Data Structures and Algorithms

Lecture 3: Stacks

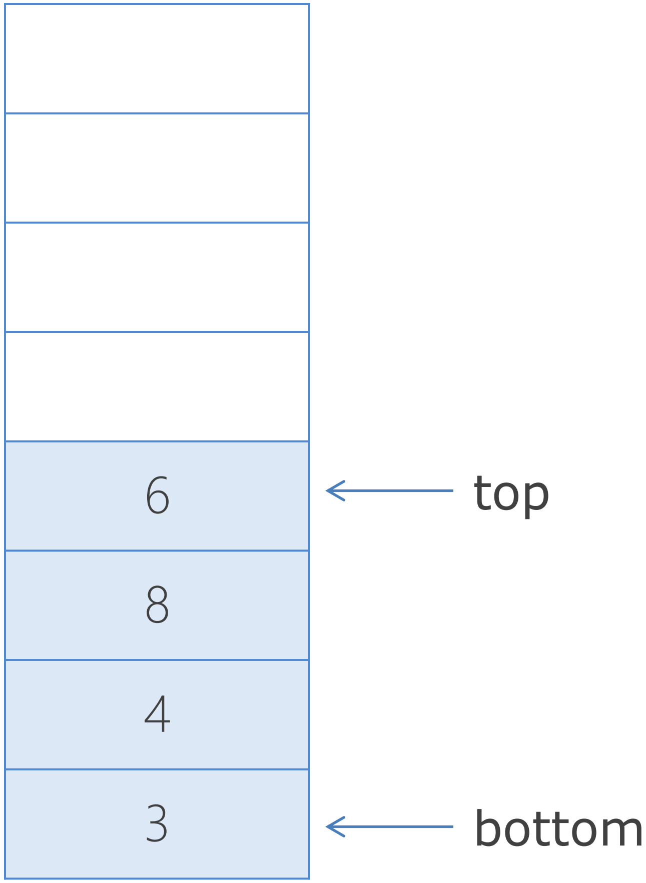
Department of Computer Science & Technology
United International College

Outline

- Stack ADT
- Basic operations of stack
  - Pushing, popping etc.
- Applications of stacks
- Implementations of stacks using
  - array
  - linked list

Stack ADT

- Stack is a special list where insertion and deletion take place at the same end
  - This end is called *top*
  - The other end is called *bottom*
- Everything happens at the top
- Nothing happens at the bottom

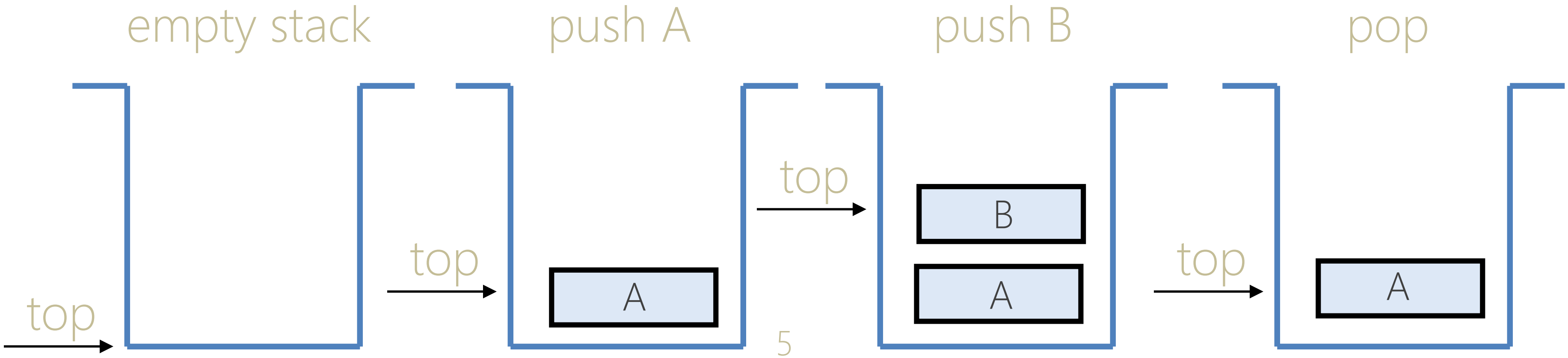


Stack Animation

- http://www.cs.armstrong.edu/liang/animation/web/Stack.html
- Stacks are known as LIFO (Last In, First Out) lists.

Push and Pop

- Primary operations: Push and Pop
- Push
  - Add an element to the top of the stack
- Pop
  - Remove the element at the top of the stack
- Top
  - Return, without removing, the element at the top



Stack Applications

- Expression evaluation
- Backtracking
- Memory Management

Implementation of Stacks

- Recall the reason why we usually don't implement the list using array?

