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