infix.length(); i++)

{

if (infix[i] == '(')

{

s.push("(");

}

else if (infix[i] == '+')

{

while (s.peek() != "+" && s.peek() != "(")

{

postfix = postfix + s.pop();

}

s.push("+");

}

else if (infix[i] == '-')

{

while (s.peek() != "+" && s.peek() != "(" && s.peek() != "-")

{

postfix = postfix + s.pop();

}

s.push("-");

}

else if (infix[i] == '\*')

{

while (s.peek() != "+" && s.peek() != "(" && s.peek() != "-" && s.peek() != "\*")

{

postfix = postfix + s.pop();

}

s.push("\*");

}

else if (infix[i] == '/')

{

while (s.peek() != "+" && s.peek() != "(" && s.peek() != "-" && s.peek() != "\*" && s.peek() != "/")

{

postfix = postfix + s.pop();

}

s.push("/");

}

else if (infix[i] == '^')

{

while (s.peek() != "+" && s.peek() != "(" && s.peek() != "-" && s.peek() != "\*" && s.peek() != "/" && s.peek() != "^")

{

postfix = postfix + s.pop();

}

s.push("^");

}

else if (infix[i] == ')')

{

while (s.peek() != "(")

{

postfix = postfix + s.pop();

}

s.pop();

}

else

{

postfix = postfix + infix[i];

}

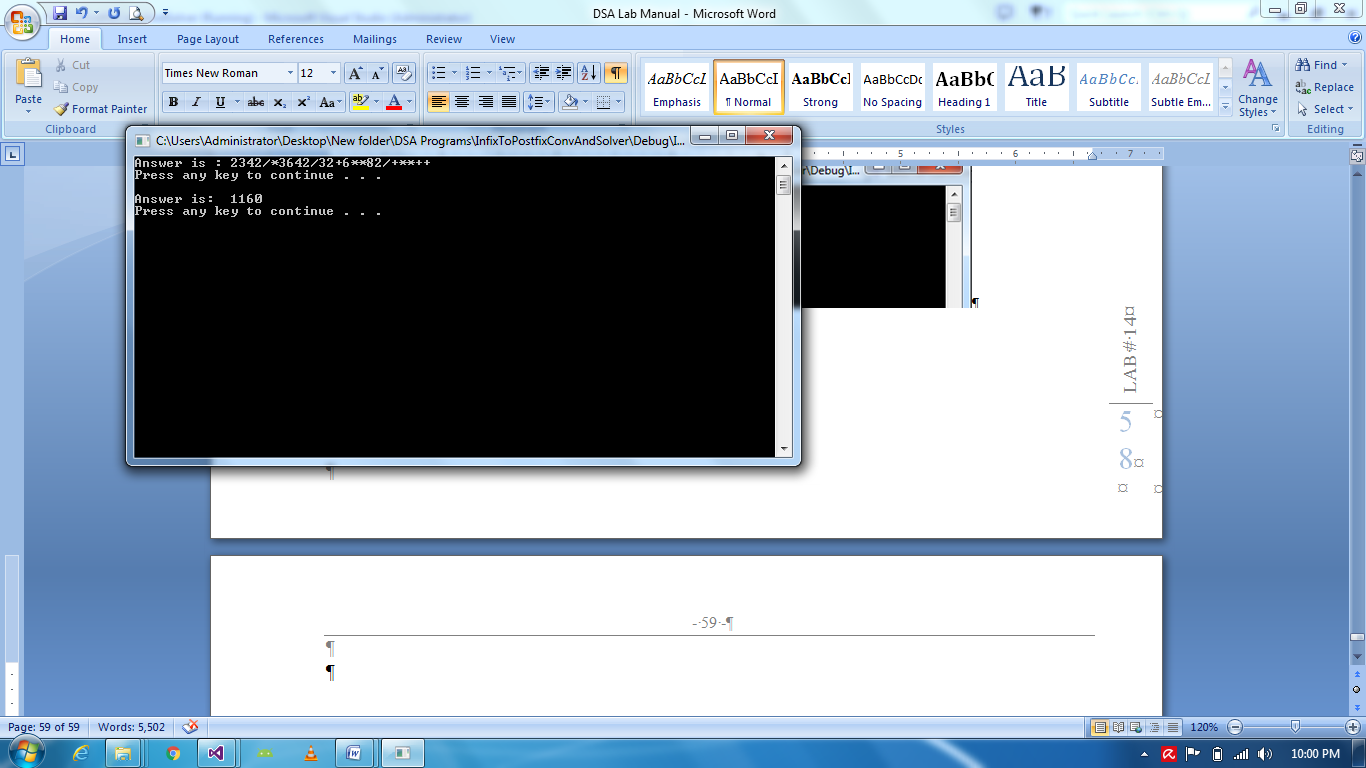
}

// cout << "post fix " << postfix << endl;

return postfix;

