Queues and Deques

Canberra Institute of Technology

aie

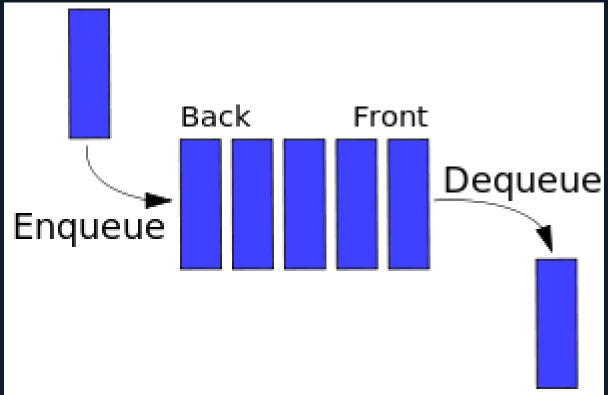
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Revision

- Stacks
  - Last In First Out

Queues

- Queues work using the FIFO concept (First In First Out).
- First object ‘pushed’ onto the queue is the first one out
- Like stacks, we can only push and pop an object on and off the queue
  - i.e., no iterators
- Often used for buffering
  - e.g, loading and playing sound



Queues in Real Life

- Waiting at the checkout
  - First person to enter the line gets served first
  - Customers join at the end… usually

Canberra Institute of Technology

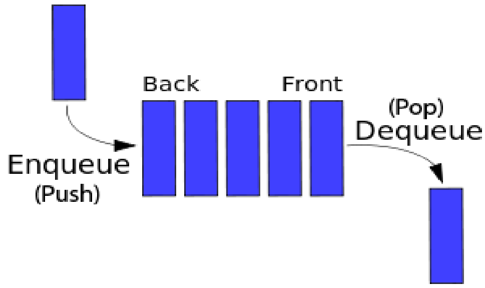
aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Properties

- The only element you have access to is the top element
  - i.e., the element that was added first
- You can not iterator or access other elements in the queue
- New objects get added (pushed) onto the end

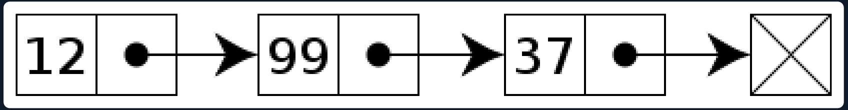
Operations of a Queue

- A queue has very similar operations to a stack:
  - Empty Returns true if queue is empty
  - Size Returns the number of elements
  - Push Adds element to the *end* (enqueue)
  - Pop Removes the *front* element of the queue (dequeue)
  - Top Returns the *front* element of the queue, without removing it



Implementation

- Often implemented using Linked Lists
- This is because it is faster to remove the head element from a linked list than from any other basic data structure



