**Faculty of Computers and Artificial Intelligence**

**Cairo University**

#### 

#### Structured Programming

**Final Term Project**

#### Submitted by :

|  |  |
| --- | --- |
| **ID** | **Name** |
| **20196020** | **دعاء جمال صالح درويش** |
| **20196025** | **سلمى ايمن شرف الدين** |

**June 2020**

**Part I :**

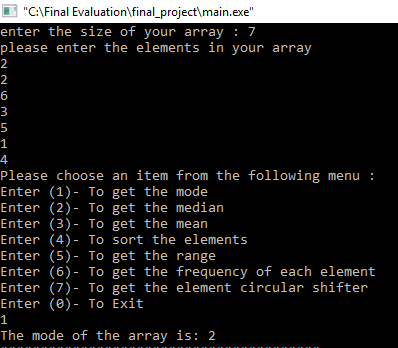
**Problem 2 :Statistical calculator**

1. **Mode Function**

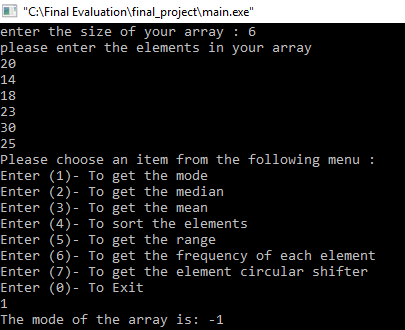
* **Input and output of the function**

|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case 1 | 7 | 2 2 6 3 5 1 4 | 2 |
| Test case 2 | 6 | 20 14 18 23 30 25 | -1 |

**Test case 1**

****

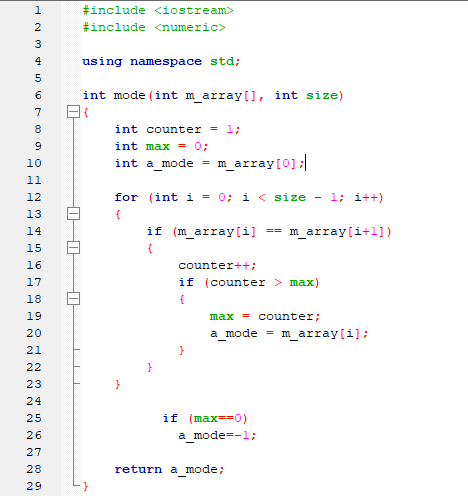
**Test case 2**



* **Description of the function**

The mode of a set of values is the most repeated value. The functions takes the array’s elements and size as arguments. We firstly assigned the variable of number of repetitions with zero (**max=0**) then assumed that the array’s mode (**a\_mode**) is equal to the first element in the array to perform a for loop where each element in the array is compared to the element that is after it to check if they are equal and if they are then the counter increases by 1 and put the counter value in the variable **max** (**number of repetitions**) and that repeated element is then placed in **a\_mode** as the array’s mode that gets returned at the end after checking the whole array to know the most repeated.

And if there is no element that gets repeated and max=0 doesn’t change then the array’s mode will return as -1 .

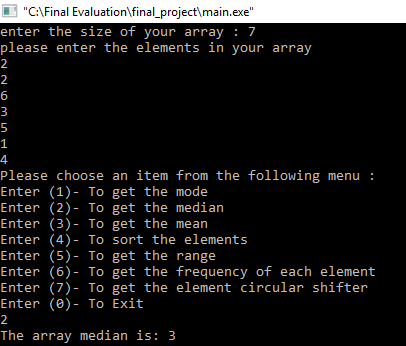


1. **Median function**

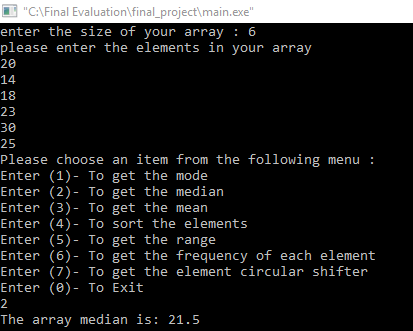
* **Input and output of the function**

|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case 1 | 7 | 2 2 6 3 5 1 4 | 3 |
| Test case 2 | 6 | 20 14 18 23 30 25 | 21.5 |

