# Department of Computing

**Name: Mahum Samar**

**CMS ID : 290647**

**CS250: Data Structure and Algorithms**

**Class: BSCS 9B**

# Lab 1: Pointers & Array-based Implementation of Lists

**Part One: Revision of Pointers in C++**

**Introduction**

This lab is about pointers, memory occupied by variables, and dynamic vs static memory allocation.

**Objectives**

This lab will revise the concepts learnt by students in the previous semesters.

**Tools/Software Requirement**

Visual Studio C++, Eclipse C++ or any other IDE.

**Description**

Pointers are used to point towards a particular memory address. In this lab we will use the pointers and perform tasks with the help of them.

**Lab Tasks**

You are required to upload the lab tasks on the LMS and the name of that tasks must be in this format YourFullName\_reg#.cpp

Remember to comment your code properly. Inappropriate or no comment will result in the deduction of marks.

# **Task 1:** Write output of the following C++ codes in your document without executing it.

## Example code a)

int a;

int b;

int \*p=&a;

int \*q=&b;

a=20;

b=35;

p=q;

\*p=83;

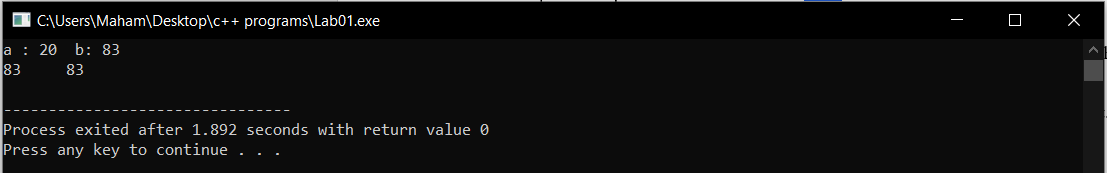
cout<<"a : "<<a<<" b: "<<b<<endl;

cout<<\*p<<" "<<\*q<<endl;

## Output:

a : 20 b: 83

83 83



## Example code b)

#include <iostream>

using namespace std;

void ExampleB(){

int x[4] = {0,4,6,9};

int \*p, a=3;

p=x; //pointer p points to the x[0]

(\*p)++; // the value at x[0] is incremented

cout<<\*p<<endl;

cout<<\*(p+1)<<endl;

p++; //pointer moves to the x[1]

\*p=\*p+a; //value at x[1] is incremented by 3

cout<<\*p<<endl;

p=p+2; //in this instruction the pointer moves to the x[3] location.

cout<<\*p<<endl;

}

int main()

{

ExampleB();

}

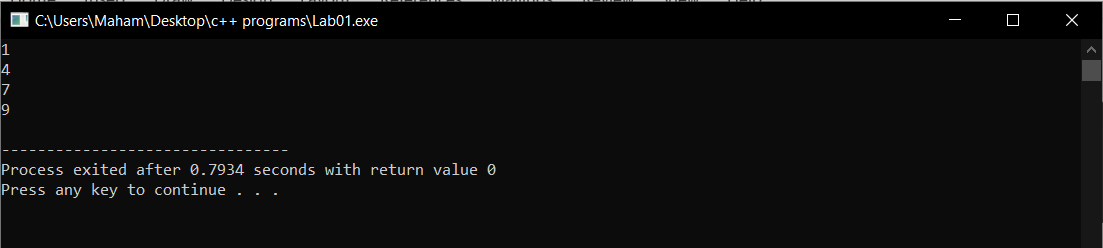
## Output:

1

4

7

9



## Example code c)

int a, \*p, \*q;

int arr[4]= {0};

p=arr;

q=p;

\*p=4;

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

a=\*p;

p++;

\*p=(a+i);

