makefile

```
###############################################################
# Program:
#     Lesson 04, DEQUE
#     Brother Helfrich, CS235
# Author:
#     Derek Calkins and David Lambertson
# Summary:
#     This program will allow a user to use the deque
#     to create a line and be able to insert at front or back
#     and remove from front or back and can access front or back.
# Time:
#     total of 8.5 hours. David 2.5 and Derek 6
# Work Load:
#     Derek: 50% and David: 50%
###############################################################

##############################################################
# The main rule
##############################################################
a.out: deque.h lesson04.o nowServing.o
       g++ -o a.out lesson04.o nowServing.o
       tar -cf lesson04.tar *.h *.cpp makefile

##############################################################
# The individual components
#      lesson04.o     : the driver program
#      nowServing.o   : the logic for the now serving program
##############################################################
lesson04.o: deque.h lesson04.cpp
       g++ -c lesson04.cpp

nowServing.o: nowServing.h nowServing.cpp deque.h
       g++ -c nowServing.cpp
```

Commented [HJ1]: This is remarkable! Well done!

nowServing.h

```
/***********************************************************************
 * Header:
 *    NOW SERVING
 * Summary:
 *    This will contain just the prototype for nowServing(). You may
 *    want to put other class definitions here as well.
 * Author
 *    David Lambertson
 ************************************************************************/

#ifndef NOW_SERVING_H
#define NOW_SERVING_H

#include "deque.h"     // for DEQUE
#include <string>
#include <iostream>



// the interactive nowServing program
void nowServing();


/*********************
 * this is a class I created to
 * help keep track of those who are in
 * in line and their information
 *********************/
class inLine
{
  public:
   //all the functions to be able to set the private member variables.
```

Commented [HJ2]: Class names need to be TitleCased, not camelCased.

```
   void setName(const std::string & name)       { this->name = name; }
   void setClass(const std::string & Class)     { this->Class = Class; }
   void setStudentMins(const int & studentMins) { this->studentMins = studentMins; }
   void setCommand(const std::string & command) { this->command = command; }

   //all the getters for my custom class
   int getStudentMins() const     { return studentMins; }
   std::string getClass() const   { return Class; }
   std::string getName() const    { return name; }
   std::string getCommand() const { return command; }

  private:
   int studentMins;
   std::string Class;
   std::string name;
   std::string command;

};


#endif // NOW_SERVING_H
```

Commented [HJ3]: Well done.

deque.h

```
/**************************************************
 * DEQUE.H
 * Summary:
 *    This class is for the deque which allows the user
 *    to add elements or remove elements from either side
 *    of the container. It also allows the user to access either
 *    end of the deque but not the elements in between.
 * Author:
 *    Derek Calkins
 * Time:
 *   Estimated: 6 hours
 *   Actual: 6 hours
 * Hardest Part:
 *   -Understanding the iHead and iTail and which was which in the deque.
 *   -Also getting it to allocate with a negative number which is where
 *    iAbsoluteFromI helped a ton.
 **************************************************/

#ifndef DEQUE_H
#define DEQUE_H

#include <cassert>
#include <cmath>
#include <iostream>

/******************************
 * This is the class for Deque.
 * It holds all the member functions
 * and variables that we use within Deque.
 ******************************/
template<class T>
class Deque
{
  public:
   //default constructor
  Deque() : myCapacity(0), iTail(-1), iHead(0), data(0x00000000) {}

   //non default constructor
   Deque(int myCapacity) throw (const char *);

   //Copy Constructor
   Deque(const Deque<T> & rhs) throw (const char *);

   //Destructor
   ~Deque() { delete [] data; }

   //checks to see if the Deque is empty
   bool empty() const { return (iHead == iTail + 1); }

   //returns the capacity of the deque
   int capacity() const { return myCapacity; }

   //returns the number of items saved in the deque
   int size() const { return ((iTail + 1) - iHead); }
```

