

Topic 5

Linked Stacks and Linked Queues

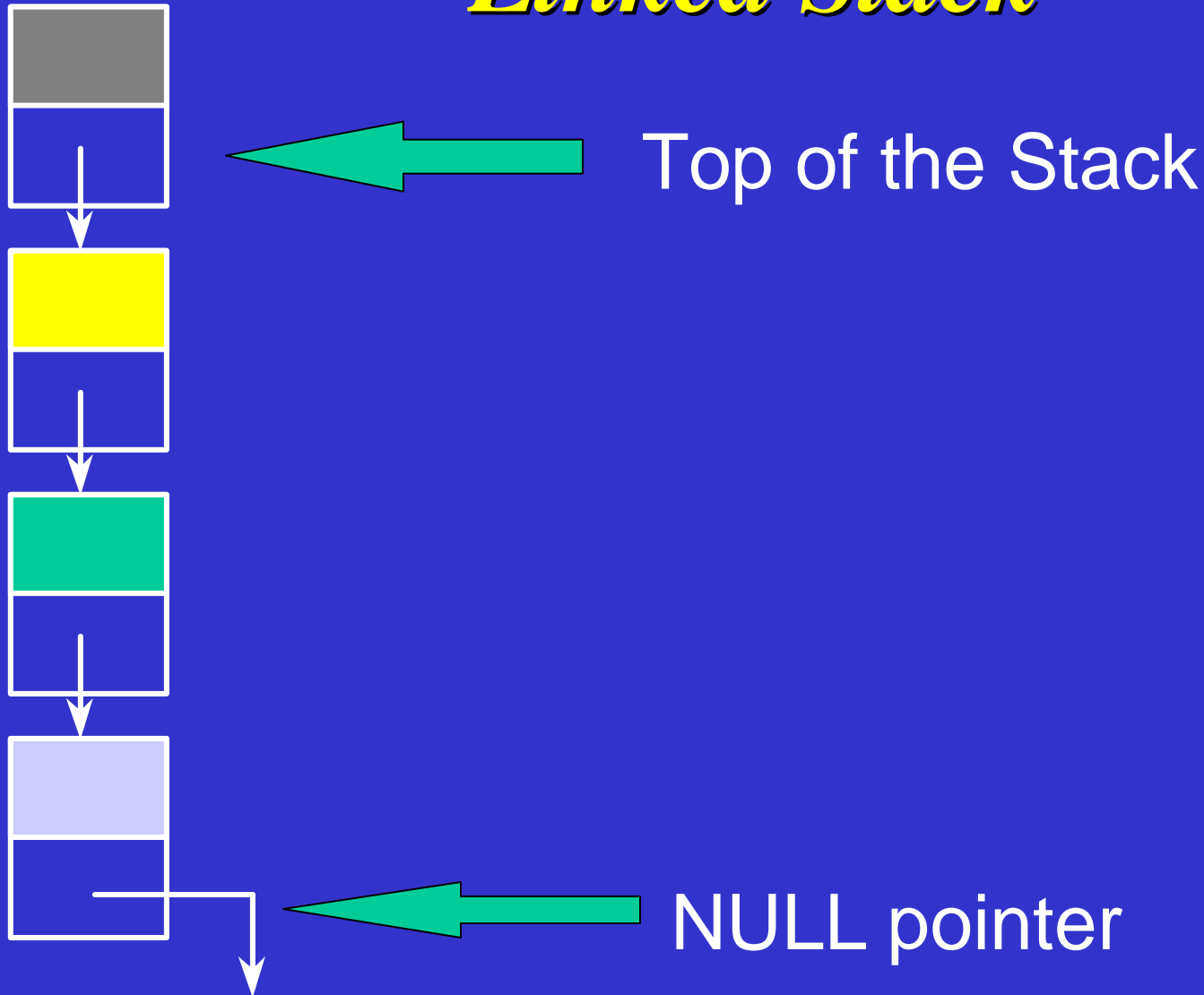
CSE1303 Part A

Data Structures and Algorithms

Overview

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

Linked Stack



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

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

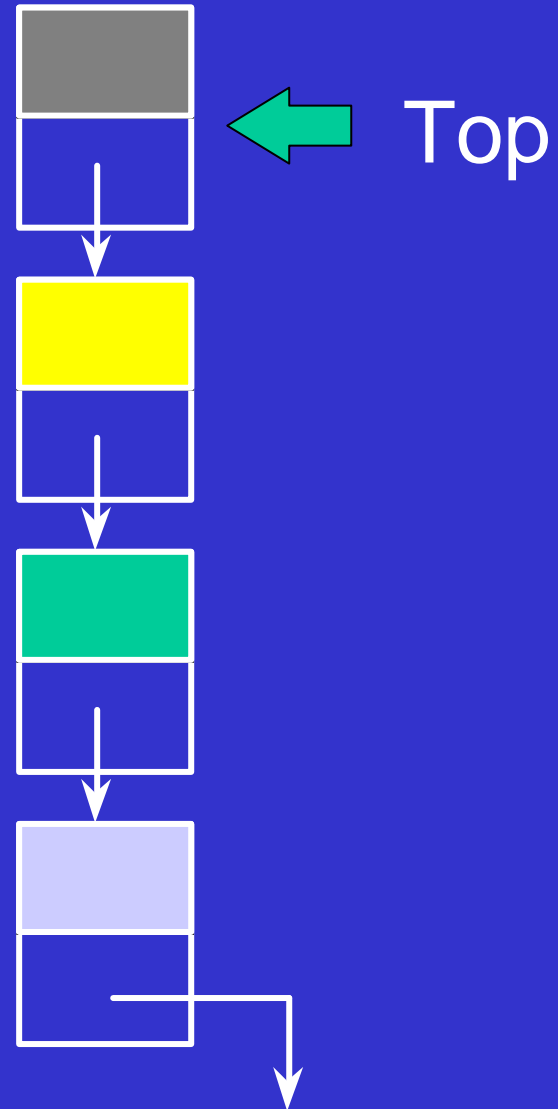
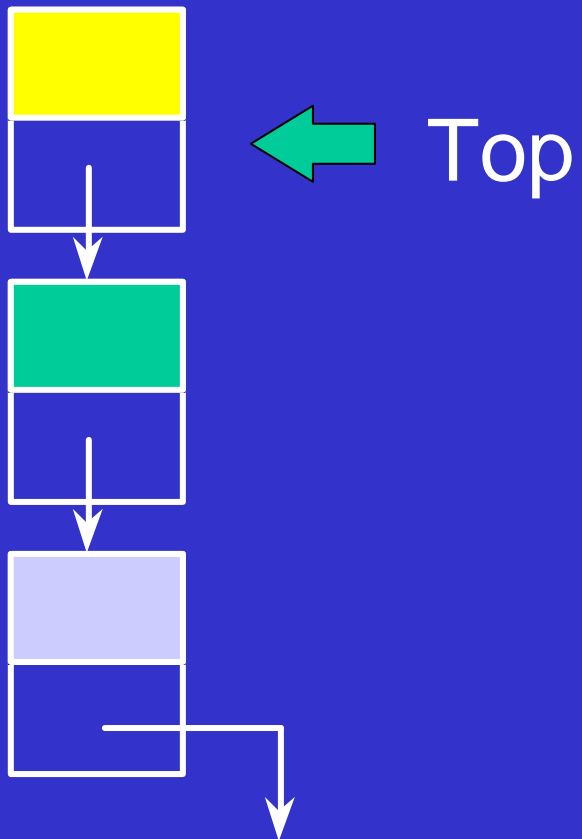
void initializeStack(Stack* stackPtr);
bool stackEmpty(const Stack* stackPtr);
bool stackFull(const Stack* stackPtr);
void push(Stack* stackPtr, float item);
float pop(Stack* stackPtr);

#endif
```

Initialize Stack

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

Push



Push

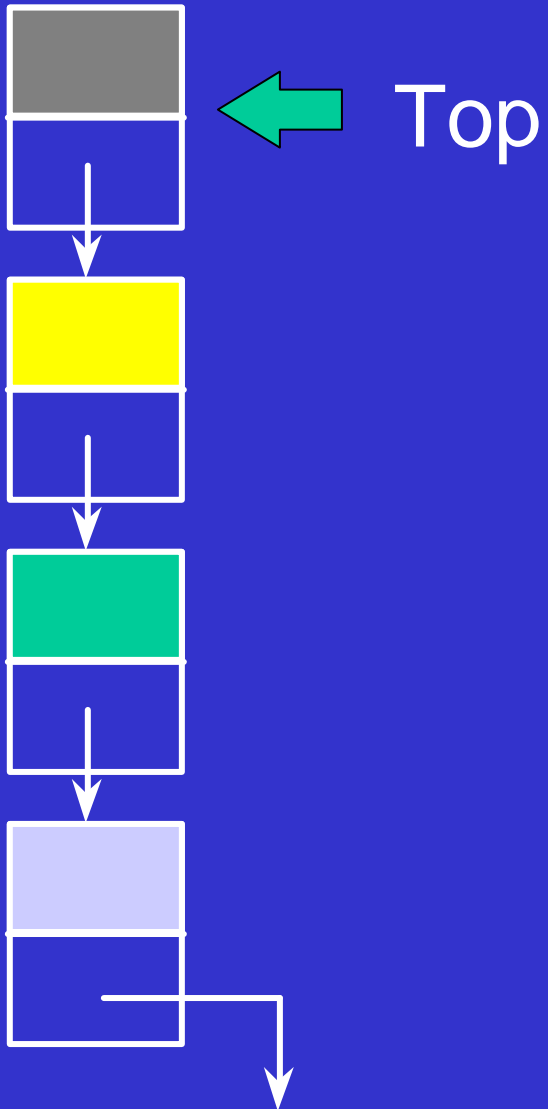
- create a **new node** for the item
- **link** the new node to the current top node
- make the new node the **new top**

Push

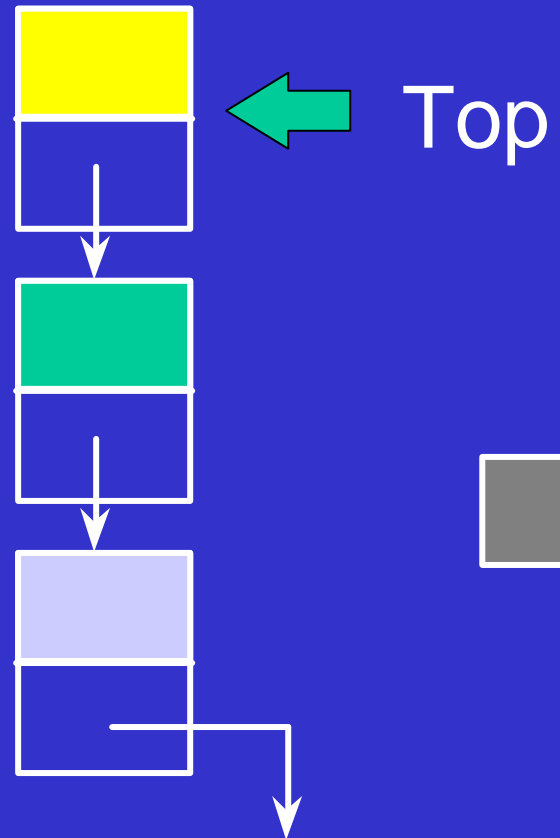
```
void push(Stack* stackPtr, float item)
{
    Node* newNodePtr = makeNode(item);

