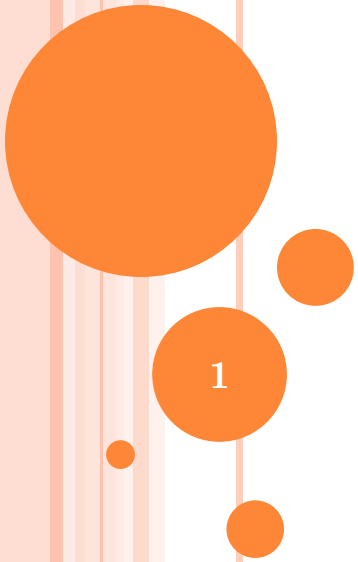
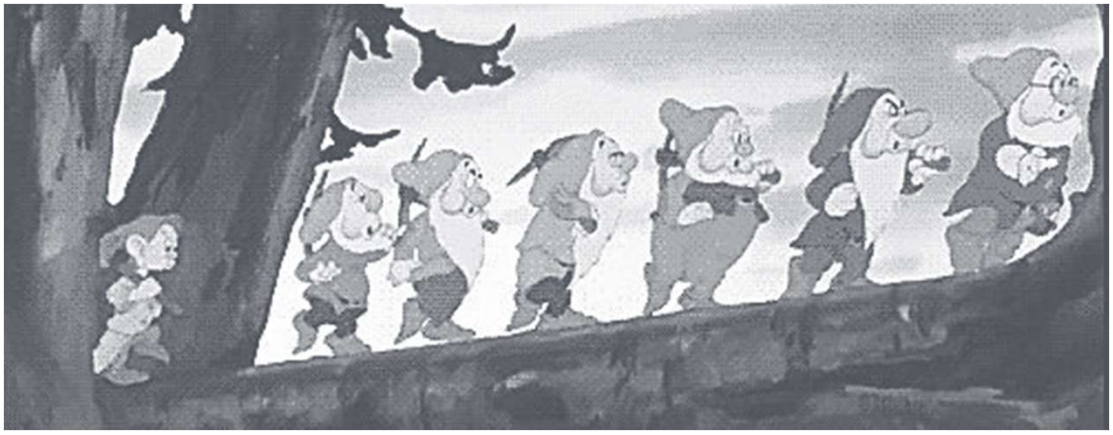


QUEUES

CS 250 – C++ Programming 2

The ADT Queue

- **Queues data structure**
  - Elements are **inserted** to the **rear** of the **queue**.
  - Elements are **removed** from the **front** of the **queue**.
  - First In First Out (**FIFO**)
- Representation of typical "line" forming
  - Like bank teller lines, movie theatre lines, etc.



Which Operations Are Needed?

- There are only a few operations needed for the **ADT queue**:
  - Test whether a queue is empty
  - Add a new item to the back of the queue
  - Remove the item at the front of the queue
    (the item that was added earliest)
  - Get the entry that was added earliest to the queue

STL Queue

- The **Standard Template Library** (STL) provides a **class** to implement a **queue**.
  - It is a **template** class

```
#include <queue>

...

queue<double> doubleQueue;     // creates a queue of doubles

queue<Student> studentQueue;   // creates a queue of objects
                               // of the class Student
```

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.
pop( )	**Removes** the **element** at the **front** of the queue.

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.
pop( )	**Removes** the **element** at the **front** of the queue.
empty( )	Returns **true** if the queue is **empty**, and returns **false** otherwise.

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.
pop( )	**Removes** the **element** at the **front** of the queue.
empty( )	Returns **true** if the queue is **empty**, and returns **false** otherwise.
front( )	**Retrieves** (**without removing**) the element at the **front** of the queue.

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.
pop( )	**Removes** the **element** at the **front** of the queue.
empty( )	Returns **true** if the queue is **empty**, and returns **false** otherwise.
front( )	**Retrieves** (**without removing**) the element at the **front** of the queue.
back( )	**Retrieves** (**without removing**) the element at the **rear** of the queue.

Queue Operations

Operation	What it does
push(obj)	**Inserts** a new **element** to the **rear** of the queue.
pop( )	**Removes** the **element** at the **front** of the queue.
empty( )	Returns **true** if the queue is **empty**, and returns **false** otherwise.
front( )	**Retrieves** (**without removing**) the element at the **front** of the queue.
back( )	**Retrieves** (**without removing**) the element at the **rear** of the queue.
size( )	Returns the **number of elements** in the queue.

Tracing Code

We will create a **queue** of **integers**, **myQueue**

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

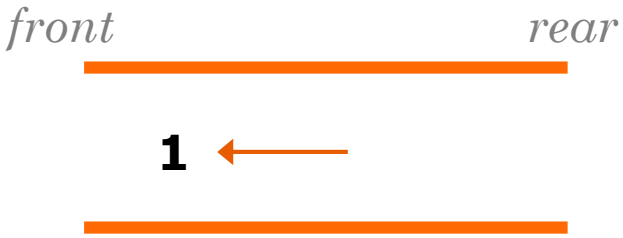
Tracing Code

front *rear*

This is our **queue** of **integers** (now empty).

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

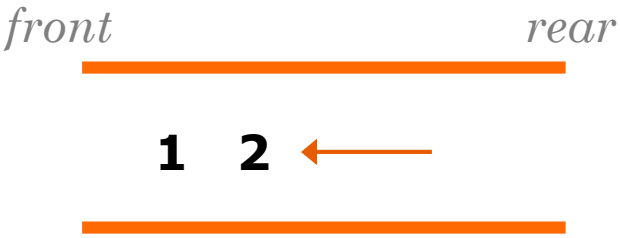
Tracing Code



We **push** integer **1** into the **queue**.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

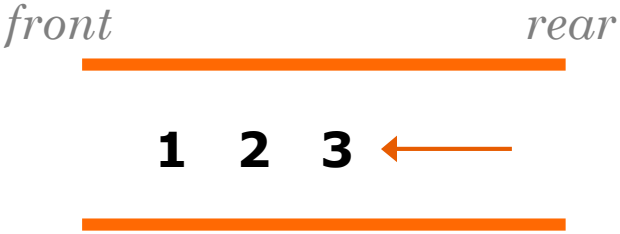
Tracing Code



We **push** integer **2** into the **queue**.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

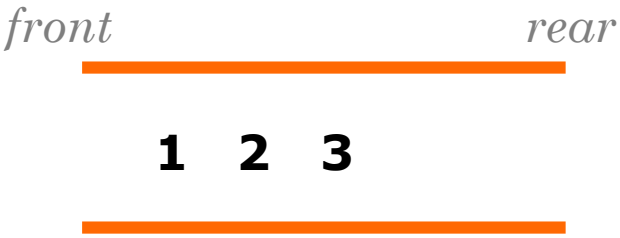
Tracing Code



We **push** integer **3** into the **queue**.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Tracing Code



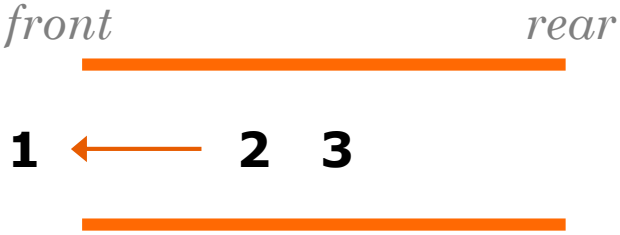
We **retrieve** (*without* removing) the **element** at the **front** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1

Tracing Code



We **pop** the **element** at the **front** of the **queue.**

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1

Tracing Code

We **retrieve** (*without* removing) the **element** at the **front** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2

Tracing Code

We **retrieve** (*without* removing) the **element** at the **rear** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3

Tracing Code

We **push** integer 4 into the **queue**.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3

Tracing Code

WHILE statement will execute as long as the **queue** is **not** empty.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3

Tracing Code

Retrieve (*without* removing) the **element** at the **front** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2

Tracing Code

Pop the **element** at the **front** of the **queue.**

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2

Tracing Code

Retrieve (*without* removing) the **element** at the **front** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2 3

Tracing Code

Pop the **element** at the **front** of the **queue.**

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2 3

Tracing Code

Retrieve (*without* removing) the **element** at the **front** of the **queue** and print it.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2 3 4

Tracing Code

Pop the **element** at the **front** of the **queue.**

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2 3 4

Tracing Code

front *rear*

Queue is now **empty**; **WHILE** statement ends.

```
queue<int> myQueue;
myQueue.push(1);
myQueue.push(2);
myQueue.push(3);
cout << myQueue.front();
myQueue.pop();
cout << myQueue.front();
cout << myQueue.back();
myQueue.push(4);
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
```

Output:

1 2 3 2 3 4

Implementing a Queue

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?

Implementing a Queue (cont.)

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?
    - Typical implementation: A **circular** **array**

Implementing a Queue (cont.)

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?
    - Typical implementation: A **circular** **array**
  - A **linked list**
    - How would you insert the elements to the back of the queue?

Implementing a Queue (cont.)

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?
    - Typical implementation: A **circular** **array**
  - A **linked list**
    - How would you insert the elements to the back of the queue?
    - In a singly-linked list, the **front** can be the the **first** node

Implementing a Queue (cont.)

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?
    - Typical implementation: A **circular** **array**
  - A **linked list**
    - How would you insert the elements to the back of the queue?
    - In a singly-linked list, the **front** can be the the **first** node
    - What else do you need?

Implementing a Queue (cont.)

- Possible ways to implement a **queue**:
  - An **array**
    - How would you insert the elements to the back of the queue?
    - Typical implementation: A **circular** **array**
  - A **linked list**
    - How would you insert the elements to the back of the queue?
    - In a singly-linked list, the **front** can be the the **first** node
    - What else do you need?
    - Need a pointer to the **back** of the list → **rear** of the queue

Common Operation Identifiers

- Other identifiers used for common operations on the queue:
  - **empty**( ) = **isEmpty( )**
  - **push**(*e*) = **enqueue(*e*)**
  - **pop**( ) = **deque( ), dequeue( )**
- **Note** that in some implementations the function **pop( )** returns a value at the front **and** removes it as well.

Queue Applications

- **Queues** are used in many **applications**:
  - Buffering
    - A "holding area" between processes
      - Example: documents waiting to be printed
  - Simulations
    - Run simulation programs to produce estimate on processes
      - Example: Estimating waiting times in a bank to determine whether there is a need for more tellers.
  - And more…

Queues (end)