Stack

CSE 2020 Computer Science II

Learning Objectives

- define stack ADT
- implement stack ADT using array and linked structure
- analyze the time complexity of operations in different implementations
- apply stack class defined in STL

Stack ADT

- A stack stores a list of elements in which insertion and deletion are performed at the same end of the list, called the `top`.
  - The first added element is at the bottom
  - The most recent added element is at the top
  - The add and remove only happen at the top
  - The most recent added element is the first to be removed
  - Last-In-First-Out (LIFO)

Operations of Stack

The operations are

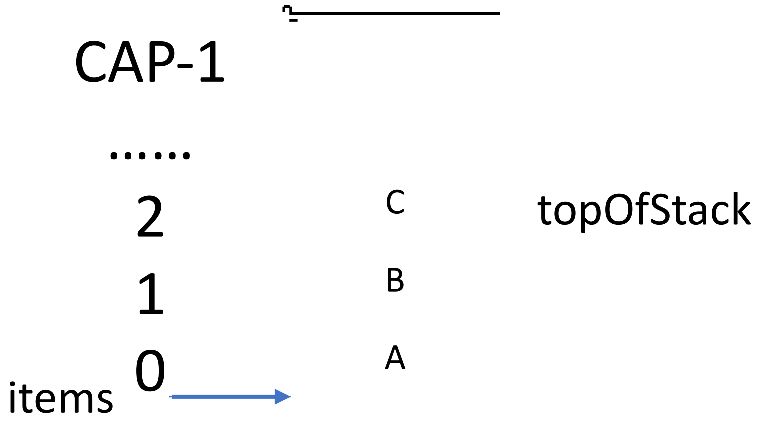
- bool empty() const: return true if the stack is empty
- void clear(): remove all elements in stack
- void push(const T & x): add x to the stack
- void pop(): remove the top element
- const T& top() const: return the top element

Implementation of Stack

- Array implementation of Stack
- Linked structure implementation of Stack

Array Impl. of Stack

- `Stack` class template defined in ArrayStack.cpp (lab exercise)
  - private attribute *`items`* is a pointer, points to a dynamic array, *`T* items`*;
  - private attribute *`topOfStack`* stores the index of the top element, *`int topOfStack;`* the stack is empty when *`topOfStack`* is -1



Array Impl. of Stack Operations

- Empty state
- push(const T & x)

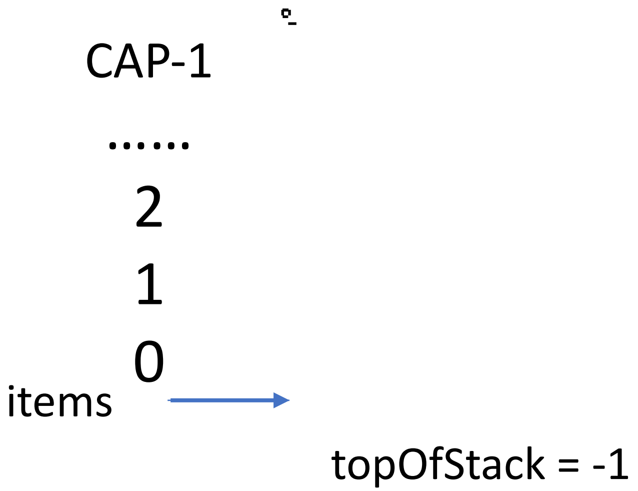
```
topOfStack++;
items[topOfStack] = x;
```

- pop()

```
topOfStack--;
```

- const T &top()

```
items[topOfStack];
```



CAP-1

......

2

1

0

items

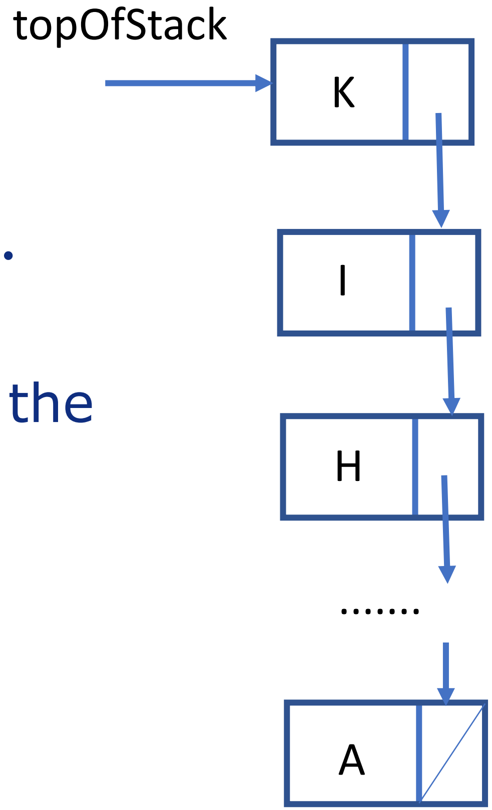
topOfStack = -1

Linked Impl. of Stack

- `Stack` class template defined in LinkedStack.cpp (in Stack.txt)
  - private struct template *NodeType* contains *T data* and *NodeType* next*. next points to next node
  - private pointer *topOfStack* points to the top node of stack,

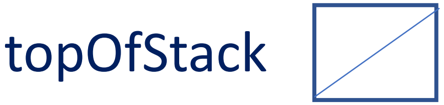
    NodeType topOfStack;*

    empty when *topOfStack* is *nullptr*



Linked Impl. of Stack Operations

- Empty state



- push(const T & x)

```
NodeType* p = new NodeType(x);
p>next = topOfStack;
topOfStack = p;
```

- pop()

```
NodeType* p = topOfStack;
topOfStack = topOfStack->next;
delete p;
```

- const T &top()

```
topOfStack->data
```

Access Stack Template Class

- Use class template to define Stack in a general way, the element type is `T`, which is bound to an actual data type in main() function, as shown in TestStack.cpp (in Stack.txt)
  - #include “ArrayStack.cpp” or “LinkedStack.cpp”
  - `Stack<int>`
  - `Stack<double>`
  - `Stack<string>`
  - `Stack<char>`
  - `Stack<Employee>`

Array vs Linked Structure Stack

- Array implementation of stack has fixed capacity
- Linked structure implementation of stack has no fixed capacity

Operations	Array Stack	Linked Stack
empty()	O(1)	O(1)
clear()	O(1)	O(n)
push(x)	O(1)	O(1)
pop()	O(1)	O(1)
top()	O(1)	O(1)

STL stack

- *stack* provides a growable array implementation of the Stack ADT

```
#include <stack>
stack<int> ints;
ints.push (10);
intv.pop();
cout << intv.top();
stack<double> dbls;
stack<string> strs;
```