University of Illinois at Urbana-Champaign
Department of Computer Science

First Examination RUBRIC

CS 225 Data Structures and Software Principles
Spring 2013
Tuesday, February 19, 7-10p

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

- This is a **closed book** and **closed notes** exam. No electronic aids are allowed.
- You should have 5 problems total on 19 pages. The last sheet is scratch paper; you may detach it while taking the exam, but must turn it in with the exam when you leave.
- The points assigned to each problem are a *rough* estimate of the time it should take you to solve the problem. Use your time wisely.
- 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.
- Please put your name at the top of each page.

Problem	Points	Score	Grader
1	35		
2	20		
3	20		
4	20		
5	5		
Total	100		

1. **[Pointers, Parameters, and Miscellany – 25 points].**

MC1 (2.5pts)

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

```
int a = 12;
int * b = &a;
a = 15;
cout << *b << endl;
```

What is the result of executing these statements?

(a) 12 is sent to standard out.

(b) 15 is sent to standard out.

(c) The memory address of `b` is sent to standard out.

(d) This code does not compile.

(e) This code may result in a runtime error.

MC2 (2.5pts)

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

```
int c = 37;
int * d;
*d = c;
c++;
cout << *d << endl;
```

What is the result of executing these statements?

(a) 37 is sent to standard out.

(b) 38 is sent to standard out.

(c) The memory address of `d` is sent to standard out.

(d) This code does not compile.

(e) This code results in a runtime error.

MC3 (2.5pts)

Consider the following class definitions:

```
class Season{
public:
        virtual void adjustTemp(int change);
private:
        int temp;
};
class Winter: public Season {
public:
        void makeColder(int change);
};
```

Where could the assignment `temp += change;` appear for the private variable `temp`?

(a) Both `adjustTemp` and `makeColder` can make the assignment.
(b) `adjustTemp` can make the assignment, but `makeColder` cannot.
(c) `makeColder` can make the assignment, but `adjustTemp` cannot.
(d) Neither `makeColder` nor `adjustTemp` can make the assignment.
(e) The answer to this question cannot be determined from the given code.

MC4 (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;
delete a;
a = new sphere(7.0);

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

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

(a) 3.0
(b) 7.0
(c) An insidious runtime memory error.
(d) A memory leak.
(e) A compiler error on line `b->setRadius(3.0);`.

MC5 (2.5pts)

Consider this definition of the function `badbad`:

```
int * badbad(int & y) {
   int * x = &y;
   y = 15;
   return x;
}
```

Which of the following statements is true?

(a) This code is bad because it returns the memory address of a local variable.
(b) This code is bad because there is at least one type mismatch.
(c) This code is bad because the parameter is not `const int & y`.
(d) This code is bad for more than one of these reasons.
(e) This code is not bad at all, despite its name.

MC6 (2.5pts)

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

```
void funwon(int y) { y = 1;}
void funtoo(int * y) { *y = *y + 2; }
void funfor(int & y) { y = y + 4; }

int main() {

    int x = 0;
    funwon(x);
    funtoo(&x);
    funfor(x);

    cout << x << endl;
    return 0;
}
```

What is the result of executing these statements?

(a) 4 is sent to standard out.
(b) 5 is sent to standard out.
(c) 6 is sent to standard out.
(d) This code does not compile.
(e) None of these options is correct.

MC7 (2.5pts)

```
class Bear {
  public:    Bear() { cout << "Growl" << endl; }
            ~Bear() { cout << "Stomp stomp stomp" << endl; }
};
int main()  {
    Bear * beary = new Bear;
    cout << "Run!" << endl;
    delete beary;
    return 0;  }
```

What is the result of compiling and executing this code (assume `iostream` is included)?

(a) Run!

(b) Growl
Run!

(c) Growl
Run!
Stomp stomp stomp

(d) Run!
Stomp stomp stomp

(e) This code does not compile.

MC8 (2.5pts)

Consider the following class definition:

```
class Sprite {
  public:
    Sprite();
    Sprite(const Sprite & other);
    // more public member functions
  private:
    // some private member variables
};
```

Which of the following functions must also be implemented for the Sprite class for it to function correctly?

(a) one parameter constructor

(b) destructor

(c) operator()

(d) setRadius

(e) operator delete

MC9 (2.5pts)

Which of the following is a correct function signature for the overloaded addition operator for the sphere class, if we want that operator to return a sphere whose radius is the sum of the radii of the object and its parameter?

(a) `sphere sphere::operator+(const sphere & right) const;`

(b) `sphere & sphere::operator+();`

(c) `sphere sphere::operator+(const sphere & left, const sphere & right);`

(d) More than one of (a), (b), (c), could be used.

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

MC10 (2.5pts)

Suppose an algorithm takes 8 seconds to run serially, and 3 seconds to run in parallel. Then the *speedup* for the parallelized code is:

(a) $\frac{8}{3}$

(b) $\frac{8-3}{8}$

(c) $\frac{3}{8}$

(d) The speedup cannot be determined because the number of processors is not known.

(e) None of these answers is correct.

MC11 (2.5pts)

Suppose class `pictureRep` contains exactly one pure virtual function: the overloaded parentheses operator, `int operator()(int i, int j)`. Also suppose that class `hardPNG` is a public `pictureRep` that implements `operator()`.

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

(a) `hardPNG * a = new pictureRep;`

(b) `hardPNG * a = new hardPNG;`

(c)
```
pictureRep * a = new hardPNG;
hardPNG * b;
a = b;
```

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

(e) None of these will result in a compiler error.

MC12 (2.5pts)

Which of the following characterizes an Abstract Data Type?

(a) Any class containing at least one pure virtual function.

(b) A collection of class data members.

(c) A description of the implementation of a data structure.

(d) A description of the functionality of a data structure.

(e) None of these options is correct.

MC13 (2.5pts)

Which of the following concepts describe the behavior of the virtual function `speak()` in the following code snippet, if the result is "ruff, moo, oink?"

```
animal ** farm;

farm = new animal*[3];
farm[0] = new dog;
farm[1] = new cow;
farm[2] = new pig;

for (int i=0; i<3;i++)
    farm[i]->speak();
```

(a) `speak()` is encapsulated

(b) `speak()` is polymorphic

(c) `speak()` is templatized

(d) `speak()` is a pure virtual function

(e) None of these options is correct.

MC14 (2.5pts)

Consider the following code, and assume the standard `iostream` and `list` class have been included.

```
template < class Iter, class Formatter >
bool mystery(Iter first, Iter second, Formatter runit) {
   bool p = true;
   while (first != second) {
        p = (p && runit(*first, *second));
        first++;
        second--;  }
   return p;
}

class noClue {
public:
   bool operator()(int a, int b) {
        return (a==b);  }
};

int main() {
   list<int> s;
   // some number of insertions to s here
   list<int>::iterator it1 = s.begin();
   list<int>::iterator it2 = s.end();
   it2--;

   noClue dunno;
   if ( mystery<list<int>::iterator, noClue>(it1, it2, dunno))
      cout << "yes" << endl;
   else  cout << "no" << endl;

   return 0; }
```

Which of the following statements is true?

(a) This code does not compile because of a type mismatch in the `mystery` parameter list.

(b) This code does not compile because of a syntax error in the template instantiation for `mystery`.

(c) If the list consists of the integers 1, 2, 3, 2, 1, 4 *in that order, with the first item on the left*, then the output is "yes".

(d) If the list consists of the integers 1, 2, 3, 2, 1 *in that order, with the first item on the left*, then the output is "yes".

(e) None of these options describes the behavior of this code.

2. **[MP2ish – 20 points].**

Consider the following partial class definition:

```
class photoLibrary
{
  private:
    PNG ** albums;
    int * albumSizes;
    int numAlbums;

    // some helper functions

  public:
    // constructors and destructor

    photoLibrary & operator=(const photoLibrary & rhs);

    // lots of public member functions
};
```

The `albums` structure is a dynamically allocated array of dynamically allocated arrays of `PNG`s. The `albumSizes` structure is a dynamically allocated array of ints, whose values are the lengths of the elements of the `albums` structure. In other words, for any `i`, `albums[i]` is an array of length `albumSizes[i]`.

Both `albums` and `albumSizes` arrays have `numAlbums` elements. You can assume that `numAlbums` is greater than zero.

In this question you will implement the overloaded assignment operator for the `photoLibrary` class.

You may assume that all pointers are valid. That is, they are either `NULL` or they point to an object of the specified type.

You will write your answers on the following pages. To grade 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.

(a) (5 points) Write a helper member function `void clear()` that frees all memory used by an instance of `photoLibrary`. Note that this function would be called by the `photoLibrary` destructor.

```
void photoLibrary::clear() // 1 Point for correct syntax
{
   for (int i=0;  i<numAlbums;   i++)
   {
      delete [] albums[i];     // 2 Points for For loop and delete [] albums[i]
   }
   delete [] albums; //  1Point
   delete [] albumSizes; //  1 Point
}
```

(b) (6 points) Write a helper member function `void copy(const photoLibrary & orig)` that makes the current object have the same *value* as the parameter `orig`. `copy` assumes that the current object has no dynamic memory associated with it. Note that this function would be called by the `photoLibrary` copy constructor.

```
void photoLibrary::copy(const photoLibrary & orig)  //  1 Point for correct syntax
{
   numAlbums=orig.numAlbums;
   album= new PNG*[numAlbums];
   albumSizes= new int[numAlbums];  //  1 pt for numAlbums,  space for album,  space i
   for (int i = 0;  i<numAlbums ; i++ )
   {
      albumSizes[i] = orig.albumSizes[i];  // 1 pt for copying into albumSizes correct
      if( orig.albums[i]!=NULL) {
         album[i]=new PNG[albumSizes[i]];   // 1 pt for creating new space for each al
         for(int j=0; j < albumSizes[i]; j++)
         {
            albums[i][j] = orig.albums[i][j];     //  2 pt for copying elements of ori
         }
      }
      else
         albums[i]=NULL;
   }
}
```

(c) (5 points) Write the member function `photoLibrary & operator=(const photoLibrary & rhs)` for the `photoLibrary` class. You should use the helper functions you wrote in parts (a) and (b).

```
photoLibrary & photoLibrary::operator=(const photoLibrary & rhs) {
 if (this != &rhs) {
   clear();
   copy(rhs);
```

```
 }
 return *this;
}
```

RUBRIC: 1 point per line.

(d) (4 points) List two situations in which the destructor is called by the system.

- When an object goes out of scope (2 points).
- When `delete` is called on a pointer to an object (2 points).

3. **[MP3ish – 20 points].**

The following code is a partial definition of a singly linked list implementation of the `List` class. In our implementation, an empty list contains a single sentinel node whose value is garbage and whose next pointer is null, and a non-empty list contains the sentinel followed by a chain of list nodes containing data.

```
template <typename Etype>
class List  {
public:
   List();
   // insertKth
   //    - parameters : newData, a reference to an Etype object
   //           which will be inserted in the Kth position of the list.
   //    - creates a new node and inserts it into the Kth position
   //            in the list (starting from 1 and ignoring sentinel).
   void insertKth(int k, Etype & newData);

   // a bunch of other List class functions
private:

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

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

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

   // Find
   //   - parameters:  starting position and number of steps
   //   - returns: a pointer to the ListNode k steps forward from start
   ListNode * Find(ListNode * start, int k);

   ListNode* head;    // points to first node of list
   int size;   // number of elements in the list
};
```

(a) (4 points) Write the no-argument constructor for this class.

```
template<Etype>
List<Etype>::List()
{
 head = new ListNode;
 size = 0;
}
```

RUBRIC:
1 pt for setting up template
1 for setting head to point to a sentinel node
1 pt for setting size
1 for function header (other than the template part)
up to 1 point for comments only
-1 for declaring head in function (ListNode * head = ...)
-1 for extra things (such as setting a tail ptr or prev ptr

(b) (5 points) Write a member function insertAt whose function signature is given below. It takes as input a non-NULL ListNode pointer called `current` and a data element of type `Etype` called `data`. The function creates a new ListNode and inserts it into the list as the target of `current`'s `next`.

```
void List::insertAt(ListNode * current, Etype \& data) \{
  ListNode  * temp = new ListNode(data);
  temp->next=current->next;
  current->next=temp;
  size++;
}
```

RUBRIC:
2 points for allocating new memory
2 points for inserting memory in the list
1 point for increasing size

(c) (2 points) We have forgotten to include the declaration of `insertAt` in our class definition above. Should we put it in the public or private section of the class? Briefly justify your response.

