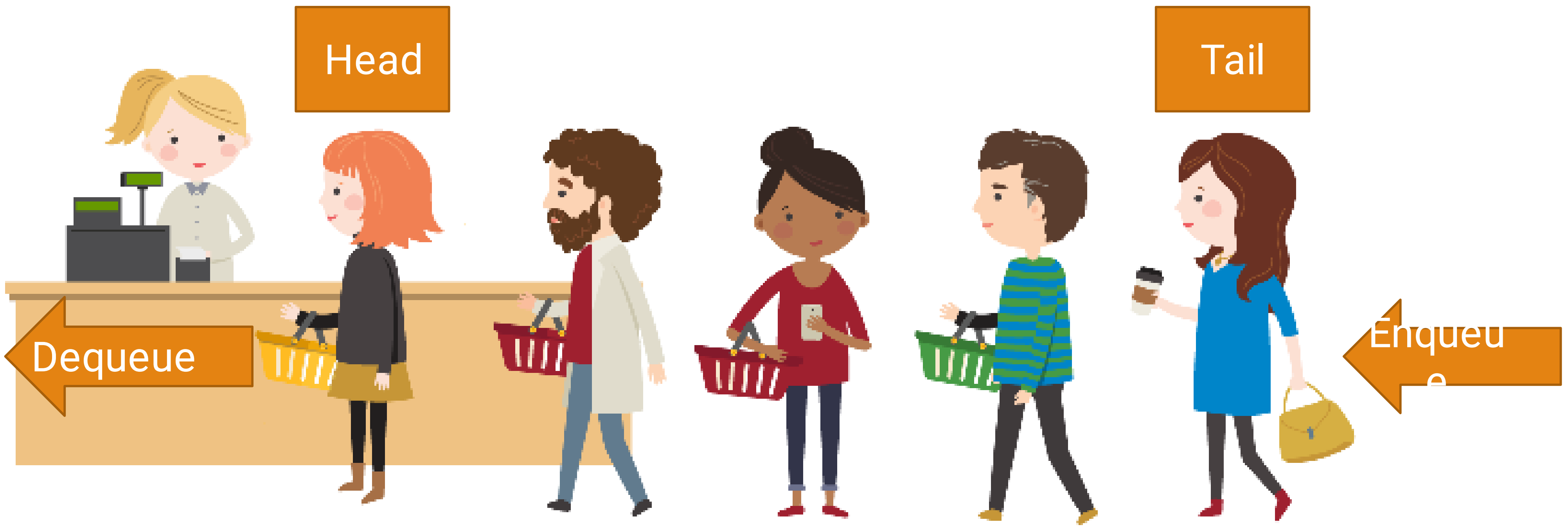
Data Structure

Lec 03 Queue



Queue

Introduction to Queue

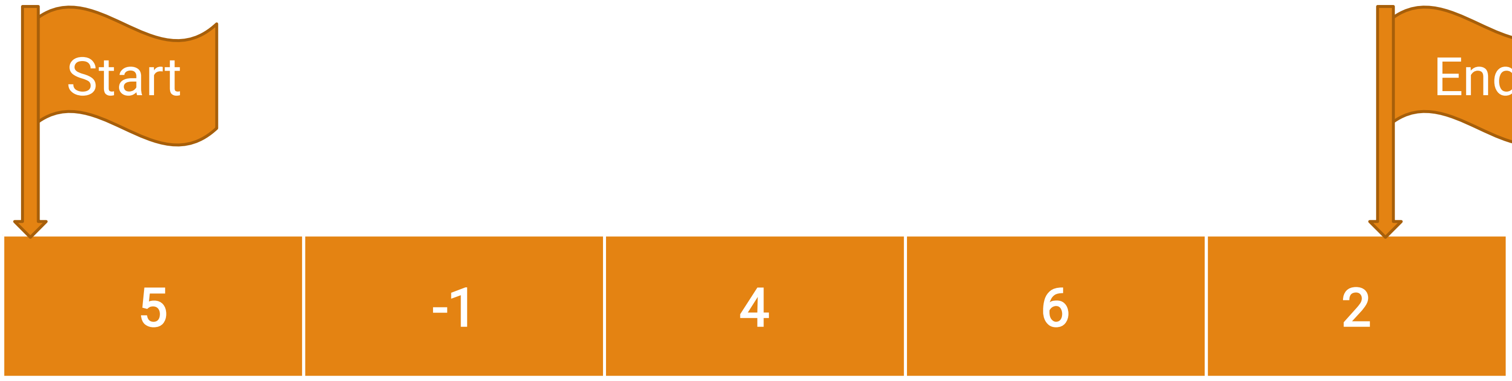
How to store them in Memory?? ???

Introduction to Queue

Where is the start
& end of the queue?

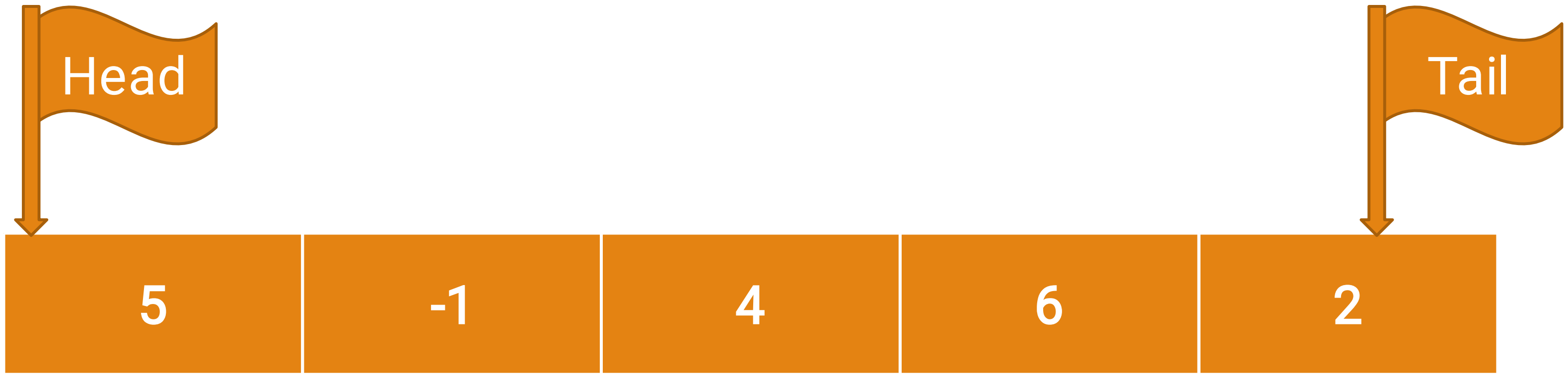
5	-1	4	6	2

Introduction to Queue

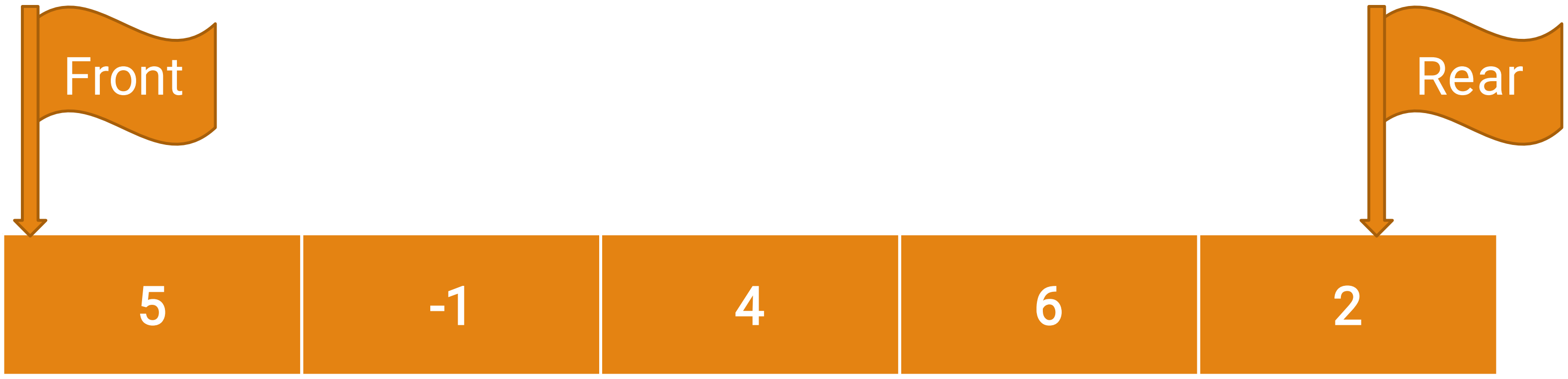


To solve the problem, use Start and End Flags

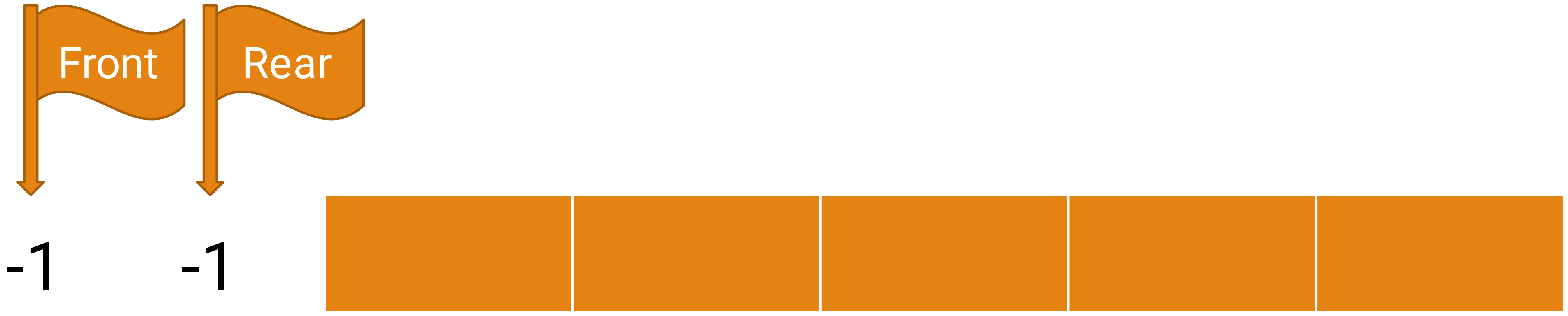
Introduction to Queue



Introduction to Queue



Introduction to Queue

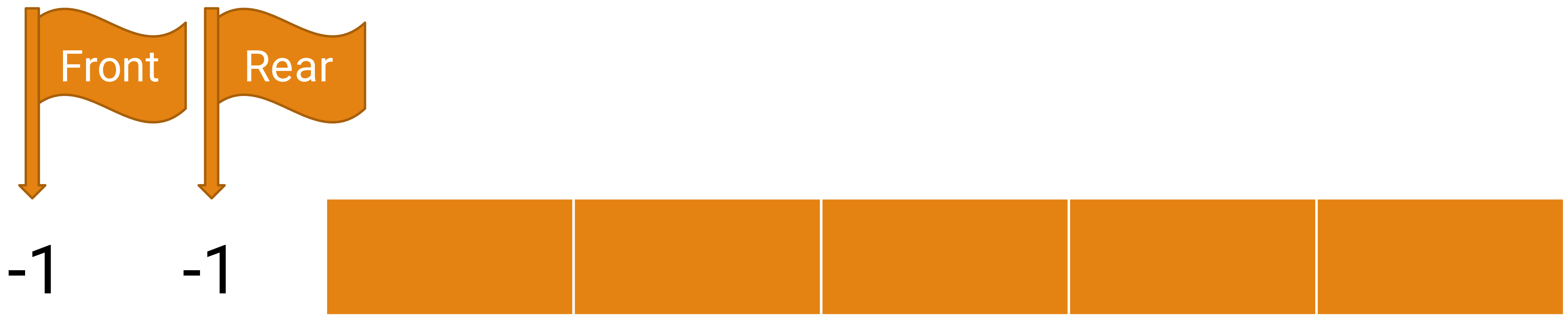


Introduction to Queue

- Queue is an *ordered collection of items* in which *new data items are added at the end*, or *tail*, of the queue while *other data are removed from the front*, or *head*, of the queue. For this reason, a queue is referred to as a FIFO structure(First-In First-Out).
- The main primitive operations of a queue are known as:
  - Insert(): adds an item to the queue.
  - Remove(): deletes an item from the queue.
- Additional primitives can be defined:
  - Is_Empty(): reports whether the queue is empty
  - Is_Full(): reports whether the queue is full

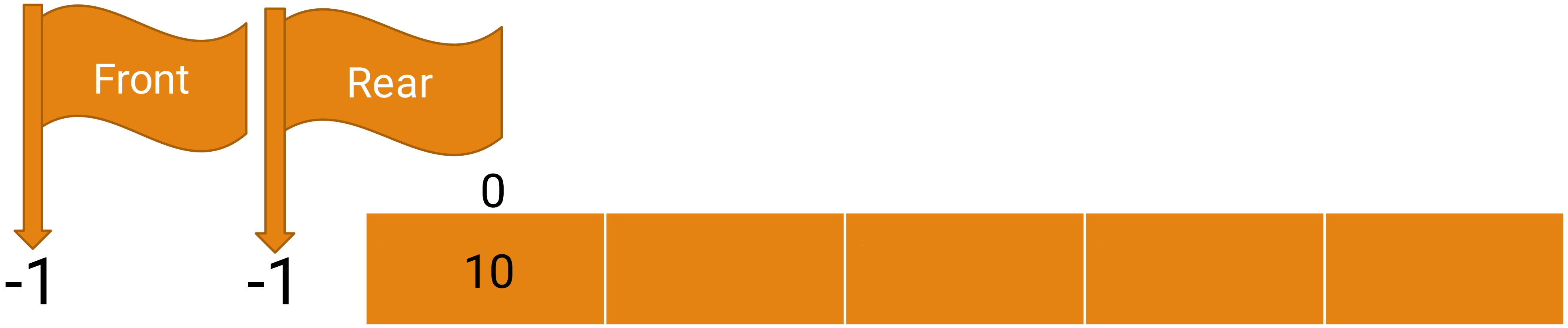
How a Queue Works?