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

Pop



1/18/02



CSE1303 Part A

9

Pop

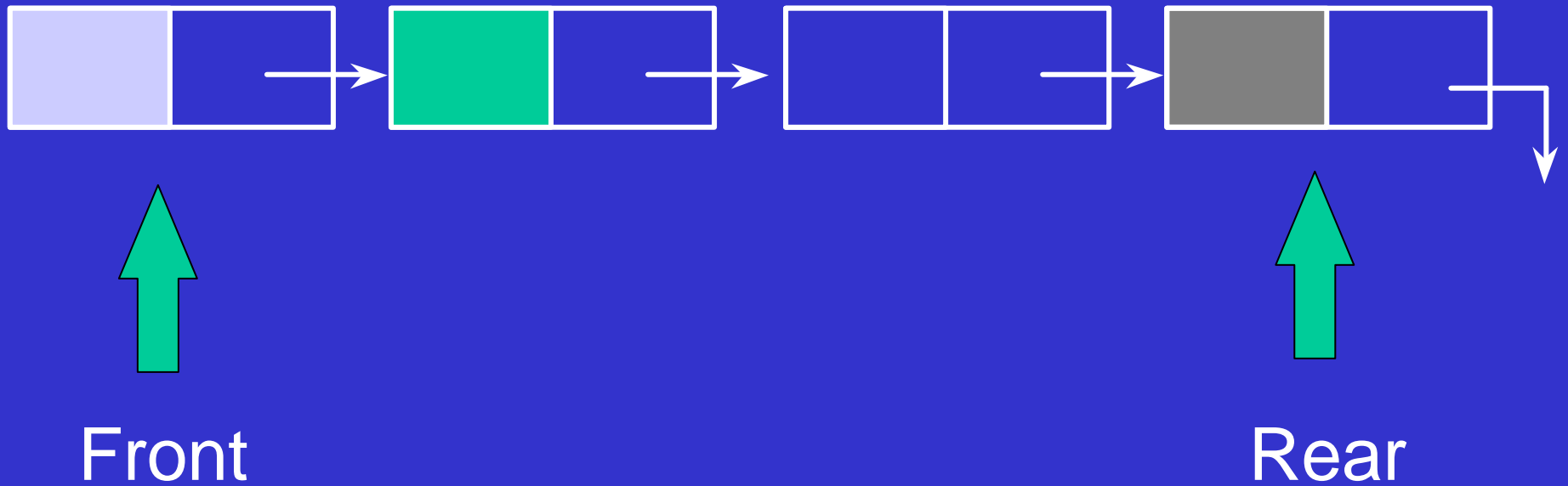
- check if the stack is empty
- remember the **item** in the top node
- remember **address** of current top node
- make the next node the **new top**
- **free** the old top node
- return the **item**

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;
}
```

Linked Queue



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

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

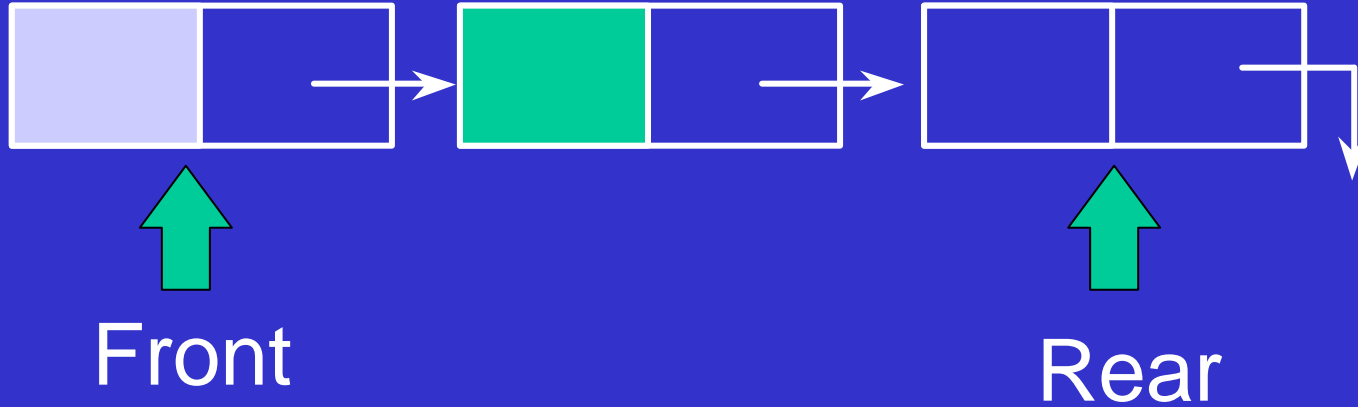
void initializeQueue(Queue* queuePtr);
bool queueEmpty(const Queue* queuePtr);
bool queueFull(const Queue* queuePtr);
void append(Queue* queuePtr, float item);
float serve(Queue* queuePtr);

#endif
```

