Data Structures in C++

CMPE226- Data Structures

Stacks (Cont.)

Adapted from Textbook (D. S. MALIK), Lecture notes (C. TURHAN) and Lecture Notes (H. QU)

Recall- Stacks

- Data structure
  - Elements added, removed from one end only
  - Last In First Out (LIFO)

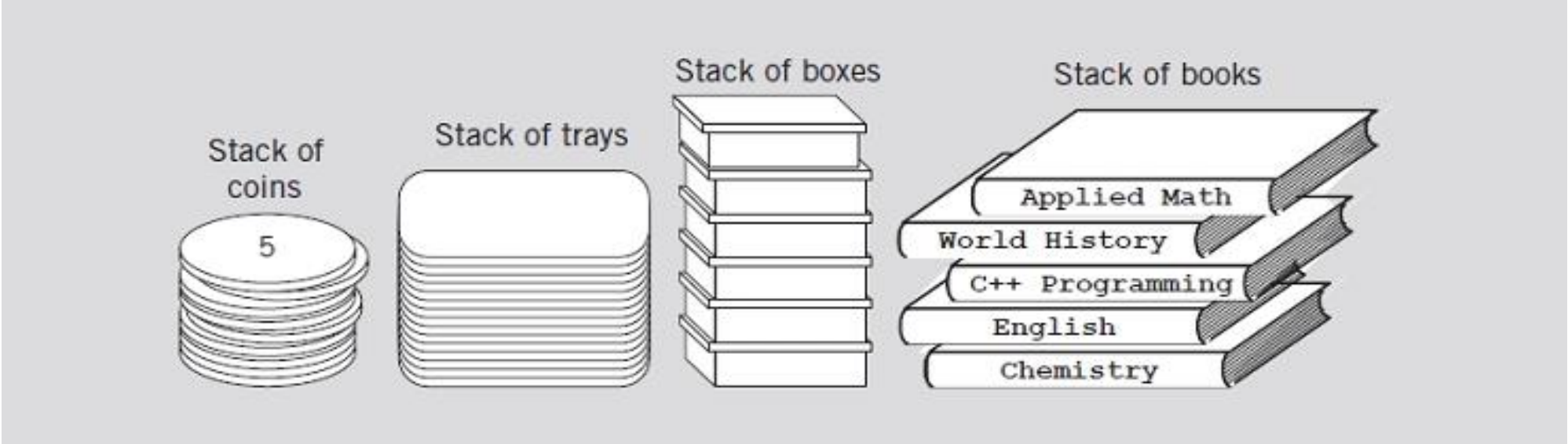
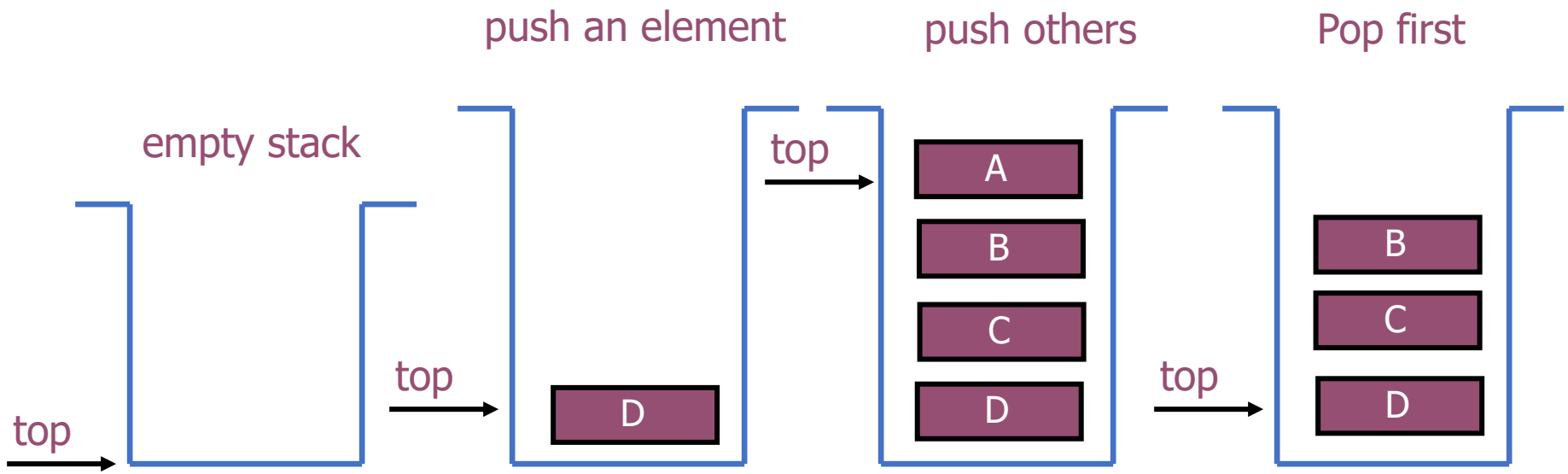


FIGURE 7-1 Various examples of stacks

Recall- Push, Pop and Top

- Push: Equivalent to an insert
- Pop: Deletes the most recently inserted element
- Top: Examines the most recently inserted element

Exp. If the characters 'D', 'C', 'B', 'A' is placed in a stack (in that order), and then removed one at a time, in what order will they be removed?