Private, because one of the parameters is a ListNode which is private so the client has no access to the class.

1 pt for private
1 pt for mentioning ListNode declaration is in the private section

(d) (4 points) Write function `insertKth` which inserts a new node in the `k`th position of the list. You may use other `List` class functions (including the one you wrote in part (b)) in your solution.

```
template<Etype>
void List<Etype>::InsertKth(int k, Etype& newData)
{
   ListNode * Kth=find(head, k-1);
   insert(Kth,newData);
}
```

RUBRIC:
1 point for template
1 point for setting up pointer to k-1 position
1 point for inserting
-1 point for not using already written functions (vs re-writing everything)

(e) (3 points) Briefly describe the role of the sentinel node in the `List` implementation. How would the code you have written for this problem have changed if we had not given you a sentinel (do not write code to answer this question)?

Simplifies coding because it the list is index starting at 1. It also eliminates the need for a special case for inserting at the front of the structure.

With insertAt, the Kth needs k-1 instead of k-2 so the sentinel node eliminates an edge case with an empty list.

RUBRIC:
1 pt for simplifies coding
2 pts for explaining why with the insertAt function

(f) (2 points) Does the `List` class require a destructor? Briefly justify your response.

Yes, because the entire structure is composed of dynamic memory, which needs to be cleared.

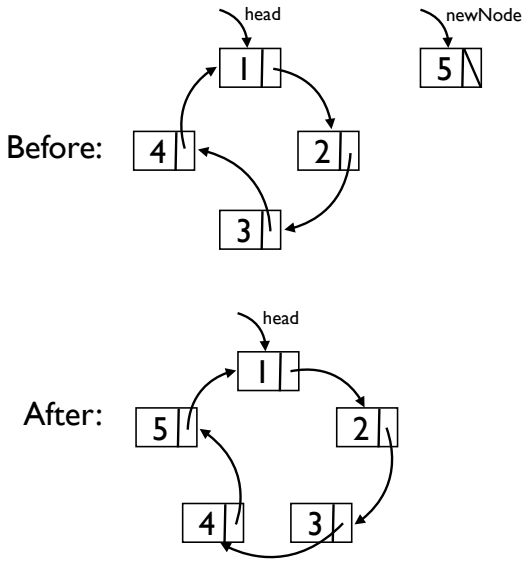
4. **[Linked Lists – 20 points].**

In each of the problem segments below, we have given you "before and after" models of linked lists. Your task is to transform the "before" into the "after" using simple pointer manipulations on the list nodes. Refer to the elements of the list nodes using the `listNode` class below. Your solutions should follow these guidelines:

- You may declare `listNode` pointer variables to use in navigating the lists. When you are finished with them, just set them to `NULL`. (Try to minimize the number of extra variables you use.)
- You must never refer to the `data` member of the `listNode` class, and you must never allocate new `listNode`s.
- You may write loops to simplify your solutions, but your answers don't need to be general... they just need to work on the given lists. (Don't worry about even/odd length, or empty lists, for example.)
- Any pointer variables named in the picture can be used in your solution.
- Do not dereference a pointer more than once. For example the expression `head->next->next` is not allowed.
- Only part (d) uses the `prev` pointer. Just ignore it for parts (a)-(c).

```
struct listNode {
  string data;
  listNode * next;
  listNode * prev;  //NULL except in part (d)
  listNode(string e): data(e), next(NULL), prev(NULL) {}
};
```

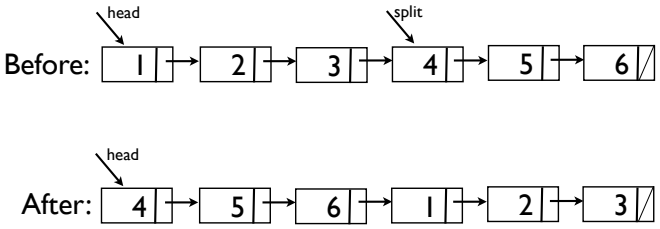
(a) (5 points)



```
listNode *temp = head;
while(temp->next != head)
       temp = temp->next;
temp->next = newNode;
newNode->next = head;
temp= NULL;
```