}

**Output**



**Lab 15:**

**Graph:**

Code:

#include <iostream>

#include<string>

using namespace std;

void main()

{

string vertex[5];

int cost[5][5];

cout << "Enter vertex: ";

for (int i = 0; i < 5; i++)

{

cin >> vertex[i];

for (int j = 0; j < 5; j++)

{

cost[i][j] = -1;

}

}

int choice;

do{

cout << "Press 1 to make edge" << endl;

cout << "Press 2 to view connected edges" << endl;

cout << "Press 3 to exit" << endl;

cin >> choice;

if (choice == 1)

{

string to, from;

cout << "Enter to: " << endl;

cin >> to; cout << endl;

cout << "Enter from: " << endl;

cin >> from; cout << endl;

int toindex = -1, fromindex = -1;

if (to != from)

{

for (int i = 0; i < 5; i++)

{

if (vertex[i] == to)

{

toindex = i;

}

if (vertex[i] == from)

{

fromindex = i;

}

}

if (toindex != -1 && fromindex != -1)

{

int c;

cout << "Enter cost: "; cin >> c; cout << endl;

cost[toindex][fromindex] = c;

}

}

}

else if (choice == 2)

{

cout << "\t";

for (int i = 0; i < 5; i++)

{

cout << vertex[i] << "\t";

}

cout << endl;

for (int i = 0; i < 5; i++)

{

cout << vertex[i] << "\t";

for (int j = 0; j < 5; j++)

{

cout << cost[j][i]<<"\t";

}

cout << endl;