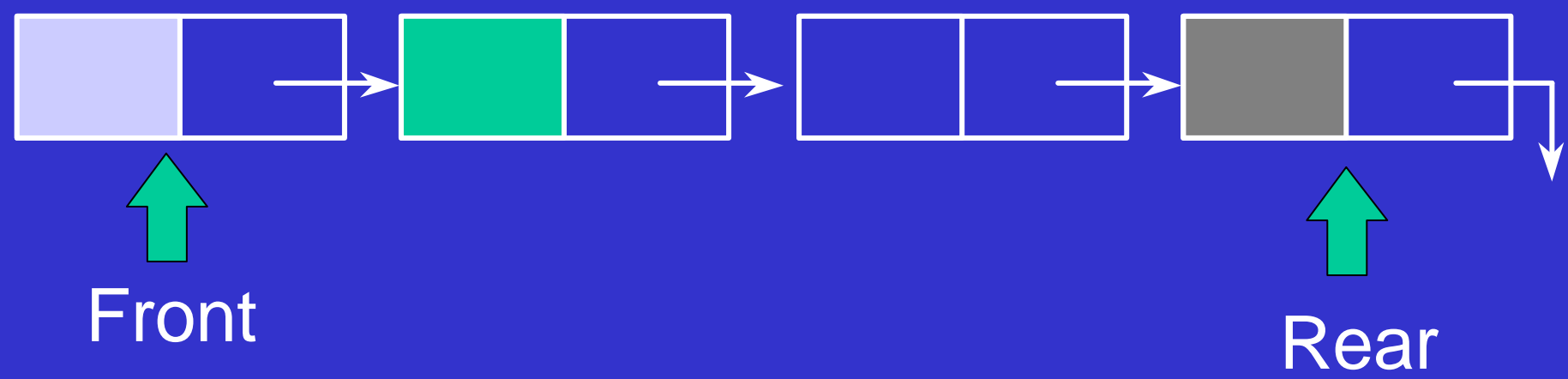
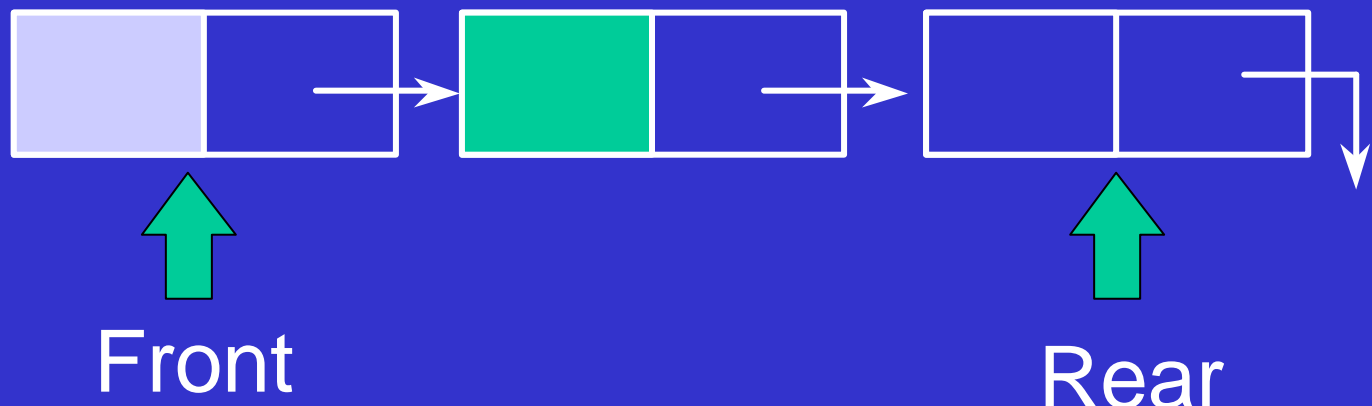
Initialize Queue

