Experiment No. 02

Pointers and Dynamic Arrays

**OBJECTIVE**:

Things that will be covered in today’s lab:

* Pointers
* Dynamic Arrays

**THEORY**:

**Pointer variable:** A variable whose content is an address (i.e., a memory address) In C++, you declare a pointer variable by using the asterisk symbol (\*) between the datatype and the variable name. The general syntax to declare a pointer variable is as follows:

datatype \* identifier;

In C++, the *ampersand* (&), address of the operator, is a unary operator that returns the address of its operand. Similarly, “\*” is a dereferencing operator, refers to the object to which its operand (pointer) points. For example, given the statements:

int x;

int \*p;

p = &x; // assigns address of x to p

cout<< \*p <<endl; // pointer p points to x

The arrays discussed in last lab are called **static arrays** because their size was fixed at compile time. One of the limitations of a static array is that every time you execute the program, the size of the array is fixed. One way to handle this limitation is to declare an array that is large enough to process a variety of data sets. However, if the array is very big and the data set is small, such a declaration would result in memory waste. On the other hand, it would be helpful if, during program execution, you could prompt the user to enter the size of the array and then create an array of the appropriate size.

**Dynamic Array:** An array created during the execution of a program**.** To create a dynamic array, we use *new* operator.

int size;

int \*p;

p = newint [size];

If you are not in need of dynamically allocated memory anymore, you can use *delete* operator, which de-allocates memory previously allocated by new operator.

delete [] p;

**For all of the exercises below save your code on the learning management system (LMS) and give the screen shot of the output you get on the console in the space provided after every exercise.**

**Exercise 1:** (**10 points)**

Write a function **resize()** that takes as arguments: *a pointer pointing to the array of integers,* its *size,* and *new\_size which* can be any number greater than 0. This function should change the size of the array. If the new size is greater than the previous one, then insert zeroes in new cells.

Example:

**Case 1: (new\_size> size)**

new\_size=7, size=5

Before calling resize function:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array => | 2 | 32 | 4 | 34 | 51 |

After calling resize function:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Array => | 2 | 32 | 4 | 34 | 51 | 0 | 0 |

**Case 2: (new\_size<size)**

new\_size=3, size=5

Before calling resize function:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array => | 2 | 32 | 4 | 34 | 51 |

After calling resize function:

|  |  |  |  |
| --- | --- | --- | --- |
| Array => | 2 | 32 | 4 |

**Exercise 2:** (**10 points)**

Implement all the functions called in the following main which creates two dynamic arrays of sizes *size\_1* and *size\_2* respectively. Take input in these arrays from the user. Now create a third array of *size* (*size\_1+size\_2*) and insert all the elements of both arrays in this array. Remove the duplicate elements from this array and resize the array to a smaller size (you may use the function you wrote in exercise1).

Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array 1=> | 10 | 2 | 13 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array 2=> | 3 | 4 | 7 | 6 | 7 |

After merging Array1 and Array2:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array 3=> | 10 | 2 | 13 | 4 | 3 | 4 | 7 | 6 | 7 |

After removing duplicate elements, this array should be of size 6:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Array 3=> | 10 | 2 | 13 | 4 | 3 | 7 | 6 |

**You may test your code using the following main function:**

**int main()**

**{**

**int size1, size2, size3;**

**cin>>size1>>size2; size3=size1+size2;**

**int \*array1= new int[size1];**

**int \*array2= new int[size2];**

**GetData(array1, size1);**

**GetData(array2, size2);**

**int \*array3= Merge(array1, size1, array2, size2];**

**DisplayData(array3, size3);**

**int nSize=RemoveDuplicates(array3, size3);// returns new size of the array**

**DisplayData(array3, nSize);**

**delete [] array1; delete [] array2; delete [] array3;**

**return 0;**

**}**

**In the space below please give the screenshot of the output you get after running your code**

**Exercise 3:**  (**10 points)**

Consider following main function and implement the ReverseSentence function

void main()

{

char input[100];

cin.getline(input,100);

//For example, user enters National library.

char \*rev=ReverseSentence(input);

cout<<rev<<endl;

// The data pointed to by rev should be yrarbil lanoitaN.

delete [] rev;

}

The length of the character array pointed to by rev should be as per the size of the sentence entered by the user.

**In the space below please give the screenshot of the output you get after running your code**