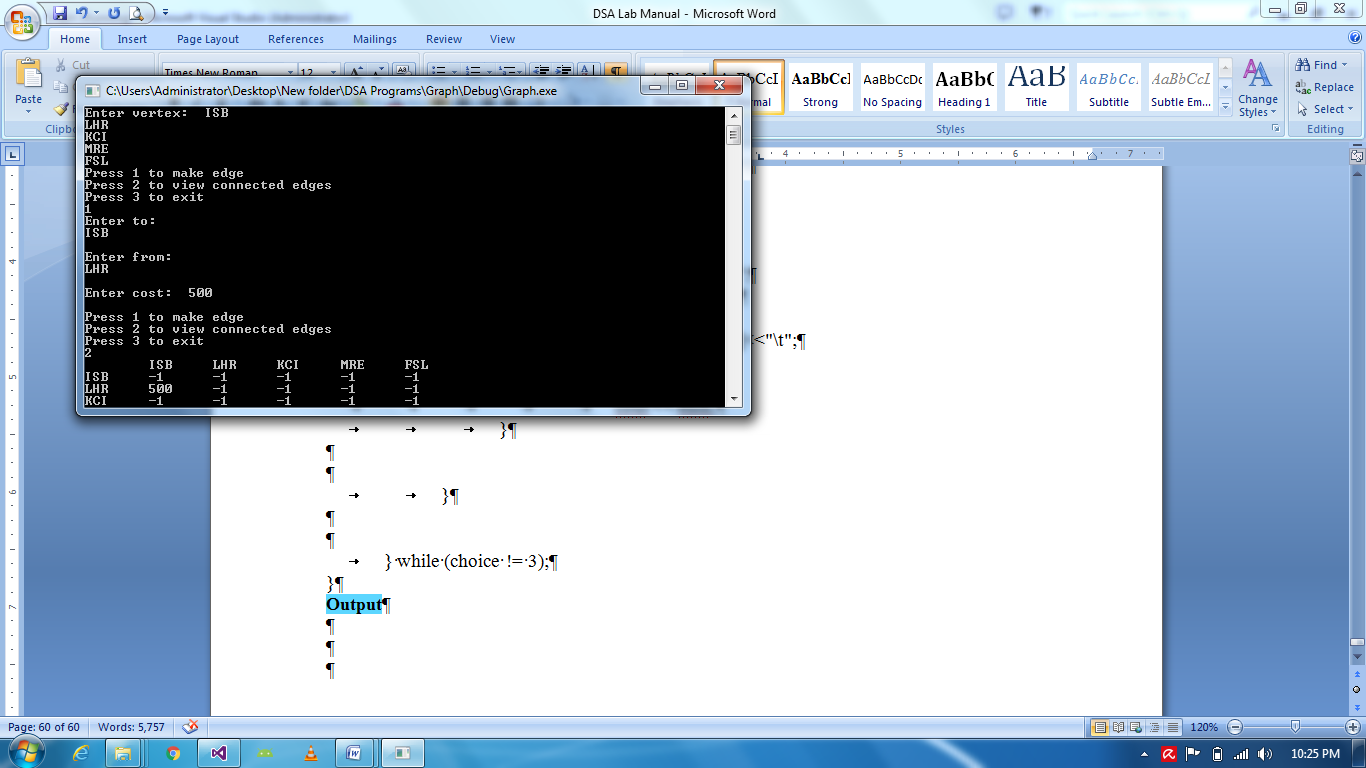
}

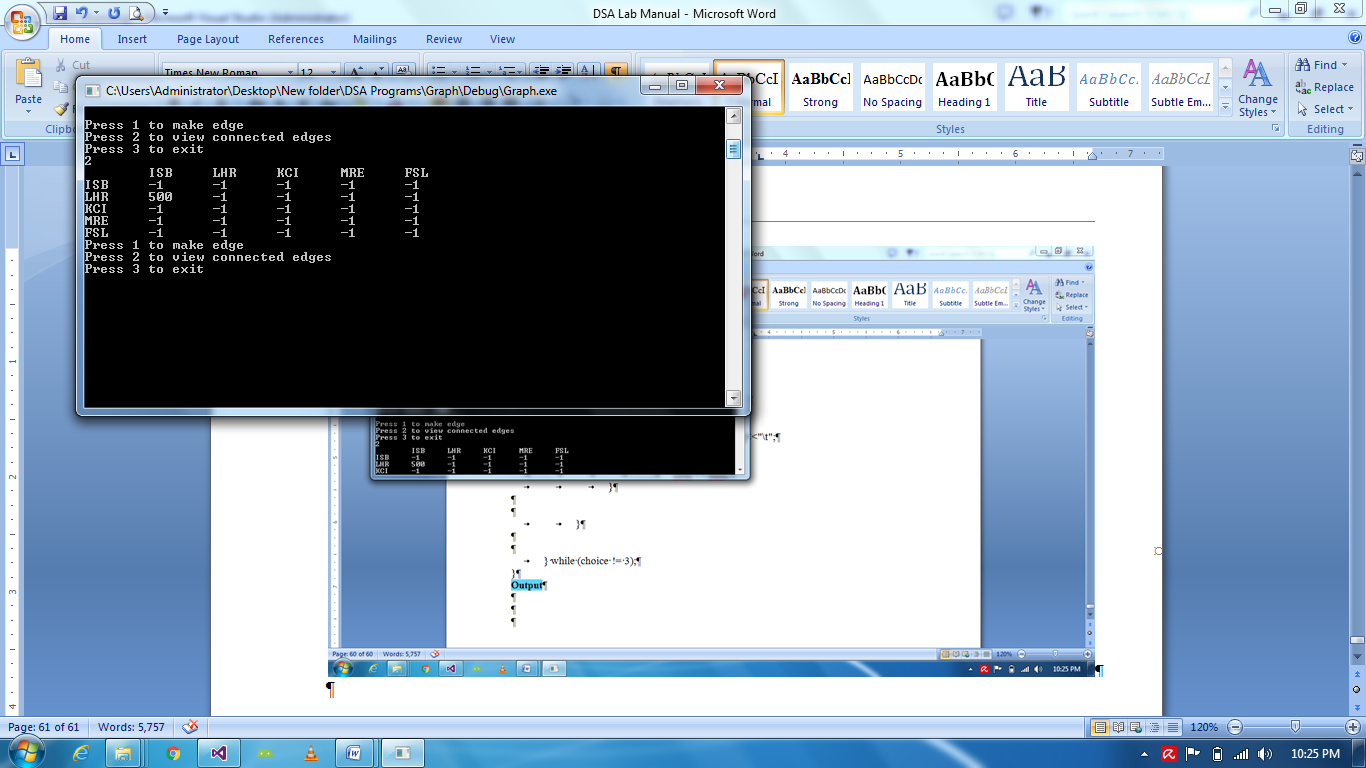
}

} while (choice != 3);

}

**Output**





**Task I: Graph via Linked List:**

Code:

#include <iostream>

#include<string>

using namespace std;

struct Edge

{

string name;

int cost;

Edge \*next;

};

void main()

{

Edge e[5];

for (int i = 0; i < 5; i++)

{

cout << "Enter name: "; cin >> e[i].name; cout << endl;

e[i].cost = -1;

e[i].next = NULL;

}

int choice;

do{

cout << "Press 1 to create edge" << endl;

cout << "Press 2 to display" << endl;

cout << "Press 3 to exit" << endl;

cin >> choice;

if (choice == 1)

{

string from;

cout << "Enter from: " << endl;

cin >> from; cout << endl;

for (int i = 0; i < 5; i++)

{

if (e[i].name==from)

{

Edge \*newedge = new Edge;

cout << "Enter name for edge: ";

cin >> newedge->name;

cout << "Enter cost for edge: ";

cin >> newedge->cost;

newedge->next = NULL;

Edge \*temp = e[i].next;

if (temp == NULL)

{

temp = newedge;

}

else

{

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newedge;

}

}

}

}

else if (choice == 2)

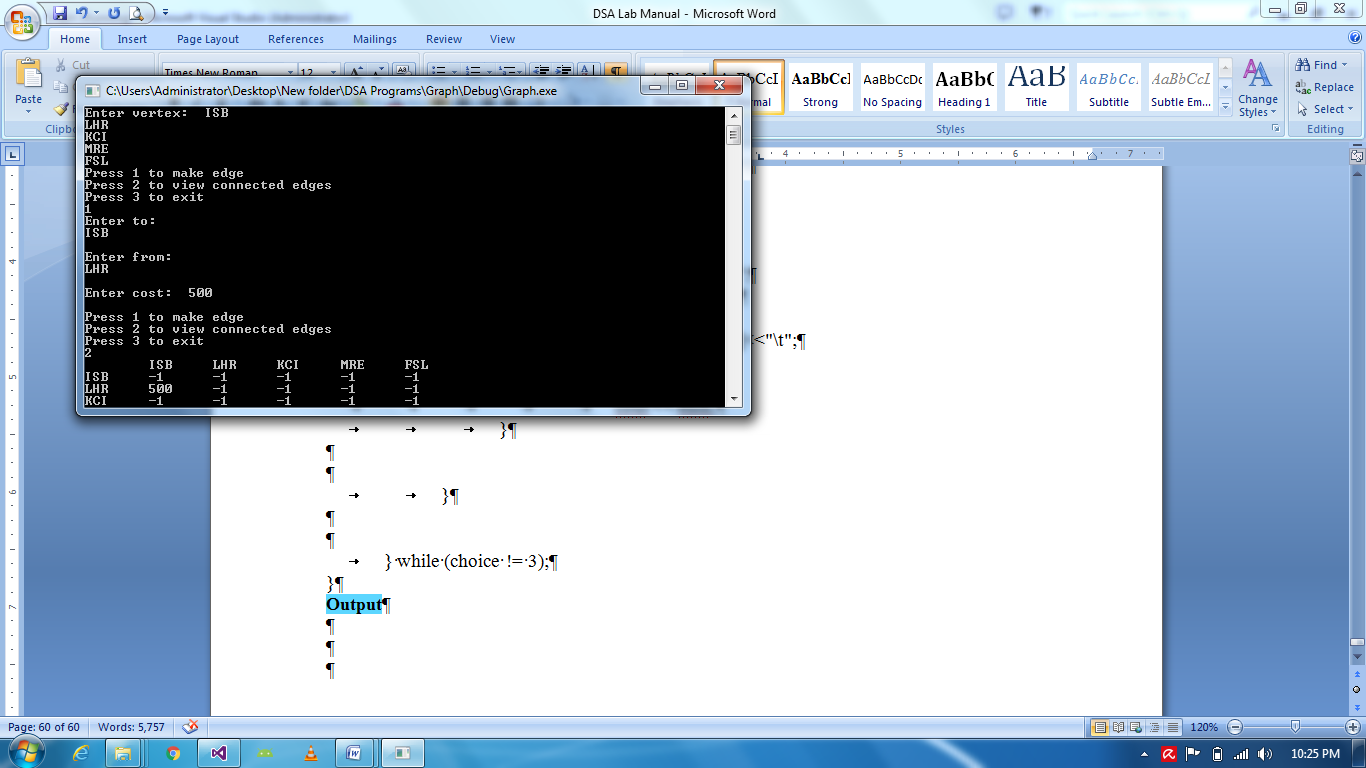
{

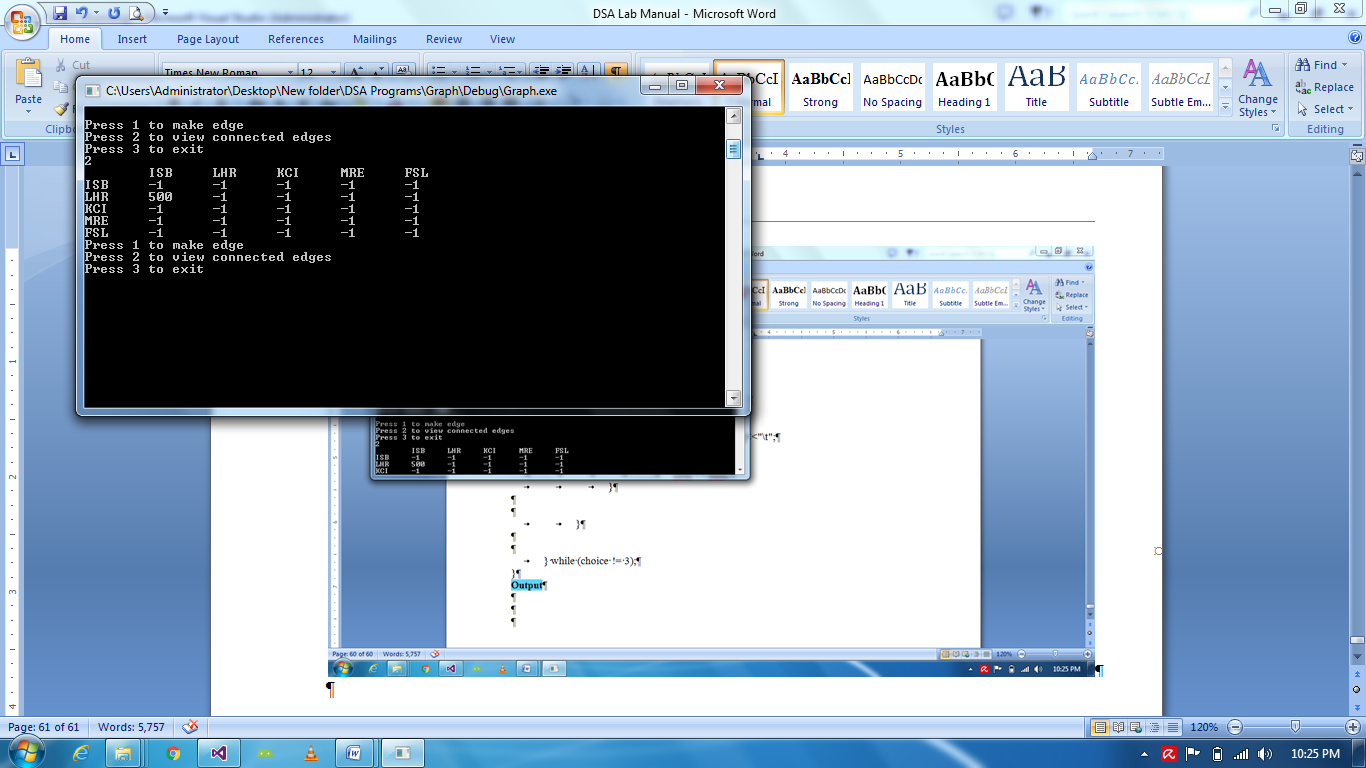
}

} while (choice != 3);

}

**Output**





**Lab 16:**

**Binary Search Tree:**

Code:

#include "iostream"

using namespace std;

struct Node

{

int nodevalue;

Node \* leftchild;

Node \* rightchild;

