```
*************************************************************
** MIDTERM EXAM: VGP220, Winter 2020
** Time: 2:00 hrs
** Student Name: Fang Zhen
** Student ID: 1930034
*************************************************************
```

1) We want to use an ordinary array as a stack. It means it should support void push(T val) and T pop(). Push() pushes an item T on top of the stack, and pop() pops out and returns the top of the stack. Note that if you push N items on the array, we like to roll back to top of top of array and this way use our array as a circular array.
Implement your solution using an ordinary fixed size c array of size N. Copy your full header code bellow. No need to add any testing code to it.

```
#pragma once
#include <iostream>

using namespace std;
template<typename T>
class Circulararray
{
	private:
		int top = -1;
		int count = 0;
		int N;
		T* cArray;
public:
	Circulararray(int arraysize)
	{
		N = arraysize;
		cArray = new T[N];
		int top = -1;
		int count = 0;

	}
	~Circulararray() = default;

	bool Isempty()
	{
		if (count <= 0)
		{
			return true;
		}
		return false;

	}

	void push(T val)
	{

			count++;
			top=(top + 1) % N;
			cArray[top] = val;
```

```
	}

	T pop()
	{

		if (!Isempty())
		{
			if (count > N)
			{
				count = N;
			}
			count--;
			top = (top + N - 1) % N;
			return cArray[top];

		}
		else
		{
			cout << "The array is empty" << endl;
			return T();
		}


	}

	void SelectionSort()
	{
		for (int k = 0; k < N - 1; ++k)
		{
			int min = k;
			for (int i = k + 1; i < N; ++i)
			{
				if (cArray[i] < cArray[min])
					min = i;
			}
			if (k != min)
			{
				T temp = cArray[min];
				cArray[min] = cArray[k];
				cArray[k] = temp;
			}
		}
	}

	void print()
	{
		for (int i = 0; i < N; i++)
		{
			cout << cArray[i] << endl;
		}
	}



};
```

2) Add a function to sort the above array using Selection Sort. Copy th function body bellow.

```
void SelectionSort()
{
	for (int k = 0; k < N - 1; ++k)
	{
		int min = k;
		for (int i = k + 1; i < N; ++i)
		{
			if (cArray[i] < cArray[min])
				min = i;
		}
		if (k != min)
		{
			T temp = cArray[min];
			cArray[min] = cArray[k];
			cArray[k] = temp;
		}
	}
}
```

3) Write a function
*T SearchSortedArray(T *theArray, N, T val);*
This function searched a sorted array of fixed size N, of T type objects, looking for val. If it finds it, return its index. Otherwise returns the largest element still smaller that val. Make sure you take advantage of the fact that the array is sorted.

```
template<typename T>
T SearchSortedArray(T *theArray, int N, T val)
{

	if (val < theArray[0])
	{
		cout << "There is no element smaller than val" << endl;
		return 0;
	}
	else if (val > theArray[N - 1])
	{
		cout << "The clooest number is:";
		return theArray[N - 1];
	}
	for (int i = 0; i < N; ++i)
	{
		if (theArray[i] == val)
		{
			cout << "The arry index is:";
			return (T)i;
		}
```

```
                else if (theArray[i] < val && val < theArray[i + 1])
                {
                        cout << "The clooest number is:";
                        return theArray[i];
                }
        }

}
```