Commented [HJ4]: Careful. What if you never got around to allocating the buffer? This is a bug.

```
   //clears the deque and sets everything to 0
   void clear() { iTail = -1; iHead = 0; }

   //saves a value from the user at the end of the deque
   void push_back(const T & value) throw (const char *)
   {
      resize(); //grows when needed
      data[iAbsoluteFromI(++iTail)] = value;
   }

   //pushes an item onto the front of the deque
   void push_front(const T & value) throw (const char *)
   {
      resize(); //grows when needed
      data[iAbsoluteFromI(--iHead)] = value;
   }

   //pops an item from the front of the deque
   void pop_front() throw (const char *)
   {
      if(empty())
         throw "ERROR: unable to pop from the front of empty deque";
      iHead++;
   }

   //pops an item from the back of the deque
   void pop_back() throw (const char *)
   {
      if(empty())
         throw "ERROR: unable to pop from the back of empty deque";
      iTail--;
   }

   //returns what item is at the back of the deque
   const T & back() throw (const char *)
   {
      if(empty())
         throw "ERROR: unable to access data from an empty deque";
      return data[iAbsoluteFromI(iTail)];
   }

   //returns what item is at the front of the deque
   const T & front() throw (const char *)
   {
      if(empty())
         throw "ERROR: unable to access data from an empty deque";
      return data[iAbsoluteFromI(iHead)];
   }

   //for finding the absolute value of the index
   const int iAbsoluteFromI(const int & i) const
   {
      return (((i % capacity()) + capacity()) % capacity());
   }

  private:
   T * data;          // holds the data
   int myCapacity;    // is the capacity of the deque
   int iTail;         // index of the tail of the deque
   int iHead;         // index of the head of the deque

   void resize() throw (const char *);

};

/***********************
 * COPY CONSTRUCTOR
 ***********************/
template <class T>
Deque<T> :: Deque(const Deque<T> & rhs) throw (const char *)
{
   this->myCapacity = rhs.myCapacity;
   this->iTail = rhs.iTail;
   this->iHead = rhs.iHead;

   try
   {
      data = new T[myCapacity];
```

Commented [HJ5]: Well done on this.

Commented [HJ6]: Not const; you should be able to change the head or the talk through back() and front()

Commented [HJ7]: Flawless.

```
   }
   catch(...)
   {
      throw "Unable to allocate buffer.";
   }
   for (int i = 0; i < (rhs.myCapacity); i++)
   {
       data[i] = rhs.data[iHead + i];
   }
}
```

Commented [HJ8]: Well done!

```
/******************
 * NON DEFAULT CONSTRUCTOR
 ****************/
template<class T>
Deque<T> :: Deque(int myCapacity) throw (const char *)
{
   assert(myCapacity > 0);

   try
   {
      data = new T[myCapacity];
   }
   catch(...)
   {
      throw "Error: Unable to allocate buffer.";
   }

   this->myCapacity=myCapacity;
   this->iHead=0;
   this->iTail=-1;
}

/**********************************
 * RESIZE
 * reallocates the data if no more space
 * is available for data to be saved.
 *********************************/
template <class T>
void Deque <T> :: resize() throw (const char *)
{
   if (myCapacity == 0)
   {
      myCapacity += 1;
      data = new T[myCapacity];
   }
   else if (myCapacity == size())
   {
      T * newData;
      myCapacity *= 2;
      try
      {
         newData = new T[myCapacity];
      }
      catch(...)
      {
         throw "Unable to allocate a new buffer for Deque";
         myCapacity /= 2;
      }
      int i = 0;
      for (; i < (size()); i++)
      {
         newData[i] = data[(iAbsoluteFromI(iHead) + i) % (myCapacity/2)];
      }
      iTail = i - 1;
      iHead = 0;

      delete [] data;
      data = newData;
   }
}

#endif
```

Commented [HJ9]: Wow that is complex!

