University of Illinois at Urbana-Champaign
Department of Computer Science

First Examination RUBRIC

CS 225 Data Structures and Software Principles
Fall 2010
7p-9p, Tuesday, September 28

Name:	
NetID:	
Lab Section (Day/Time):	

- This is a **closed book** and **closed notes** exam. No electronic aids are allowed, either.
- You should have 5 problems total on 20 pages. The last two sheets are scratch paper; you may detach them while taking the exam, but must turn them in with the exam when you leave.
- Unless otherwise stated in a problem, assume the best possible design of a particular implementation is being used.
- Unless the problem specifically says otherwise, (1) assume the code compiles, and thus any compiler error is an exam typo (though hopefully there are not any typos), and (2) assume you are NOT allowed to write any helper methods to help solve the problem, nor are you allowed to use additional arrays, lists, or other collection data structures unless we have said you can.
- We will be grading your code by first reading your comments to see if your plan is good, and then reading the code to make sure it does exactly what the comments promise. In general, complete and accurate comments will be worth approximately 30% of the points on any coding problem.
- Please put your name at the top of each page.

Problem	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **[Pointers, Parameters, and Miscellany – 20 points].**

MC1 (2.5pts)

Consider the following statements, and assume the standard `iostream` library has been included:

```
int v;
int * w;
v = 10;
w = v;
*w = 8;
cout << v << endl;
```

What is the result of executing these statements?

(a) 8 is sent to standard out.
(b) 10 is sent to standard out.
(c) This code does not compile.
(d) This code results in a runtime error.
(e) None of these options is correct.

MC2 (2.5pts)

Consider the following statements, and assume the standard `iostream` library has been included:

```
int b;
int * a = new int(8);

b = *a;
*a = 10;

cout << b << endl;
delete b;
```

What is the result of executing these statements?

(a) 8 is sent to standard out.
(b) 10 is sent to standard out.
(c) This code does not compile.
(d) This code results in a runtime error.

(e) None of these options is correct.

Note that this problem contains a typo. I intended the correct response to be a), but the typo (`delete b` should be `delete a`) causes in a compiler error so we took either answer.

MC3 (2.5pts)

What is the output of the following sequence of C++ statements? (The sphere class interface is included at end of the exam.)

```
sphere * a, * b;

a = new sphere(1.0);
b = a;
b->setRadius(2.0);
delete b;

a->setRadius(4.0);
sphere * c = new sphere(5.0);
b = new sphere(3.0);

cout << a->getRadius() << endl;
```

(a) 4.0

(b) 3.0

(c) A segmentation fault.

(d) Compiler error.

(e) The behavior cannot be predicted. (insidious bug)

MC4 (2.5pts)

Consider the following statements, and assume the standard `iostream` library has been included:

```
void doub(int x) { x = x * 2;}
void trip(int * x) { *x = *x * 3; }
void quin(int & x) { x = x * 5; }

int main() {

    int x = 1;

   doub(x);
   trip(&x);
   quin(x);

    cout << x << endl;

    return 0;
}
```

What is the result of executing these statements?

(a) 1 is sent to standard out.

(b) 5 is sent to standard out.

(c) 6 is sent to standard out.

(d) 10 is sent to standard out.

(e) 15 is sent to standard out.

MC5 (2.5pts)

Which of the following correctly declares a dynamic array of `List`s of pointers to `string`s?

(a) `List<string> ** name;`

(b) `List<string *> * name;`

(c) `List * name = new string *[size]`

(d) More than one of (a), (b), (c), are correct.

(e) None of (a), (b), (c), are correct.

MC 6 (2.5pts)

Suppose class `stringGetter` contains exactly one pure virtual function: the overloaded parentheses operator, `string operator()(int x)`. Also suppose that class `getPageString` is a public `stringGetter` that implements `operator()`.

Which of the following C++ statements will certainly result in a compiler error?

(a) `stringGetter * a = new stringGetter;`

(b) `stringGetter * a = new getPageString;`

(c)
```
stringGetter * a;
getPageString * b = new getPageString;
a=b
```

(d) Exactly two of these will result in a compiler error.

(e) It is possible that none of these will result in a compiler error.

MC 7 (2.5pts)

Which of the following concepts is mentioned in the "Rule of the Big Three?"

(a) copy constructor

(b) constructor

(c) encapsulation

(d) header file

(e) None of these concepts is mentioned in the rule.

