

**CSE1303 Part A**  
**Data Structures and Algorithms**  
**Summer Semester 2003**

**Lecture A3 – Basic Data Structures (Stacks)**

Kymerly Fergusson

**Basic Data Structures**

- Stacks
- Queues
- Lists

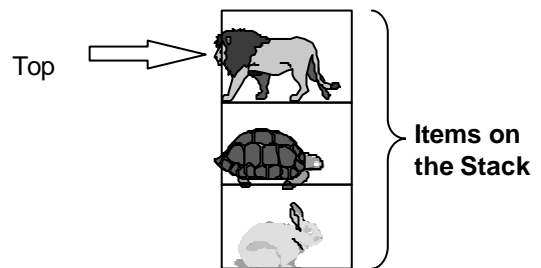
2

**Overview of Stacks**

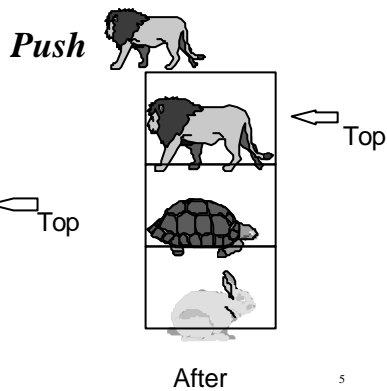
- What is a Stack?
- Operations on a Stack
  - push, pop
  - initialize
  - status: empty, full
- Implementation of a Stack.
- Example: Reversing a sequence.

3

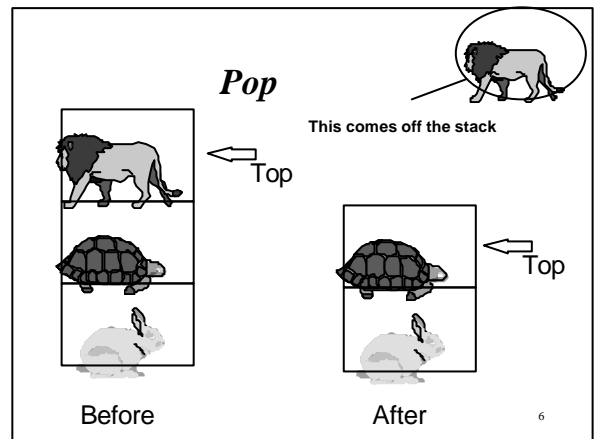
**A Stack**



4



5



6

## Operations

- Initialize the Stack.
- Pop an item off the top of the stack.
- Push an item onto the top of the stack.
- Is the Stack empty?
- Is the Stack full?
- Clear the Stack
- Determine Stack Size

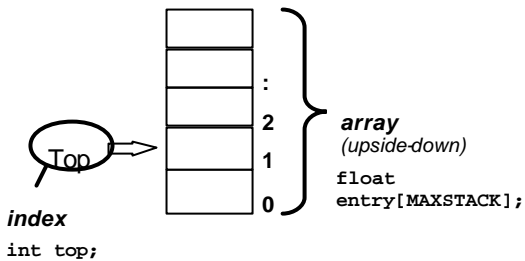
7

## Stack Properties

- Sequence of items, where insertions and deletions are done at the top.
- Main operations are pop and push.
- Last-In First Out (LIFO).
- Used when calling functions.
- Used when implementing recursion.

8

## Implementation

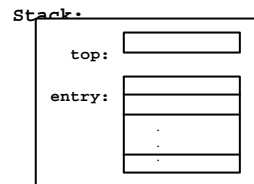


9

```
#define MAXSTACK 20

struct StackRec
{
    int top;
    float entry[MAXSTACK];
};

typedef struct StackRec Stack;
```



10

```
#ifndef STACKH
#define STACKH
#include <stdbool.h>
#define MAXSTACK 20

struct StackRec
{
    int top;
    float entry[MAXSTACK];
};

typedef struct StackRec Stack;

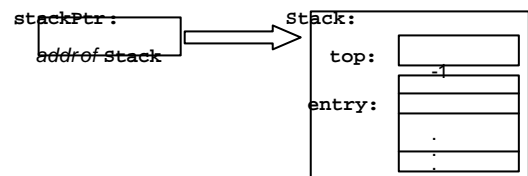
void initializeStack(Stack* stackPtr);
bool stackEmpty(const Stack* stackPtr);
bool stackFull(const Stack* stackPtr);
void push(Stack* stackPtr, float item);
float pop(Stack* stackPtr);

#endif
```

