Program 1

Problem Statement:

Implement two stacks sharing the same array. Stacks must include the following fuctions: adding, and removing elements, listing elements, checking is stack full or empty. Find the maximum and minimum elements of the stack and swap them (do not brake LIFO principle).

Code:

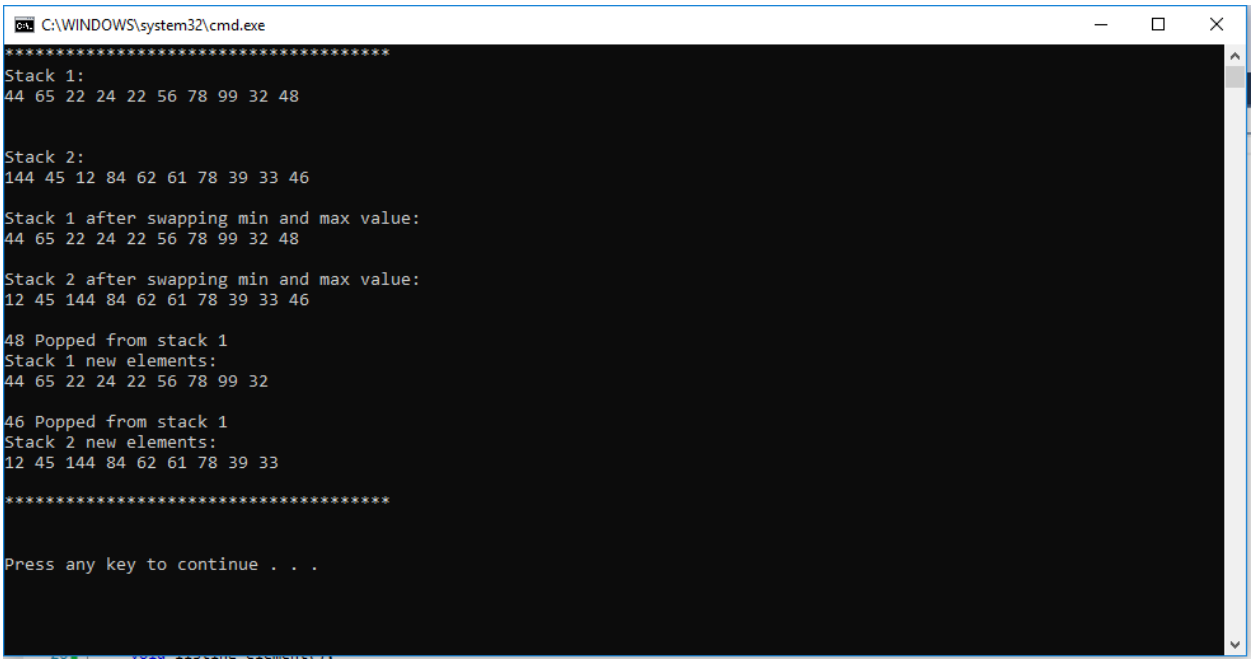
```
/*

Implement two stacks sharing the same array. Stacks must include the following fuctions:
adding, and removing elements, listing elements, checking is stack full or empty. Find
the
maximum and minimum elements of the stack and swap them (do not brake LIFO principle).

*/

#include<iostream>
using namespace std;

//implementation of Class stack
class Stack
{
	//declaring class variables
	const int STACK_SIZE = 100;
	int top;
	int a[100];    //Maximum size of Stack
public:
	//declaring class functions
	Stack();
	bool adding(int x);
	int removing();
	bool isEmpty();
	bool isFull();
	void listing_element();
	void minMaxElements();
};
//implementation of default constructor of class 'Stack'
Stack::Stack()
{
	top = -1;
}
//implementation of 'adding method' of class 'Stack'
```

```
bool Stack::adding(int x)
{
	if (isFull())
	{
		cout << "OH dear!!!! No element can be inserted because the stack is full
now." << endl;
		return false;
	}
	else
	{
		top = top + 1;
		a[top] = x;
		return true;
	}
}
//implementation of 'removing method' of class 'Stack'
int Stack::removing()
{
	if (isEmpty())
	{
		cout << "Oh dear!!! No element is present in stack." << endl;
		return 0;
	}
	else
	{
		int value = a[top];
		top = top - 1;
		return value;
	}
}
//implementation of 'isEmpty' method of class 'Stack'
bool Stack::isEmpty()
{
	if (top < 0)
	{
		return true;
	}
	else
		return false;
}
//implementation of 'isFull' method of class 'Stack'
bool Stack::isFull()
{
	if (top >= STACK_SIZE - 1)
	{
		return true;
	}
	else
		return false;
}
//implementation of 'listingElement' method of class 'Stack'
void Stack::listing_element()
{
	for (int i = 0; i <= top; i++)
	{
		cout << a[i] << " ";
	}
	cout << endl << endl;
```

```
}

//implementation of 'minMaxElements' method of class 'Stack'
void Stack::minMaxElements()
{
	int maxValue = a[0];
	int minValue = a[0];
	int minIndex = 0;
	int maxIndex = 0;
	for (int i = 0; i <= top; i++)
	{
		if (a[i]<minValue)
		{
			minValue = a[i];
			minIndex = i;
		}
		if (a[i]>maxValue)
		{
			maxValue = a[i];
			maxIndex = i;
		}
	}
	int temp;
	a[minIndex] = maxValue; //min value has max value swaped
	a[maxIndex] = minValue; //max has min value
}

// Driver program to test above functions
int main()
{
	const int ELEMENTS = 10;
	class Stack s, s1;

	//initializing two arrays
	int first[ELEMENTS] = { 44, 65, 22, 24, 22, 56, 78, 99, 32, 48 };
	int second[ELEMENTS] = { 144, 45, 12, 84, 62, 61, 78, 39, 33, 46 };
	for (int i = 0; i<ELEMENTS; i++)
	{
		s.adding(first[i]);
		s1.adding(second[i]);
	}
	//calling functions of stack

	cout << "************************************\n";
	cout << "Stack 1:" << endl;
	s.listing_element();
	cout << "\nStack 2:" << endl;
	s1.listing_element();
	s1.minMaxElements();
	cout << "Stack 1 after swapping min and max value:" << endl;
	s.listing_element();
	cout << "Stack 2 after swapping min and max value:" << endl;
	s1.listing_element();
	cout << s.removing() << " Popped from stack 1\n";
	cout << "Stack 1 new elements:" << endl;
	s.listing_element();
	cout << s1.removing() << " Popped from stack 1\n";
	cout << "Stack 2 new elements:" << endl;
```

```
        s1.listing_element();
        cout << "************************************\n";

        cout << endl << endl;
        system("pause");
        return 0;
}
```