RUBRIC:
Next node of newNode not set properly -1
Correct solution by dereferencing more than once : 2
Extra variable not set to NULL -1
Head not set properly after insert -1
Some meaningful attempt 1/2

(b) (5 points)

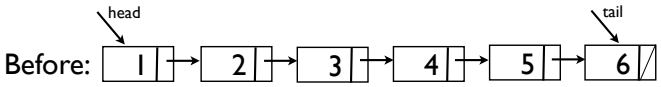


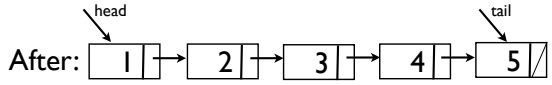

```
listNode *temp=head;
while(head ->next != split)
head = head->next;
head->next = NULL;
head = split;

while(split->next != NULL)
split = split->next;
split->next = temp;
```

RUBRIC:
Next of last node not set to NULL -1
Correct solution by dereferencing more than once : 2
Extra variable not set to NULL -1
Head not set properly after insert -1
Some meaningful attempt 1/2

(c) (5 points)

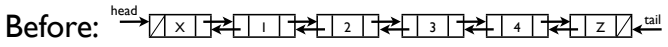


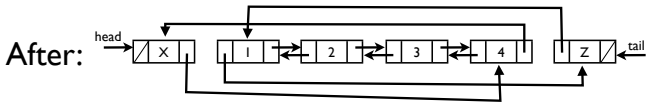



```
listNode * temp = head;
while (temp->next != tail) temp = temp->next;
temp->next = NULL;
delete tail ;
tail = temp
temp = NULL;
```

RUBRIC:
Tail not set properly -1
Correct solution by dereferencing more than once : 2
Extra variable not set to NULL -1
Next of last node not set to NULL -1
Some meaningful attempt 1/2

(d) (5 points)






```
listNode *temp1 = tail->prev;
listNode *temp2 = head->next;

head->next = temp1;
temp1->next = head;

tail->prev = temp2;
temp2->prev = tail;

temp1 = NULL;
temp2 = NULL;
```

RUBRIC:
For each of the 4 changed pointers set incorrectly -1
Correct solution by dereferencing more than once : 2
Extra variable not set to NULL -1

Some meaningful attempt 1

5. **[Miscellaneous – 5 points].**

(a) (6 points) For this problem, you will be a code critic. Please answer the questions below about the following `sphere` class member function. `setPictureGetRadius` is supposed to change the member variable `thePicture` and return the current value of member variable `theRadius`. The adapted `sphere` class appears at the end of the exam.

```
double  sphere::setPictureGetRadius(PNG & newPicture) const {

   thePicture = newPicture;
   return theRadius;

}
```

i. (2 points) Comment on the type specification in the parameter list. Is there a better way to specify that parameter? Why is it pass-by-reference?
SOLUTION:
The type of the parameter should be const since we have no reason to modify it. (1 point)
We pass by reference so that we avoid doing a possibly expensive copy to the parameter. (1 point)

ii. (2 points) Comment on the `const` keyword in the function prototype.
SOLUTION:
The const keyword means we aren't allowed to modify the classes member vars. (1 point)
The method itself is not const (it modifies thePicture), so the result is a compiler error. (1 point)

iii. (1 points) What does the assignment `thePicture = newPicture;` assume about the `PNG` class? Is this a reasonable assumption?
SOLUTION:
It reasonably assumes the `PNG` class as an overloaded assignment operator.

(b) (0 points) Please give us feedback about the course, entering your responses on items 15 thru 17 of the scantron form you used for your multiple choice responses:

i. On a scale of 1 to 5, how much are you learning in the class? (1 is not much, 5 is a ton)

(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

ii. On a scale of 1 to 5, how is the pace of the course so far? (1 is too slow, 5 is too fast)

(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

iii. On a scale of 1 to 5, rate your general satisfaction with the course. (1 is profoundly dissatisfied, 5 is happy) If your response is not 4 or 5, please suggest a specific improvement we can make.

(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

```
class sphere {
public:
   sphere();
   sphere(double r);

   double getDiameter() const;
   double getRadius() const;
   void setRadius(double r);
   double setPictureGetRadius(PNG & newPicture) const;
private:
   double theRadius;
   PNG thePicture;
};
```

scratch paper