Empty Queue



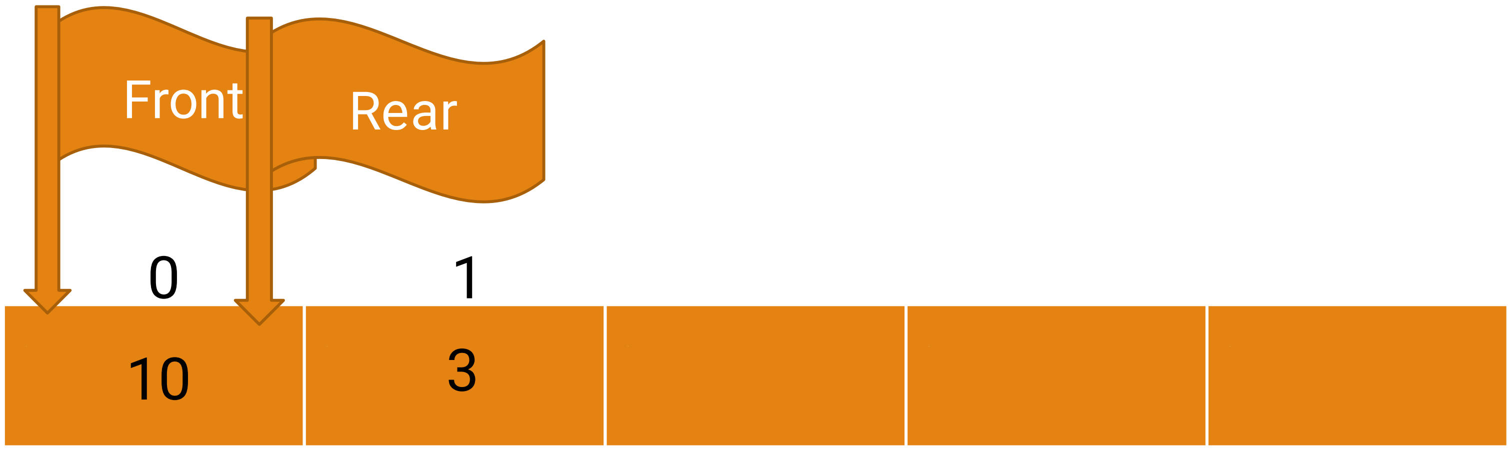
How a Queue Works?

Inserting 10 to the Queue



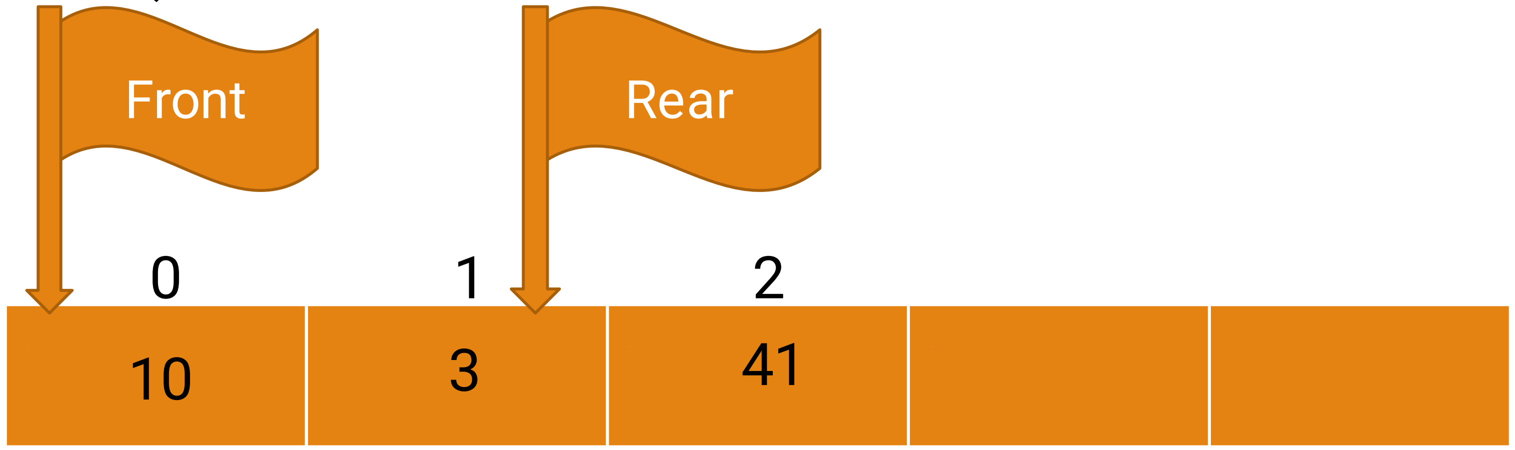
How a Queue Works?

Inserting 3 to the Queue



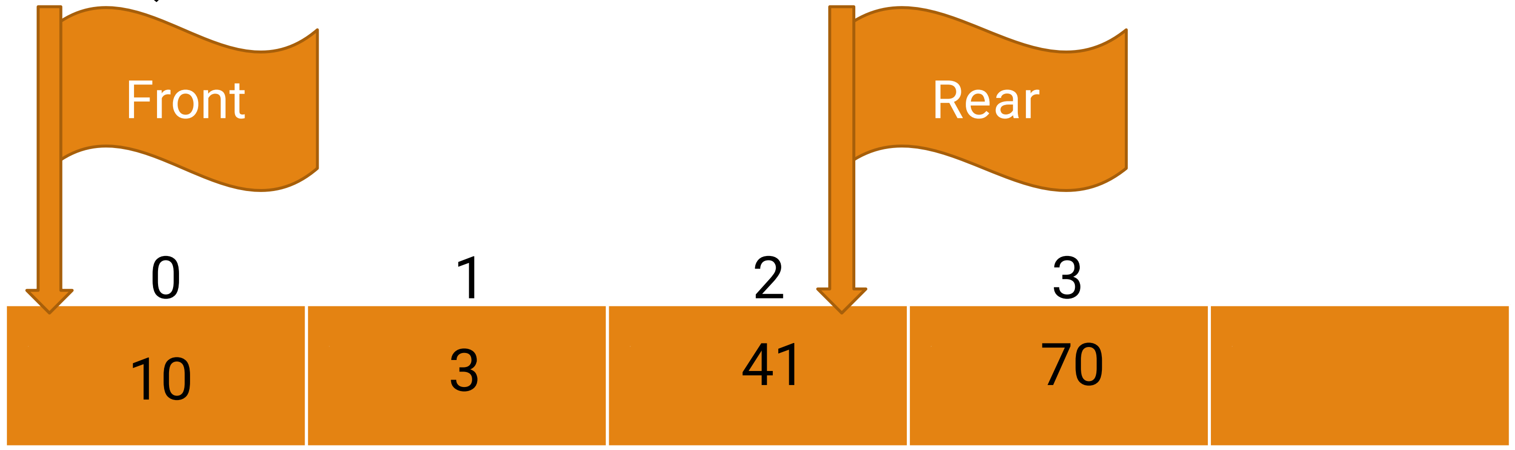
How a Queue Works?

Inserting 41 to the Queue



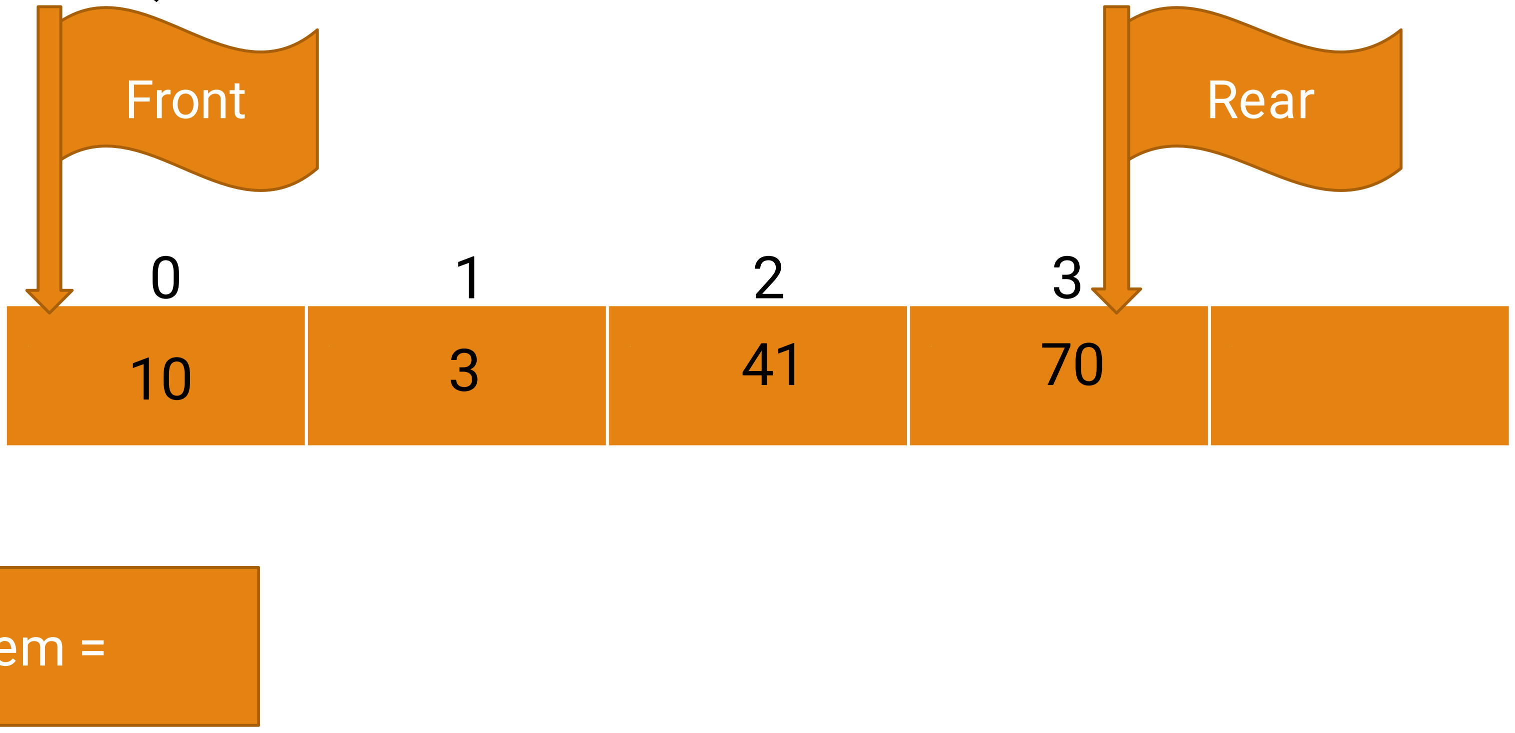
How a Queue Works?

Inserting 70 to the Queue



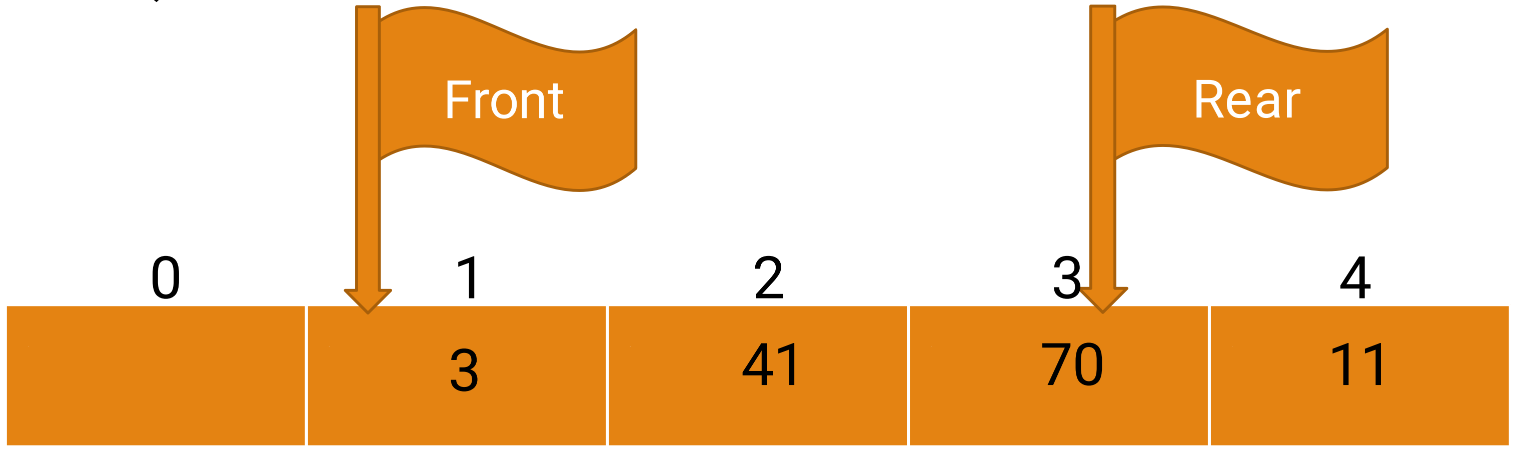
How a Queue Works?

Deleting Item from the Queue



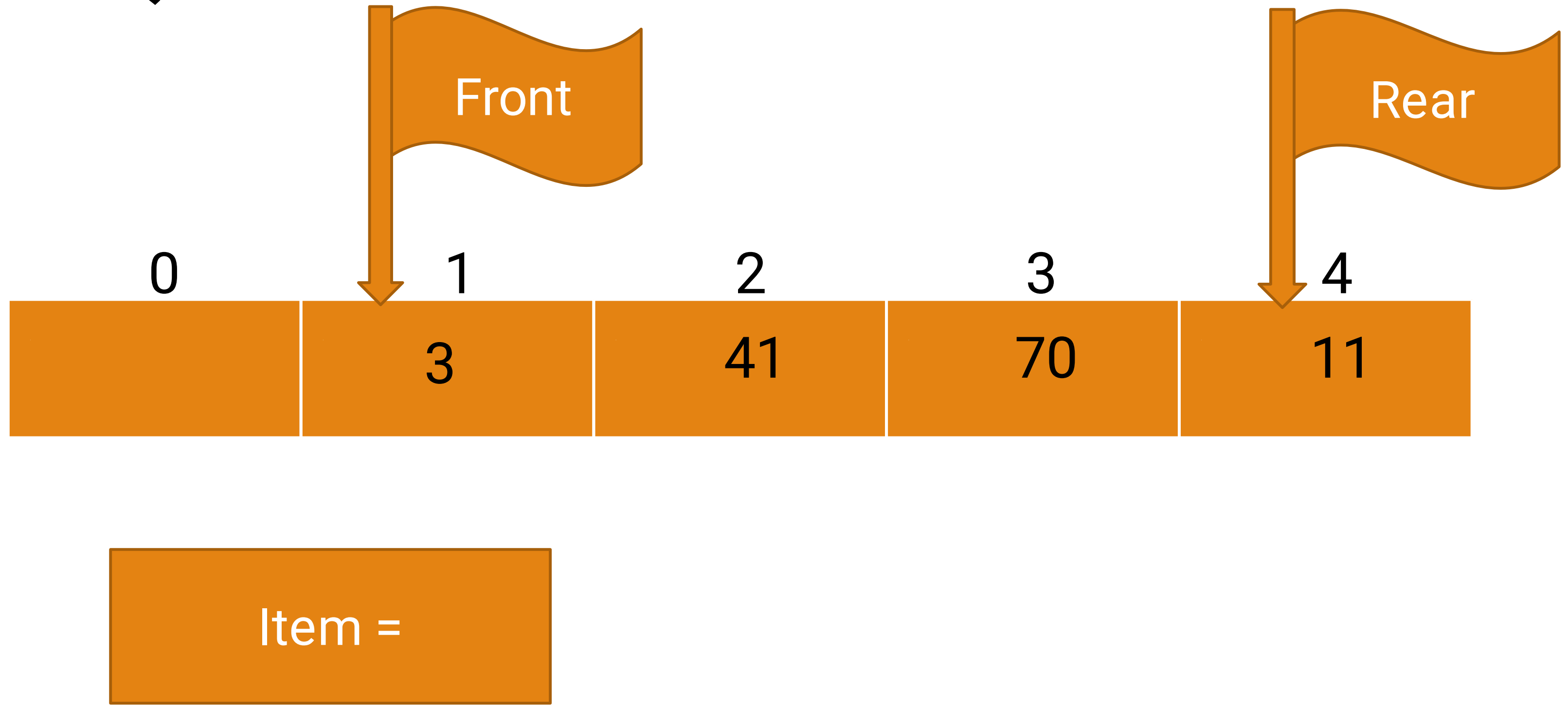
How a Queue Works?

Inserting 11 to the Queue



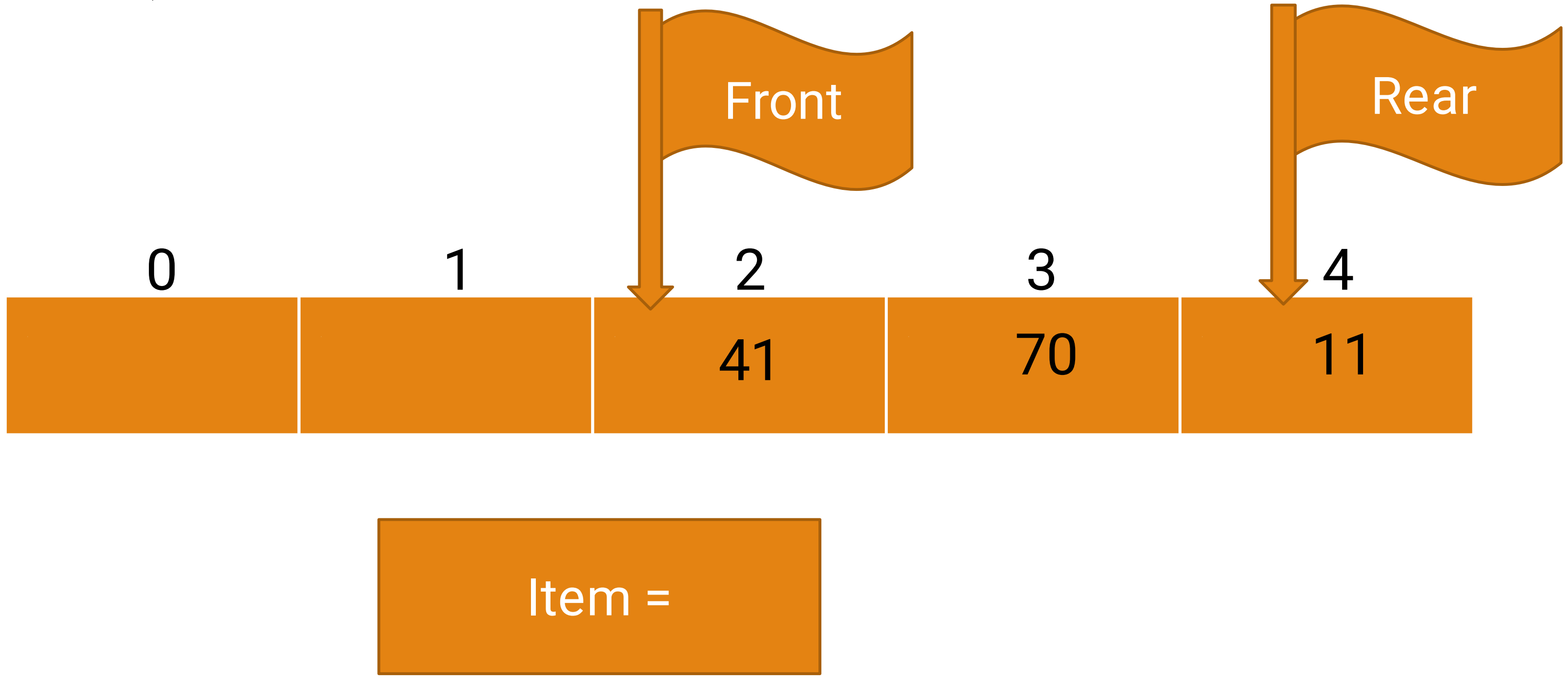
How a Queue Works?

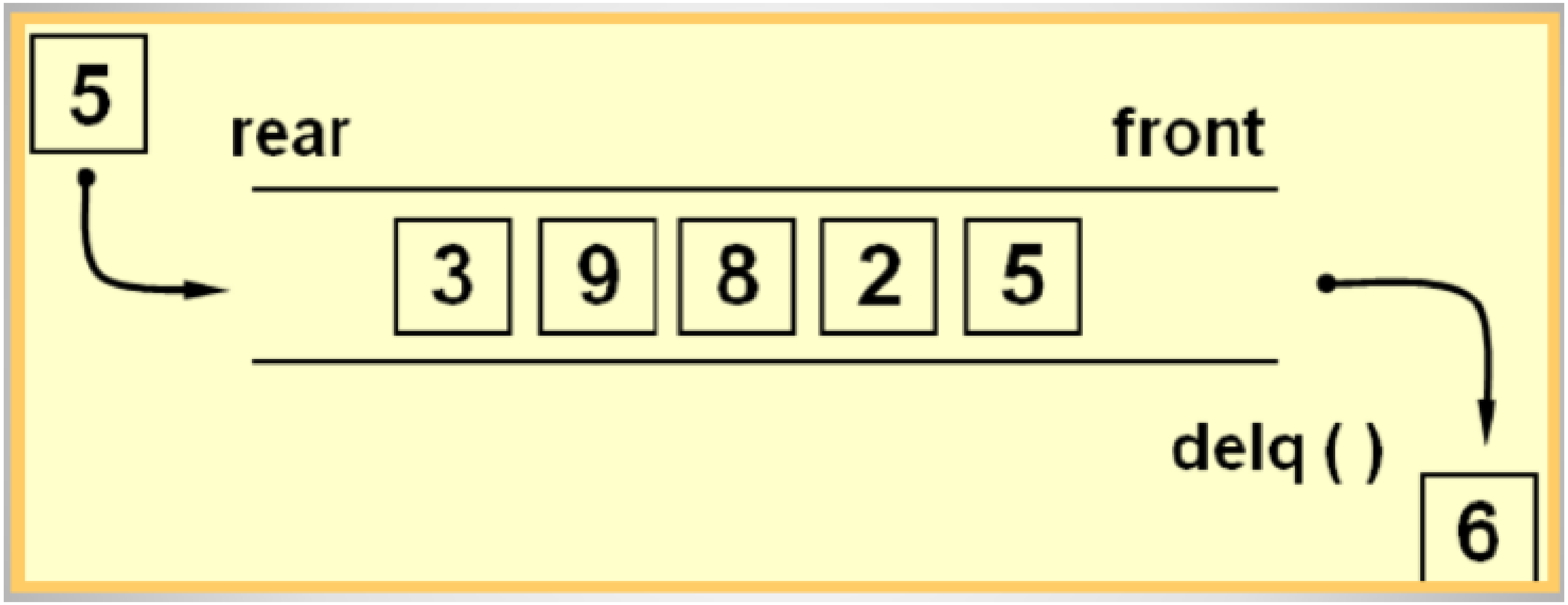
Deleting Item from the Queue



How a Queue Works?

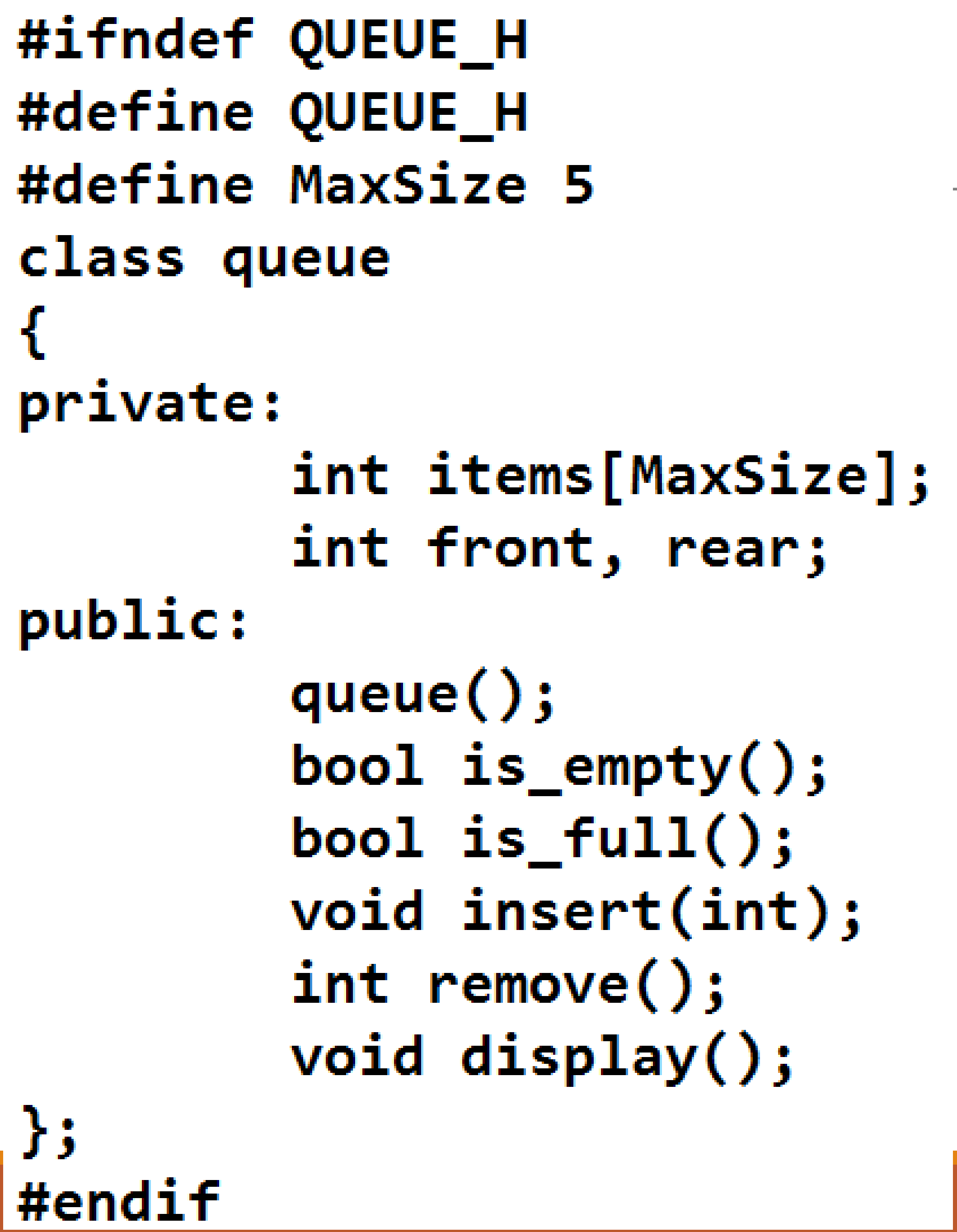
Deleting Item from the Queue





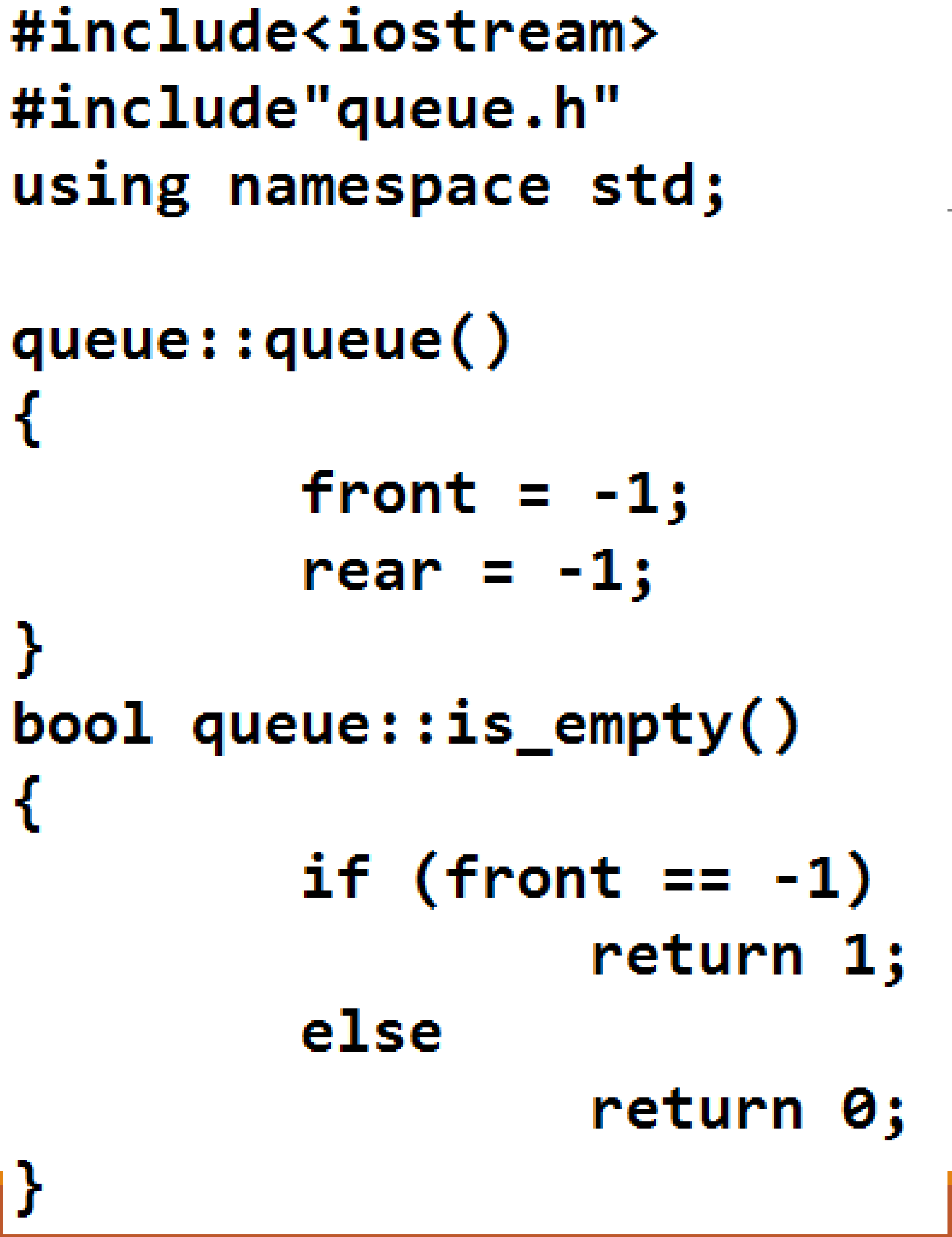
Queue Implementation

Queue Implementation



```
#ifndef QUEUE_H
#define QUEUE_H
#define MaxSize 5
class queue
{
private:
        int items[MaxSize];
        int front, rear;
public:
        queue();
        bool is_empty();
        bool is_full();
        void insert(int);
        int remove();
        void display();
};
#endif
```

Queue Implementation

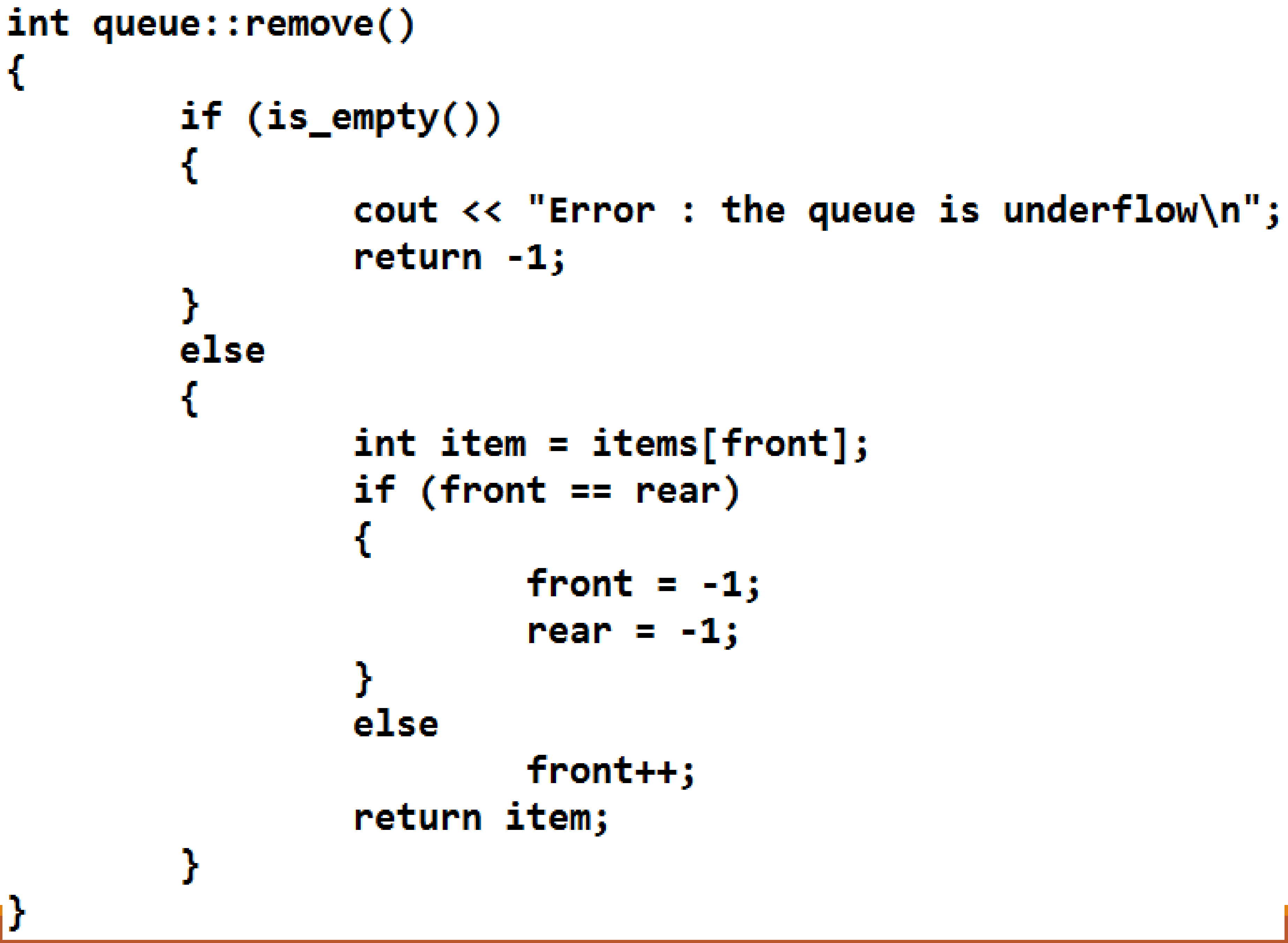


```
#include<iostream>
#include"queue.h"
using namespace std;

queue::queue()
{
        front = -1;
        rear = -1;
}
bool queue::is_empty()
{
        if (front == -1)
                return 1;
        else
                return 0;
}
```

Queue Implementation

```
bool queue::is_full()
{
        if (rear == MaxSize - 1)
                return 1;
        else
                return 0;
}

void queue::insert(int item)
{
        if (is_full())
                cout << "Error : the queue is overflow\n";
        else
        {
                if (is_empty())
                        front++;
                rear++;
                items[rear] = item;
        }
}
```

Queue Implementation



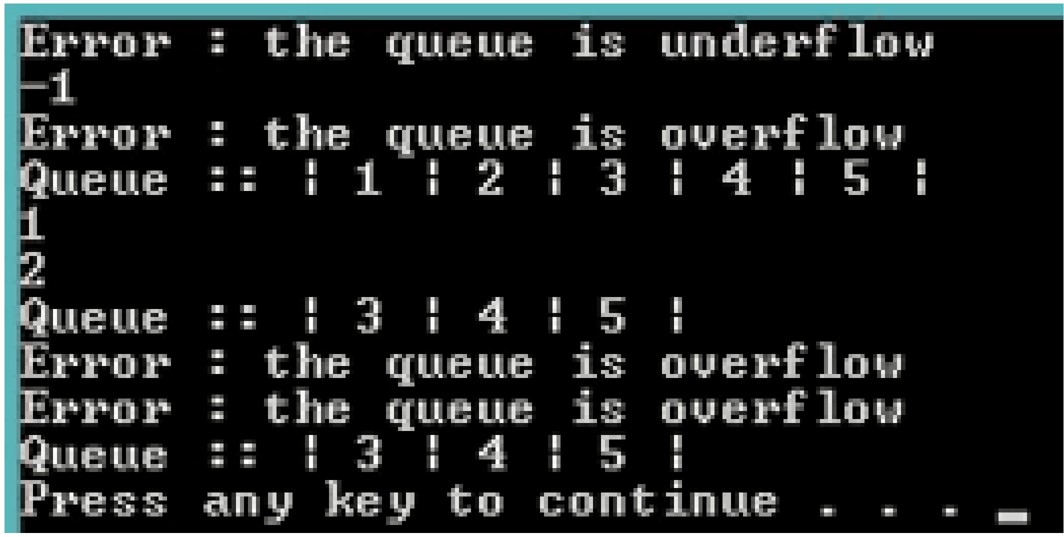
```
int queue::remove()
{
        if (is_empty())
        {
                cout << "Error : the queue is underflow\n";
                return -1;
        }
        else
        {
                int item = items[front];
                if (front == rear)
                {
                        front = -1;
                        rear = -1;
                }
                else
                        front++;
                return item;
        }
}
```

Queue Implementation

```
void queue::display()
{
        cout << "Queue :: | ";
        if (!is_empty())
                for (int i = front; i <= rear; i++)
                        cout << items[i] << " | ";
        cout << endl;
}
```

Queue Implementation

```
#include<iostream>
#include"queue.h"
using namespace std;

void main()
{
        queue q;
        cout << q.remove() << endl;

        q.insert(1);
        q.insert(2);
        q.insert(3);
        q.insert(4);
        q.insert(5);
        q.insert(6);

        q.display();
        cout << q.remove() << endl;
        cout << q.remove() << endl;
        q.display();

        q.insert(7);
        q.insert(8);
        q.display();
}
```



```
Error : the queue is underflow
-1
Error : the queue is overflow
Queue :: | 1 | 2 | 3 | 4 | 5 |
1
2
Queue :: | 3 | 4 | 5 |
Error : the queue is overflow
Error : the queue is overflow
Queue :: | 3 | 4 | 5 |
Press any key to continue . . . _
```