**Test case 1**

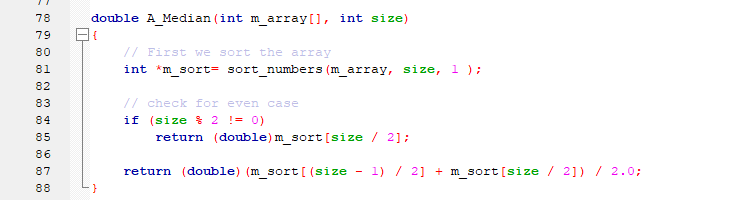


**Test case 2**



* **Description of the function**

The median is the middle value of a sorted set of values in ascending or descending order, the function takes the elements and size as arguments. If size of array is an even number, then the median is the average of the two middle values. After we sort the array using the sorting function and assign a pointer to the sorted array, we check if the array size entered is odd or even, if it’s odd then the function returns the middle value present in the sorted array, and if it’s even it returns the average of the two middle values.

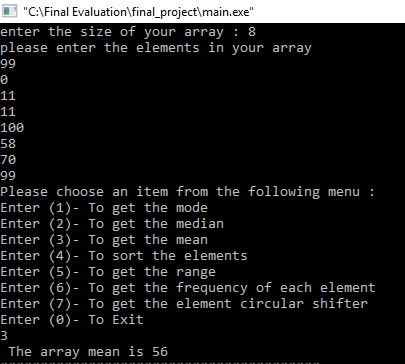


1. **Mean function**

* **Input and output of the function**

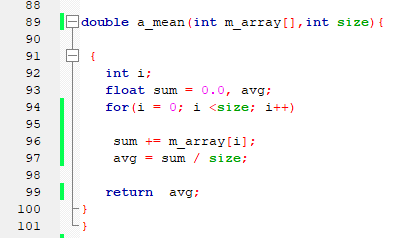
|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case | 8 | 99 0 11 11 100 58 70 99 | 56 |

**Test case**



* **Description of the function**

This function takes the elements and size as arguments and returns the array’s mean which is the average (**avg**) of the array of the elements by adding all the elements, saving them in the variable **sum** then dividing over the **size.**

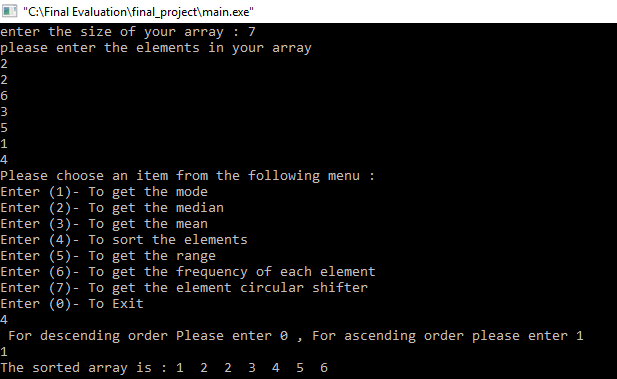


1. **Sort function**

* **Input and output of the function**

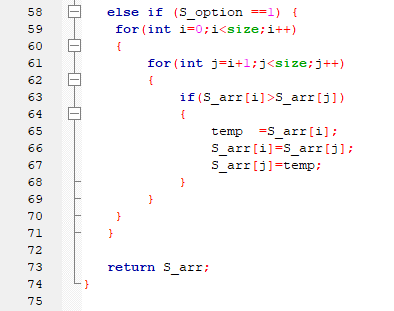
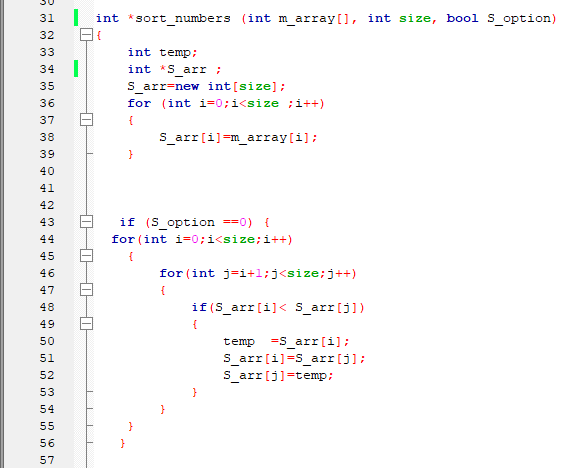
|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case | 7 | 2 2 6 3 5 1 4 | 1 2 2 3 4 5 6 |

**Test case**



* **Description of the function**

This function sorts the array in 2 modes, ascending and descending, by taking the elements, size and mode as arguments. Since the function returns multiple values (sorted array **S\_array**) we declare a pointer using dynamic memory allocation where we save the array’s elements. Then, based on the user’s choice ( Ascending / Descending) the array is sorted using for loops to re-arrange the elements by saving them in temp temporarily to be returned then printed.