MC 8 (2.5pts)

Consider the following class definitions:

```
class Sport{
public:
    int winner() const;
private:
    int score;
};

class VolleyBall: public Sport {
public:
    int loser();
};
```

Where could the assignment `score=20;` appear for the private variable `score`?

(a) Both `winner()` and `loser()` can make the assignment.

(b) `loser()` can make the assignment, but `winner()` cannot.

(c) `winner()` can make the assignment, but `loser()` cannot.

(d) Neither `loser()` nor `winner()` can make the assignment.

(e) The answer to this question cannot be determined from the given code.

2. **[MP2ish – 20 points].** Consider the following partial class definition:

```
class RoadTrip
{
  private:
    string ** destinations;
    int duration;

    // some helper functions

  public:
    // constructors and destructor
    RoadTrip(int num); // constructor for a RoadTrip of num days

    //  operator=   declaration
    //  operator+   declaration

    // lots of other public member functions not relevant to this problem
};
```

The `destinations` structure is a dynamically allocated array of `string` pointers. The array `destinations` has `duration` elements. A place name (of type `string`) is added to the destination structure in the cell whose number corresponds to the first day of arrival at that place. We assume that cells with no destination are simply layovers in the most recent place, and that day 0 contains our starting location. For example, if we start in Urbana, arrive in Milwaukee on the first day, and end in Lincoln on the fifth day, then `destinations[0] == "Urbana"`, `destinations[1] == "Milwaukee"`, `destinations[5] == "Lincoln"`. In this example, the `duration` variable should be 6. You can assume that `duration`, when it is specified, is greater than zero. You can also assume the `string` class has been included and scoped.

You may assume that all pointers are valid. That is, they are either `NULL` or they point to an object of the specified type. In particular, the `RoadTrip(int num)` constructor builds a dynamic array of length `num`, whose elements are all `NULL`, and sets `duration` to `num`.

In this question you will help us implement some of the member functions for the `RoadTrip` class.

You will write your answers on the following pages. To grade the coding portions of this problem, we will first read your comments to make sure you intend to do the right thing, and then we'll check your code to make sure it does what your comments say it should. As a result, be sure your comments are coherent, useful, and reflective of your approach to the problem. Comments will be worth up to 1/3 of the total points for any part of the problem. Adding comments to our code skeletons can get you partial credit, but it's not required.

(a) (8 points) In this part of the problem, you will write the code for an overloaded addition operator, so that two `RoadTrip`s can be added together. The addition of two road trips is a simple concatenation of the days of the trips into one larger adventure. Specifically, If road trips `m` and `n` are declared like: `RoadTrip m(10), n(20);`, then `m + n` is a road trip of length 30 consisting of all the days of `m` followed by all the days of `n`. Recall that some days might be `NULL`. These should be preserved in the `RoadTrip` sum.

```
RoadTrip RoadTrip::operator+(const RoadTrip& rhs) {  // 2 points for function signatu

   RoadTrip m(this->duration + rhs.duration);  // 1 point

   for (int i = 0; i < this->duration; i++) {  // 1 point for correct initialization

      if (this->destination[i] != NULL) {  // 1 point for correct if condition
         m.destination[i] = new string;   // 2 points for copying the elements
         *m.destination[i] = *(this->destination[i]);
      }
   }

   for (int i = 0; i < rhs.duration; i++) {
      if (rhs.duration[i] != NULL) {
         m.destination[duration + i] = new string(*rhs.destination[i]);
      }
   }

return m; // 0.5 for return statement
}
```

(b) (6 points) Write the destructor for the `RoadTrip` class.

Solution:

```
RoadTrip::~RoadTrip()
{
   for( int i = 0; i < duration; i++)
   {
      if( destination[i] != NULL)
      {
         delete destination[i];
         destination[i] = NULL;
      }
   }

delete [] destination;
destination = NULL;
}
```

2 pts for only deleting the array destination

(c) (6 points) The following function is intended to take one destination in a `RoadTrip` and move it to a different location within the same trip. Specifically, the contents of `destinations[target]` are to be replaced by the contents of `destinations[source]`, which should then be left empty, or `NULL`. Unfortunately, the code below doesn't behave the way we intend. Please complete and correct the function. Your code will be partially graded on efficiency (don't copy data if you don't need to). You may assume that `source` and `target` are valid indices, though you cannot assume that their contents are non-`NULL`.