Topic		Array	Linked List
Efficiency	push		
	pop		
	Top		
space			

Stack Implementation

- Data
  - maxTop: the max size of stack
  - top: the index of the top element of stack
  - values: point to an array which stores elements of stack
  - values can of any data type but we use double for demonstration

```
typedef struct{
        double* values;
        int top;
        int maxTop;
} Stack;
```

Stack Implementation

- Methods

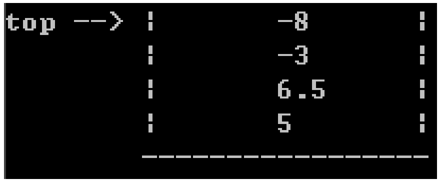
```
bool CreateStack(Stack *stack, int size);
bool IsEmpty(Stack* stack);
bool IsFull(Stack* stack);
bool Top(Stack* stack, double* x);
bool Push(Stack* stack, double x);
bool Pop(Stack* stack, double* x);
void DisplayStack(Stack* stack);
void DestroyStack(Stack* stack);
```

Methods

- bool CreateStack(Stack *stack, int size);
  - Creates an empty stack whose capacity is *size*
- bool IsEmpty(Stack* stack);
  - Returns true if the stack is empty and false otherwise
- bool IsFull(Stack* stack);
  - Returns true if the stack is full and false otherwise
- bool Top(Stack* stack, double* x);
  - Returns true if the operation is successful and false otherwise
  - Passes the value of the top element to *x*

Methods

- bool Push(Stack* stack, double x);
  - Add a new element with value *x* to the top of the stack
  - Returns true if the operation is successful and false otherwise
- bool Pop(Stack* stack, double* x);
  - Remove an element from the top of the stack
  - Returns true if the operation is successful and false otherwise
  - Passes the value of the top element to *x*
- void DisplayStack(Stack* stack);



```
top --> ¦       -8       ¦
        ¦       -3       ¦
        ¦       6.5      ¦
        ¦       5        ¦
       ------------------
```

- void DestroyStack(Stack* stack);
  - Frees the memory occupied by the stack

CreateStack

```
#include <stdlib.h>
bool CreateStack(Stack *stack, int size){
   if (size <= 0)
      return false;
   stack->values = (double*)malloc(sizeof(double)*size);
   stack->top = -1;
   stack->maxTop = size - 1;
   return true;
}
```

Why?

Push

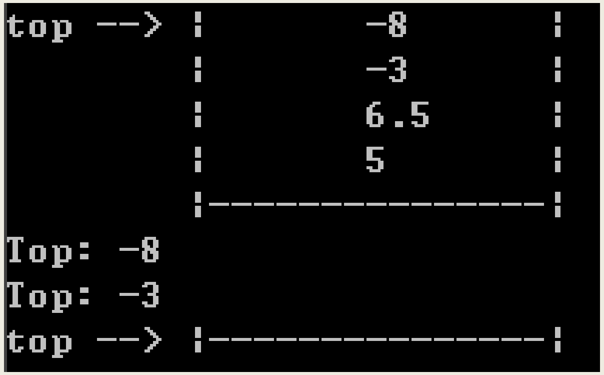
```
bool Push(Stack* stack, double x){
    if(IsFull(stack))
        return false;
    stack->values[++stack->top] = x;
    return true;
}
```

This explains.

Using Stack

```c
#include "stack.h"
int main(void) {
   Stack stack;
   double val;
   CreateStack(&stack, 5);
   Push(&stack, 5);
   Push(&stack, 6.5);
   Push(&stack, -3);
   Push(&stack, -8);
   DisplayStack(&stack);
   if(Top(&stack, &val))
      printf("Top: %g", val);
   Pop(&stack, &val);
   if(Top(&stack, &val))
      printf("Top: %g", val);
   while(!IsEmpty(&stack))
      Pop(&stack, &val);
   DisplayStack(&stack);
   DestroyStack(&stack);
}
```

```c
int main(void) {
   Stack stack;
   double val;
   CreateStack(&stack, 5);
   Push(&stack, 5);
   Push(&stack, 6.5);
   Push(&stack, -3);
   Push(&stack, -8);
   DisplayStack(&stack);
   if(Top(&stack, &val))
      printf("Top: %g", val);
   Pop(&stack, &val);
   if(Top(&stack, &val))
      printf("Top: %g", val);
   while(!IsEmpty(&stack))
      Pop(&stack, &val);
   DisplayStack(&stack);
   DestroyStack(&stack);
}
```



```
top --> ¦       -8      ¦
        ¦       -3      ¦
        ¦       6.5     ¦
        ¦       5       ¦
        ¦---------------¦
Top: -8
Top: -3
top --> ¦---------------¦
```

Side Notes

- All these values evaluate to false in C:

0

NULL

false

Side Notes II

- A typical file structure for a C program with user defined data structures
  - stack.h
    - Declares all the data types and functions
    - No implementation
  - stack.cpp
    - Includes stack.h
    - Implements all the functions in stack.h
  - main.cpp
    - Includes stack.h
    - Implements the main function

Task

- Write *stack.h* and *stack.cpp* which implement the stack data structure.
- Refer to *list.h* and write proper comments in *stack.h* to describe every function.
- Submit *stack.h* and *stack.cpp* to iSpace.