}

for (int j=0; j<4; j++){

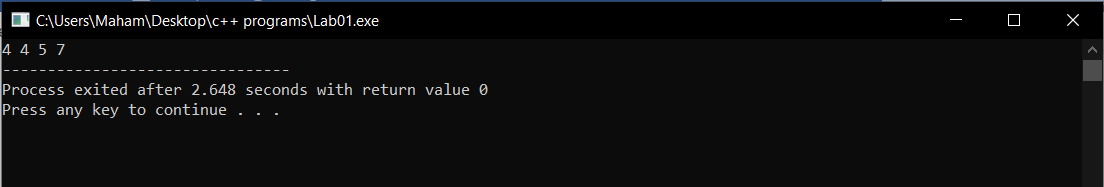
cout<<\*q<<" ";

q++;

}

## Output:

4 4 5 7



# Task 2:

int a=5, b=10;

int \*pa=&a; //pa and pb are pointer variables of type int.

int \*pb=&b;

int \*\*ppa=&pa; //ppa and ppb are called double pointers or pointers-to-pointers.

int \*\*ppb=&pb;

1. Write code of a function that swaps values of variables a and b. Input to the function should be the address of both the variables.

#include <iostream>

using namespace std;

void swapVariablesByAddress(int \*x, int \*y)

{

//this methods take the address of the variable a and b and swaps their values by using an additional temporary variable

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int main()

{

int a=5, b=10;

int \*pa=&a; //pa and pb are pointer variables of type int.

int \*pb=&b;

int \*\*ppa=&pa; //ppa and ppb are called double pointers or pointers-to-pointers.

int \*\*ppb=&pb;

cout << "Before swapping" << endl;

cout << "Value in a is " << \*pa <<endl;

cout << "Value in b is " << \*pb <<endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

swapVariablesByAddress(&a,&b);

cout << "After swapping" << endl;

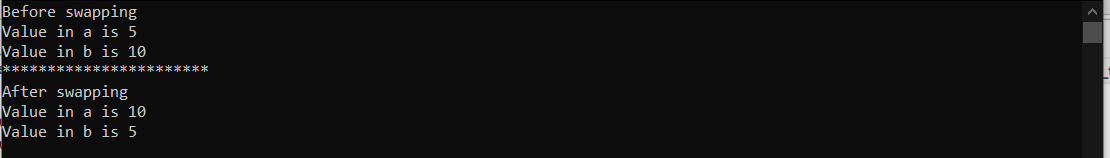
//displays the result on the screen.

cout << "Value in a is " << a <<endl;

cout << "Value in b is " << b <<endl;

}

## Output:



1. Write code of a function that swaps values of pointer variables pa and pb. Input to the function should be the address of both the pointer variables.

#include <iostream>

using namespace std;

void swapPointersByDoublePointers(int \*\*x, int \*\*y)

{

//this method takes the addresses of the pointer variables in the double pointers and then swaps the addresses in the pointer ..

//so that the pa points to variable b and pb points to variable a.

int \*temp = \*x;

\*x = \*y;

\*y = temp;

}

int main()

{

int a=5, b=10;

int \*pa=&a; //pa and pb are pointer variables of type int.

int \*pb=&b;

int \*\*ppa=&pa; //ppa and ppb are called double pointers or pointers-to-pointers.

int \*\*ppb=&pb;

cout << "Before swapping" << endl;

cout << "Value in pa is " << pa <<endl;

cout << "Value in bb is " << pb <<endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

swapPointersByDoublePointers(&pa,&pb);

cout << "After swapping" << endl;

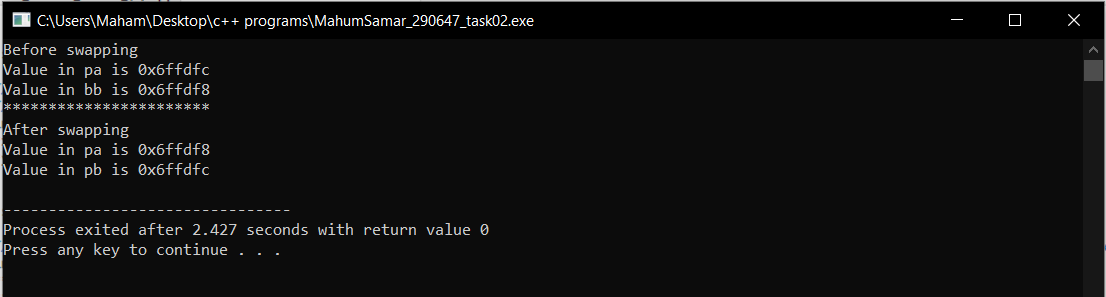
//displays the result on the screen.