Solution:

```
void RoadTrip::changeDestination(int source, int target)
{
   if ( target == source )
      return;

   delete destination[target];
   destination[target] = NULL;
   // *(destination[target]) = *(destination[source]);
   if ( destination[source] != NULL )
   {
      destination[target] = destination[source];
      destination[source] = NULL;
   }
}
```

2 pts for check if target equals to source
1 pts for deleting destination[target] before copying value
-2 pts if use the existing sentence to copy from source to target
-2 pts if not setting destination[source] to NULL.

3. **[MP3ish – 20 points].**

The following code is a partial definition of a doubly linked list implementation of the `List` class that you used for MP3. Note in particular that it does not contain sentinels, but it *does* have head and tail pointers.

```
template <typename Etype>
class List   {
public:

   // splitList
   //    - parameters : rank - an integer; the rank-th node of the list
   //                       is the first node of the split-off list
   //    - returns the portion of the current list from
   //         the rank-th node onward; the current list is reduced
   //         to the portion occurring before the rank-th node
   List<Etype> splitList(int rank);

   // a bunch of other List class functions

private:

   class ListNode
   {
   public:
      // ListNode constructor
      //    - initializes element to default Etype, and pointers to NULL
      ListNode();

      // ListNode constructor
      //    - parameters : value - the value to store in the element field
      //    - initializes node to hold value and NULL pointers
      ListNode(Etype const & value);

      // Maybe some other functions here.

      ListNode* next; // pointer to next node in list
      ListNode* prev; // pointer to prior node in list
      Etype element;   // holds element of node

   };

   ListNode* head;    // points to first node of list
   ListNode* tail;    // points to last node of list
   int size;
};
```

(a) (10 points) Please implement the `splitList` function (as you did for MP3) This function finds the rank-th element of the list, and detaches the portion of the list starting there from the preceding part of the list. The detached portion is then placed in a new list. If rank is greater than size, then return an empty list (leaving the original intact). The possible values for rank are integers greater than or equal to 1. That is, You can assume that the integer argument rank is positive. Note that the node pointed to by the List class' head pointer is considered to be the "first" node, and not the "zeroeth". For this function, you may declare a single variable of type`List<Etype>`, for the sole purpose of holding the list that is returned to the calling function. However, you are not allowed to create any new nodes in this function – you can only manipulate the existing nodes.

Be sure to make your code very neat and easy to read. Draw pictures where it will help us, and comment profusely.

Solution:

```
template <typename Etype>
void List<Etype>::appendList(List<Etype> & endList)
{

   typename List<Etype>::ListNode* lastCur = tail;
   typename List<Etype>::ListNode* firstEnd = endList.head;

   size += endList.size;

   if (head == NULL)            // current list is empty
   {
      head = endList.head;
   }
   else
   {
      lastCur->next = firstEnd;
   }

   if (endList.head != NULL)    // param list is nonempty
   {
      tail = endList.tail;
      firstEnd->prev = lastCur;
   }

   endList.head = NULL;
   endList.current = NULL;
   endList.tail = NULL;
   endList.size = 0;
}
```

+1 Overall Correctness.
+2 If the first list is empty, set the head of the first list.
+1 If the second list is empty, return.
+2 Appending lists.
+1 Setting the tail of the first list.
+2 Setting the size of the first and second list.
+1 Setting the head and tail of second list to NULL.

(b) (5 points) Give a tight worst case upper bound for the running time of your implementation of `splitList` if the list contains n items.

Solution: The running time depends on the solution provided for 3a, but the best worst case running time is $O(n)$. This is because when the location to split is at the ? $n/2$ location of the list (worst case), it takes ?n/2? operations to iterate to get there (by starting at the front of the list or at the back of the list), and then a constant amount of work to split the list. Thus the total running time is $O(n/2 + c) = O(n)$.

+5 For correct running time and brief explanation.
+5 For correct running time and no explanation.
+2 For incorrect running time and correct explanation. (some answers had two separate for loops as $O(n^2)$?but their explanation of the running time was correct)
+0 For incorrect running time and incorrect/no explanation.

(c) (3 points) Does the `ListNode` class require a destructor? Briefly justify your answer.

Solution: No, it is not required to implement a destructor since the ListNode class itself never allocates any dynamic memory. The default destructor that is provided will call the destructor of the Etype if it is an Object type. Additionally, the memory that the ListNode?s prev and next pointers point to will be deleted by the List class that contains them.

+3 For a correct answer explaining that a destructor is not required since it doesn?t allocate any dynamic memory.
+2 For saying it isn?t required but an incorrect explanation such as if the destructor was implemented it will chain react and delete all the ListNodes in the List.
+0 For saying the destructor is required.

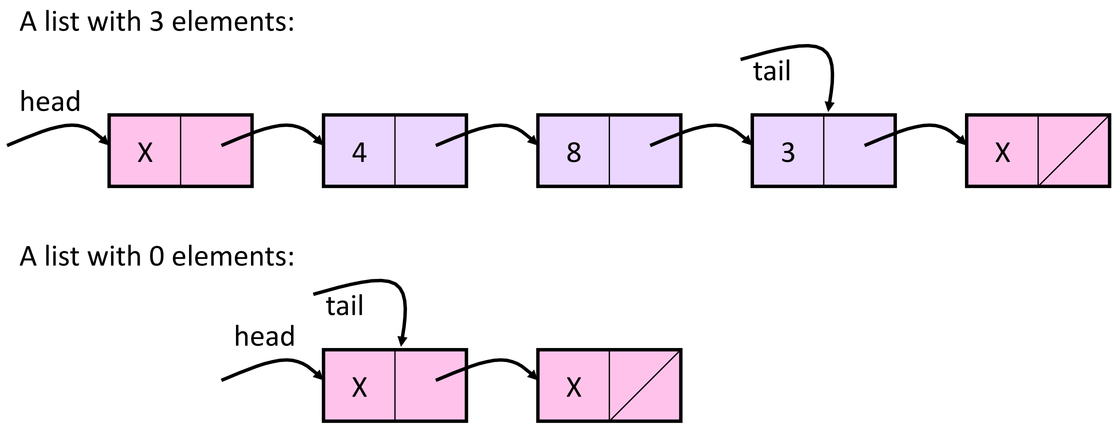
(d) (2 points) Explain why the `ListNode` class is private within the `List` class.

Solution: The ListNode class is declared in the private section of the List class to hide the implementation details of the List class. This is because the client using the List class shouldn?t need to know how the List is implemented (i.e. it may be implemented using ListNodes and pointers, or it could be array based.).

+2 For a correct answer explaining that it is to hide the implementation details.
+0 For saying it is to prevent people from modifying the List. (This is incorrect because we could use any object that is declared any where and as long as it is declared as a private member variable others won?t be able to modify it in the List class.)

4. **[Links – 20 points].**

Suppose you have implemented a List using a singly linked list with sentinels at the head and tail, and a tail pointer pointing to the node *before* the tail sentinel node. In the general case, this corresponds to the last data item in the list as in the figure below, and in an empty list, it corresponds to the head sentinel. In this problem you will implement and analyze some of the member functions for this list class.



Here is a partial `List` class definition (continued on next page):

```
template <class LIT>
class List{
public:
   List(const LIT & e);
   List(const List & orig);
   ~List();
   void insert(int loc, const LIT & e);
   void removeLast();
   // lots more member functions
private:
   listNode * head;
   listNode * tail;
   listNode * curr;
   int size;        // the number of data elements in the list

   listNode * Find(int k, listNode * curr);

   struct listNode {
     LIT data;
     listNode * next;
     listNode(LIT e): data(e), next(NULL) {} ;
     listNode(): next(NULL) {} ;
   };};
```

The `Find` private helper function returns a pointer to the listNode that is `k` nodes beyond the input listNode *. For example, `Find( 2, head)` returns a pointer to the node containing the value 8 in the example above. You may use this helper function anywhere you'd like.

(a) (5 points) Write the default (no argument) constructor for the `List` class. To make this easy for us to grade, place the function prototype above the first line, and then write the rest of your code between the brackets. (Note: the `next` pointer for the tail sentinel should be `NULL`.)

```
template <class LIT>
//template <typename LIT> also acceptable
List < LIT> :: List() // +0.5 point
{
head = new listNode(); // +1 for first sentinel node
tail = head; // +1 for proper handling tail pointer
head -> next = new listNode(); // +1 for second sentinel node
size = 0; // +0.5
} // +1 for comments
```

(b) (5 points) We've written part of the code for the member function `removeLast` below. Your task is to complete the function. `removeLast()` removes the last data item in the list if the list is not empty, and does nothing, otherwise. In the non-empty example on the previous page, `removeLast()` would remove the node containing value 3.

```
template < class LIT>
void List <LIT> :: removeLast()
{
if (size == 0)  // +0.5 for if statement
return;
else
{
listNode * temp = Find (size-1, head); // +0.5 for locating previous to last node
temp -> next = tail -> next; // +1 for reassigning pointer to sentinel node
delete tail; // +1 for de-allocating tail node
tail = temp; // +1 for maintaining tail pointer
size --; // +0.5
} // +0.5 for comments
}
```

(c) (8 points) Complete the table below with tight asymptotic running times (using big-O notation) for the following `List` class functions on data of size n. In the table we compare the linked memory implementation of this problem with the best array-based implementation we can imagine. You may assume that the space in the array is sufficient for all list operations.