11

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

void initializeStack(Stack* stackPtr)
{
    stackPtr -> top = -1;
}
```



12

```

bool stackEmpty(const Stack* stackPtr)
{
    if (stackPtr-> top < 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool stackFull(const Stack* stackPtr)
{
    if (stackPtr -> top >= MAXSTACK-1)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

13

```

void push(Stack* stackPtr, float item)
{
    if (stackFull(stackPtr))
    {
        fprintf(stderr, "Stack is full\n");
        exit(1);
    }
    else
    {
        stackPtr-> top++;
        stackPtr-> entry[stackPtr-> top] = item;
    }
}

```

14

```

float pop(Stack* stackPtr)
{
    float item;

    if (stackEmpty(stackPtr))
    {
        fprintf(stderr, "Stack is empty\n");
        exit(1);
    }
    else
    {
        item = stackPtr-> entry[stackPtr-> top];
        stackPtr-> top--;
    }

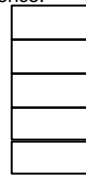
    return item;
}

```

15

## Reversing a Sequence

Take a sequence: **-1.5 2.3 6.7**



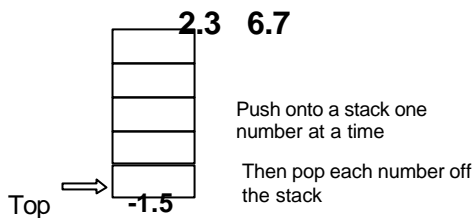
Push onto a stack one number at a time

Then pop each number off the stack

Top →

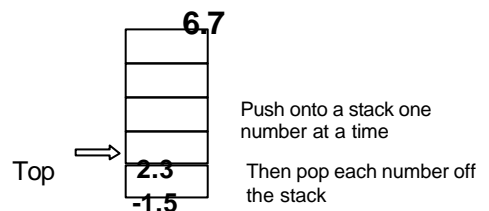
16

## Reversing a Sequence



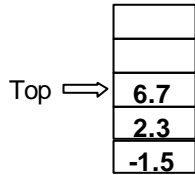
17

## Reversing a Sequence



18

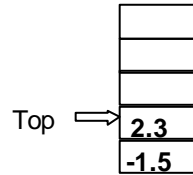
## Reversing a Sequence



Push onto a stack one number at a time  
Then pop each number off the stack

19

## Reversing a Sequence

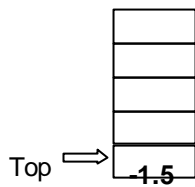


Push onto a stack one number at a time  
Then pop each number off the stack

6.7

20

## Reversing a Sequence

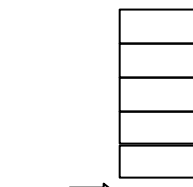


Push onto a stack one number at a time  
Then pop each number off the stack

6.7 2.3

21

## Reversing a Sequence



Push onto a stack one number at a time  
Then pop each number off the stack

6.7 2.3 -1.5

22

```

module reverse ()
{
  initialize the Stack
  loop{
    input nextNumber
    if (end of input) then {exit loop}
    if (Stack is not full) then {push nextNumber onto Stack}
  }
  loop {
    if (Stack is empty) then {exit loop}
    pop nextNumber off the Stack
    output nextNumber
  }
}

```

23

```

#include <stdio.h>
#include "stack.h"
int main()
{
  Stack theStack;
  float next;

  initializeStack(&theStack);
  printf("Enter number sequence: ");
  while (scanf("%f", &next) != EOF) {
    if (!stackFull(&theStack)) {
      push(&theStack, next);
    }
  }
  while (!stackEmpty(&theStack)) {
    next = pop(&theStack);
    printf("%f", next);
  }
  printf("\n");
}

```

24

## ***Revision***

- Stack
- Operations on a Stack
  - push, pop
  - initialize
  - status: empty, full
  - others: clear, size
- Example: Reversing a sequence.
- Implementation.

25

## ***Next Lecture***

- Queue
- Main Operations
- Implementation

## ***Revision: Reading***

- Kruse - Chapters 3.1.1 – 3.1.5
- Deitel & Deitel – Chapter 12.5
- Langsam - Chapter 2
- Standish – Chapters 7.1 – 7.6

## ***Preparation***

*Next lecture: **Queues***

- Read 4.1 to 4.2 in Kruse et al.
- Deitel & Deitel 12.6

26