```
#ifndef __MYQUEUE_H__
#define __MYQUEUE_H__

#include <iostream>
#include <algorithm>

#include "MyVector_a676m513.h"

template <typename DataType>
class MyQueue : private MyVector<DataType>
{
  private:
    size_t dataStart, dataEnd;

    // changes the size of the array to newSize
    void resize(size_t newSize)
    {
        // code begins
        MyVector<DataType>::resize(newSize);
        // code ends
    }

    // requests for newCapacity amount of space
    void reserve(size_t newCapacity)
    {
        // code begins
        MyVector<DataType>::reserve(newCapacity);
        // code ends
    }

  public:

    // default constructor
    explicit MyQueue(size_t initSize = 0)
    {
        // code begins
        dataStart = 0;
        dataEnd = 0;
        MyVector<DataType>::resize(initSize);
        // code ends
    }

    // copy constructor
    MyQueue(const MyQueue & rhs)
    {
        // code begins
        dataStart = rhs.dataStart;
        dataEnd = rhs.dataEnd;
        MyVector<DataType>::operator=(rhs);
        // code ends
    }

    // move constructor
    MyQueue(MyQueue && rhs)
    {
        // code begins
        dataStart = rhs.dataStart;
        dataEnd = rhs.dataEnd;
        MyVector<DataType>::operator=(std::move(rhs));
```

```
    // code ends
}

// destructor
~MyQueue()
{
    // code begins

    // code ends
}

// copy assignment
MyQueue& operator= (const MyQueue& rhs)
{
    // code begins
    dataStart = rhs.dataStart;
    dataEnd = rhs.dataEnd;
    MyVector<DataType>::operator=(rhs);
    return *this;
    // code ends
}

// move assignment
MyQueue& operator= (MyQueue&& rhs)
{
    // code begins
    dataStart = rhs.dataStart;
    dataEnd = rhs.dataEnd;
    MyVector<DataType>::operator=(std::move(rhs));
    return *this;
    // code ends
}

// insert x into the queue
void enqueue(const DataType& x)
{
    // code begins
    if (this -> empty())
    {
        this -> theSize++;
        dataEnd = this -> theSize;
        this -> data[dataStart] = std::move(x);

    }
    else
    {
        dataEnd = this -> theSize++;
        this -> data[dataEnd] = std::move(x);
    }
    // code ends
}

// insert x into the queue
void enqueue(DataType && x)
{
    // code begins
    if (this -> empty())
    {
        this -> theSize++;
```

```
        dataEnd = this -> theSize;
        this -> data[dataStart] = std::move(x);

    }
    else
    {
        dataEnd = this -> theSize++;
        this -> data[dataEnd] = std::move(x);
    }

    // code ends
}

// remove the first element from the queue
void dequeue(void)
{
    // code begins
    size_t temp = dataStart + 1;
    if (dataEnd == dataStart)
    {
        this -> pop_back();
        dataEnd = this -> theSize;
    }
    else
    {
        for (int i = temp; i < this -> theSize; i++)
        {
            this -> data[i - 1] = std::move(this -> data[i]);
        }
        this -> pop_back();
        dataEnd = this -> theSize;
    }


    // code ends
}

// access the first element of the queue
const DataType & front(void) const
{
    // code begins
    return this -> data[dataStart];
    // code ends
}

// check if the queue is empty; return TRUE is empty; FALSE otherwise
bool empty(void) const
{
    // code begins
    return dataStart == dataEnd;
    // code ends
}

// access the size of the queue
size_t size() const
{
    // code begins
    return MyVector<DataType>::size();
    // code ends
```

```
    }

    // access the capacity of the queue
    size_t capacity(void) const
    {
        // code begins
        return MyVector<DataType>::capacity();
        // code ends
    }

};


#endif // __MYQUEUE_H__
```