};

class BST

{

Node \* root;

void Link(Node \* linker, Node \*n)

{

if (n->nodevalue > linker->nodevalue)

{

if (linker->rightchild == NULL)

{

linker->rightchild = n;

return;

}

Link(linker->rightchild, n);

return;

}

else

{

if (linker->leftchild == NULL)

{

linker->leftchild = n;

return;

}

Link(linker->leftchild, n);

return;

}

}

void Display(Node \* n)

{

if (n == NULL)

return;

else

{

//cout<<n->nodevalue<<","; //PreOrder Tree Traversal

Display(n->leftchild);

cout << n->nodevalue << ","; //InOrder Tree Traversal

Display(n->rightchild);

//cout<<n->nodevalue<<","; //PostOrder Tree Traversal

}

}

public:

BST()

{

root = NULL;

}

void Insert(int val)

{

Node \* newnode = new Node;

newnode->nodevalue = val;

newnode->leftchild = NULL;

newnode->rightchild = NULL;

if (root == NULL)

root = newnode;

else

Link(root, newnode);

}

void Inordertraversal()

{

if (root == NULL)

cout << "No node";

else

Display(root);

}

};

void main()

{

BST b;

b.Insert(36);

b.Insert(18);

b.Insert(17);

b.Insert(42);

b.Insert(20);

b.Insert(40);

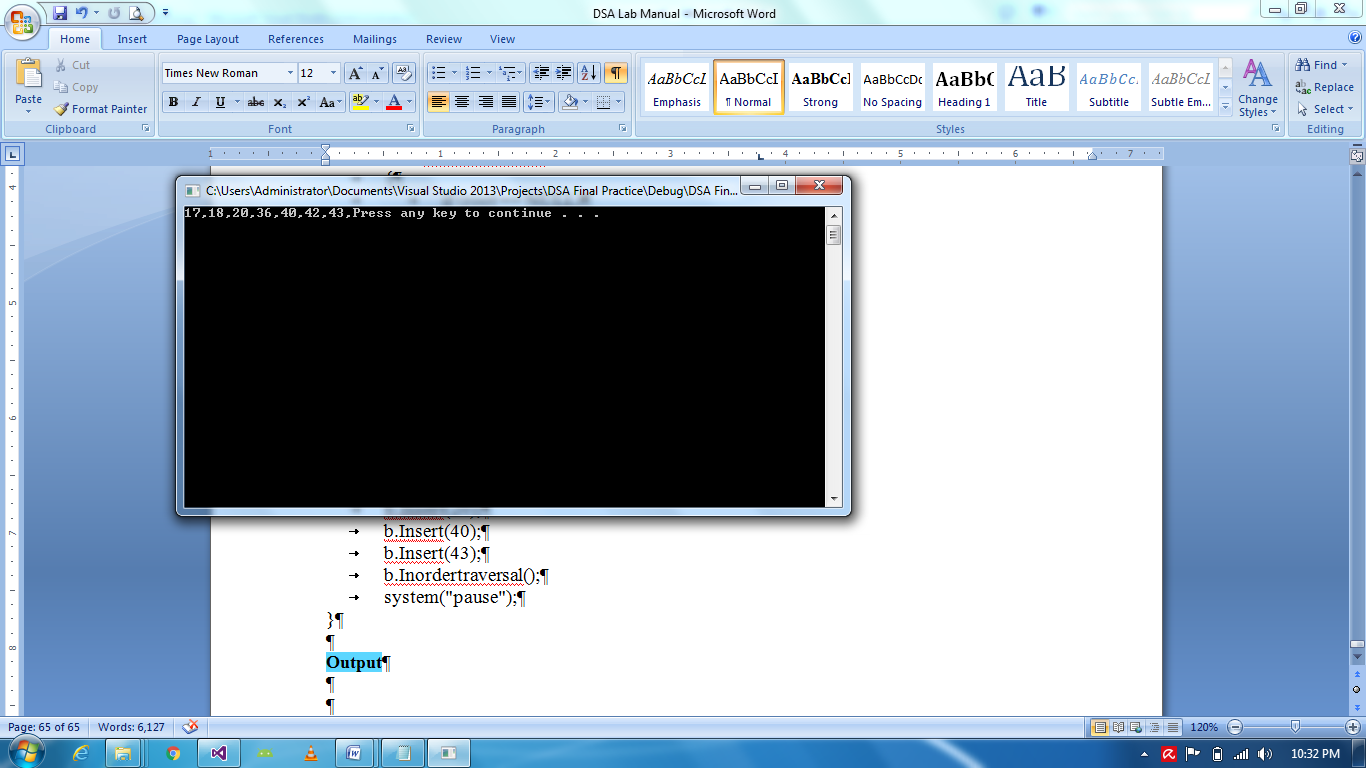
b.Insert(43);

b.Inordertraversal();

system("pause");

