Multiple Stacks

ICT 4303

Multiple Stacks

```
class twoStacks
{
    int arr[size];
    int top1, top2;
public:
  twoStacks()  // constructor
  {
      top1 = -1;
      top2 = size;
  }
```

Multiple Stacks

```
// Method to push an element x to stack1
  void push1(int x)
  {
     // There is at least one empty space for new element
     if (top1 < top2 - 1)
     {
        top1++;
        arr[top1] = x;
     }
     else
     {
        cout << "Stack Overflow";

     }
}
```

Multiple Stacks

```
// Method to push an element x to stack2
  void push2(int x)
  {
     // There is at least one empty space for new element
     if (top1 < top2 - 1)
     {
        top2--;
        arr[top2] = x;
     }
     else
     {
        cout << "Stack Overflow";

     }
  }
```

Multiple Stacks

```
// Method to pop an element from first stack
  void pop1()
  {
     if (top1 >= 0 )
     {
       int x = arr[top1];
       top1--;
       cout<<“popped element is”<<x;
     }
     else
     {
        cout << "Stack UnderFlow";


     }
  }
```

Multiple Stacks

```
// Method to pop an element from second stack
  void pop2()
  {
     if (top2 < size)
     {
       int x = arr[top2];
       top2++;
       cout<<“popped element is”<<x;
     }
     else
     {
        cout << "Stack UnderFlow";


     }
  }
};
```

Multiple Stacks

```
int main()
{
    twoStacks ts;
    ts.push1(5);
    ts.push2(10);
    ts.push2(15);
    ts.push1(11);
    ts.push2(7);
    cout << "Popped element from stack1 is " << ts.pop1();
    ts.push2(40);
    cout << "\nPopped element from stack2 is " << ts.pop2();
    return 0;
}
```

Multiple Stacks

n>=3

```
#include<iostream>
using namespace std;
#define max_size 20
class stack
{
   int top[10];
   int a[50];
   int boundary[10];
   public:
      stack(int);
      void push(int ,int);
      void pop(int);
      void display(int);
};
```

Multiple Stacks

n>=3

```
stack::stack(int n)
{
    for(int i=0;i<n;i++)
    boundary[i]=top[i]=(max_size/n)*i-1;
}
```

Multiple Stacks

n>=3

```
void stack::push(int i,int x)
{
        if((top[i]==boundary[i+1])||(top[i]==(max_size-1)))
                cout<<"Stack is full \n";
        else
                a[++top[i]]=x;
}
```

Multiple Stacks

n>=3

```
void  stack::pop(int i)
{
   if(top[i]==boundary[i])
        cout<<"stack is empty\n";
   else
        cout<<"deleted element is "<<(a[top[i]--]);
}
```

Multiple Stacks

n>=3

```
void stack::display(int i)
{
   if(top[i]==boundary[i])
            cout<<"stack is empty\n";
   else
            for(int j=top[i];j>boundary[i];j--)
            cout<<"\nThe elements of stack are "<<"\n"<<a[j];
}
```

Multiple Stacks

n>=3

```
int main()
{
    stack s(5);
    s.push(1,10);
    s.push(2,30);
    s.push(3,40);
    s.push(3,28);
    s.pop(3);
    s.display(2);
    return 0;
}
```

Queues

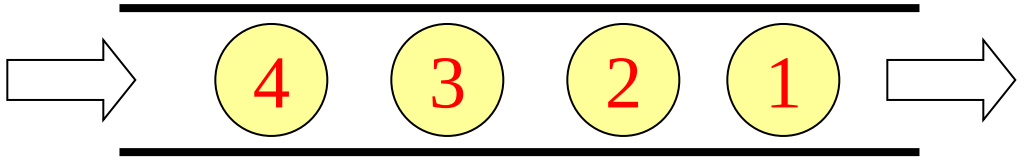
ICT 4303

Contents

- Definition
- Operations on Queues
- Types of Queues
- Linear Queue
- Circular Queue

What is a Queue?