cout << "Value in pa is " << pa <<endl;

cout << "Value in pb is " << pb <<endl;

}

## Output:



1. Write code of a function that swaps values of the variables a and b using pointer-to-pointer variables ppa and ppb.

#include <iostream>

using namespace std;

void swapVariablesByDoublePointer(int \*\*x, int \*\*y)

{

//this method thakes the double pointer addresses as an argument and swaps the values of the varables a and b by using the temporary variable.

int temp = \*\*x;

\*\*x = \*\*y;

\*\*y = temp;

}

int main()

{

int a=5, b=10;

int \*pa=&a; //pa and pb are pointer variables of type int.

int \*pb=&b;

int \*\*ppa=&pa; //ppa and ppb are called double pointers or pointers-to-pointers.

int \*\*ppb=&pb;

cout << "Before swapping" << endl;

cout << "Value in a is " << a <<endl;

cout << "Value in b is " << b <<endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

swapVariablesByDoublePointer(ppa,ppb);

cout << "After swapping" << endl;

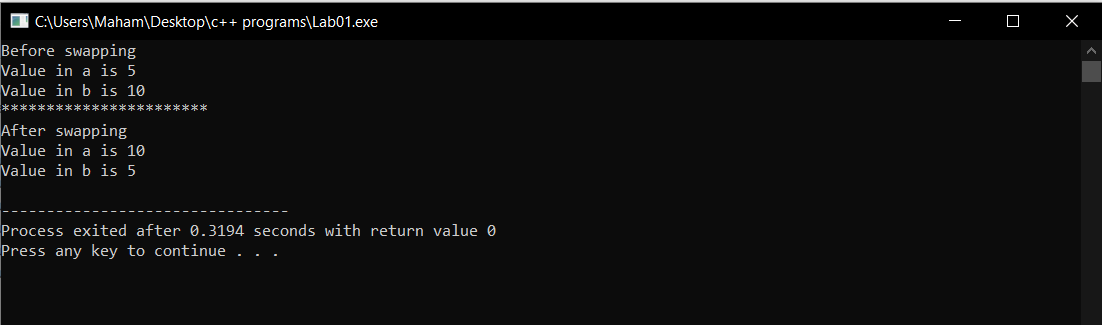
//displays the result on the screen.

cout << "Value in a is " << a <<endl;

cout << "Value in b is " << b <<endl;

}

## Output:



# Task 3:

int list[5]={3,6,9,12,15};

int \*pArr= list;

Your task is to write a piece of code that prints all values stored in the array **list** using only pointer variable pArr. Do not use the conventional way of printing values by numbering indexes.

#include <iostream>

using namespace std;

int main()

{

int list[5]={3,6,9,12,15};

int \*pArr= list; //pArr is the pointer variable to the array list and it is pointing at the 1st index of the array.

int size = sizeof(list)/sizeof(list[0]); //this instruction calculates the number of elements of the array.