STACKS- Overview

- Stacks are less flexible
  - ✓but are more efficient and easier to implement
- Last In First Out (LIFO) structure

  The last element inserted will be the first to be retrieved.
- Any list implementation could be used to implement a stack, can be implemented as array or linked list
  - Arrays: limited number of elements

    static: the size of stack is given initially
  - Linked lists: allow dynamic element addition

    dynamic: never become full

STACKS- Implementation

Stack ADT Definition

- Initialize: Initialize stack to empty stack
- Destroy: remove all elements
- isEmpty: return true if stack is empty, false otherwise
- isFull: return true if stack is full, false otherwise
- Push: add a new element to the top
- Pop: remove and return top element
- Top: return top element (not remove only return)

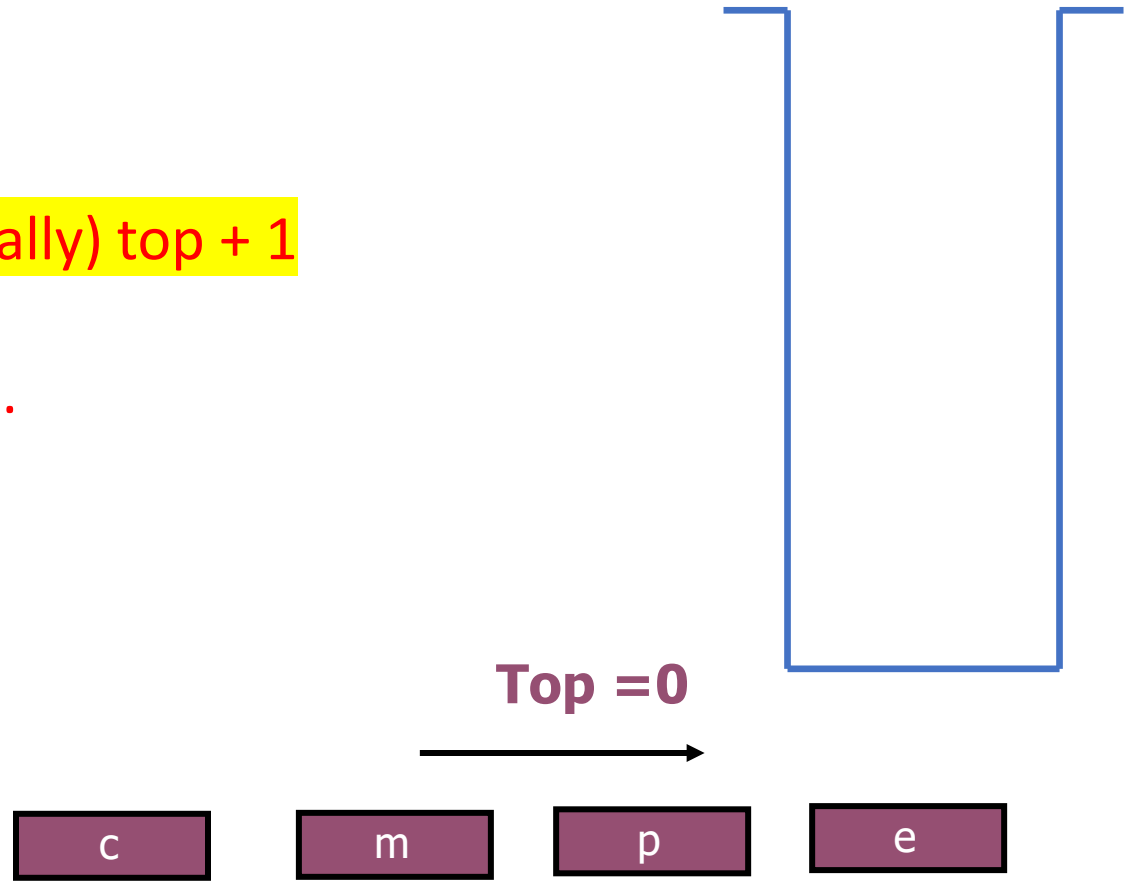
Implementation of Stacks using Arrays

Implementation of Stacks- with Arrays
Stack Class: Attributes & Operations

```
template <class T>
class Stack {
protected:
        T *arr;//array hold elements (point to array which stores elements of stack)
        int top;//index of empty space after topmost element
        int size;//size of array
public:
        Stack(int stackSize);
        bool isEmpty(); //return true if stack is empty
        bool isFull(); //return true if stack is full
        void destroy();
        void push(T &); //add an element to the top of the stack
        void copy(Stack<T> &);
        T pop(); //delete an element at the top of the stack
        T top(); //return an element at the top of the stack
        ~Stack();
};
```

