Data Structures and Algorithms
Lab 4

Roadmap

- Template classes
- Function templates
- Stack
- Applications of stack

Template Classes

- KeyStorage class from lab2

```
include <iostream>
using namespace std;

template<typename T>
class KeyStorage
{
    public:
       KeyStorage();
            KeyStorage(int k, T m);
            ~KeyStorage();
            T GetMember();
            void SetMember(T element);
    private:
             int key;
            T member; //a generic member: we don't know its type when creating the class
};

template<typename T>
KeyStorage<T>::KeyStorage()
{

}
```

```
template<typename T>
KeyStorage<T>::KeyStorage(int k, T m)
{
    key=k;
    member=m;
}

template<typename T>
KeyStorage<T>::~KeyStorage()
{
}

template<typename T>
T KeyStorage<T>::GetMember()
{
    return this->member;
}

template<typename T>
void KeyStorage<T>::SetMember(T m)
{
    this->member=m;
}
```

```
int main()
{
        //Everything happens to compile time, not to
run time
        //The compiler analyses the way in which you
use the class
        KeyStorage<long> keyElement1(2, 31456);
        KeyStorage<int> keyElement2(3,2);

        cout<<keyElement2.GetMember();

        KeyStorage<long> keyElement3;
        KeyStorage<int> keyElement4;
        return 0;
}
```

Function templates

- Returning the maximum between two numbers

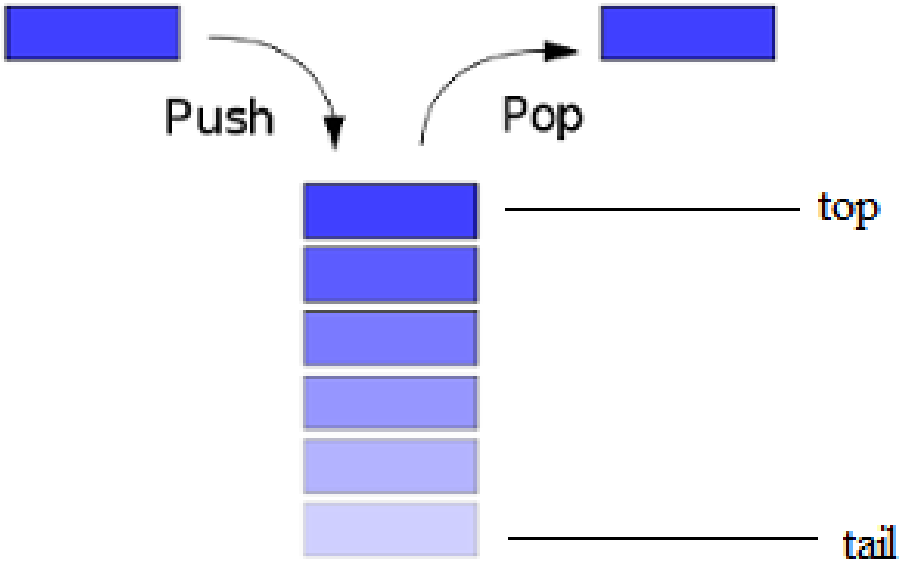
```
#include<iostream.h>
using namespace std;

template<class T>
 T getMax(T a, T b) {
        return a > b ? a : b;
 }

int main()
{
   cout<<getMax<int>(2, 3)<<" ";
   cout<<getMax<double>(3.2, 4.6);
}
```

Stack

- instance of an abstract data type (ADT) that formalizes the concept of restricted collection (LIFO = last in first out)
- ADT vs. Data Structures: ADT is in the logical level and data structure is in the implementation level
- Example:
  - ADT: stack
  - Data Structures:
    - stack implement with an array
    - stack implemented with a linked list

Access to the Elements of a Stack

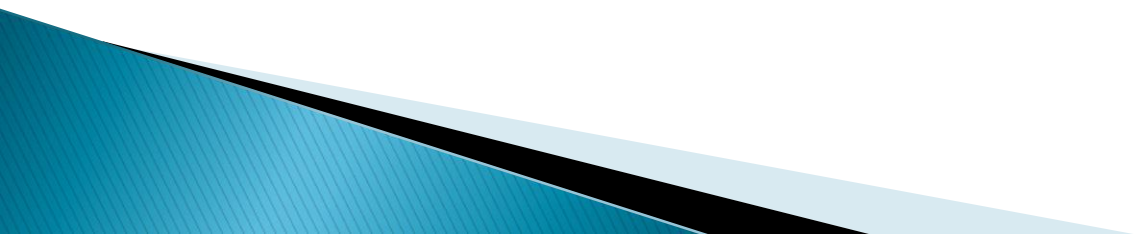
- Through its top



Basic Operations

- push(x)
  - Adds the element x at the top of the stack
- pop()
  - Removes the element from the top of the stack and returns it
  - Returns an error if the stack is empty
- peek()
  - Returns (but does not remove) the element at the top of the stack
- isEmpty()
  - Returns 1 if the stack is empty and 0 otherwise

Stack: Array-based Implementation



```
#include <iostream>
using namespace std;
#define NMAX 10 // pre-processing directive

template<typename T>
class Stack {
   private:
                // an array of NMAX dimension
                T stackArray[NMAX];
                /* the top of the stack, representing the INDEX of last
    element of the
                stackArray:0, 1, 2,....*/
    int topLevel;
   public:
    void push(T x) {
    //puts an element in the stack array
    //check if the stack array has the maximum dimension
                                    if (topLevel >= NMAX - 1)
                                    {
                cout<<"The stack is full: we have already NMAX
    elements!\n";
          //exit the function without making anything
                                                    return;
    }
    /*add an element=> the index of the last element of the stack Array
    increases and put the value of the new element in the stack array*/
       stackArray[++topLevel] = x;
    }

    int isEmpty() {
    //returns 1, if topLevel>=0, meaning the stack array has elements
    // returns 0, otherwise
       return (topLevel < 0);
                }
 T pop() {
       // extracts and element from the stack array and returns the new top
          if (isEmpty()) {
              // the extraction is made only if the array is not empty
             cout<<"The stack is empty! \n";
              T x;
              return x;
          }
          // the topLevel decreases and the new top is changed
          return stackArray[--topLevel];
}

T peek() {
           // returns the top of the stack
            if (isEmpty()) {
               // the extraction is made only if the array is not empty
               cout<<"The stack is empty! \n";
               T x;
               return x;
             }
          return stackArray[topLevel];
}

Stack() { // constructor
            topLevel = -1; // the stack is empty in the beginning
}

~Stack() { // destructor
 }
};

int main()
{
                    Stack<int> myStack;
                    myStack.peek();
                    myStack.push(5);
                    myStack.push(2);
                    myStack.push(3);
                    cout<<myStack.peek()<<"\n";
                    cout<<myStack.pop();
                    return 0;
}
```

```
#include <iostream>
using namespace std;
#define NMAX 10 // pre-processing directive

template<typename T>
class Stack {
   private:
                    // an array of NMAX dimension
                    T stackArray[NMAX];
                    /* the top of the stack, representing the INDEX of last element of the
                    stackArray:0, 1, 2,....*/
      int topLevel;
   public:
      void push(T x);
      int isEmpty();
      T pop();
       T peek();
       Stack();
        ~Stack();
};

template<typename T>
void Stack<T>::push(T x) {
       //puts an element in the stack array

       //check if the stack array has the maximum dimension
                                        if (topLevel >= NMAX - 1)
                                        {
                    cout<<"The stack is full: we have already NMAX elements!\n";
            //exit the function without making anything
                                                        return;
       }
       /*add an element=> the index of the last element of the stack Array
       increases and put the value of the new element in the stack array*/
          stackArray[++topLevel] = x;
      }

template<typename T>
int Stack<T>::isEmpty() {
       //returns 1, if topLevel>=0, meaning the stack array has elements
       // returns 0, otherwise
          return (topLevel < 0);
                    }
```

```
template<typename T>
T Stack<T>::pop() {
                    // extracts and element from the stack array and returns the new top
         if (isEmpty()) {
            // the extraction is made only if the array is not empty
                                        cout<<"The stack is empty! \n";
            T x;
            return x;
         }
                                                    // the topLevel decreases and
the new top is changed
         return stackArray[--topLevel];
      }

template<typename T>
T Stack<T>::peek() {
                                                    // returns the top of the stack
                    if (isEmpty()) {
                          // the extraction is made only if the array is not empty
                                                    cout<<"The stack is empty! \n";
                          T x;
                          return x;
                    }
         return stackArray[topLevel];
                                        }

template<typename T>
Stack<T>::Stack() { // constructor
                     topLevel = -1; // the stack is empty in the beginning
                                        }

template<typename T>
Stack<T>::~Stack() { // destructor
                                        }

int main()
{
                    Stack<int> myStack;
                    myStack.peek();
                    myStack.push(5);
                    myStack.push(2);
                    myStack.push(3);
                    cout<<myStack.peek()<<"\n";
                    cout<<myStack.pop();
                    return 0;
}
```