for(int i=0 ; i < size ; i++) //this loop prints all the values in the array

{

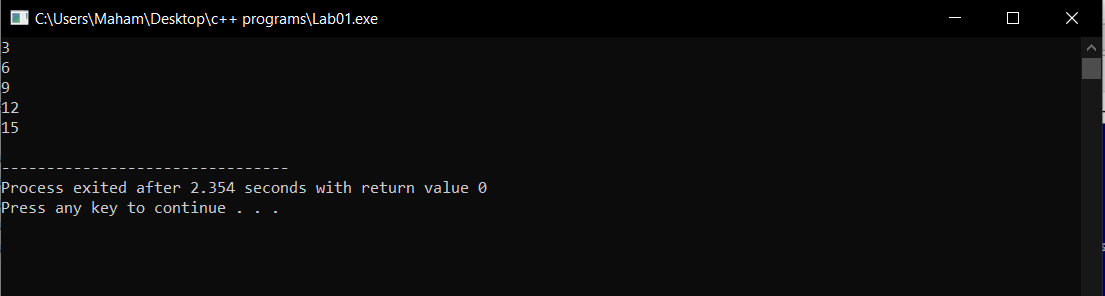
cout << \*pArr << endl; //print element on the screen

pArr++; //increments the pointer location through the array until the last element is reached.

}

}

## Output:



# **Task 4:** Write a code to find the memory in bytes occupied by int, long, double, float and char.

#include <iostream>

using namespace std;

int main()

{

//this instruction displays the size of int data type by using the method "sizeof() ie 4 bytes"

cout << "Size of int " << sizeof(int) << " byte " << endl;

//this instruction displays the size of long data type by using the method "sizeof() ie 4 bytes"

cout << "Size of long " << sizeof(long) << " byte " << endl;

//this instruction displays the size of double data type by using the method "sizeof() ie 8 bytes"

cout << "Size of double " << sizeof(double) << " byte " << endl;

//this instruction displays the size of float data type by using the method "sizeof() ie 4 bytes"

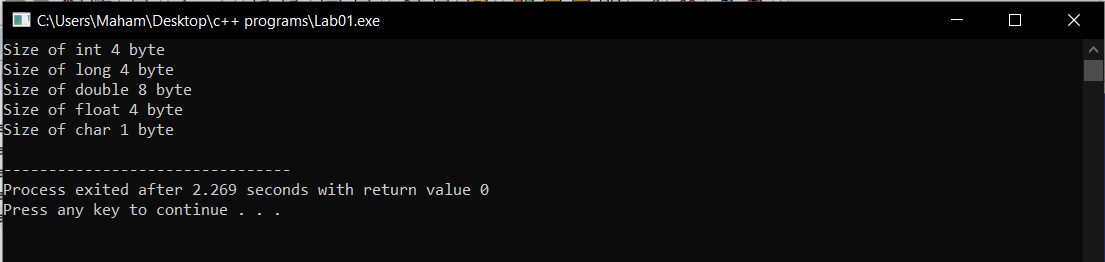
cout << "Size of float " << sizeof(float) << " byte " << endl;

//this instruction displays the size of char data type by using the method "sizeof() ie 1 byte"

cout << "Size of char " << sizeof(char) << " byte " << endl;

}

## Output:



**Part Two: Static vs Dynamic Arrays**

Consider two variants of declaring arrays below. Memory for the first variant gets allocated on the Stack. The lifetime of an array created using the method A depends on its scope. If it is defined globally, its life is equal to the lifetime of the application. If it is declared in a function, memory for it gets allocated on the stack when the function gets called. It gets deallocated when the function call terminates. All the data related to the function call including the array gets removed from the stack. On the other hand, memory for the array created using new operator gets allocated on the heap at runtime. The lifetime of such an array is at max equal to the execution time of the application. If the array is no more required, the memory allocated for it can be freed using **delete []** command.

# Method A:

const int size;

cout << "Enter size of array: ";

cin >> size; //In c++, cin command is used to take input from users.

int x[size];

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

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

In this method the static array of size “size” is created. And filled with the elements in the array.

# **Method B**

int size; // Note that size variable is const in variant A whereas it isn’t in variant B. Find out the logic behind it.

cout << "Enter size of array: ";

cin >> size;

int \*x = new int[size];

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

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

This is the dynamic array and here the size is not constant because dynamic array size can be increased and decreased at runtime and constant value cannot be changed. Hence to change the size of array “size” variable can be changed and hence array size be changed.