Queue.h

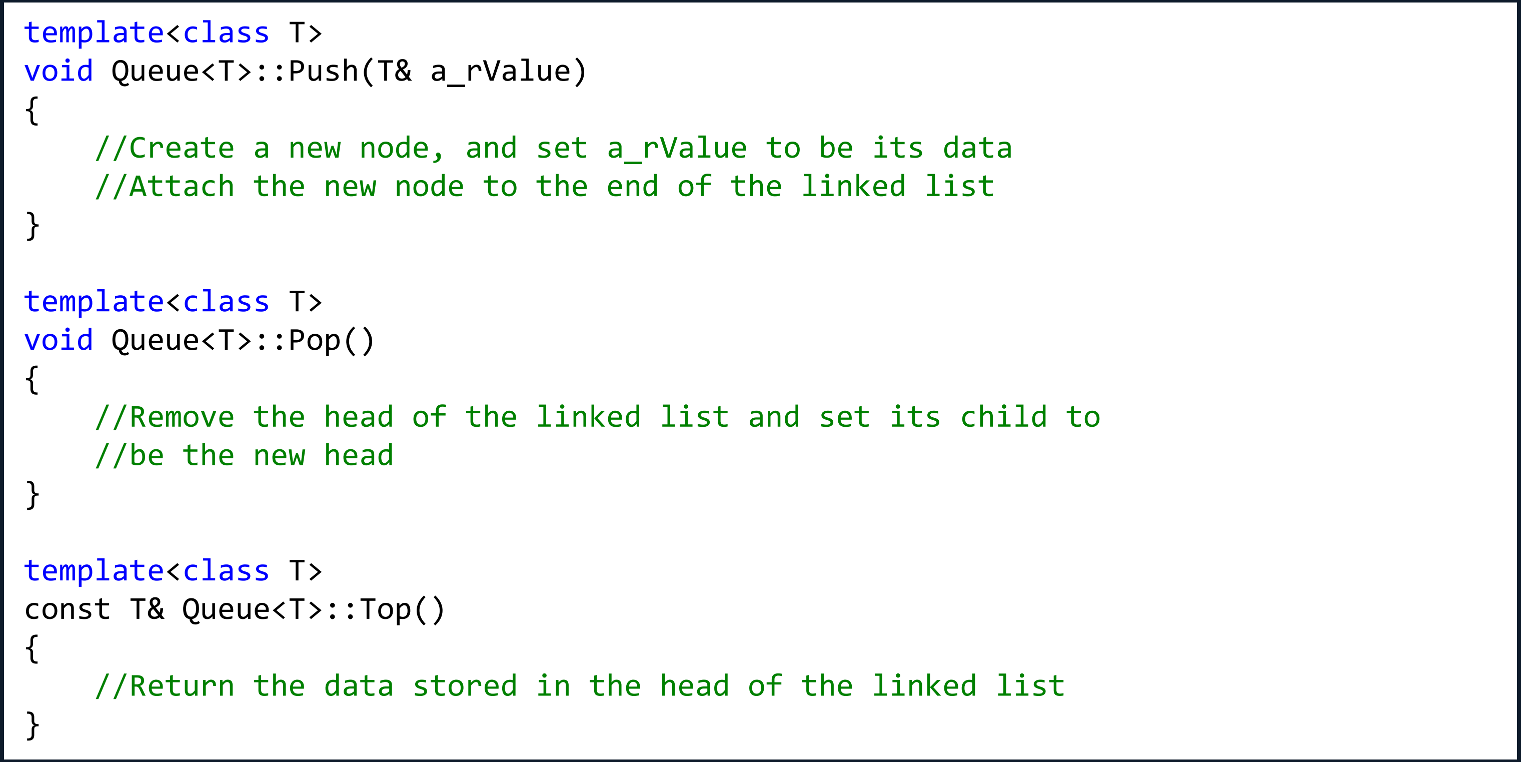
```
template<class T>
class Queue {
public:
    class Node {
    public:
        // Linked list node Implementation goes here...
    };

    Queue() : m_pData(NULL), m_uiSize(0) {};
    Queue(const Queue& a_rPointer);
    ~Queue();

    //Accessory functions
    boolEmpty()const;
    UIntGetSize()const;

    //Access functions
    void Push(T& a_rValue);
    void Pop();
    const T& Top();

private:
    Node*m_pData;
    UIntm_uiSize;
};
```

Canberra Institute of Technology

aie

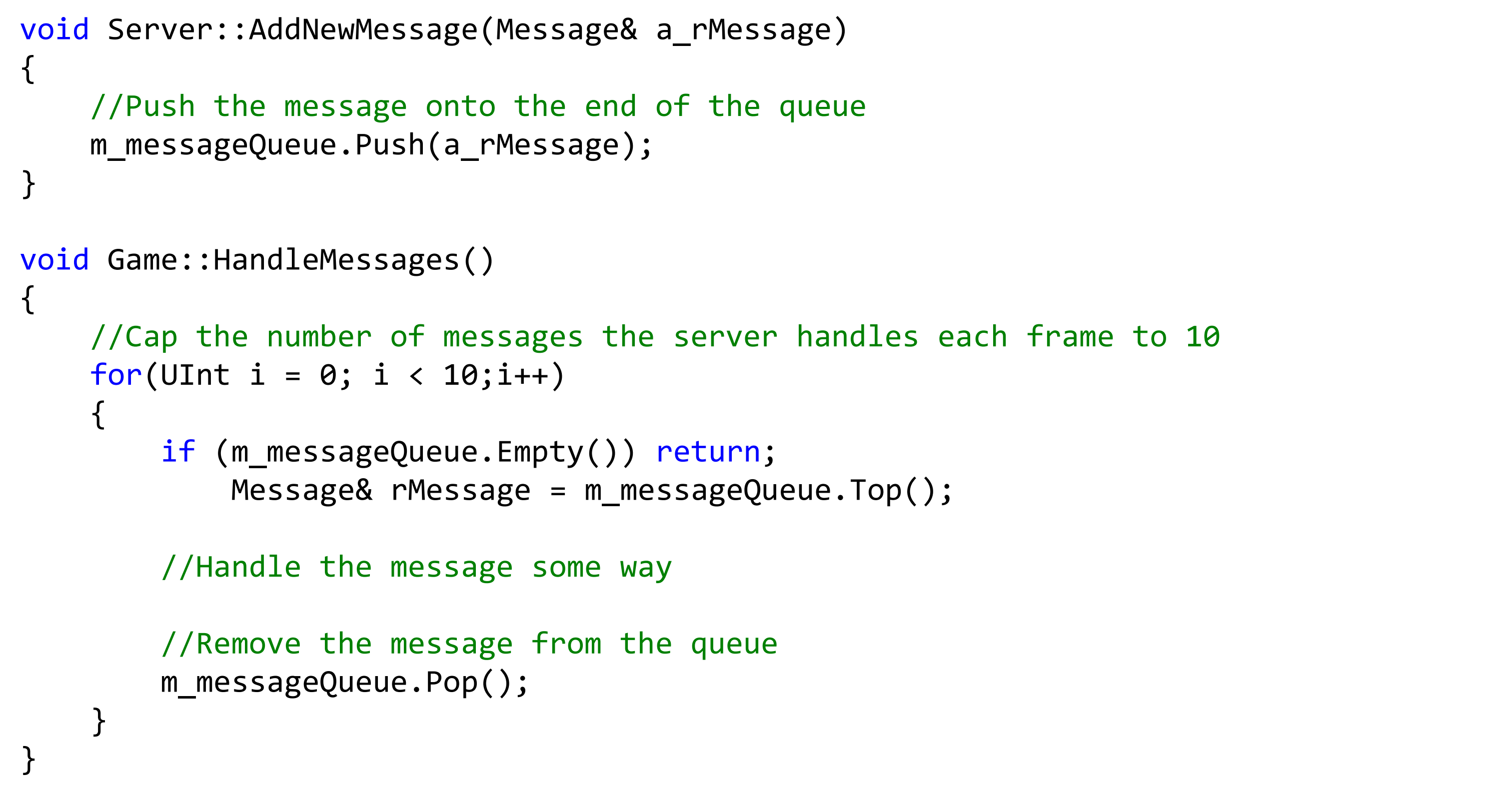
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Queue.h



```
template<class T>
void Queue<T>::Push(T& a_rValue)
{
    //Create a new node, and set a_rValue to be its data
    //Attach the new node to the end of the linked list
}

template<class T>
void Queue<T>::Pop()
{
    //Remove the head of the linked list and set its child to
    //be the new head
}

template<class T>
const T& Queue<T>::Top()
{
    //Return the data stored in the head of the linked list
}
```

A Practical Use



```
void Server::AddNewMessage(Message& a_rMessage)
{
    //Push the message onto the end of the queue
    m_messageQueue.Push(a_rMessage);
}

void Game::HandleMessages()
{
    //Cap the number of messages the server handles each frame to 10
    for(UInt i = 0; i < 10;i++)
    {
        if (m_messageQueue.Empty()) return;
            Message& rMessage = m_messageQueue.Top();

        //Handle the message some way

        //Remove the message from the queue
        m_messageQueue.Pop();
    }
}
```

Deques

- Suppose you wanted to get access to the end of the queue?
  - You cannot in a standard Queue
- A Deque is a ‘double-ended queue’ and allows items to be inserted and removed from either ends of a sequence.

Applications of Deques

- Trivially, a palindrome checker.
  - Is RADAR a palindrome? (spelled the same way backwards and forwards)
- More seriously, in operating systems, the Adaptive-Steal job scheduling algorithm for multi-processors relies on deques.
  - More info here: http://chargueraud.org/research/2013/ppopp/full.pdf

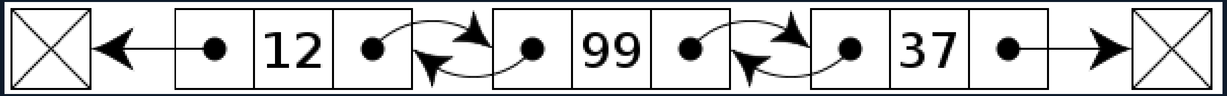
Operations of a Deque

- A deque has some additional and modified operations to a queue:

– PushBack	Adds element to the *end*
– PopBack	Removes element from the *end*
– PushFront	Adds elements to the *front*
– PopFront	Removes element front the *front*

Implementation

- Can be implemented as a doubly-linked list:



  - The start and end pointers provide access to the front and back of the deque
  - The next and prev node pointers provide a means to insert or remove elements from the front or back
- Can also be implemented a dynamic array:
  - Provides random access to any of the deque’s elements

Why not just use a vector STL container?

- Why can’t we just use the vector STL container type for everything?
- While it may fulfil the functional requirements of what we need...it may not have the best performance.
- A deque for example can be faster than vector if you had to insert items at the front of the sequence
  - Deque: O(1) – just need to readjust pointers
  - Vector: O(N) – need to move and copy items

Conclusion

- Queues are important and useful in programming when we wish operate on items in a sequence in the order they were inserted
- Deques are useful when we need to operate on and insert to both the front and back of a queue