- It is an ordered group of homogeneous items of elements.
- Queues have two ends:
  - Elements are added at one end, indicated by ***rear***.
  - Elements are removed from the other end, indicated by ***front***.
- The element added first is also removed first **(FIFO: First In, First Out).**

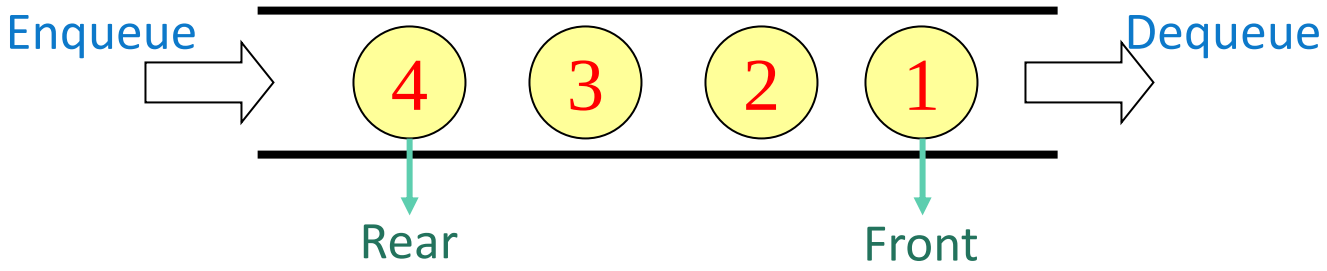


Operation on Queues

- MAXSIZE: Maximum number of items that might be on the queue.
- ItemType: Data type of the items on the queue.

Operations

- MakeEmpty()
- Boolean IsEmpty()
- Boolean IsFull()
- Enqueue ()
- Dequeue ()
- Peek()



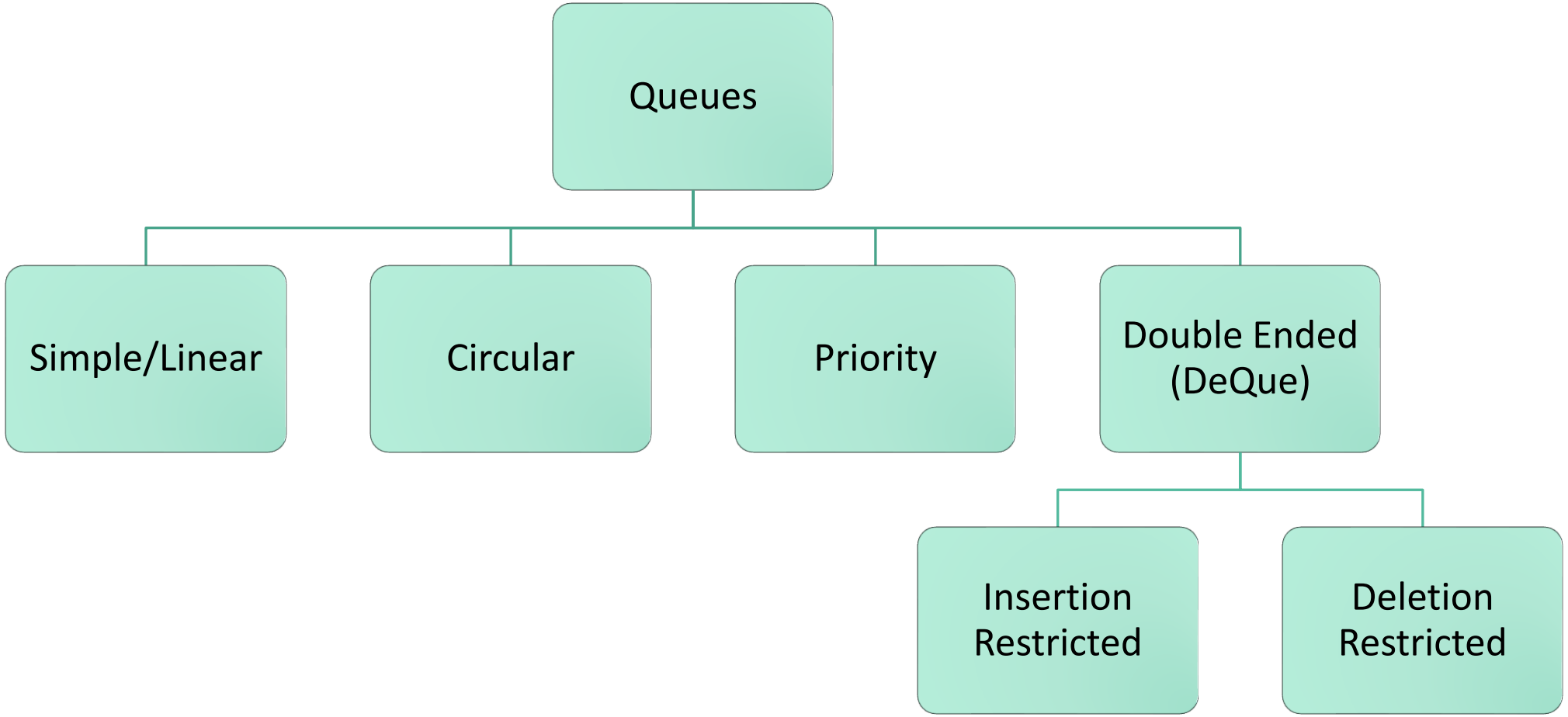
Queue

- Applications
  - Waiting List (CPU, Printer, or any other resources)
  - Song playlist
  - Interrupts

- Time Complexity

Enqueue, Dequeue	O (1)
Search	O (n)

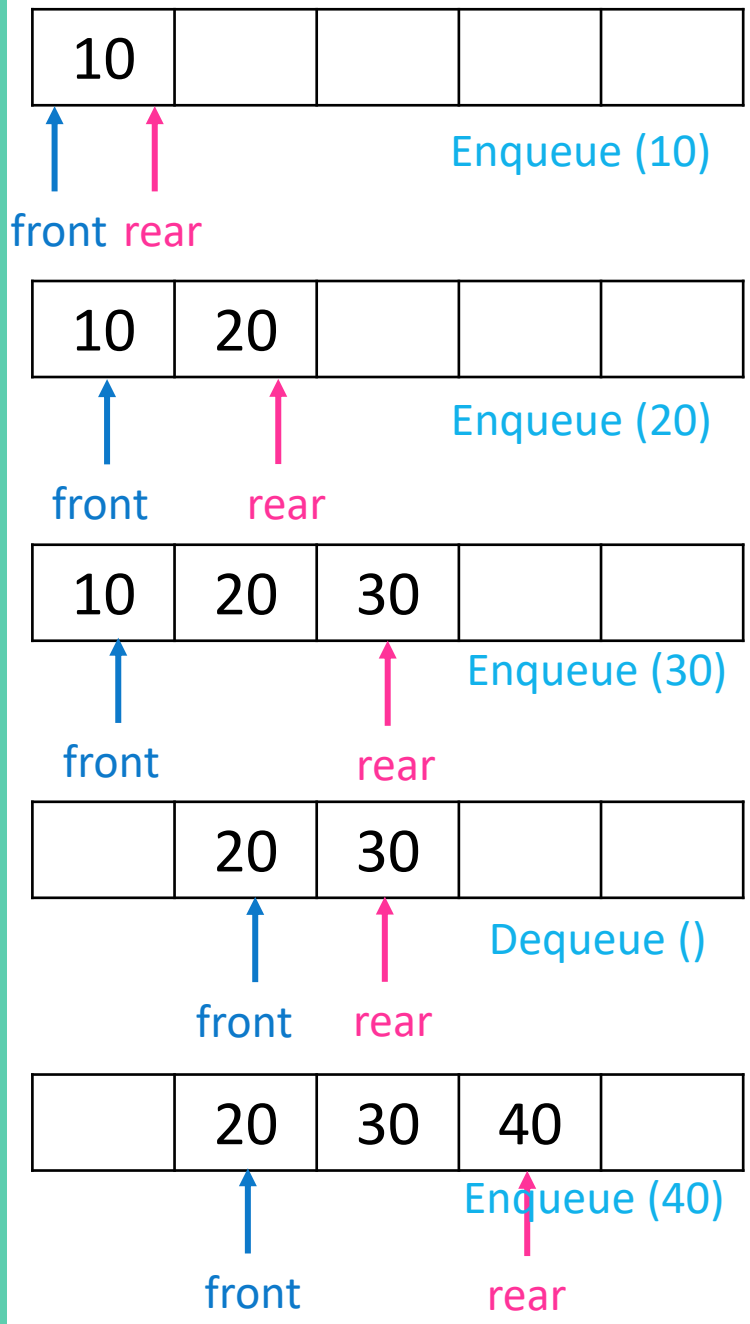
Types of Queues

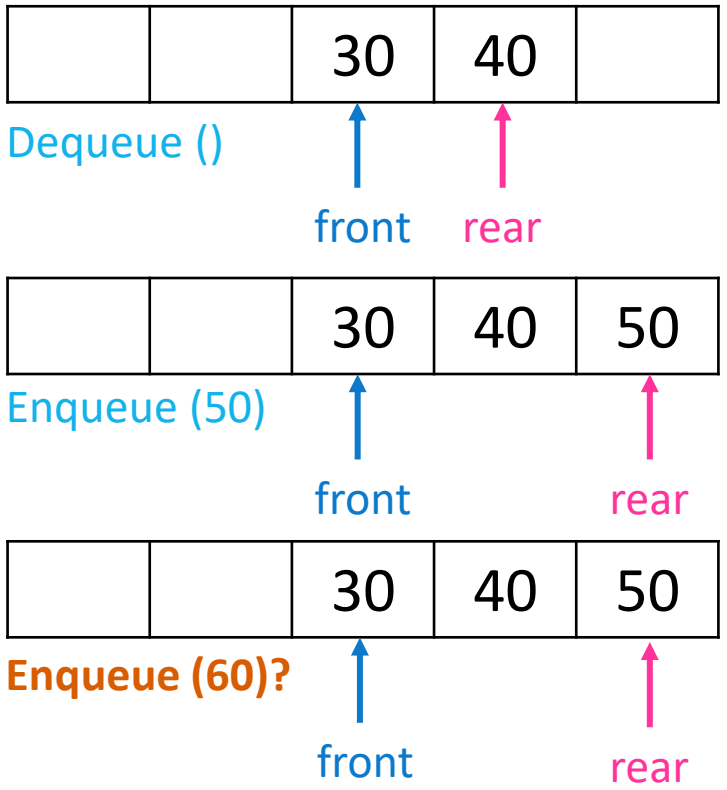


Simple Queue Operations

Initially, rear =-1, front =-1

1. Enqueue (10)
2. Enqueue (20)
3. Enqueue (30)
4. Dequeue()
5. Enqueue(40)
6. Dequeue()
7. Enqueue(50)
8. Enqueue(60)



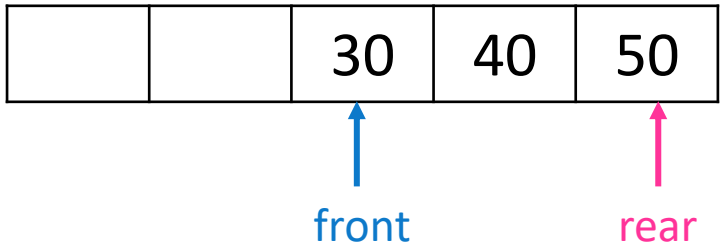


Though there are empty slots available, 60 could not be enqueued.

Queue Underflow and Overflow

- Overflow Condition

  rear = MAXSIZE-1

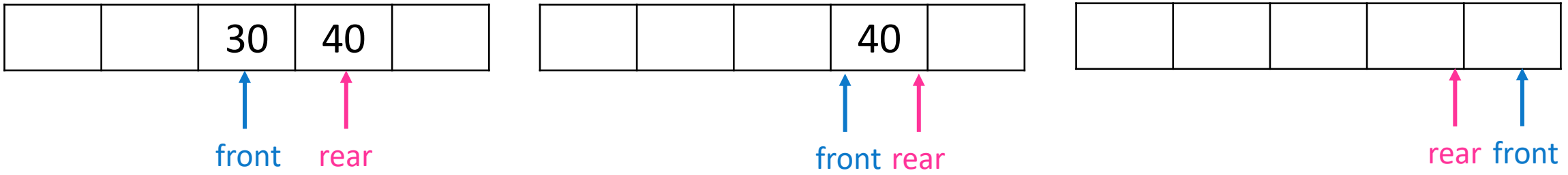


- Underflow Condition

  front = -1 or front > rear

Case I:

Case II:



Queue Methods

```
int enqueue(int x){
        if (rear == MAXSIZE-1)
                cout<<“Overflow”;
        else{
                if(front == -1 && rear == -1){
                        front = 0;
                        rear = 0;
                        }
                else
                        rear = rear +1;
                q[rear] = x;
        }
}
```

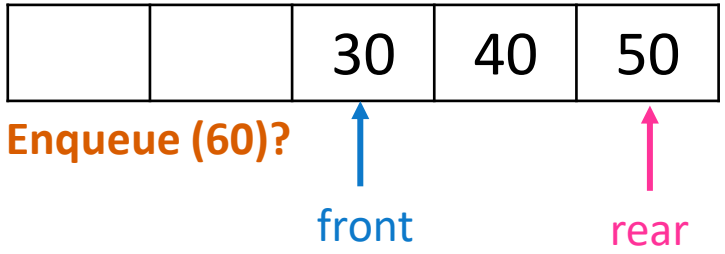
Queue Methods

```
int dequeue(int x){
        if (front == -1 || front >rear)
                cout<<“Underflow”;
        else{
                int value = q[front];
                if(front == rear){
                        front = -1;
                        rear = -1;
                        }
                else
                         front = front +1;
        }
}
```

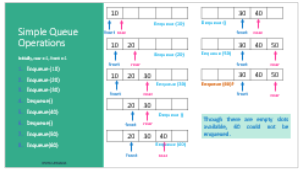
Queue Methods

```
int display(int x){
        if (!(IsEmpty()){
                for(int i=front; i<=rear; i++)
                        cout<<q[i]<<“\t”;
        }
}
```

Circular Queue



- The problem faced in linear queue is addressed using circular queue.



Circular Queue Methods

```
int enqueue(int x){
        if (rear +1 % MAXSIZE == front)
                cout<<“Overflow”;
        if(front == -1 && rear == -1){
                front = 0;
                rear = 0;
                }
        elseif(rear = MAXSIZE-1 && front !=0)
                rear=0;
        else
                rear = rear +1%MAXSIZE;
                q[rear] = x;
}
```

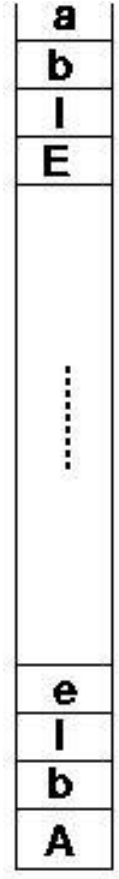
Queue Methods

```
int dequeue(int x){
        if (front == -1)
                cout<<“Underflow”;
        int value = q[front];
        if(front == rear){
                front = -1;
                rear = -1;
                }
        elseif (front==MAXSIZE-1)
                front = 0;
        else
                 front = front +1;
}
```

Example: recognizing palindromes

- We will read the line of text into a stack and a queue.
- Compare the contents of the stack and the queue character-by-character to see if they would produce the same string of characters.

Example: recognizing palindromes



Stack

Queue

Example: recognizing palindromes

Check if the code gives expected results.

```
#include <iostream.h>
#include <ctype.h>
#include "stack.h"
#include "queue.h“
int main()
{
 StackType<char> s;
 QueType<char> q;
 char ch;
 char sItem, qItem;
 int mismatches = 0;
 cout << "Enter string: " << endl;
 while(cin.peek() != '\\n') {
   cin >> ch;
   if(isalpha(ch)) {
     if(!s.IsFull())
       s.Push(toupper(ch));
     if(!q.IsFull())
q.Enqueue(toupper(ch));
   }
}
```

Example: recognizing palindromes

```
while( (!q.IsEmpty()) && (!s.IsEmpty()) ) {

  s.Pop(sItem);
  q.Dequeue(qItem);

  if(sItem != qItem)
    ++mismatches;
}
if (mismatches == 0)
  cout << "That is a palindrome" << endl;
else
 cout << That is not a palindrome" << endl;

return 0;
}
```

Books

- Ellis Horowitz, Sartaj Sahni, Susan Anderson-Freed, Fundamentals of Data structures in C (2e), Silicon Press, 2008.
- Ellis Horowitz, Sartaj Sahni, Dinesh Mehta, Fundamentals of Data Structures in C++ (2e), Galgotia Publications, 2008.