

CSE1303 Part A
Data Structures and Algorithms
Summer Semester 2003

Lecture A4 – Basic Data Structures – Continued (Queues)

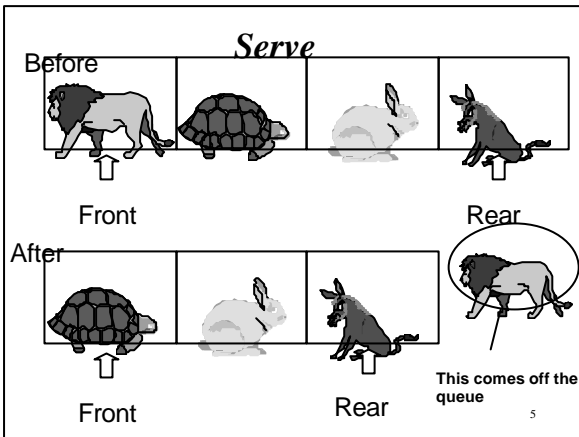
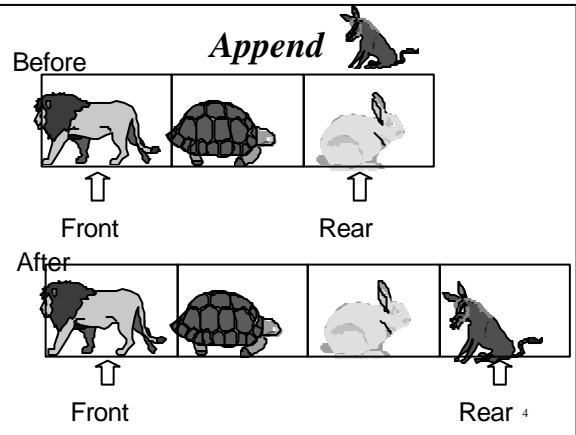
Kymerly Fergusson

Basic Data Structures

- Stacks
- Queues
- Lists

Overview

- What is a Queue?
- Queue Operations.
- Applications.
- Linear Implementation.
- Circular Implementation.



Operations

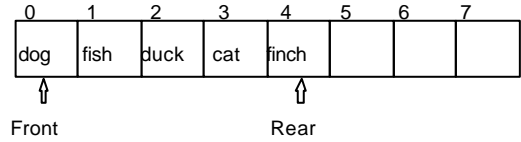
- Initialize the queue.
- Append an item to the rear of the queue.
- Serve an item from the front of the queue.
- Is the queue empty?
- Is the queue full?
- What size is the queue?

Applications

- In operating systems, e.g. printer queues, process queues, etc.
- Simulation programs.
- Algorithms.

7

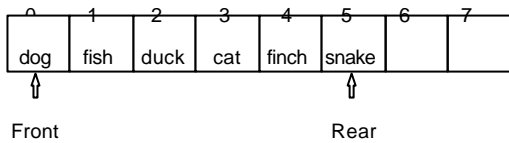
Linear Implementation



8

Append

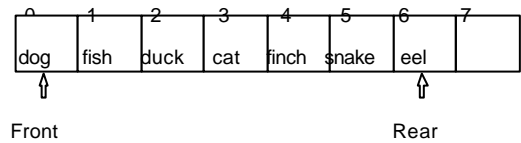
snake



9

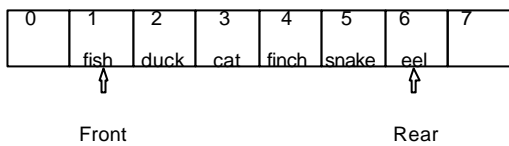
Append

eel



10

Serve

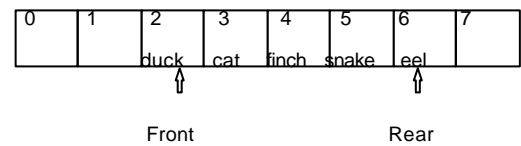


dog

This comes off the queue

11

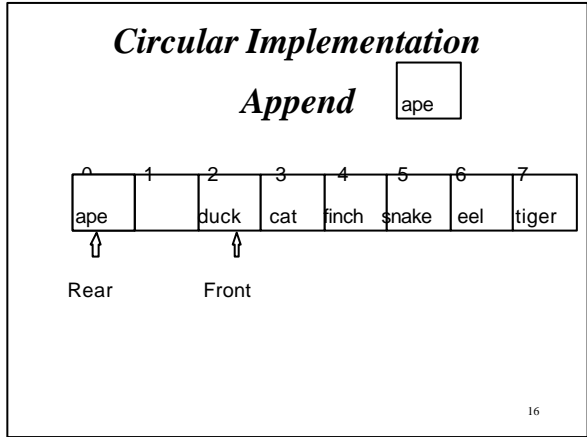
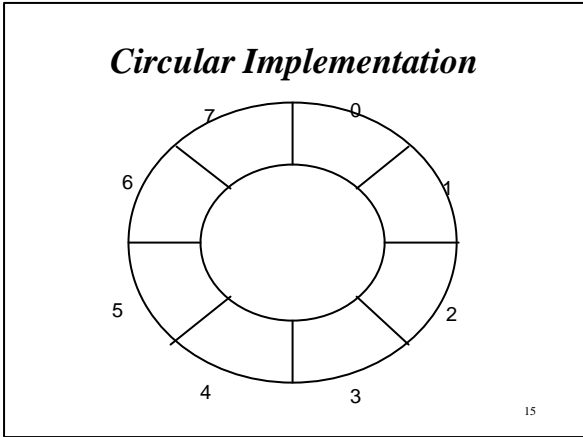
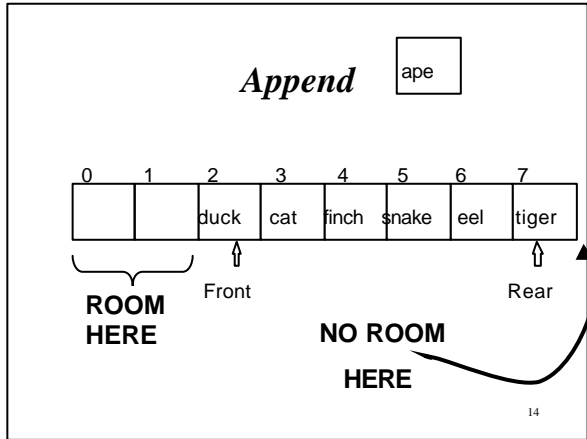
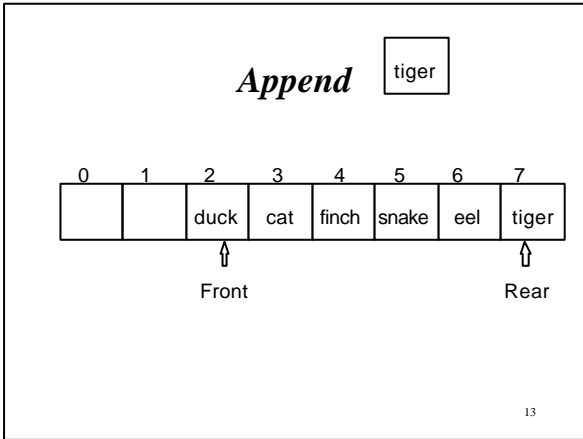
Serve



fish

This comes off the queue

12



```
#define MAXQUEUE 20

struct QueueRec
{
    int    count;
    int    front;
    int    rear;
    float  entry[MAXQUEUE];
};

typedef struct QueueRec Queue;
```

Q

count:

front:

rear:

entry:

17

```
#ifndef QUEUEH
#define QUEUEH
#include <stdbool.h>
#define MAXQUEUE 20

struct QueueRec
{
    int    count;
    int    front;
    int    rear;
    float  entry[MAXQUEUE];
};

typedef struct QueueRec 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
```

18

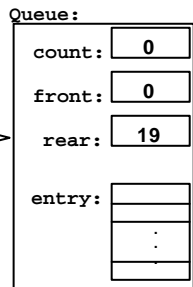
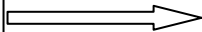
```

#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

void initializeQueue(Queue* queuePtr)
{
    queuePtr -> count = 0;
    queuePtr -> front = 0;
    queuePtr -> rear = MAXQUEUE-1;
}
queuePtr:

```

addrOf Queue



19

```

bool queueEmpty(const Queue* queuePtr)
{
    if (queuePtr->count <= 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
bool queueFull(Queue* queuePtr)
{
    if (queuePtr->count >= MAXQUEUE)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

20

```

void append(Queue* queuePtr, float item)
{
    if (queueFull(queuePtr))
    {
        fprintf(stderr, "Queue is full\n");
        exit(1);
    }
    else
    {
        queuePtr->rear++;
        if (queuePtr->rear == MAXQUEUE)
        {
            queuePtr->rear = 0;
        }
        queuePtr->entry[queuePtr->rear] = item;
        queuePtr->count++;
    }
}

```

21

```

float serve(Queue* queuePtr)
{
    float item;

    if (queueEmpty(queuePtr)) {
        fprintf(stderr, "Queue is empty\n");
        exit(1);
    }
    else {
        item = queuePtr->entry[queuePtr->front];
        queuePtr->front++;
        if (queuePtr->front == MAXQUEUE)
        {
            queuePtr->front = 0;
        }
        queuePtr->count--;
    }
    return item;
}

```

22

Revision

- Queue
- Main Operations
- Implementation.

Revision: Reading

- Kruse: Chapter 4.1 to 4.2
- Deitel & Deitel: Chapter 12.6
- Standish: Chapter 7
- Langsam: Chapter 4.1

Preparation

Next lecture: Lists

- Kruse 4.5, 4.6, 4.8
- Deitel & Deitel(2e) 12.1, 12.2, 12.4

23