**Part Three: Array-based Implementation of List Data Structure**

**Introduction**

We have studied List as Abstract Data Type in the class. We have discussed various operations that can be performed on a list. The objective of this lab is to implement an array-based list. First, you shall implement static array-based list which is not resizable. Then, you shall implement a dynamic version of an array-list; you shall allocate memory for array at runtime using the **new** operator.

Your task is to implement all the following operations.

1. bool IsEmpty() operation. It returns true value if the list is empty. Otherwise, false.
2. bool isFull() function. It returns true value if the list is full. False otherwise.
3. InsertSorted(value) a value at its logical position in a list.
4. CreateRoom(int pos); If a new value has to be inserted into a list at position pos, this function creates room for it, by shifting all values from position pos till last one index to the right.
5. FillGap(int pos); If an existing value has to be deleted from position pos, this function fills the gap by shifting all values from pos till last element to the left side.
6. DynamicExpansion( ); If an array-list is already full, this function dynamically creates a new array of size greater than the current size of the array, copies all contents from the older array to the new one, updates the pointer which stores the address of an array, and lastly deletes the older array using **delete** ptr[] command.
7. DynamicReduction(); If the size of a list falls below 50% capacity of the array after a deletion operation, this function dynamically creates a new array of size smaller than the current size of the existing array, copies all contents from the older array to the new one, updates the pointer that stores the address of an array, and lastly deletes the older bigger array using **delete** ptr[] command.

# **Task 1:** Implement a version of an array-list in which the array is static.

/\*#include <iostream>

using namespace std;

const int sizeOfArray = 50; //the maximum size of the array

struct staticArrayList{

int lengthOfList = 0; //variable to store the number of list elements

int array[sizeOfArray];

bool isEmpty()

{

//method to check if the array is empty or not

return lengthOfList == 0;

}

bool isFull()

{

//method to check if the array is full or not

return lengthOfList == sizeOfArray;

}

void insertAtPosition(int value,int pos)

{

//method to insert the value at the specific location given by the user.

for(int i = 0 ; i <= lengthOfList -1 ; i++)

{

if (array[i] == value)

{

cout << "This value already exists." << endl;

return;

}

}

if((pos <= lengthOfList+1) && (pos > 0) && !isFull())

{

for(int i = lengthOfList ; i >= (pos -1); i--)

{

array[i + 1] = array[i];

}

array[pos-1] = value;

lengthOfList++;

}

else

{

cout << "The position entered is not valid." << endl;

}

}

void insertSorted(int value)

{

//method that inserts the value in the array

for(int i = 0 ; i <= lengthOfList -1 ; i++)

{

if (array[i] == value)

{

cout << "This value already exists." << endl;

return;

}

}

int pos = 0;

bool found = false;

for(int i = 0 ; i <= lengthOfList - 1 ; i++)

{

if ((array[i] > value))

{

pos = i;

found = true;

break;

}

}

if(found)

{

for(int i = lengthOfList - 1 ; i > pos - 1; i--)

{

array[i + 1] = array[i];

}

array[pos] = value;

lengthOfList++;

}

}

void printList()

{

//method that prints the elements of the array

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

{

cout << array[i] << endl;

}

}

void DeleteValue(int value)

{

//method to delete the value specified by the user if the value is present in the array

if(!isEmpty())

{

int pos =0;

bool found = false;

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

{

if (array[i] == value)

{

pos = i;

found = true;

break;

}

}

if (found)

{

for (int i = pos ; i < lengthOfList ; i++)

{

array[i] = array[i+1];

}

lengthOfList--;

}

}

}

void DeleteValueAtPosition(int pos)

{

// deletes a value if a user enters a valid position (1 <= pos <= size of a list.)

//this method deletes the value at the position specified by the user.

if((pos < lengthOfList) && (pos > 0) && (!isEmpty()))

{

for(int i = pos-1 ; i < lengthOfList+1 ; i++)

{

array[i] = array[i+1];

}

lengthOfList--;

}

}

};

int main()

{

staticArrayList A;

A.insertAtPosition(1,1); //Insert value at position 1.