Implementation of Stacks- with Arrays

```
//Stack Constructor- Initialize the elements
template <class T>
Stack <T>::Stack(int stackSize) {
        size = stackSize;
        arr = new T[size];
        top = 0; //shows the index that is (actually) top + 1
}
//Destructor- Delete contents and set size to 0.
template <class T>
Stack<T>::~Stack() {
        delete[] arr;
        size = 0;
}
```



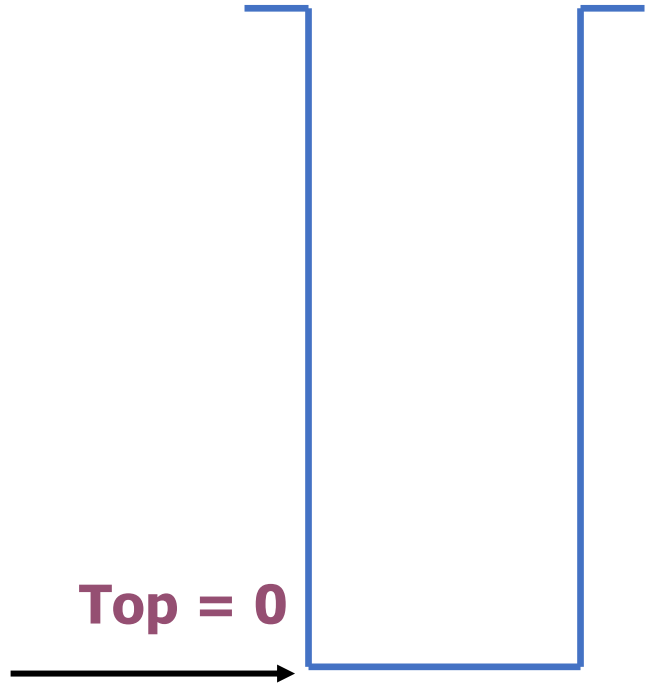
Implementation of Stacks- with Arrays

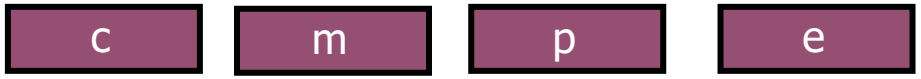
```
template <class T>
bool Stack<T>::isEmpty() {
        return (top == 0);
}
template <class T>
bool Stack<T>::isFull() {
        return (top == size);
}
template <class T>
void Stack<T>::destroy() { //elements already in the stack become garbage.
        top = 0;
}
```

Top always shows top +1. Assume that the size of array is 4, including “cmpe”.

Indices	0	1	2	3
values	c	m	p	e

Top=0





Implementation of Stacks- with Arrays

Push()- Add a new element

```
template <class T>
void Stack<T>::push(T &data) {
    if (!isFull()) {//when adding a new item check if it is full?
                    arr[top++] = data;
          }else {
                    std::cout << "Stack Full" << std::endl;
          }
}
```

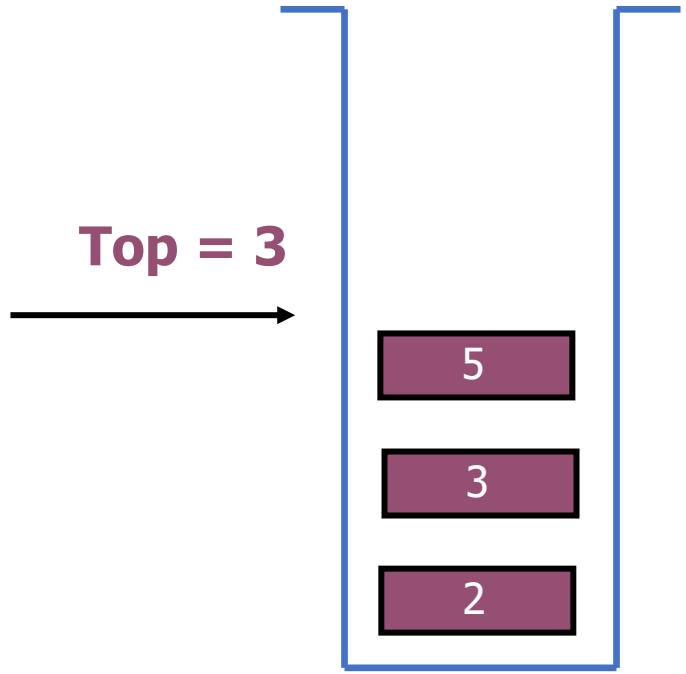
Ex. Push(item); *(Assume item=5)*

Set arr[top] =item;

Increment top by 1.

Indices	0	1	2	3
values	2	3	5	

top



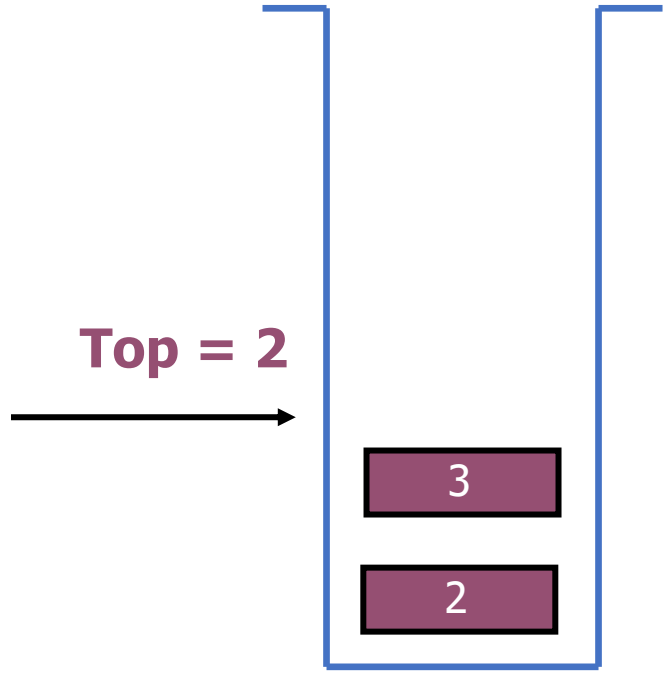
Implementation of Stacks- with Arrays

Pop()- Return and remove the element which is at the top

```
template <class T>
T Stack<T>::pop() {
        assert(!isEmpty()); //if false abort
        return arr[--top];
}
```

Top= top-1;
Return Arr[top]

Ex. Pop();
Decrement top value by 1.
return value in arr[top]



Indices	0	1	2	3
values	2	3		

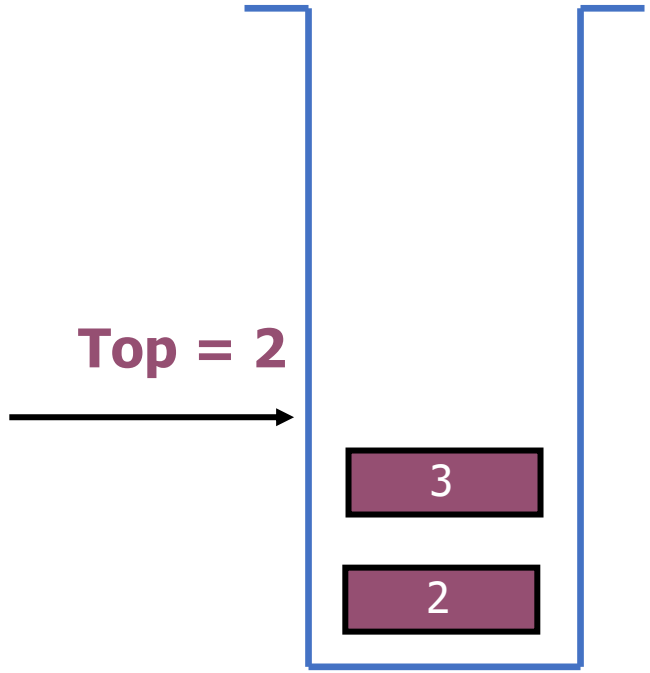
top

Implementation of Stacks- with Arrays

Top()- Return the top element

```
template <class T>
T Stack<T>::topData() {
        assert(!isEmpty()); //if false abort
        return arr[top-1];

}
```

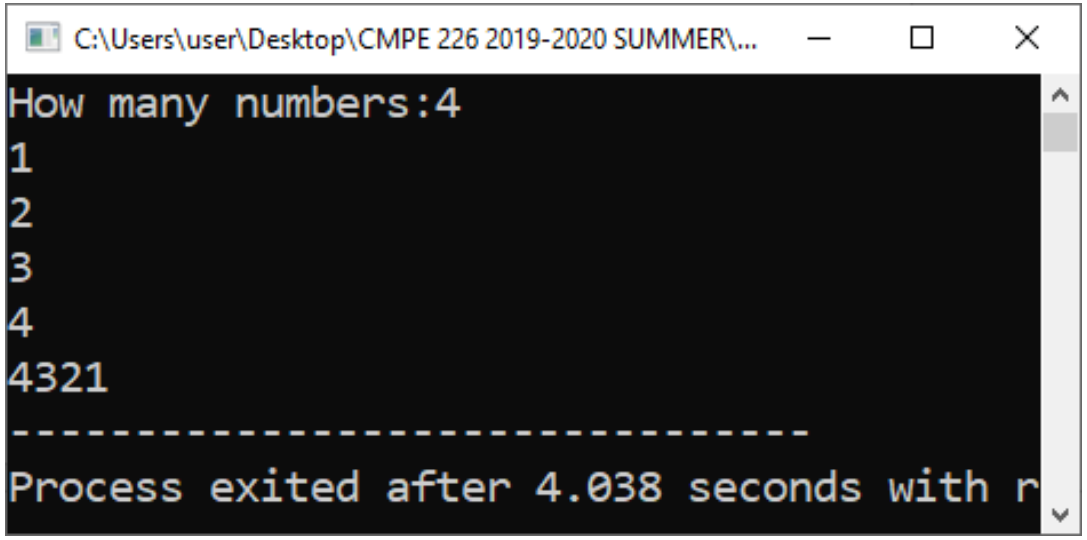


Indices	0	1	2	3
values	2	3		

Top=2

Example

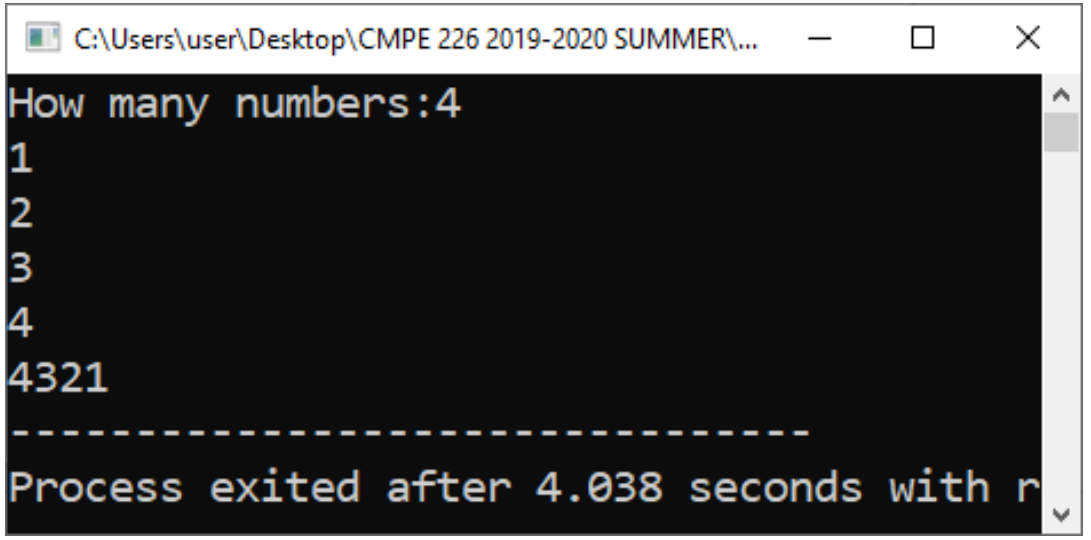
- Input an integer n from user and input n integers into a stack. Output them in reverse order. Use stack.h file.



Example

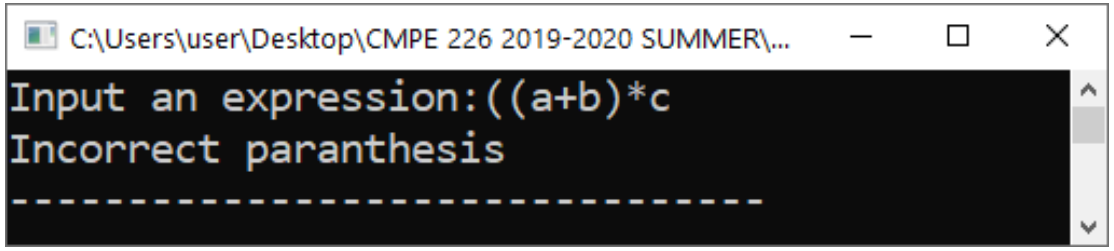
- Input an integer n from user and input n integers into a stack. Output them in reverse order. Use stack.h header file.

