1	2	3	4	5	6	7

Name: _____

Student ID #: _____

Section #: _____

CMSC 114 Final Exam Spring 2003

This is a closed book, closed notes, 120 minutes exam. No calculators or other aids are allowed. If you have a question during the exam, please raise your hand. Each question's value is written next to its number. Make sure there are *seven* problems in the exam. First answer those questions that you find are easy! YOU WILL LOSE POINTS IF YOU DO NOT WRITE YOUR NAME, STUDENT ID NUMBER, AND CORRECT SECTION NUMBER AT THE TOP OF THE EXAM (IN THE DESIGNATED AREA).

1. [25 points]

   1. Write the Unix command we use to create a directory.

   2. Write the Unix command necessary to remove a file called data .

   3. Write the Unix command that will remove all the files in the
      current directory ending with the extension .bak .

   4. Can we have more than one destructor for a class ? YES/NO
      No explanation necessary.

   5. Mention three sorting algorithms (besides quicksort) that
      were discussed in class or lab.

   6. Which of the following arrays can a binary search be used upon?

      i. `int a[] = { 10, 20, 19, 45 };`

      ii. `float b[] = { -7.5, 0.0, 2.0, 7.5 };`

      iii. `int c[] = { 1, 9, 10 };`

      Write down your choices here:

   7. Which of the following arrays could have been generated by a
      quicksort partition function assuming the pivot value is 15?

      i. `{ -20, 2, 15, 40, 19 }`

      ii. `{ 4, 7, 8, 15, 18 }`

      iii. `{ 7, 5, 3, 15, 21 }`

      Write down your choices here:

   8. Which Unix command allows us to determine the amount of disk
      quota a user has available?

   9. Write the Unix command used to compile a program called p1.cc
      so that the executable name is called p1.exe ?

   10. Which command (used with your projects) allows us to put a
       set of files into one file?

   11. Which command (used with your projects) allows us to compare two
       files?

12. Is it safe to delete a NULL pointer? For example:

```
char *p = NULL;
delete p;
```

Answer YES or NO. No explanation necessary.

13. What is the order (running time) for the binary search algorithm?

i. O(log n)
ii. O(n)
iii. O(n log n)
iv. O(n * n)

Write your choice here:

14. Can all the operators in C++ be overloaded?

Answer YES or NO. No explanation necessary.

15. Can we overload the new operator?

Answer YES or NO. No explanation necessary.

16. Can we change the associativity of an operator by using operator overloading?

Answer YES or NO. No explanation necessary.

17. Write the Unix command we use to list the files in a directory.

18. Write the Unix command we use to copy files.

19. What command, besides telnet, can you use to connect to a remote Unix machine?

20. Write the compilation command that will generate the .o file associated with the C++ file Train.cc .

21. Write the Unix command we use to change directories.

2. [10 points]

OVERVIEW

Each line of a file called data has two numeric values. You will write a C++ program that will compute the average of those two values by reading the data from a file (NOT using Unix redirection) and sending the results to the standard output. If a line begins with the character 'E' (followed by the two numeric values) then no average computation will be done for that particular line and the line will be ignored.

REQUIREMENTS/ASSUMPTIONS/RESTRICTIONS

a. You do not have to check whether the opening of the data file is successful or not.

b. Remember that your program must handle a file with a lot of entries (not just the entries provided in the example at the end).

c. You may assume there will not be leading or trailing spaces in any line nor will there be any blank lines.

d. Use string objects from the C++ string class (<string>) in order to manipulate strings.

e. The only iteration statement you can use is while loops (no for loops, no do whiles).

EXAMPLE RUN

To the right is an example of running the program you will write. The % symbol represents the Unix prompt. Remember, this is just an example and your program must work with a data file having different entries.

```
% cat data
10 20
E 40 50
5 6
% a.out
Avg: 15
Avg: 5.5
%
```

3. [15 points] The following structure will be used for the questions in this problem:

```
struct Node
{
   int data;
   Node *next;
};
```

   a. [10 points] Write a NON-RECURSIVE function called selectEvenNodes that creates a linked-list with copies of the even nodes of another linked-list. (When we say "even nodes", we mean node #2, node #4, node #6, etc.) The prototype of the function is:

```
void selectEvenNodes(Node *head, Node *&result);
```

   For example, if the list represented by head has the elements

```
9  17  57  34  22  5  8
```

   then the function will generate a list with the following elements:

```
17  34  5
```

   The parameter result represents the list generated by the function.

   The following assumptions/restrictions apply for the implementation of this function:

   i. When the function is called the value associated with the result parameter is NULL.
   ii. You can assume the list represented by head has at least two or more nodes.
   iii. The first node of the list is considered to be an odd node.
   iv. You MAY NOT modify the list represented by head.
   v. YOU WILL GET 0 CREDIT FOR THIS FUNCTION if you use recursion, auxiliary functions, traverse the list more than once or use more than one iteration statement.

b. [5 points] Write a RECURSIVE function called find which has the following prototype:

```
bool find(Node *head, int x);
```

The function will return true if there is a node in the linked-list with a data value x false otherwise. Your function must not modify the original list. You will get 0 credit if your solution is not recursive, uses auxiliary functions, any iteration statements or static variables.