A.insertAtPosition(2,2); //Insert value at position 2.

A.insertAtPosition(4,3); //Insert value at position 3.

cout << "Printing array after inserting the values." << endl ;

A.printList();

cout << "Deleting the value 2 and printing array." << endl ;

A.DeleteValue(2);

A.printList();

cout << "Inserting value 3 at its logical position and printing array." << endl ;

A.insertSorted(3);

A.printList();

cout << "Deleting the value at position 2 and printing array." << endl ;

A.DeleteValueAtPosition(2);

A.printList();

if(A.isEmpty())

{

cout << "Array is empty." << endl;

}

else

{

cout << "Array is not empty." << endl;

}

if(A.isFull())

{

cout << "Array is full." << endl << endl;

}

else

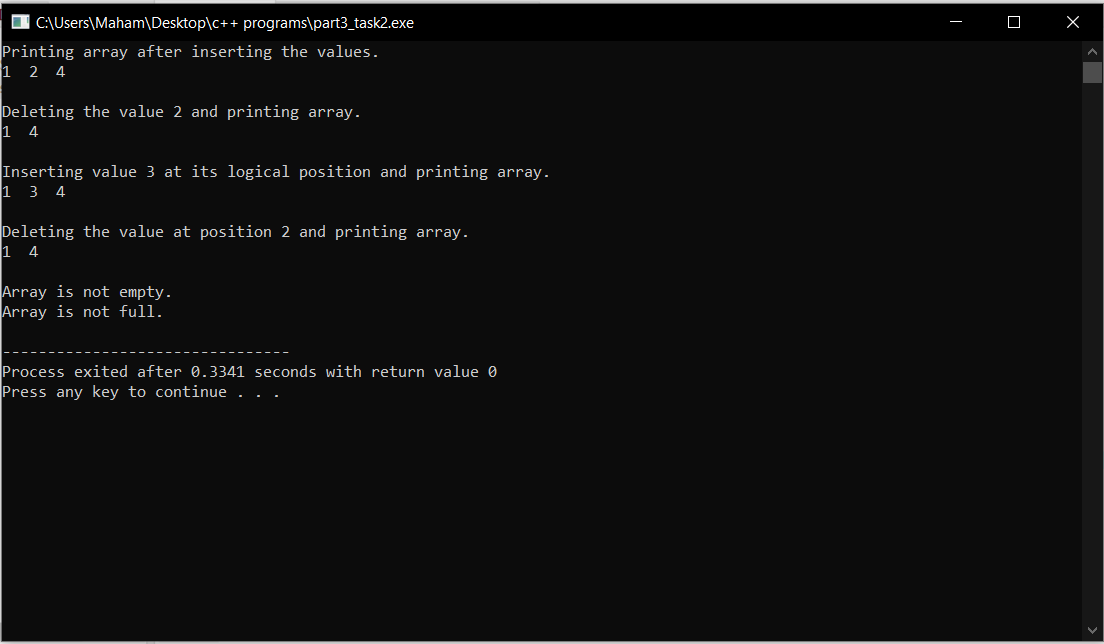
{

cout << "Array is not full." << endl;

}

}

## Output:



# **Task 2:** Implement a version of an array-list in which the array is dynamic. Memory for an array of capacity *size*can be allocated at runtime using new operator. It can be deallocated using **delete** operator. Moreover, array-list can be resized if it becomes full, or the number of elements in it fall below a certain threshold.