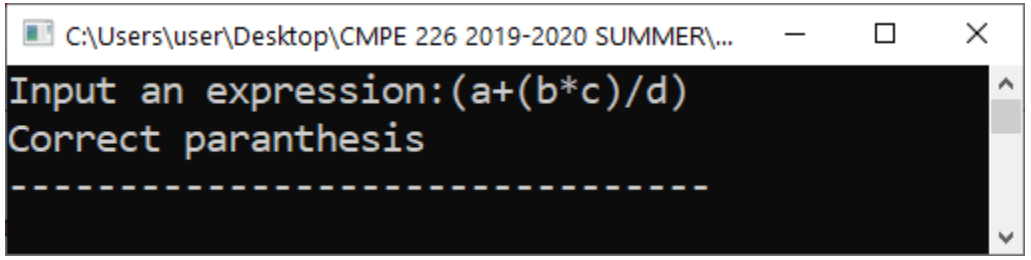
```
#include<iostream>
#include "stack.h"
using namespace std;
main(){
    Stack<int> numbers(10);
    int n,x;
    cout<<"How many numbers:";
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>x;
        numbers.push(x);
    }
    while(!numbers.isEmpty()){
        cout<<numbers.pop();
    }
}
```



Example

- Input a mathematical expression and check if parenthesis are nested correctly. Use stack.h file.
- Ex. *(x-y))* *incorrect.*
  - *)x+y)+z* *incorrect.*
  - *((3+(x*y)/z)*)5* *correct.*
  - *How many parenthesis you open? Should be equal to the closed ones.*





Example

- Input a mathematical expression and check if parenthesis are nested correctly. Use stack.h file.

Algorithm

(1) Create an empty stack.

(2) Read characters until the end of expression

   i. If the character is an opening symbol, push it onto the stack
   ii. If it is a closing symbol, then if the stack is empty, incorrect
   iii. Otherwise, pop the stack.

(3) At the end, if the stack is not empty, incorrect

Solution

```
#include<iostream>
#include "stack.h"
using namespace std;
int main(){
    Stack<char>parser(100);
    bool valid = true;
    string expr;
    char next;
    cout<<"Input an expression:";
    cin >> expr;
    for (int i = 0; valid & i < expr.length(); i++){
        next = expr[i];
        if (next == '(') {
            parser.push(next);
        } else if (next == ')') {
            if (parser.isEmpty()) {
                valid = false;
            } else {
                parser.pop();
            }
        }
    }
    if (!parser.isEmpty())//if stack is not empty- incorrect
        valid = false;
    if (valid) {
        cout << "Correct paranthesis";
    } else {
        cout << "Incorrect paranthesis";
    }
    return 0;
}
```

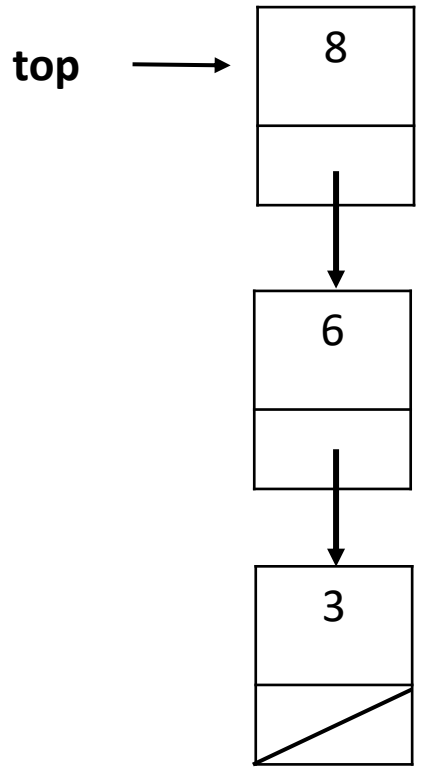
Implementation of Stacks using LinkedList

Implementation of Stacks- with LinkedList

In Stack: Insert and delete from the same end, top.

We use Linked Lists' insertFirst() method and we should also use delete first.

To keep the data, create a node structure including info and the address of next node (top pointing to bottom node)



Top is a pointer to the top node

Implementation of Stacks- with LinkedList

```
#ifndef LINKEDSTACK_H
#define LINKEDSTACK_H
#include <iostream>
#include <cassert>
template <class T>
struct node {
        T info;
        node<T> *link;
};
```