lesson04.cpp

```
/***********************************************************************
* Program:
```

```
*    Lesson 04, DEQUE
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
* Summary:
*    This is a driver program to exercise the Deque class.  When you
*    submit your program, this should not be changed in any way.  That being
*    said, you may need to modify this once or twice to get it to work.
***********************************************************************/

#include <iostream>     // for CIN and COUT
#include <string>       // for the String class
#include "deque.h"      // your Deque class should be in deque.h
#include "nowServing.h" // your nowServing() function
using namespace std;


// prototypes for our four test functions
void testSimple();
void testPush();
void testPop();
void testErrors();

// To get your program to compile, you might need to comment out a few
// of these. The idea is to help you avoid too many compile errors at once.
// I suggest first commenting out all of these tests, then try to use only
// TEST1.  Then, when TEST1 works, try TEST2 and so on.
#define TEST1   // for testSimple()
#define TEST2   // for testPush()
#define TEST3   // for testPop()
#define TEST4   // for testErrors()

/**********************************************************************
 * MAIN
 * This is just a simple menu to launch a collection of tests
 ***********************************************************************/
int main()
{
   // menu
   cout << "Select the test you want to run:\n";
   cout << "\t0. Now Serving\n";
   cout << "\t1. Just create and destroy a Deque\n";
   cout << "\t2. The above plus push items onto the Deque\n";
   cout << "\t3. The above plus pop items off the Deque\n";
   cout << "\t4. The above plus exercise the error Deque\n";

   // select
   int choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 0:
         nowServing();
         break;
      case 1:
         testSimple();
         cout << "Test 1 complete\n";
         break;
      case 2:
         testPush();
         cout << "Test 2 complete\n";
         break;
      case 3:
         testPop();
         cout << "Test 3 complete\n";
         break;
      case 4:
         testErrors();
         cout << "Test 4 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}

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

```
 * TEST SIMPLE
 * Very simple test for a Deque: create and destroy
 ******************************************/
void testSimple()
{
#ifdef TEST1
   // Test1: a bool Deque with default constructor
   cout << "Create a bool Deque using the default constructor\n";
   Deque <bool> d1;
   cout << "\tSize:     " << d1.size()                   << endl;
   cout << "\tEmpty?    " << (d1.empty() ? "Yes" : "No") << endl;
   cout << "\tCapacity: " << d1.capacity()               << endl;

   // Test2: double Deque with non-default constructor
   cout << "Create a double Deque using the non-default constructor\n";
   Deque <double> d2(10 /*capacity*/);
   cout << "\tSize:     " << d2.size()                   << endl;
   cout << "\tEmpty?    " << (d2.empty() ? "Yes" : "No") << endl;
   cout << "\tCapacity: " << d2.capacity()               << endl;

   {
      // Test3: copy the bool Deque
      cout << "Copy the double Deque using the copy-constructor\n";
      Deque <double> d3(d2);
      cout << "\tSize:     " << d3.size()                   << endl;
      cout << "\tEmpty?    " << (d3.empty() ? "Yes" : "No") << endl;
      cout << "\tCapacity: " << d3.capacity()               << endl;
   }
   cout << "\tDestroying the third Deque\n";
#endif //TEST1
}

/*******************************************
 * TEST PUSH
 * Add a whole bunch of items to the Deque.  This will
 * test the Deque growing algorithm
 ******************************************/
void testPush()
{
#ifdef TEST2
   // create
   cout << "Create an integer Deque with the default constructor\n";
   Deque <int> d;

   cout << "\tEnter integer values, type 0 when done\n";
   int value;
   do
   {
      cout << "\t> ";
      cin  >> value;
      if (value)
      {
         d.push_back(value);

         // display the value and the capacity
         cout << "\t\tPushed " << d.back()
              << " size="      << d.size()
              << " capacity="  << d.capacity()
              << endl;
      }
   }
   while (value);

   // empty it and do it again in the front.
   d.clear();
   cout << "\tEnter integer values, type 0 when done\n";
   do
   {
      cout << "\t> ";
      cin  >> value;
      if (value)
      {
         d.push_front(value);

         // display the value and the capacity
         cout << "\t\tPushed " << d.front()
              << " size="      << d.size()
              << " capacity="  << d.capacity()
              << endl;
      }
   }
   while (value);
#endif // TEST2
}

/******************************************
 * TEST POP
 * We will test pop_front(), pop_back(),
 * push_front(), and push_back() to make
 * sure the dEque looks the way we expect
 * it to look.
 ******************************************/
void testPop()
{
#ifdef TEST3
   // create
   cout << "Create a string Deque with the non-default constructor\n";
   Deque <string> d(4);

