

CSE1303 Part A
Data Structures and Algorithms
Summer Semester 2003

Lecture A8 – Linked Stacks and
Linked Queues

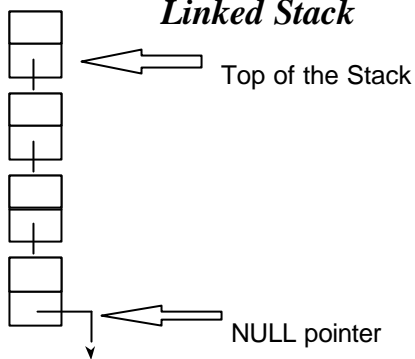
Kymberly Fergusson

Overview

- Linked Stack.
 - Push
 - Pop
- Linked Queue.
 - Append
 - Serve

2

Linked Stack



3

```
#ifndef LINKEDSTACKH
#define LINKEDSTACKH
#include <stdbool.h>
#include "node.h"

struct LinkedStackRec
{
    Node* topPtr;
};
typedef struct LinkedStackRec 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
```

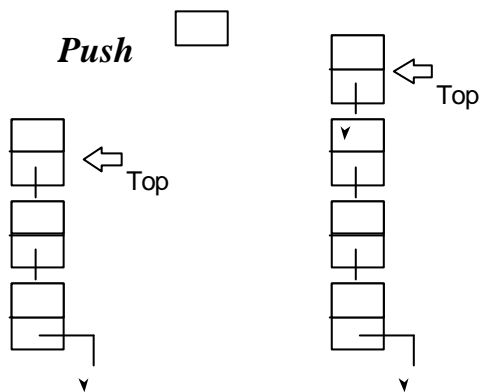
4

Initialize Stack

```
void initializeStack(Stack* stackPtr)
{
    stackPtr->topPtr = NULL;
}
```

5

Push



6

Push

- create a new node for the item
- link the new node to the current top node
- make the new node the new top

7

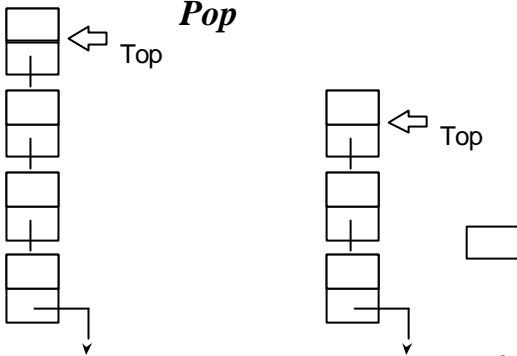
Push

```
void push(Stack* stackPtr, float item)
{
    Node* newNodePtr = makeNode(item);

    newNodePtr->nextPtr = stackPtr->topPtr;
    stackPtr->topPtr = newNodePtr;
}
```

8

Pop



9

Pop

- check if the stack is empty
- remember the item in the top node
- remember address of current top node
- make the next node the new top
- free the old top node
- return the item

10

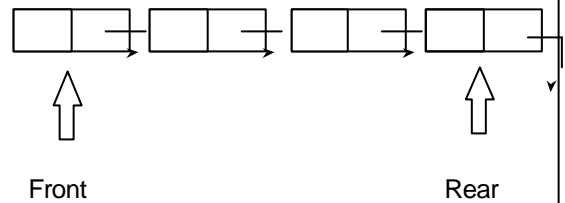
Pop

```
float pop(Stack* stackPtr)
{
    float item;
    Node* oldNodePtr = stackPtr->topPtr;

    if (stackEmpty(stackPtr)) {
        fprintf(stderr, "Stack is empty\n");
        exit(1);
    }
    else {
        item = oldNodePtr->value;
        stackPtr->topPtr = oldNodePtr->nextPtr;
        free(oldNodePtr);
    }
    return item;
}
```

11

Linked Queue



12

```

#ifndef LINKEDQUEUEH
#define LINKEDQUEUEH
#include <stdbool.h>
#include "node.h"

struct LinkedQueueRec
{
    int    count;
    Node*  frontPtr;
    Node*  rearPtr;
};
typedef struct LinkedQueueRec Queue;

void initializeQueue(Queue* queuePtr);
bool queueEmpty(const Queue* queuePtr);
bool queueFull(const Queue* queuePtr);
void append(Queue* queuePtr, float item);
float serve(Queue* queuePtr);

#endif

```

13

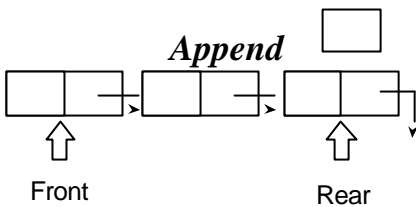
Initialize Queue

```

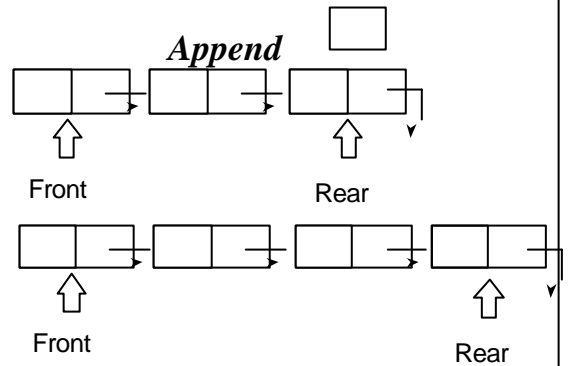
void initializeQueue(Queue* queuePtr)
{
    queuePtr->frontPtr = NULL;
    queuePtr->rearPtr = NULL;
    queuePtr->count = 0;
}

```

14



15



16

Append

- create a new node for the item
- if the queue is empty:
 - the new node becomes both front and rear of the queue
- otherwise:
 - make a link from the current rear to the new node
 - the new node becomes the new rear
- increment the count

17

```

void append(Queue* queuePtr, float item)
{
    Node* newNodePtr = makeNode(item);

    if (queueEmpty(queuePtr)) {
        queuePtr->frontPtr = newNodePtr;
        queuePtr->rearPtr = newNodePtr;
        queuePtr->count = 1;
    }
    else {
        queuePtr->rearPtr->nextPtr = newNodePtr;
        queuePtr->rearPtr = newNodePtr;
        queuePtr->count++;
    }
}

```

18

Revision: Reading

- Kruse 3.1.6, 4.6
- Standish 7.6, 7.8

Preparation

Next lecture: Linked Lists

- Read 5.2.2 - 5.2.4 in Kruse et al.