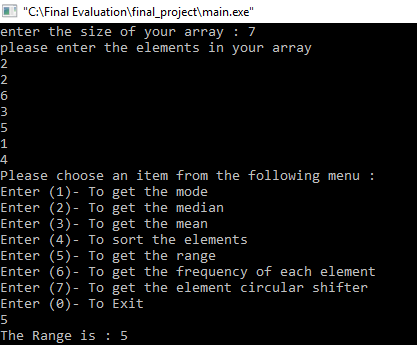


1. **Range Function**

* **Input and output of the function**

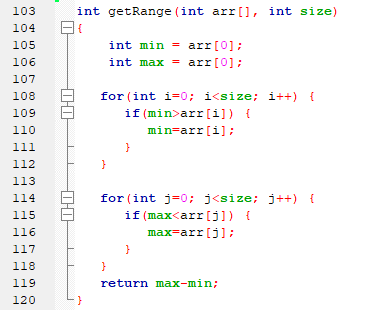
|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case | 7 | 2 2 6 3 5 1 4 | 5 |

**Test case**



* **Description of the function**

This function returns the Range which is the difference between the highest and lowest elements in the array. It takes the elements and size as arguments. We first assume the **min** and **max** value is equal to the first element and by using for loops to compare array’s elements, the **max** and **min** get updated to their actual values to later return their difference ( Range).

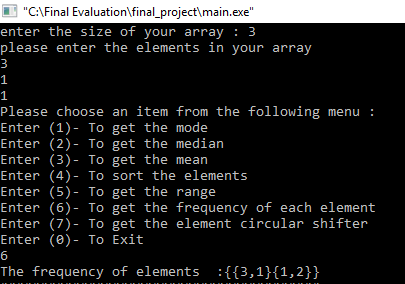


1. **Element Frequency Function**

* **Input and output of the function**

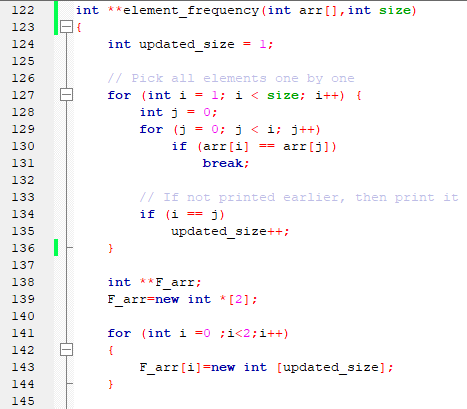
|  |  |  |
| --- | --- | --- |
| **Array size** | **Array elements** | **output** |
| Test case | 3 | 3 1 1 | {{3,1},{1,2}} |

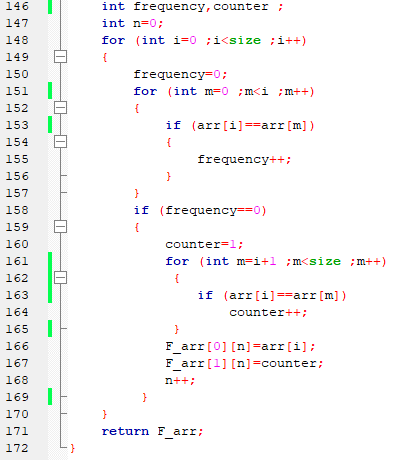
**Test case**

****

* **Description of the function**

This function counts the frequency of each element in the array and returns a 2D array with the results. It takes the elements and size as arguments. The **updated\_size** declared with value 1 at the beginning is the size of the unique elements ( elements with no frequency) and its value is decided through the for loop that checks for repetitions then the 2D pointer array is which will be returned declared (**F\_arr**) using dynamic memory allocation with a size of the unique elements. Then each element is checked to look if it matches the second element and increase the frequency variable when it’s found to be repeated, otherwise it enters the second loop where **counter=1** and the element is then checked again with the elements to save the number of repetitions in counter to finally return **F\_arr** .



