EECE.3220: Data Structures

Spring 2017

Exam 2

March 31, 2017

Name: ____

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cell phones) are prohibited. If you have a cell phone, please turn off your ringer prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer. In particular, note that:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—we will assume it is part (c) if you mark none of them.**
- For each part of Question 3, you must complete a short function. We have provided comments to describe what your program should do and written some of the code.
  - Note that each function contains both lines that are partially written and blank spaces in which you must write additional code. **You must write all code required to make each program work as described—do not simply fill in the blank lines.**
- You can solve each part of Question 3 using only the variables that have been declared, but you may declare and use other variables if you want.

You will have 50 minutes to complete this exam.

Q1: Multiple choice	/ 20
Q2: Stacks and queues	/ 40
Q3: Operator overloading; lists	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

1. (20 points, 5 points per part) ***Multiple choice***
For each of the multiple choice questions below, clearly indicate your response by circling or underlining the one choice you think best answers the question.

a. Say you want to design a matrix class (i.e., a two-dimensional table of numbers). Which of the following possible implementations requires you to write your own copy constructor and assignment operator, rather than just using the compiler-generated default versions?

A.
```
class Matrix {
…
private:
   double m[MAX_ROWS][MAX_COLS]; // Statically allocated
}                                //  2D array to hold matrix
```

B.
```
class Matrix {
…
private:
   int nr, nc;   // Number of rows/cols
   double **m;   // Pointer to dynamically allocated
}                //  2D array to hold matrix
```

C.
```
class Matrix {
…
private:
   double m[MAX_ROWS*MAX_COLS];  // Statically allocated
}                                //  1D array to hold matrix
```

D.
```
class Matrix {
…
private:
   Node *m;      // Pointer to linked list to hold matrix
}                // Assume each Node holds row #, col #, and
                 //   actual value in that matrix entry
```

i. Only A

ii. Only B

iii. A and C

iv. B and D

v. B, C, and D

1 (continued)

b. You are given the following function prototype and definition:

```
int f(int x, int y = 32);      // Prototype

int f(int x, int y) {          // Definition
    return y / x;
}
```

Which of the following function calls will result in the variable v being set to 4?

A. `v = f(8);`

B. `v = f(4);`

C. `v = f(10, 40);`

D. `v = f(2, 8);`

i. Only A

ii. Only B

iii. C and D

iv. B and C

v. A, C, and D

1 (continued)

c. Given two integers, `x` and `y`, if `x = 32` and `y = 20`, which of the following function calls will print an error message and terminate the program?

i. `assert(x > y);`

ii. `assert(x < y);`

iii. `assert(x != 0);`

iv. None of the above

d. Which of the following statements accurately reflect your opinion(s)? Circle all that apply (but please don't waste too much time on this "question")!

i. "Are we done yet?"

ii. "I hope we get Program 4 before the end of the semester."

iii. "I hope we get assignment grades before the end of the semester."

iv. "I hope the rest of the exam is as easy as this question."

2. (40 points) ***Stacks and queues***
For each short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so that I can easily recognize your final answer.

Please refer to the additional handout provided with the exam for a description of `Stack` and `Queue` functions.

a. (14 points)

```
int main() {
   Stack st1;
   int i;
   int m[16] = { 0,   1, -2,  -3,
                -4,   5,  6,  -7,
                -8,  -9, 10, -11,
               -12, -13, 14,  15};

   for (i = 0; i < 16; i++) {
      if (m[i] >= 0)
         st1.push(i);
      else {
         cout << i << " ";
         i = i + 2;
      }
   }
   cout << "\n";

   while (!st1.empty()) {
      cout << st1.top() << " ";
      st1.pop();
   }
   cout << "\n";
   return 0;
}
```

2 (continued)
b. (12 points)

<u>Please refer to the additional handout provided with the exam for a description of `Stack` and `Queue` functions.</u>

```
int main() {
   Queue Qbert;
   int i;
   for (i = 10; i > 0; i--) {
      switch (i % 3) {
         case 0:
            if (!Qbert.empty())
               Qbert.enqueue(Qbert.front());
            break;
         case 1:
            Qbert.enqueue(i);
            break;
         case 2:
            Qbert.dequeue();
            break;
      }
   }

   while (!Qbert.empty()) {
      cout << Qbert.front() << "\n";
      Qbert.dequeue();
   }
   return 0;
}
```

2 (continued)
c. (14 points)