}

**Output**



**Task I: Tree Traversals:**

Code:

#include<iostream>

using namespace std;

struct Node

{

char Data;

Node \*Left = NULL;

Node \*Right = NULL;

};

Node \*NN(char Val)

{

Node \*n = new Node;

n->Data = Val;

n->Left = NULL;

n->Right = NULL;

return n;

};

void Post\_Order(Node \*n) // Left , Right, Root

{

if (n == NULL)

{

return;

}

else

{

Post\_Order(n->Left);

Post\_Order(n->Right);

cout << n->Data << " ";

}

}

void Pre\_Order(Node \*n) // Root, Left, Right

{

if (n == NULL)

{

return;

}

else

{

cout << n->Data << " ";

Pre\_Order(n->Left);

Pre\_Order(n->Right);

}

}

void In\_Order(Node \*n) // Left, Root, Right

{

if (n == NULL)

{

return;

}

else

{

In\_Order(n->Left);

cout << n->Data << " ";

In\_Order(n->Right);

}

}

void main()

{

Node \*Root = NN('A');

Root->Left = NN('B');

Root->Right = NN('C');

Root->Left->Left = NN('D');

Root->Left->Right = NN('E');

Root->Left->Left->Left = NN('F');

Root->Left->Left->Right = NN('G');

Root->Right->Left = NN('H');

Root->Right->Left->Left = NN('J');

Root->Right->Right = NN('I');

cout << "\n\nTree Traversals : " << endl << endl << endl;

cout << "\n\nPost Order of Tree is : " << endl;

Post\_Order(Root);

cout << "\n\nPre Order of Tree is : " << endl;

Pre\_Order(Root);

cout << "\n\nIn Order of Tree is : " << endl;

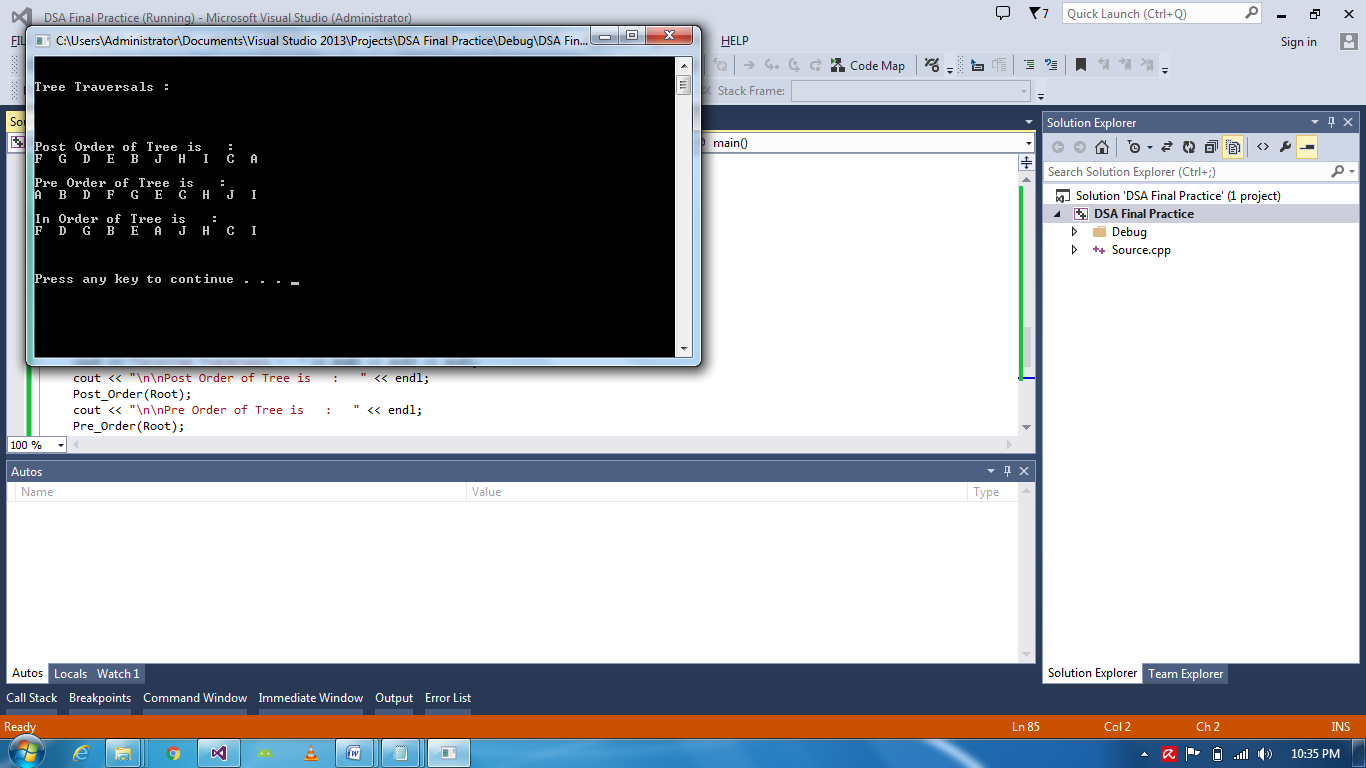
In\_Order(Root);