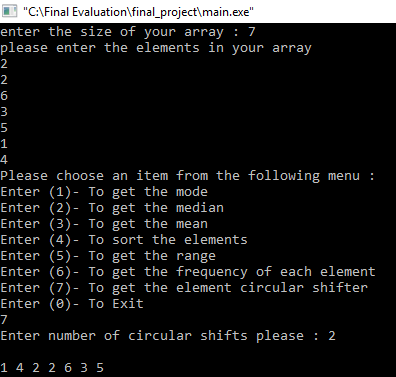


1. **Element Circular Shifter Function**

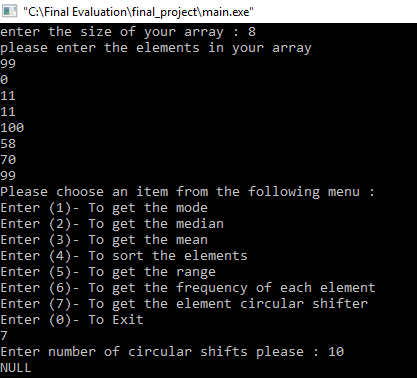
* **Input and output of the function**

|  |  |  |  |
| --- | --- | --- | --- |
| **Array size** | **Array elements** | **Number of shifts** | **output** |
| Test case 1 | 7 | 2 2 6 3 5 1 4 | 2 | 1 4 2 2 6 3 5 |
| Test case 2 | 8 | 99 0 11 11 100 58 70 99 | 10 | Null |

**Test case 1**

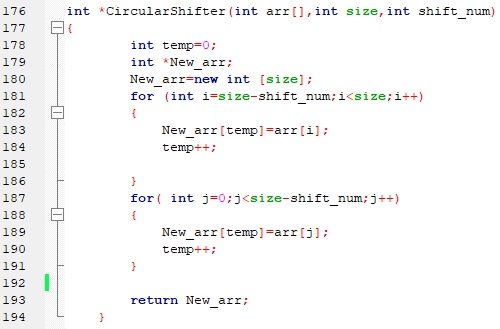


**Test case 2**



* **Description of the function**

This function makes a circular right shift of N elements, it takes the array’s size , elements and number of shifts as arguments (**shift\_num**) and then shifts the N elements to the right and inserts them from the left and returns a pointer to the new array (**New\_arr**)

****

**Part II :**

**Problem 2 :Statistical calculator (code text)**

#include <iostream>

#include <numeric>

using namespace std;

int mode(int m\_array[], int size)

{

int counter = 1;

int max = 0;

int a\_mode = m\_array[0];

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

{

if (m\_array[i] == m\_array[i+1])

{

counter++;

if (counter > max)

{

max = counter;

a\_mode = m\_array[i];

}

}

}

if (max==0)

a\_mode=-1;

return a\_mode;

}

int \*sort\_numbers (int m\_array[], int size, bool S\_option)

{

int temp;

int \*S\_arr ;

S\_arr=new int[size];

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

{

S\_arr[i]=m\_array[i];

}

if (S\_option ==0) {

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

{

for(int j=i+1;j<size;j++)

{

if(S\_arr[i]< S\_arr[j])

{

temp =S\_arr[i];

S\_arr[i]=S\_arr[j];

S\_arr[j]=temp;

}

}

}

}

else if (S\_option ==1) {

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

{

for(int j=i+1;j<size;j++)

{

if(S\_arr[i]>S\_arr[j])

{

temp =S\_arr[i];

S\_arr[i]=S\_arr[j];

S\_arr[j]=temp;

}

}

}

}

return S\_arr;

}

double A\_Median(int m\_array[], int size)

{

// First we sort the array

int \*m\_sort= sort\_numbers(m\_array, size, 1 );

// check for even case

if (size % 2 != 0)

return (double)m\_sort[size / 2];

return (double)(m\_sort[(size - 1) / 2] + m\_sort[size / 2]) / 2.0;

}

double a\_mean(int m\_array[],int size){

{

int i;

float sum = 0.0, avg;

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

sum += m\_array[i];

avg = sum / size;

return avg;

}

}

int getRange(int arr[], int size)

{

int min = arr[0];

int max = arr[0];

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

if(min>arr[i]) {

min=arr[i];

}

}

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

if(max<arr[j]) {

max=arr[j];

}

}

return max-min;

}

int \*\*element\_frequency(int arr[],int size)