4. [15 points] Write the output generated by the following program. You can assume that the compiler has not generated any optimizations for this code.

```
#include <iostream>
#include <string>

using namespace std;

class Book {
   public:
      Book(string titleIn = ``NONE'');
      Book(const Book &b);
      void print();
      ~Book();

   private:
      string title;
};

Book::Book(string titleIn) {
   title = titleIn;
   cout << ``BC'' << endl;
}

Book::Book(const Book &b) {
   title = b.title;
   cout << ``BCOPYC'' << endl;
}

Book::~Book() {
   cout << ``BDES'' << endl;
}

void Book::print() {
   cout << title << endl;
}

class Shelf {
   public:
      Shelf();
      ~Shelf();
      void print();
      void updateBook(Book &b, int n);

   private:
      Book b1;
      Book b2;
};

Shelf::Shelf() {
   cout << ``ShelfC'' << endl;
}

Shelf::~Shelf() {
   cout << ``ShelfDES'' << endl;
}

void Shelf::print() {
   b1.print();
   b2.print();
}

void Shelf::updateBook(Book &b, int n) {
   if (n == 1)
      b1 = b;
   else
      b2 = b;
}

int main()
{
   cout << ``****First'' << endl;
   Shelf *s;
   s = new Shelf;
   s->print();

   Shelf s2(*s);
   delete s;
   cout << ``****Second'' << endl;

   return 0;
}
```

---

5. [35 points] For this problem you will implement a class called Section. The class represents a section of a college course. Below we describe the data members, the methods associated with the class and what you must implement. At the end there is a driver illustrating the use of the class; you may want to take a look at it as you read the description of the methods.

   a. **Supporting class**

   The class called Person is used to implement the class Section. **You do not have to implement anything related to the Person class. You may not modify the Person class in any way.**

```
class Person {
   public:
      Person(string nameIn);
      string getName();
      void print(ostream &);

   private:
      string name;
};

Person::Person(string nameIn) {
   name = nameIn;
}

string Person::getName() {
   return name;
}

void Person::print(ostream &out) {
   out << name << endl;
}
```

   b. **Data members of the Section class**

   The two private data members of the Section class are array and size.

      i. **array** - represents a pointer to a dynamically-allocated array of Person object pointers. That means that this data member will be declared as follows:

```
Person **array;       // NOTICE THERE ARE TWO ASTERISKS
```

      The following is a visual representation of the array, assuming the size of the array is three:

```
-------------
| o | o | o |
--|---|---|--
  v   v   v
  P1  P2  P3     Each Pi represents a dynamically-allocated
                 Person object
```

      ii. **size** - integer value representing the size of the array.

   IMPORTANT: If you declare the array data member as a pointer to an array of Person objects as in:

```
~~Person *array;~~       // WRONG DECLARATION, USES ONLY ONE ASTERISK
```

   then you will get **ZERO** points for this problem (YOU WILL LOSE ALL THE POINTS). You must implement your code with the clear understanding that the array is an array of pointers to Person objects, not an array of Person objects.

c. **Methods (public)**

i. **Constructor** - initializes an object so that it has no elements. You must initialize both the array and size data members.

ii. **Destructor** - returns any allocated resources.

iii. **Copy Constructor** - Defines the appropriate copy constructor for the class.

iv. **add** - This method has one parameter (a Person object) which is passed by value. The method adds the Person object to the end of the array by:

A. Growing the array of pointers by one.

B. Making a copy of the Person object parameter and inserting a pointer to the copy at the end of the array.

The method has void as returned type.

d. **Operators to Overload**

i. **operator<<** - You must overload this operator so that output operations can be generated. If the array has no elements the message "Section empty" will be generated. Otherwise, each of the persons in the section will be printed. See the Sample Output included at the end for the formatting to follow. You must overload the operator so that we can do cascading.

ii. **operator==** - You must overload this operator so that we can compare two Section objects. Two section objects will be considered equal if they have the same size and if the names of the Persons in both sections appear in the same order and are the same. See the Sample Output for clarification about the meaning of equality. YOU MUST OVERLOAD THIS OPERATOR AS A MEMBER FUNCTION OTHERWISE YOU WILL GET 0 CREDIT.

e. **HINT**

If you need to make a copy of a Person object use the Person copy constructor.

f. **WHAT YOU MUST IMPLEMENT**

i. **Class declaration**
Write the class declaration (what usually goes in the .h file) for the Section class described above. The declaration must include any necessary declarations associated with operators that you need to overload. If you forget the class declaration, you will lose a significant number of points. You do not have to include #ifndef / #define directives as part of the declaration.

ii. **Implementation of the Methods**
Implement the public methods described above. Write them as they will appear in a .cc file.

iii. **Overloading the operators**
Provide the overloading functions associated with the operators described above. Write the functions as they will appear in a .cc file.

iv. **Restrictions/Assumptions**
i. You cannot add any methods (private or public) besides the ones we have specified.
ii. You cannot add any data members (private or public) besides the ones we have specified.

[SAMPLE DRIVER AND OUTPUT ARE ON THE NEXT PAGE.]

g. **Example Driver (and output)**

```
int main()
{
   Section s1;
   Person p1(``John'');
   Person p2(``Rose'');
   Person p3(``Mary'');

   cout << ``***First'' << endl;
   cout << s1;

   cout << ``***Second'' << endl;
   s1.add(p1);
   s1.add(p2);
   cout << s1;

   cout << ``***Third'' << endl;
   Section s2(s1);
   s2.add(p3);
   cout << s2;

   if (s2 == s1)
      cout << ``equal'' << endl;
   else
      cout << ``not equal'' << endl;

   Section s3;
   s3.add(p2);
   s3.add(p1);
   if (s3 == s1)
      cout << ``equal'' << endl;
   else
      cout << ``not equal'' << endl;

   return 0;
}
```

OUTPUT

```
***First
Section empty
***Second
John
Rose
***Third
John
Rose
Mary
not equal
not equal
```