cout << "\n\n\n\n";

system("pause");

}

**Output**



**Task II: Displaying Maximum and Minimum value in BST:**

Code:

#include "iostream"

using namespace std;

struct Node

{

int nodevalue;

Node \* leftchild;

Node \* rightchild;

};

class BST

{

Node \* root;

void Link(Node \* linker, Node \*n)

{

if (n->nodevalue > linker->nodevalue)

{

if (linker->rightchild == NULL)

{

linker->rightchild = n;

return;

}

Link(linker->rightchild, n);

return;

}

else

{

if (linker->leftchild == NULL)

{

linker->leftchild = n;

return;

}

Link(linker->leftchild, n);

return;

}

}

void Display(Node \* n)

{

if (n == NULL)

return;

else

{

//cout<<n->nodevalue<<","; //PreOrder Tree Traversal

Display(n->leftchild);

cout << n->nodevalue << ","; //InOrder Tree Traversal

Display(n->rightchild);

//cout<<n->nodevalue<<","; //PostOrder Tree Traversal

}

}

public:

BST()

{

root = NULL;

}

void Insert(int val)

{

Node \* newnode = new Node;

newnode->nodevalue = val;

newnode->leftchild = NULL;

newnode->rightchild = NULL;

if (root == NULL)

root = newnode;

else

Link(root, newnode);

}

void Inordertraversal()

{

if (root == NULL)

cout << "No node";

else

Display(root);

}

void DisplayMin()

{

if (root == NULL)

cout << "No Node \n";

else

{

Node \*minval = root;

while (minval->leftchild != NULL)

minval = minval->leftchild;

cout << "Min val " << minval->nodevalue << endl;

}

}

void DisplayMax()

{

if (root == NULL)

cout << "No Node \n";

else

{

Node \*maxval = root;

while (maxval->rightchild != NULL)

maxval = maxval->rightchild;

cout << "Max val " << maxval->nodevalue << endl;

}

}

void DeleteRoot()

{

if (root == NULL)

cout << "Nothing to delete " << endl;

else if (root->leftchild == NULL && root->rightchild == NULL)

{

Node \* temp = root;

root = NULL;

temp = NULL;

delete temp;

}

else if (root->leftchild != NULL)

{

Node \*parent = root;

Node \* child = root->leftchild;

while (child->rightchild != NULL)

{

parent = child;

child = child->rightchild;

}

root->nodevalue = child->nodevalue;

parent->rightchild = child->leftchild;

}

else

{

Node \*parent = root;

Node \* child = root->rightchild;

while (child->leftchild != NULL)

{

parent = child;

child = child->leftchild;

}

root->nodevalue = child->nodevalue;

parent->leftchild = child->rightchild;

}

}

};

void main()

{

BST b;

b.Insert(36);

b.Insert(18);

b.Insert(17);

b.Insert(42);

b.Insert(20);

b.Insert(40);

b.Insert(43);

b.DisplayMin();

b.DisplayMax();

system("pause");

}

**Output**