{

int updated\_size = 1;

// Pick all elements one by one

for (int i = 1; i < size; i++) {

int j = 0;

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

if (arr[i] == arr[j])

break;

// If not printed earlier, then print it

if (i == j)

updated\_size++;

}

int \*\*F\_arr;

F\_arr=new int \*[2];

for (int i =0 ;i<2;i++)

{

F\_arr[i]=new int [updated\_size];

}

int frequency,counter ;

int n=0;

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

{

frequency=0;

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

{

if (arr[i]==arr[m])

{

frequency++;

}

}

if (frequency==0)

{

counter=1;

for (int m=i+1 ;m<size ;m++)

{

if (arr[i]==arr[m])

counter++;

}

F\_arr[0][n]=arr[i];

F\_arr[1][n]=counter;

n++;

}

}

return F\_arr;

}

int \*CircularShifter(int arr[],int size,int shift\_num)

{

int temp=0;

int \*New\_arr;

New\_arr=new int [size];

for (int i=size-shift\_num;i<size;i++)

{

New\_arr[temp]=arr[i];

temp++;

}

for( int j=0;j<size-shift\_num;j++)

{

New\_arr[temp]=arr[j];

temp++;

}

return New\_arr;

}

int main()

{

cout << "enter the size of your array : ";

int e\_number;

cin>>e\_number;

int i;

// int temp = 0;

int array[e\_number];

cout<<"please enter the elements in your array"<<endl;

for (i = 0; i < e\_number; i++) {

cin >> array[i];

}

label2:

cout<<"Please choose an item from the following menu : "<<endl;

cout<<"Enter (1)- To get the mode"<<endl;

cout<<"Enter (2)- To get the median"<<endl;

cout<<"Enter (3)- To get the mean"<<endl;

cout<<"Enter (4)- To sort the elements"<<endl;

cout<<"Enter (5)- To get the range"<<endl;

cout<<"Enter (6)- To get the frequency of each element"<<endl;

cout<<"Enter (7)- To get the element circular shifter"<<endl;

cout<<"Enter (0)- To Exit"<<endl;

int choice;

while (choice!=0)

{

cin>>choice;

switch(choice)

{

case 1:

{

cout << "The mode of the array is: " << mode(array,e\_number) << endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 2:

{

cout << "The array median is: " << A\_Median(array,e\_number) << endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 3:

{

cout <<" The array mean is "<<a\_mean(array,e\_number)<<endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 4:

{

bool option ;

cout << " For descending order Please enter 0 , For ascending order please enter 1" << endl;

cin>> option;

int \*sort= sort\_numbers(array, e\_number, option );

cout <<"The sorted array is : " ;

for (int i=0 ;i<e\_number ;i++ )

{

cout << sort[i] <<" " ;

}

cout<<endl ;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 5:

{

cout <<"The Range is : "<<getRange(array,e\_number)<<endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 6:

{

//first we repeat the step of changing the size of the array based on the number of unique elements present in the array to pass that size to the result\_array

cout <<"The frequency of elements :";

int updated\_size = 1;

// Pick all elements one by one

for (int i = 1; i < e\_number; i++)

{

int j = 0;

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

if (array[i] == array[j])

break;

// If not printed earlier, then print it

if (i == j)

updated\_size++;

}

int \*\*result\_array;

result\_array=new int\*[2];

for (int i=0;i<2;i++)

{

result\_array[i]=new int [updated\_size];

}

result\_array=element\_frequency(array,e\_number);

cout <<"{";

for (int i=0;i<updated\_size;i++)

{

cout <<"{"<<result\_array[0][i]<<","<<result\_array[1][i]<<"}";

}

cout <<"}"<<endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 7:

{

int shift\_argument;

int \*Shifted\_array;

cout<<"Enter number of circular shifts please : ";

cin>>shift\_argument;

if (shift\_argument<0||shift\_argument>e\_number)

{

cout<<"NULL";

}

else

{

Shifted\_array= CircularShifter(array,e\_number,shift\_argument);

for (int i =0 ;i<e\_number ;i++ )

{

cout <<Shifted\_array[i]<<" ";

}

}

cout<<endl;

cout<<"########################################"<<endl;

goto label2;

break;

}

case 0:

{

cout<<"Thank you for using our program"<<endl;

continue;

default:cout<<"Your choice is invalid"<<endl;

break;

}

}

}

return 0;

}