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

Append



Append



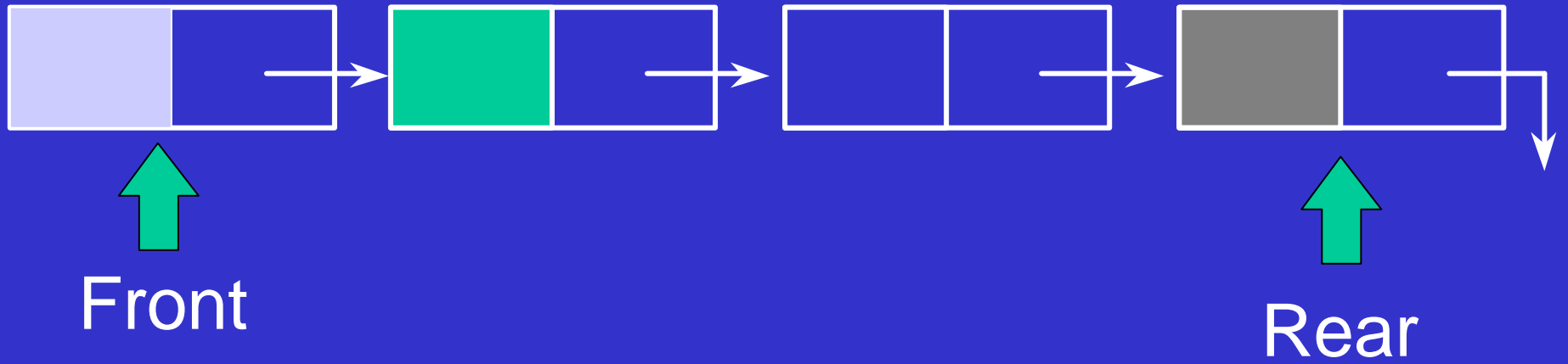
Append

- 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**

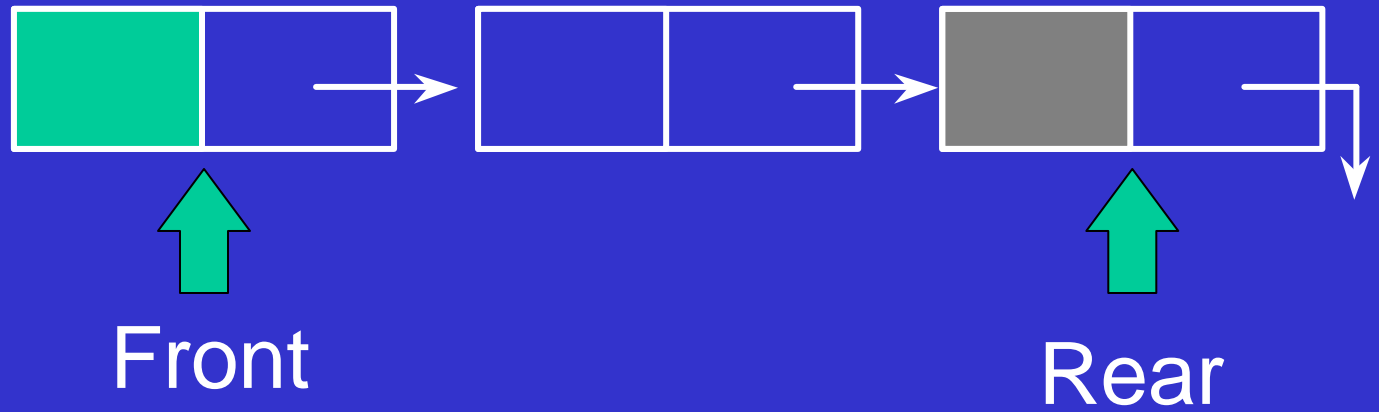
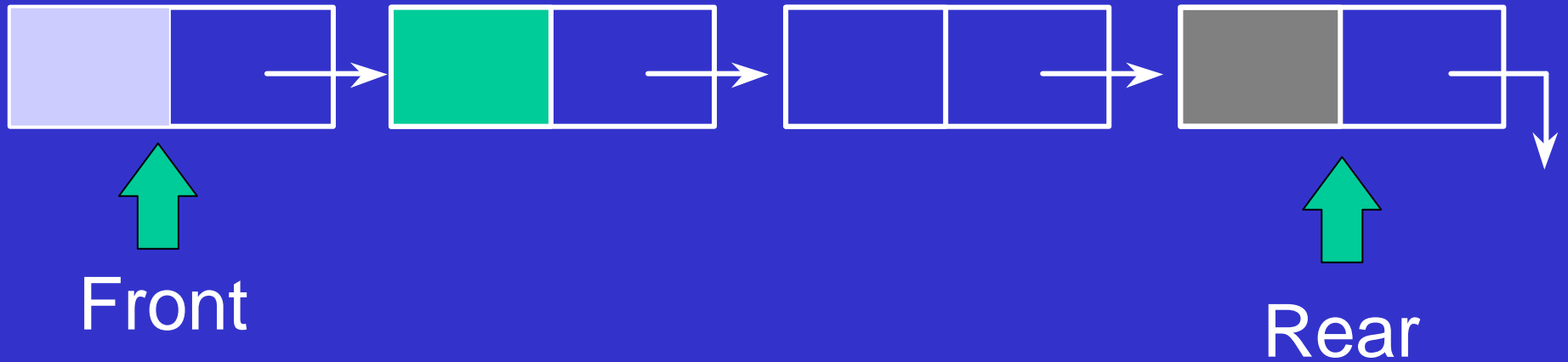
```
void append(Queue* queuePtr, float item)
{
    Node* newNodePtr = makeNode(item);

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

Serve



Serve



Serve

- check that the queue is not empty
- remember the **item** in the front node
- remember the **address** of the front node
- make the next node the **new front**
- **free** the old front node
- decrement **count**
- if queue is now empty, set **rear** to NULL
- return the **item**

```
float serve(Queue* queuePtr)
{
    float item;
    Node* oldNodePtr = queuePtr->frontPtr;

    if (queueEmpty(queuePtr)) {
        fprintf(stderr, "Queue is empty\n");
        exit(1);
    }
    else {
        item = oldNodePtr->value;
        queuePtr->frontPtr = oldNodePtr->nextPtr;
        queuePtr->count--;
        free(oldNodePtr);
        if (queuePtr->count == 0) {
            queuePtr->rearPtr = NULL;
        }
    }
    return item;
}
```

```
#include <stdio.h>
#include "linkedqueue.h"

main()
{
    Queue  theQueue;
    float  item;

    initializeQueue(&theQueue);

    while (scanf("%f", &item) != EOF)
    {
        append(&theQueue, item);
    }

    while (!queueEmpty(&theQueue))
    {
        item = serve(&theQueue);
        printf("%f\n", item);
    }
}
```

Revision

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

Preparation

- Read 5.2.2 - 5.2.4 in Kruse et al.