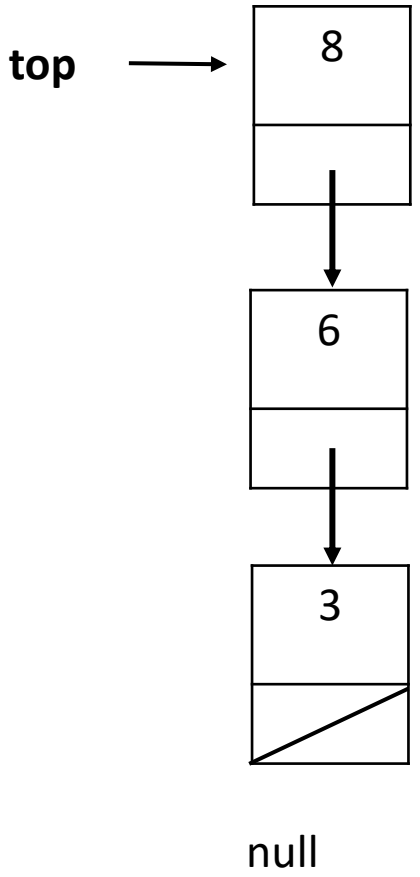
```
template <class T>
class LinkedStack {
protected:
        node<T> *top; //head becomes top. Items should be inserted from top.
        int cnt;
public:
        //Constructor
        LinkedStack(){
                top=NULL;//sets top initial value to NULL
        }
        //Destructor
        ~LinkedStack(){
                destroy();
        }
```

```
bool isEmpty(){
                return top==NULL;
        }
//There is no size limitation in Stack using LinkedList implementation. We do not need to check if the stack is full or not.
T showTop(){
        assert( !isEmpty() );//if not empty
        return top->info;//return the value on the top
        }
T pop();
void push(T&);
void destroy();
};
```

Implementation of Stacks- with LinkedList

Destroy()

```
//resets the stack to its initial state
template <class T>
void LinkedStack<T>::destroy(){
      node<T> *p;
      while (top != NULL) {
                  p = top;
                  top = top->link;
                  delete p;
      }
      cnt = 0;
}
```

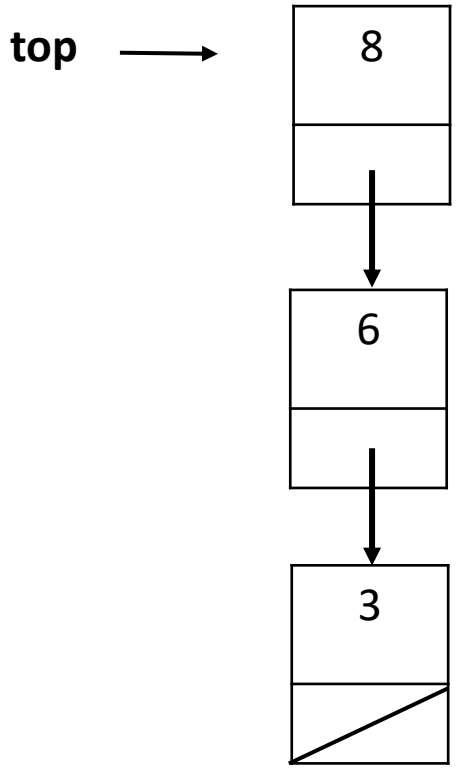


Top is a pointer to the top node

Implementation of Stacks- with LinkedList

Push()

```
//insert a new item to the top of stack
template <class T>
void LinkedStack<T>::push(T &item){
      node<T> *newNode = new node<T>;
      newNode->info=item;
      newNode->link=top;
      top=newNode;
}
```

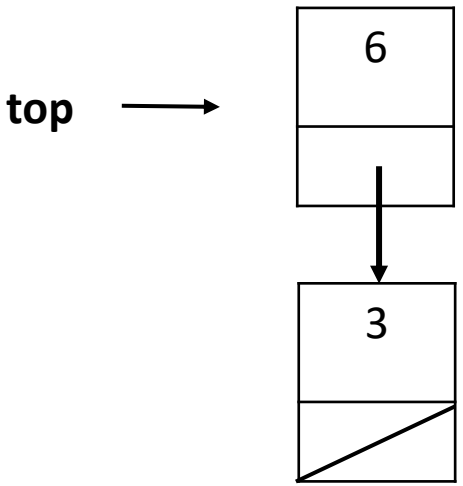


Top is a pointer to the top node

Implementation of Stacks- with LinkedList

Pop()