Output Screen Shot:



Program 2

Problem Statement:

One possible improvement for Bubble Sort would be to add a flag variable and a test that determines if an exchange was made during the current iteration. If no exchange was made, then the list is sorted and so the algorithm can stop early. Modify the Bubble Sort implementation to add this flag and test. Compare the modified implementation on a range of inputs to determine if it does or does not improve performance in practice.

Code:

```
/*

One possible improvement for Bubble Sort would be to add a flag variable and a test that
determines if an exchange was made during the current iteration. If no exchange was made,
then the list is sorted and so the algorithm can stop early. Modify the Bubble Sort
implementation to add this flag and test. Compare the modified implementation on a range
of
inputs to determine if it does or does not improve performance in practice.

*/

#include <iostream>
#include <chrono>
using namespace std;

//functions prototype
void swap(int *, int *); // swap two values
void bubbleSort(int arr[], int);
void bubbleSortModified(int arr[], int);
void displayValues(int arr[], int);

int main()
{
	//record start time
	auto start = std::chrono::high_resolution_clock::now();
	int array1[] = { 64, 34, 25, 32, 52, 21, 28, 56, 100, 120, 95 };
	int n = sizeof(array1) / sizeof(array1[0]);

	bubbleSort(array1, n);
	cout << "\nSorted array without flag variable: \n\n";
	displayValues(array1, n);
	// Record end time
	auto finish = std::chrono::high_resolution_clock::now();
	std::chrono::duration<double> elapsed = finish - start;

	cout << "\nThe execution time of bubble sort without flag is:   "<<
elapsed.count() << endl;
```

```
	start = std::chrono::high_resolution_clock::now();
	int array2[] = { 64, 34, 25, 32, 52, 21, 28, 56, 100, 120, 95 };
	n = sizeof(array2) / sizeof(array2[0]);
	cout << "\n\nSorted array with flag variable: \n\n";
	bubbleSortModified(array2, n);
	displayValues(array2, n);
	// Record end time
	finish = std::chrono::high_resolution_clock::now();
	std::chrono::duration<double> elapsd = finish - start;

	cout << "\nThe execution time of bubble sort with flag is:   " << elapsd.count()
<< endl;

	cout << endl << endl;
	system("pause");
	return 0;
}

//implementation of function 'swap'
void swap(int *a, int *b) // swap two values
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

//implementation of function 'bubble sort'
void bubbleSort(int arr[], int n)
{
	for (int i = 0; i < n - 1; i++)
	{
		for (int j = 0; j < n - i - 1; j++)
		{
			if (arr[j] > arr[j + 1])
			{
				swap(&arr[j], &arr[j + 1]);
			}
		}
	}
}

//implementation of function 'bubble sort with flag'
void bubbleSortModified(int arr[], int n)
{
	bool flag = false; //flag variable
	for (int i = 0; i < n - 1; i++)
	{
		// Last i elements are already in place
		for (int j = 0; j < n - i - 1; j++)
		{
			if (arr[j] > arr[j + 1])
			{
				swap(&arr[j], &arr[j + 1]);
				flag = true;
			}
		}
		if (flag == false)
```

```
			{
				break; // no current ilteration made
			}
			else
				flag = false; // check for ilteration again
		}
}

//implementation of function 'displayValues'
void displayValues(int arr[], int size)
{
	for (int i = 0; i < size; i++)
	{
		cout << arr[i] << "  ";
	}

	cout << endl;
}
```

Output ScreenShot: