```
//****************************************************************************************************
**
//
//              File:                                   heap.h
//
//              Student:                                Sean Herrick
//
//              Assignment:                             Program #10
//
//              Course Name:                    Data Structures II
//
//              Course Number:                  COSC 3100-01
//
//              Due:                                    November 16th, 2022
//
//
//              This program is an example of a Heap's member functions
//
//              Other files required:
//                      1.      waitList.cpp
//                      2.      customer.h
//
//****************************************************************************************************
**

#ifndef HEAP_H
#define HEAP_H

#include "customer.h"

//****************************************************************************************************
**

template <typename TYPE>
class Heap
{
private:
    TYPE* heap;
    int capacity,
        numValues;
    void _siftUp ( int c );
    void _siftDown ( int p );
    int _leftChildOf ( int p ) const;
    int _parentOf ( int c ) const;


public:
    Heap ( int c = 100 );
    ~Heap ( );
    bool insert ( const TYPE & dataIn );
    bool remove ( TYPE & dataIn );
    int getCapacity ( ) const;
    int getNumValues ( ) const;
    bool viewMax ( TYPE & dataOut ) const;
    bool isEmpty ( ) const;
    bool isFull ( ) const;

};

//****************************************************************************************************
**

template <typename TYPE>
Heap <TYPE>::Heap ( int capacity )
```

```
{
    this->capacity = capacity;
    heap = new TYPE [ capacity ];
    numValues = 0;
}

//*****************************************************************************************************
**

template <typename TYPE>
Heap<TYPE>::~Heap ( )
{
    delete [ ] heap;
    this->heap = nullptr;
    this->numValues = 0;
    this->capacity = 0;
}

//*****************************************************************************************************
**

template <typename TYPE>
int Heap <TYPE>::_leftChildOf ( int p ) const
{
    return ( ( 2 * p ) + 1 );
}

//*****************************************************************************************************
**

template <typename TYPE>
int Heap <TYPE>::_parentOf ( int c ) const
{
    return ( ( c - 1 ) / 2 );
}

//*****************************************************************************************************
**

template <typename TYPE>
bool Heap <TYPE>:: insert ( const TYPE & dataIn )
{
    bool success = false;

    if ( numValues < capacity )
    {
        heap [ numValues ] = dataIn;
        _siftUp ( numValues );
        numValues++;
        success = true;
    }

    return success;
}

//*****************************************************************************************************
**

template <typename TYPE>
bool Heap<TYPE>:: remove ( TYPE & dataIn )
{
    bool success = false;

    if ( numValues > 0 )
    {
        dataIn = heap [ 0 ];
        heap [ 0 ] = heap [ ( numValues - 1 ) ];
        numValues--;
        _siftDown ( 0 );
        success = true;
    }

    return success;
}

//****************************************************************************************************
**

template <typename TYPE>
bool Heap<TYPE>:: viewMax ( TYPE & dataOut ) const
{
    bool success = false;

    if ( numValues > 0 )
    {
        dataOut = heap [ 0 ];
        success = true;
    }

    return success;
}

//****************************************************************************************************
**

template <typename TYPE>
void Heap <TYPE>:: _siftUp ( int c )
{
    int parent;

    if ( c > 0 )
    {
        parent = _parentOf ( c );

        if ( heap [ c ] > heap [ parent ] )
        {
            swap ( heap [ c ],heap [ parent ] );
            _siftUp ( parent );
        }
    }

}

//****************************************************************************************************
**

template <typename TYPE>
void Heap<TYPE>:: _siftDown ( int p )
{
    int child;

    child = _leftChildOf ( p );

    if ( child < numValues )
    {
        if ( ( child + 1  < numValues ) && ( heap [ child ] < heap [ child + 1 ] ) )
        {
            child++;
        }
        if ( heap [ p ] < heap [ child ] )
        {
            swap ( heap [ p ], heap [ child ] );
            _siftDown ( child );
        }
    }
}

//****************************************************************************************************
**

template <typename TYPE>
int Heap<TYPE>::getCapacity ( ) const
{
    return capacity;
}

//****************************************************************************************************
**

template <typename TYPE>
int Heap<TYPE>::getNumValues ( ) const
{
    return numValues;
}

//****************************************************************************************************
**

template <typename TYPE>
bool Heap<TYPE>::isEmpty ( ) const
{
    return ( numValues == 0 );
}

//****************************************************************************************************
**

template <typename TYPE>
bool Heap<TYPE>::isFull ( ) const
{
    return ( numValues >= capacity );
}

//****************************************************************************************************
**

#endif
```