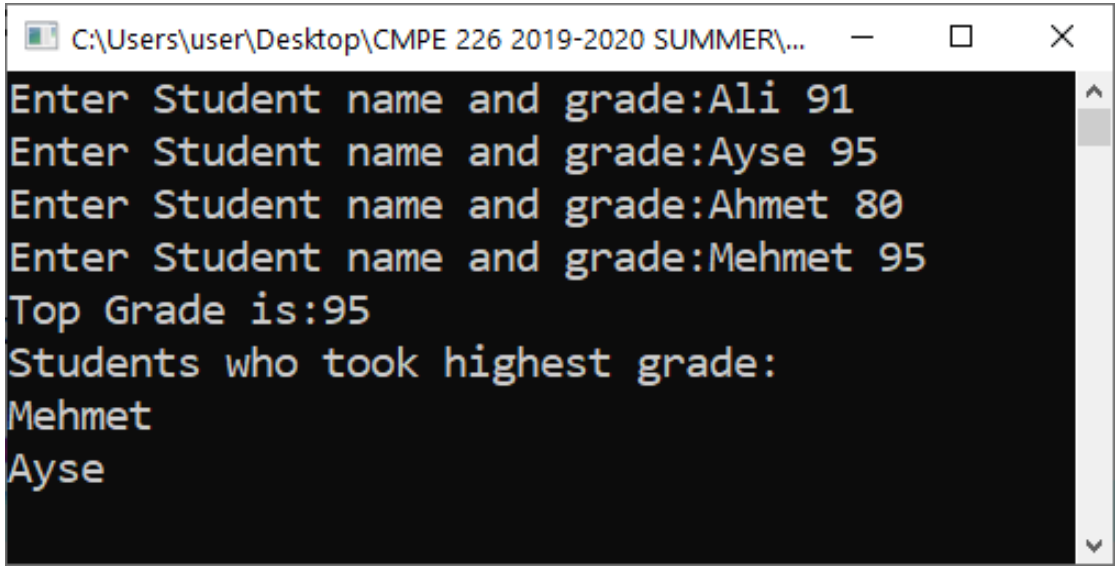
```
//retrieve (remove and return) the top value
template <class T>
T LinkedStack<T>::pop(){
      node<T> *p;//pointer to traverse
      T item;
      assert(!isEmpty());//if the stack is empty terminate
      p=top;
      item=top->info;
      top=top->link;
      delete p;
      return item;
}
```



Top is a pointer to the top node

Example

Write a program to enter the exam grades of 50 students and find the names of the students who received the top grade. Use a stack of objects to store name&grade of students who received the top grade. There may be more than 1 student who received the top grade. Output the highest grade and names of students with top grade.



Solution

```
#include <iostream>
#include <string>
#include "LinkedStack.h"
using namespace std;
class Student {
public:
string name;
int grade;
friend istream & operator>>(istream &is, Student &s){
		is>>s.name>>s.grade;
		return is;
		}
friend ostream& operator<<(ostream &os, Student &s){
			os<<s.name<<endl;
			return os;
		}
};
main(){
		int max=0;
		Student s;
		LinkedStack<Student> st;
		for (int i=1;i<5; i++){
			cout<<"Enter student name and
grade:";
			cin >> s;
			if (s.grade > max){
				max=s.grade;
				st.destroy();
				st.push(s);
			} else if (s.grade==max){
				st.push(s);
			}
		}
cout << "Top Grade="<<max<<endl;
cout << "Students who received top grade:"<<endl;
		while(!st.isEmpty()){
			s=st.pop();
			cout << s;
		}
}
```

Stacks Derived from LinkedList Class

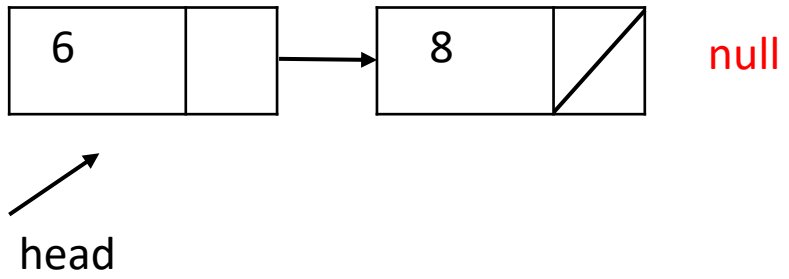
Stacks Derived from LinkedList Class

- Inherited from LinkedList header file.
  - ➢InsertFirst() – for Push() in Stack.
  - ➢Pop() should be defined.

```
template <class T>
class LinkedStack :public Linkedlist<T>{
public:
        bool isEmpty(){
                return emptyList();
        }
        void push(T &item){
                insertFirst(item);
        }
        T showTop(){
                return front();
        }
        T pop();
}
```

Stacks Derived from LinkedList Class

```
template <class T>
T LinkedStack<T>::pop(){
        node<T> *p;
        T item;
        assert(!isEmpty());
        p=head;
        head=head->link;
        count--;
        item =p->info;
        delete p;
        if (head==NULL){
                last=NULL;
        }
        return item;
}
```



References

- CMPE226- Data Structures Lecture Notes by Cigdem Turhan
- Data Structures Using C++, D.S. Malik, Thomson Course Technology, 2nd Edition.
- Lecture Slides by Huamin Qu, The Hong Kong University of Science and Technology (2005)