Exercises

1. Show the minimum and the maximum element from a stack.
2. Check if a given text is well-written i.e. all the propositions and phrases should contain a ". Or ? Or !" at the end.

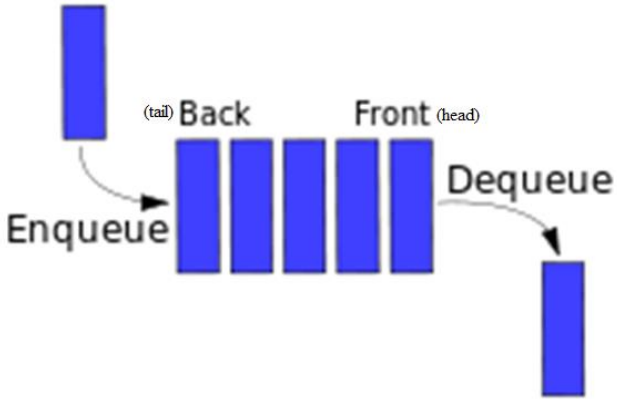
Hint! Use substring =>s.substr(pos, noChars);

E.G. string s ="Home sweet home" -> s.substr(5,5) returns "sweet"

Example of parsing a given string

```
#include <iostream.h>
#include <string>
using namespace std;
int main(){
    string s ="121";
    char string[10];
    for(int i=0; i<s.length(); i++){
            string[i]=s[i];
            cout<<string[i];
    }
}
```

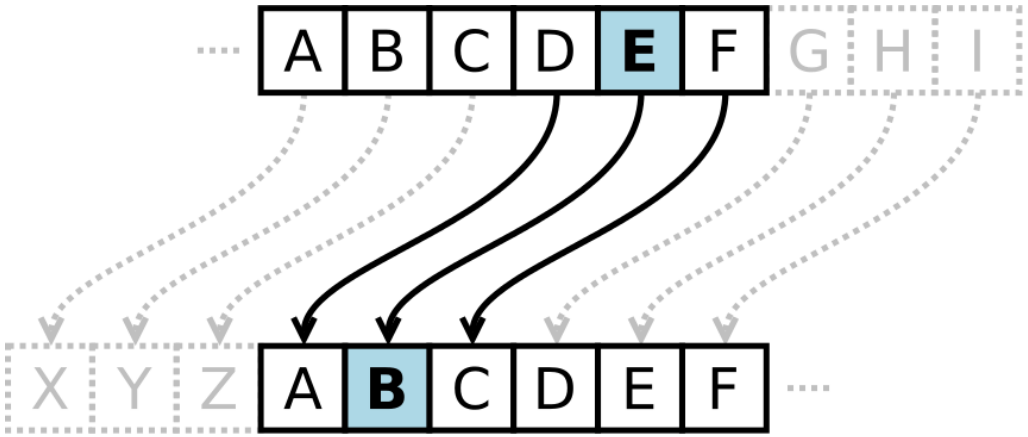
Homework



1. Using the model of the STACK class, implement a QUEUE template class with an array (QUEUE is an ADT that formalizes the concept of restricted collection (FIFO = first in first out):

- Enqueue(x)
  - Adds the element x at the tail of the queue
- Dequeue()
  - Removes the element from the head of the queue and returns it
  - Returns an error if the stack is empty
- Peek()
  - Returns (but does not remove) the element at the head of the queue
- isEmpty()
  - Returns 1 if the queue is empty and 0 otherwise

2. Given a string where A = 0, B = 1,..., Z = 25, encrypt and decrypt according to the Caesar cipher through the use of the stack data structure.
Hint: transform the letters into numbers.



3. Considering that you store points inside a stack, sort the elements of the stack in the increasing order of the distance from the origin (0, 0) using two stacks. Display the first 3 closest points to the origin.