   // instructions
   cout << "instructions:\n"
        << "\t+f dog   pushes dog onto the front\n"
        << "\t+b cat   pushes cat onto the back\n"
        << "\t-f       pops off the front\n"
        << "\t-b       pops off the back\n"
        << "\t*        clear the deque\n"
        << "\t?        shows the statistics of the deque\n"
        << "\t!        quit\n";

   string command;
   string text;
   do
   {
      cout << "> ";
      cin  >> command;

      try
      {
         if (command == "+f")
         {
            cin >> text;
            d.push_front(text);
         }
         else if (command == "+b")
         {
            cin >> text;
            d.push_back(text);
         }
         else if (command == "-f")
         {
            cout << "\tpop: " << d.front() << endl;
            d.pop_front();
         }
         else if (command == "-b")
         {
            cout << "\tpop: " << d.back() << endl;
            d.pop_back();
         }
         else if (command == "?")
         {
            cout << "\tSize:     " << d.size()     << endl;
            cout << "\tCapacity: " << d.capacity() << endl;
         }
         else if (command == "*")
         {
            d.clear();
         }
         else if (command != "!")
         {
            cout << "Unknown command\n";
            cin.ignore(256, '\n');
         }
      }
      catch (const char * e)
      {
         cout << '\t' << e << endl;
      }
```

```
    }
   while (command != "!");
#endif // TEST3
}

/*********************************************
 * TEST ERRORS
 * Numerous error conditions will be tested
 * here, including bogus popping and the such
 *******************************************/
void testErrors()
{
#ifdef TEST4
   // create
   cout << "Create a char deque with the default constructor\n";
   Deque <char> d;

   // test using front() with an empty deque
   try
   {
      d.front();
      cout << "BUG! We should not be able to front() with an empty deque!\n";
   }
   catch (const char * error)
   {
      cout << "\tDeque::front() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }

   // test using back() with an empty deque
   try
   {
      d.back();
      cout << "BUG! We should not be able to back() with an empty deque!\n";
   }
   catch (const char * error)
   {
      cout << "\tDeque::back() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }

   // test using pop_front() with an empty deque
   try
   {
      d.pop_front();
      cout << "BUG! We should not be able to pop_front() "
           << "with an empty deque!\n";
   }
   catch (const char * error)
   {
      cout << "\tDeque::pop_front() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }

   // test using pop_back() with an empty deque
   try
   {
      d.pop_back();
      cout << "BUG! We should not be able to pop_back() "
           << "with an empty deque!\n";
   }
   catch (const char * error)
   {
      cout << "\tDeque::pop_back() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }

#endif // TEST4
}
```

nowServing.cpp

```
/***********************************************************************
 * Implementation:
 *    NOW SERVING
 * Summary:
 *    This will contain the implementation for nowServing() as well as any
 *    other function or class implementations you may need
 * Author
 *    David Lambertson
 * Time
 *    2.5 hours
 ***********************************************************************/

#include "nowServing.h" // for nowServing() prototype
#include "deque.h"      // for DEQUE
#include <iostream>
#include <string>
using namespace std;

//prototypes for the functions I call in nowServing()
void setSpot(inLine & spot, const string & Class, const string & name,
             const int & studentMins, const string & command);
void getData(inLine & spot, string & Class, string & command,
             Deque<inLine> & waiting);

/************************************************
 * NOW SERVING
 * The interactive function allowing the user to
 * handle help requests in the Linux lab
 ***********************************************/
void nowServing()
{
   // instructions
   cout << "Every prompt is one minute.  The following input is accepted:\n";
   cout << "\t<class> <name> <#minutes>    : a normal help request\n";
   cout << "\t!! <class> <name> <#minutes> : an emergency help request\n";
   cout << "\tnone                         : no new request this minute\n";
   cout << "\tfinished                     : end simulation\n";

   // your code here

   int minute;                 //the current minute
   string command;             //the command given to us (IE !! or finished)
   inLine spot;                //creates an inLine to save the information
   Deque<inLine> waiting;      //a deque of inLine data for the waiting line
   string Class;               //saves the class the person is in

   inLine serving;             //the person currently being served
   int time = 0;               //allows us to minus one for time elapsed
                               //for serving

   do //loop until we have command == finished
   {
      cout << "<" << minute << "> ";
      cin >> command;
      minute++;

      if(command == "!!")             //if there is an emergency do this
      {
         cin >> Class;
         getData(spot, Class, command, waiting);
      }

      else if (command == "none")     // no command we do nothing
         ;

      else if (command == "finished") //we are done, get thee out of the loop
         break;

      else                            //default case of someone lining up.
      {
         Class = command;             //since our class got saved as a command
         getData(spot, Class, command, waiting);
      }

      if (time == 0)                 //when the last person has been finished
                                     //being helped
      {
         if (!waiting.empty())       //only copy if there is stuff to copy
         {
            serving = waiting.front();
            waiting.pop_front();
            time = serving.getStudentMins();
         }
      }
```

Commented [HJ10]: No wrapper class around the line of students?

Commented [HJ11]: You should never have an IF statement with an empty body

```
      if (serving.getStudentMins() != 0) //only does this while the student
                                         //has time left
      {
         time--;

         cout << (serving.getCommand() == "!!" ? "\tEmergency for "
                  : "\tCurrently serving ")
              << serving.getName()
              << " for class " << serving.getClass()
              << ". Time left: " << serving.getStudentMins() << endl;
         serving.setStudentMins(time);
      }
   }
   while( command != "finished");

   // finished!
   cout << "End of simulation\n";
}


/*******************************
 *a simple function just setting everything
 *for the one in line.
 *********************************/
void setSpot(inLine & spot, const string & Class, const string & name,
             const int & studentMins, const string & command)
{
   spot.setName(name);
   spot.setClass(Class);
   spot.setStudentMins(studentMins);
   spot.setCommand(command);
}

/************************************
 *gets and sets the data within the deque
 ************************************/
void getData(inLine & spot, string & Class, string & command, Deque<inLine> & waiting)
{
   string name;
   int studentMins;
   cin >> name >> studentMins;
   setSpot(spot, Class, name, studentMins, command);
   if (command == "!!")
      waiting.push_front(spot);
   else
      waiting.push_back(spot);
}
```

Test Bed Results

```
cs235d.out:

Started program
   > Select the test you want to run:
   >    0. Now Serving
   >    1. Just create and destroy a Deque
   >    2. The above plus push items onto the Deque
   >    3. The above plus pop items off the Deque
   >    4. The above plus exercise the error Deque
   > > 1
   > Create a bool Deque using the default constructor
   >    Size:     0
   >    Empty?    Yes
   >    Capacity: 0
   > Create a double Deque using the non-default constructor
   >    Size:     0
   >    Empty?    Yes
   >    Capacity: 10
   > Copy the double Deque using the copy-constructor
   >    Size:     0
   >    Empty?    Yes
   >    Capacity: 10
   >    Destroying the third Deque
   > Test 1 complete
Program terminated successfully

Started program
```

```
   > Select the test you want to run:
   >    0. Now Serving
   >    1. Just create and destroy a Deque
   >    2. The above plus push items onto the Deque
   >    3. The above plus pop items off the Deque
   >    4. The above plus exercise the error Deque
   > > 2
   > Create an integer Deque with the default constructor
   >    Enter integer values, type 0 when done
   >    > 10
   >       Pushed 10 size=1 capacity=1
   >    > 11
   >       Pushed 11 size=2 capacity=2
   >    > 12
   >       Pushed 12 size=3 capacity=4
   >    > 0
   >    Enter integer values, type 0 when done
   >    > 100
   >       Pushed 100 size=1 capacity=4
   >    > 90
   >       Pushed 90 size=2 capacity=4
   >    > 80
   >       Pushed 80 size=3 capacity=4
   >    > 70
   >       Pushed 70 size=4 capacity=4
   >    > 60
   >       Pushed 60 size=5 capacity=8
   >    > 50
   >       Pushed 50 size=6 capacity=8
   >    > 0
   > Test 2 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. Now Serving
   >    1. Just create and destroy a Deque
   >    2. The above plus push items onto the Deque
   >    3. The above plus pop items off the Deque
   >    4. The above plus exercise the error Deque
   > > 3
   > Create a string Deque with the non-default constructor
   > instructions:
   >    +f dog   pushes dog onto the front
   >    +b cat   pushes cat onto the back
   >    -f       pops off the front
   >    -b       pops off the back
   >    *        clear the deque
   >    ?        shows the statistics of the deque
   >    !        quit
   > > +b one
   > > +b two
   > > +b three
   > > ?
   >    Size:     3
   >    Capacity: 4
   > > -b
   >    pop: three
   > > -b
   >    pop: two
   > > -b
   >    pop: one
   > > ?
   >    Size:     0
   >    Capacity: 4
   > > +f alfa
   > > +f beta
   > > +f charlie
   > > ?
   >    Size:     3
   >    Capacity: 4
   > > -f
   >    pop: charlie
   > > -f
   >    pop: beta
   > > -f
   >    pop: alfa
   > > ?
   >    Size:     0
   >    Capacity: 4
   > > +f three
   > > +f two
   > > +f one
   > > +b four
   > > +b five
   > > +b six
   > > ?
   >    Size:     6
   >    Capacity: 8
   > > -f
   >    pop: one
   > > -f
   >    pop: two
   > > -f
   >    pop: three
   > > -f
   >    pop: four
   > > -f
   >    pop: five
   > > -f
   >    pop: six
   > > ?
   >    Size:     0
   >    Capacity: 8
   > > +b delta
   > > +b echo
   > > +b foxtrot
   > > +f charlie
   > > +f bravo
   > > +f alfa
   > > -b
   >    pop: foxtrot
   > > -b
   >    pop: echo
   > > -b
   >    pop: delta
   > > -b
   >    pop: charlie
   > > -b
   >    pop: bravo
   > > -b
   >    pop: alfa
   > > !
   > Test 3 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. Now Serving
   >    1. Just create and destroy a Deque
   >    2. The above plus push items onto the Deque
   >    3. The above plus pop items off the Deque
   >    4. The above plus exercise the error Deque
   > > 4
   > Create a char deque with the default constructor
   >    Deque::front() error message correctly caught.
   >    "ERROR: unable to access data from an empty deque"
   >    Deque::back() error message correctly caught.
   >    "ERROR: unable to access data from an empty deque"
   >    Deque::pop_front() error message correctly caught.
   >    "ERROR: unable to pop from the front of empty deque"
   >    Deque::pop_back() error message correctly caught.
   >    "ERROR: unable to pop from the back of empty deque"
   > Test 4 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. Now Serving
   >    1. Just create and destroy a Deque
   >    2. The above plus push items onto the Deque
   >    3. The above plus pop items off the Deque
   >    4. The above plus exercise the error Deque
   > > 0
   > Every prompt is one minute.  The following input is accepted:
   >    <class> <name> <#minutes>    : a normal help request
   >    !! <class> <name> <#minutes> : an emergency help request
   >    none                         : no new request this minute
   >    finished                     : end simulation
   > <0> cs124 Sam 2
   >    Currently serving Sam for class cs124. Time left: 2
   > <1> none
   >    Currently serving Sam for class cs124. Time left: 1
   > <2> none
   > <3> cs124 Sue 3
   >    Currently serving Sue for class cs124. Time left: 3
   > <4> cs165 Steve 2
   >    Currently serving Sue for class cs124. Time left: 2
   > <5> !! cs124 Joseph 1
   >    Currently serving Sue for class cs124. Time left: 1
   > <6> none
   >    Emergency for Joseph for class cs124. Time left: 1
   > <7> none
   >    Currently serving Steve for class cs165. Time left: 2
   > <8> cs124 Sam 1
   >    Currently serving Steve for class cs165. Time left: 1
   > <9> none
   >    Currently serving Sam for class cs124. Time left: 1
   > <10> none
   > <11> finished
   > End of simulation
Program terminated successfully

No Errors
```

Grading Criteria

Criteria	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%	Weight	Score
Deque interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	lesson04.cpp compiles without modification	All of the methods in Deque match the problem definition	Deque has many of the same interfaces as the problem definition	The public methods in the Deque class do not resemble the problem definition	20	-1
Deque Implementation	Passes all four Deque testBed tests	Passes three testBed tests	Passes two testBed tests	Passes one testBed test	Program fails to compile or does not pass any testBed tests	20	
Now Serving	The code demonstrates Object-Oriented design principles	Passes the Now Serving testBed test	The code essentially works but with minor defects	Elements of the solution are present	The Now Serving problem was not attempted	30	-1
Code Quality	There is no obvious room for improvement	All the principles of encapsulation and modularization are honored	One function is written in a "backwards" way or could be improved	Two or more functions appears "thrown together."	The code appears to be written without any obvious forethought	20	-1
Style	Great variable names, no errors, great comments	No obvious style errors	A few minor style errors: non-standard spacing, poor variable names, missing comments, etc.	Overly generic variable names, misleading comments, or other gross style errors	No knowledge of the BYU-I code style guidelines were demonstrated	10	
Total							97

Commented [HJ12]: These are just a bunch of nit-picks. Well done!