<u>Please refer to the additional handout provided with the exam for a description of `Stack` and `Queue` functions.</u>

```
int main() {
   Stack S;
   Queue Q;

   int list[10] = {3, 10, 0, -5, 9,
                   78, 6, 3220, 1, 1146};

   for (int i = 0; i < 10; i = i + 2)
      S.push(list[i]);

   while (!S.empty()) {
      cout << "S " << S.top() << "\n";
      Q.enqueue(list[S.top()]);
      S.pop();
   }

   while (!Q.empty()) {
      cout << "Q " << Q.front() << "\n";
      Q.dequeue();
   }
   return 0;
}
```

4. (40 points) ***Operator overloading; lists***

For each part of this problem, you are given a short function to complete. **CHOOSE TWO OF THE THREE PARTS** and fill in the spaces provided with appropriate code.

You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.

Remember, you must write all code required to make each function work as described—**do not assume you can simply fill in the blank lines and get full credit.**

Also, remember that if examples are provided, each example is only applicable in one specific case—**it does not cover all possible results of using that function.**

In order to allow plenty of space to solve each problem, this page is essentially just a "cover sheet" for Question 3—**the actual problems start on the next page**.

Each of these functions works with one of the List implementations. **You can find the class definitions on the extra sheet provided with the exam.**

4 (continued)

a. `AList operator -(const AList &rhs);`

This binary subtraction operator, used with static array-based lists, returns a new list which contains values that are only in the list to the left of the - operator, not in both the left and right.

For example, if `AList L1` holds {1, 2, 5, 9, 13} and `AList L2` holds {2, 5, 13}, the result of `L1 - L2` would be a list holding {1, 9}—in essence, the values in `L2` are removed from `L1`.

To simplify your solution, assume (1) both lists are sorted from lowest to highest values, and (2) the last value in the right hand list is greater than or equal to the last value in the left hand list.

```
AList AList::operator -(const AList &rhs) {
  AList result;          // Result of subtract operation
  int LHindex, RHindex;  // Index vars for left/right hand sides
  int i;                 // Index into result

  // Initialize variables




  // Loop while LHindex/RHindex point to valid array locations

  while (_______________________________________________________

         ___________________________________________________) {
    // Search RHS until you find value that matches or exceeds
    // current value in LHS, or run out of room in RHS

    while (_________________________________________________) {


    }
    // If value should be placed in result, add it

    if (____________________________________________________) {



    }
    // Otherwise, if match found, skip to next value in LHS

    else if (_________________________________________________)

  }
  return result;
}
```

4 (continued)

b. `bool operator ==(const DList &rhs);`

This binary comparison operator, used with dynamic array-based lists, returns true if the two lists (left- and right-hand-side) match and false otherwise. In order for two lists to match, the following conditions must be true:

- The lists must be the same size (but not necessarily the same capacity)
- The lists must store the same set of values

```
bool DList::operator ==(const DList &rhs) {
  int i;      // Index

  // If sizes don't match, lists don't match—return appropriate
  //    value

  if (_______________________________________________________) {



  }

  // If sizes do match, check to see if array contents match
  // If mismatch found, can end function at that point
  else {









  }

  // If sizes and array contents match, lists match—return
  //    appropriate value




}
```

4 (continued)

c. `LList & operator ~();`

This unary operator, used with linked lists, reverses the node order of the calling object. For example, if `LList L1` contains nodes `1 -> 2 -> 3 -> 4`, the statement `~L1;` would change the list so the nodes were in the order `4 -> 3 -> 2 -> 1`.

The solution requires you to keep three pointers: the current node, its original predecessor, and its original successor. Your function should visit all nodes in the list. For each node, (1) make the node point to its original predecessor, then (2) move all three pointers ahead by one node.

<u>Assume—as with the linked list covered in class—that the `insert()` function adds nodes so they're ordered from lowest to highest value, so you can't use that function in your solution.</u>

```
LList & LList::operator~() {
   Node *pred;       // Original predecessor of current node
   Node *cur;        // Current node
   Node *succ;       // Original successor of current node

   // Make sure list isn't empty

   if (_______________________________________________________) {
      // Initialize node pointers



      // Visit & reorder nodes while there's a non-NULL successor

      while (_________________________________________________) {







      }
      // Make sure old last node points to its new successor, and
      //   "first" pointer points to new first node


   }
   // Return reference to calling object


}
```