	Singly linked list	Array (front -> rear)	Array (rear -> front)	wrapping around array
insertAtFront	O(1)	O(n)	O(1)	O(1)
insertAtRear	O(1)	O(1)	O(n)	O(1)
removeFront	O(1)	O(n)	O(1)	O(1)
removeRear	O(n)	O(1)	O(n)	O(1)

(d) (2 points) Explain the role of the tail sentinel node in the functions listed in the table above. Would the running times of these functions change if there were no sentinel?

Solution:

Running time doesnt change due to a presence of tail sentinel node in our case.

Note: O(1) hack for removeRear doesnt work in our case as we still need to reassign tail pointer to the previous element in the singly linked list and thus have to iterate through the list giving us running time of O(n).

The role of tail sentinel node is mainly to simplify the function implementation by eliminating the need for conditional checks in edge cases (no need to check for NULL for tail-¿next pointer)

5. **[Miscellaneous – 20 points].**

(a) (5 points) We have tried to write the overloaded assignment operator for the `Food` class, but our code isn't working! Assume we have tested the `copy` and `clear` functions, and they do what they're supposed to do. Circle the errors in our function, and then write the correct code on the right, very clearly.

```
Food::operator=(Food & rhs){

   if( *this !=  rhs) {
      copy(rhs);
      clear();
   }

   return this;
}
```

Rubric:
5 Errors
1. Return Type: 0.5 points for identifying return type was missing. 0.5 points for correctly fixing it. (Food & const Food::operator=)
2. Parameter: 0.5 points for identifying the parameter was wrong. 0.5 points for changing it to const
3. If Statement: 0.5 points for identifying the if statement was wrong. 0.5 points for correcting it to compare addresses. (if (this != &rhs))
4. Copy and Clear: 0.5 points for identifying something wrong with the function calls. 0.5 points for switching the order they are called in.
5. Return value: 0.5 points for identifying the return is wrong. 0.5 for correcting it to return an object, not a pointer, (return *this;).

Now, answer the 3 questions below based on your corrected code.

i. (2 points) Explain the type specification in the parameter list. In particular, why is the parameter passed by reference?
Solution: 2 points for saying it is more efficient to pass by reference, but const so that we cant change it.
1 point for saying we dont want to copy the parameter but didnt say why we didnt want to copy it.

ii. (2 points) Justify the conditional (`if`) by explaining its purpose.
*Solution:*2 points for saying we want to avoid self assignment by comparing the address of the lhs and the rhs.
1 point for explaining what the IF did, but not WHY we made that check.

iii. (2 points) Explain the return value. What is its purpose?

Solution: 2 points for explaining that we return the value to be used by chaining operators (or so that we can directly use the lhs without an extra line of code).
1 point for saying we return the object by reference, but didnt explain its purpose.

(b) (2 points) Briefly describe a situation in which we would choose to pass an object by reference, rather than by value. Please give an example different from the one in the previous problem.

Solution: 2 points for an example that was either to save memory on a large object, or the copy constructor, where passing by value would result in an infinite loop.
1 point if the example had some logical flaws in it, or it didnt compare passing by reference to passing by value.

(c) (2 points) Briefly describe one scenario in which we would choose to pass an object by reference, over passing a pointer to the object by value. Please give a reason that is more substantive than mere syntactic simplicity.

Solution: 2 points their example said it was slower and more dangerous to dereference points all the time as opposed to just using the reference object. Or if you wanted to change where that pointer was pointing toyou couldnt do that with pointer by value.
1 point if their example looked like they knew the benefit of pass by reference, but didnt exactly compare it to pass pointer by value (as opposed to pass object by value)

(d) (2 points) Briefly describe two instances when the copy constructor is invoked.

1 point for each example, up to 2 points. (Taken from the three examples)

- Explicitly called Sphere a = new Sphere(); Sphere b = new Sphere(a);
- Pass an object by value
- Return an object by value.