#include <iostream>

using namespace std;

int sizeOfArray = 50; //the maximum size of the array

struct DynamicArrayList{

int lengthOfList = 0; //variable to store the number of list elements

int \*pArrayList = new int [sizeOfArray];

bool isEmpty()

{

//method to check if the array is empty or not

return lengthOfList == 0;

}

bool isFull()

{

//method to check if the array is full or not

return lengthOfList == sizeOfArray;

}

void insertAtPosition(int value, int pos)

{

DynamicExpansion();

//method to insert the value at the specific location given by the user.

if((pos <= lengthOfList+1) && (pos > 0) && !isFull())

{

for(int i = lengthOfList ; i >= (pos -1); i--)

{

pArrayList[i + 1] = pArrayList[i];

}

pArrayList[pos-1] = value;

lengthOfList++;

}

else

{

cout << "The position entered is not valid." << endl;

}

}

void insertSorted(int value)

{

DynamicExpansion();

//method that inserts the value in the pArrayList

for(int i = 0 ; i <= lengthOfList -1 ; i++)

{

if (pArrayList[i] == value)

{

cout << "This value already exists." << endl;

return;

}

}

int pos = 0;

bool found = false;

for(int i = 0 ; i <= lengthOfList - 1 ; i++)

{

if ((pArrayList[i] > value))

{

pos = i;

found = true;

break;

}

}

if(found)

{

for(int i = lengthOfList - 1 ; i > pos - 1; i--)

{

pArrayList[i + 1] = pArrayList[i];

}

pArrayList[pos] = value;

lengthOfList++;

}

}

void printList()

{

//method that prints the elements of the pArrayList

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

{

cout << pArrayList[i] << " ";

}

cout << endl << endl;

}

void DeleteValue(int value)

{

DynamicReduction();

//method to delete the value specified by the user if the value is present in the pArrayList

if(!isEmpty())

{

int pos =0;

bool found = false;

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

{

if (pArrayList[i] == value)

{

pos = i;

found = true;

break;

}

}

if (found)

{

for (int i = pos ; i < lengthOfList ; i++)

{

pArrayList[i] = pArrayList[i+1];

}

lengthOfList--;

}

}

}

void DeleteValueAtPosition(int pos) // deletes a value if a user enters a valid position (1 <= pos <= size of a list)

{

DynamicReduction();

// deletes a value if a user enters a valid position (1 <= pos <= size of a list.)

//this method deletes the value at the position specified by the user.

if((pos < lengthOfList) && (pos > 0) && (!isEmpty()))

{

for(int i = pos-1 ; i < lengthOfList+1 ; i++)

{

pArrayList[i] = pArrayList[i+1];

}

lengthOfList--;

}

}

void DynamicExpansion() //Increases size of the pArrayList when the list becomes full.

{

if(lengthOfList == (sizeOfArray - 1))

{

sizeOfArray = 2 \* sizeOfArray;

int \*ptempArray = new int[sizeOfArray];

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

{

ptempArray[i] = pArrayList[i];

}

delete[] pArrayList;

pArrayList = ptempArray;

}

}

void DynamicReduction() //decreases size of the array when the size of a list becomes less than 50% after deletion operation

{

if(lengthOfList == 1/2 \*(sizeOfArray - 1))

{

sizeOfArray = 1/2 \* sizeOfArray;

int \*ptempArray = new int[sizeOfArray];

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

{

ptempArray[i] = pArrayList[i];

}

delete[] pArrayList;

pArrayList = ptempArray;

}

}

};

int main()

{

DynamicArrayList A;

A.insertAtPosition(1,1); //Insert value at position 1.

A.insertAtPosition(2,2); //Insert value at position 2.

A.insertAtPosition(4,3); //Insert value at position 3.

cout << "Printing array after inserting the values." << endl ;

A.printList();

cout << "Deleting the value 2 and printing array." << endl ;

A.DeleteValue(2);

A.printList();

cout << "Inserting value 3 at its logical position and printing array." << endl ;

A.insertSorted(3);

A.printList();

cout << "Deleting the value at position 2 and printing array." << endl ;

A.DeleteValueAtPosition(2);

A.printList();

if(A.isEmpty())

{

cout << "Array is empty." << endl;

}

else

{

cout << "Array is not empty." << endl;

}

if(A.isFull())

{

cout << "Array is full." << endl << endl;

}

else

{

cout << "Array is not full." << endl;

}

}

## Output:

