```
//***************************************************************************************************
**
//
//              File:                                   customer.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.      heap.h
//                      2.      waitList.cpp
//
//***************************************************************************************************
**

#ifndef CUSTOMER_H
#define CUSTOMER_H

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

struct Customer
{
    char fname [ 15 ],
         lname [ 15 ];
    int mileage,
        years,
        arrivalNum,
        priority;

    Customer ( );
    friend ostream & operator << ( ostream & out, const Customer& data );
    bool operator == ( const Customer & rhs ) const;
    bool operator == ( int priority ) const;
    bool operator != ( const Customer & rhs ) const;
    bool operator != ( int priority ) const;
    bool operator < ( const Customer & rhs ) const;
    bool operator < ( int priority ) const;
    bool operator > ( const Customer & rhs ) const;
    bool operator > ( int priority ) const;
    bool operator <= ( const Customer & rhs ) const;
    bool operator <= ( int priority ) const;
    bool operator >= ( const Customer & rhs ) const;
    bool operator >= ( int priority ) const;
    int operator % ( int priority ) const;
    Customer & operator = ( int priority );

};

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

Customer :: Customer ( )
{
    priority = 0;
}

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

ostream & operator << ( ostream & out, const Customer & data )
{
    out << left << setw ( 12 ) << data.priority << setw ( 15 ) << data.arrivalNum << setw ( 9 ) <<
data.fname
        << setw ( 13 ) << data.lname << setw ( 15 ) << data.mileage << data.years;

    return out;
}

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

bool Customer :: operator == ( const Customer & rhs ) const
{
    return ( this->priority == rhs.priority );
}

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

bool Customer :: operator == ( int priority ) const
{
    return ( this->priority == priority );
}

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

bool Customer :: operator != ( const Customer & rhs ) const
{
    return ( this->priority != rhs.priority );
}

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

bool Customer :: operator != ( int value ) const
{
    return ( this->priority != value );
}

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

bool Customer :: operator < ( const Customer & rhs ) const
{
    return ( this->priority < rhs.priority );
}

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

bool Customer :: operator < ( int priority ) const
{
    return ( this->priority < priority );
}

//**********************************************************************************************
**
```

```
bool Customer :: operator > ( const Customer & rhs ) const
{
    return ( this->priority > rhs.priority );
}

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

bool Customer :: operator > ( int value ) const
{
    return ( this->priority > value );
}

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

bool Customer :: operator <= ( const Customer & rhs ) const
{
    return ( this->priority <= rhs.priority );
}

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

bool Customer :: operator <= ( int value ) const
{
    return ( this->priority <= value );
}

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

bool Customer :: operator >= ( const Customer & rhs ) const
{
    return ( this->priority >= rhs.priority );
}

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

bool Customer :: operator >= ( int value ) const
{
    return ( this->priority >= value );
}

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

Customer & Customer :: operator = ( int value )
{
    this->priority = value;

    return *this;
}

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

int Customer :: operator % ( int value ) const
{
    return ( this->priority % value );
}

//***************************************************************************************************
```

```
**

#endif
```

```
//****************************************************************************************************
**
//
//              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
```

```
//****************************************************************************************************
**
//
//              File:                                   waitList.cpp
//
//              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.      heap.h
//                      2.      customer.h
//
//****************************************************************************************************
**

#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <algorithm>

using namespace std;

#include "heap.h"
#include "customer.h"


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

void getData ( Heap <Customer>& waitList );
void printWaitList ( Heap <Customer>& waitList );

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

int main ( )
{
        Heap <Customer> waitList;

        getData ( waitList );
        printWaitList ( waitList );

        return 0;
}

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

void getData ( Heap <Customer>& waitList )
{
        Customer cust;
        ifstream inFile;
        inFile.open ( "overbooked.bin", ios :: binary );

        while ( inFile.read ( ( char * ) & cust, sizeof ( cust ) ) )
        {
                cust.priority =   ( ( ( cust.mileage / 1000 ) + cust.years ) - cust.arrivalNum );
                waitList.insert ( cust );
        }

        inFile.close ( );
}

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

void printWaitList ( Heap <Customer>& waitList )
{
        Customer cust,
                tempCust;
        ofstream outFile;
        int numVals,
                capacity;

        outFile.open ( "waitList.txt" );
        numVals = waitList.getNumValues ( );
        capacity = waitList.getCapacity ( );

        if ( waitList.viewMax ( cust ) )
        {
                tempCust = cust;
        }

        outFile << string ( 78, '=' ) << endl;
        outFile << setw ( 43 ) << "Priority List" << endl;
        outFile << string ( 78, '=' ) << endl;
        outFile << setw ( 5 ) << "Priority:" << setw ( 15 )
                    << "Arrival Num:" << setw ( 8 ) << "Name:" << setw ( 25 )
                    << "Mileage:" << setw ( 13 ) << "Years:" << endl;

        while ( waitList.remove ( cust ) )
        {
                outFile << cust << endl;
        }

        outFile << endl << "There are " << numVals << " people on the priority list" << endl;
        outFile << "The person with the highest priority is:\n" << tempCust << endl;

        if ( waitList.isEmpty ( ) )
        {
                outFile << "There are no more people in the list" << endl;
        }

        if ( waitList.isFull ( ) )
        {
                outFile << "The list is full" << endl;
        }

        else
        {
                outFile << "The list is not full" << endl;
        }

        outFile << "The list can hold up to " << capacity << " people" << endl;
        outFile << string ( 78, '=' ) << endl;
}
```

```
//****************************************************************************************************
**

/*
=============================================================================
                                                          Priority List
=============================================================================
Priority:   Arrival Num:   Name:                  Milage:       Years:
93          3              Baclan   Nguyen        93000         3
90          2              Amanda   Trapp         89000         3
74          5              Warren   Rexroad       72000         7
61          6              Jorge    Gonzales      65000         2
57          1              Bryan    Devaux        53000         5
56          10             Dave     Lightfoot     63000         3
37          9              Steve    Chu           42000         4
30          7              Paula    Hung          34000         3
24          11             Joanne   Brown         33000         2
19          8              Lou      Mason         21000         6
14          4              Sarah    Gilley        17000         1

There are 11 people on the priority list
The person with the highest priority is:
93          3              Baclan   Nguyen        93000         3
There are no more people in the list
The list is not full
The list can hold up to 100 people
=============================================================================
*/
```