CSCI 202 Computer Science II   Fall 2002

FINAL EXAMINATION Version 1A (200 points max)

Print your name and Student Id. number on the back of the last sheet. **Closed book, open sheet of notes. Calculator OK. No wireless communication or computers. Write answers on this paper. Attempt every question** (9 Questions, 1 per page).

1. (22 points)

a. (12 points) The following C++ program compiles and runs. I compile it into an executable program called *puzzle*. Write what it outputs **in the box provided** when I run it with this command:

```
puzzle file.in   file.out
```

```
#include <iostream>
#include <string>
int main(int argc, char* argv[])
{    cout << argc << endl;
     int i;
     for ( i = 0;   i < argc ;   i=i+2 )
           cout << argv[ i ] <<endl;
     return 0;
}
```

Put output below:

b.(10 points). Use the Unified Modeling Language (**UML**) **to draw a diagram** of the classes and relationships in **either** your last laboratory **or** your last programming project, but **not** both. You can omit operations and attributes to make the diagram fit below, Study the following incomplete class. *Mystery* hides a Standard C++ Library vector of doubles so that it can only be manipulated in particular ways. The name is chosen to be meaningless. **This class will be used in several questions this exam.**

2(22 points)

a. (12 points). Fill in each blank _____ with **one** reserved word, number, identifier, or symbol (1 point per blank).

```
#include <string>
#include <fstream>

#_______________<__________>

_____________ Mystery
{
____________: ___________< double> v; //hidden vector of double items

____________:

        ___________(string filename);//constructor reads file

        ____________( );// constructor that asks user for values

        void print() _______;//accessor, prints to cout
        void print(string filename) const; // accessor, prints to file

        __________ get_item( int i)__________ { return v[ i ];} //accessor

        __________ size() const {return v . size(); }//accessor
        void swap(int i, int j);//mutator, swaps items numbered i and j in v.
        void sort();//mutator, sorts vector into increasing order
        int min( int first, int end) const;
        // accessor, pre: 0 <= first < end <=size()
        // returns the index i with the minimum v[i] where first <=i < end.
}; //Mystery
```

b. (10 points) Draw a diagram below of the above *Mystery* class, using the **UML**, showing: attributes, operations, data types, and visibility(+,#,-), but **no** stereotypes:

3.(22 points) More Mystery.... Fill in the blanks (2 points each).

a.(6 points) The default constructor *Mystery* reads a series of doubles input by the user.

```
Mystery::Mystery()
{
       ______________ << "item = ? ";// prompt user for next item
       double next;
       while( cin >> next) // read or stop at end
       {
            v . _____________( next );//put it at end of vector

            __________________ << "item = ? ";// prompt
       }

}
```

b.(6 points) The other constructor for *Mystery* is given the name of a file. It opens the file for input, reads numbers from it until the stream fails, and then closes it.

```
#include <fstream>
Mystery::Mystery(string filename)
{
       _______________  input;// Declare variable

       input . ____________________________ //Open file for input

       double next;

       while (input >> next)
       {
             v. push_back ( next );
       }
       input . _______________________// Close file
}
```

c. (10 points) The Mystery::*print*() accessor with no arguments, accesses all the items in vector '*v*' in turn. It sends each one to the standard output *cout*, with one item per line.

```
_______________ Mystery::print( )_______________

{      for ( int i=0;  i < v . ________________() ; i++)

       {

            cout << ____________ << ________________;

       }//end for

}
```

4(24 points) More Mystery...

a.(12 points) The other *Mystery*::*print* member function is given the name of a file. It opens the file for output, and puts the items in the vector *v* into that file with one item per line, and then closes the file. Complete the code below.

```
void Mystery::print(string filename) const
{








}
```

b.(12 points max) Here are four algorithms that might be used to implement *Mystery* functions:

```
Algorithm A.
If there is only one item in the collection, exit.
Else:
  Divide it into two nearly equal halves.
  Apply Algorithm A to each half in turn.
  Merge the halves taking smallest items first.
```

```
Algorithm B.
  For each item v[i] except the last one,
    find the smallest item in the following items and
      if it is not the current item then
           swap current and smallest items.
```

```
Algorithm C.
  Try each v[i] in v in turn
         if v[i] == val, return i.

  If no v[i] matched then return -1.
```

```
Algorithm D.
  If the range in the vector is empty return -1 else
 if it has only one item return it's number,
  else divide it into two nearly equal halves, and
     apply Algorithm D to the half that contains
           the item.
```

(Put letters below. A wrong letter can score -1 point, but the right A, B, C, D can score 2 points)

Which of A, B, C, D are "sort" algorithms ?______________

Which of A, B, C, D are "search" algorithms? ______________

Which of A, B, C, D are O(n)? ______________

Which of A, B, C, D are O(log(n))? ______________

Which algorithm would you choose for *Mystery*::*sort*( )? ______________

Why? (short sentence, 2 points)____

____

5. (24points) Yet More on Mystery...
a. (6 points) Correct the errors in the *Mystery*::*swap(...)* implementation below. ***Hint****: it compiles, runs, but does not swap v[i] with v[j].*

```
void Mystery::swap( int i, int j)//mutator, swaps items numbered i and j in v.

{

              v [ i ] = v [ j ];


              v [ j ] = v [ i ];

}
```

b. (9 points) Complete the code below:

```
int Mystery::min( int first, int end) const
              // accessor, pre: 0 <= first < end <=size()
              // returns the i with minimum v [i] where first <=i < end.
```

c.(9 points) Write down the implementation for the member function *Mystery::sort* () using the **sorting algorithm** from question **4b** that can call *Mystery::swap(...), Mystery::min(...), and Mystery::size()* to save work. (reusing swap, min, & size 1 point each, choosing correct algorithm 2 points, correctness 2 points, style 2 points)

```
void Mystery::sort() //mutator, sorts vector v into increasing order
```

6. (20 points max, blank answer = wrong=0 points, correct=2 point, near miss=1 point) STL.

a. (14 points) For each situation below, **write down** the single **answer** that fits the situation or description **best**:

Answers: array, deque, list, queue, stack, vector

A card game where cards are placed on top of other cards and only the top card can be removed.____________

Any existing item can be accessed by number. Items can be added and deleted at only *one* end _____________

Items can be accessed, inserted, and deleted at one end only._____________

Items are best accessed in sequence and can be inserted and deleted at any place.____________

A bank stores transactions as they happen but processes them at night in the order that they came in.____________

An ordered collection where items are inserted at one end and can only be deleted at the *other* end._____________

Holding operands and operations while evaluating an arithmetic expression._____________

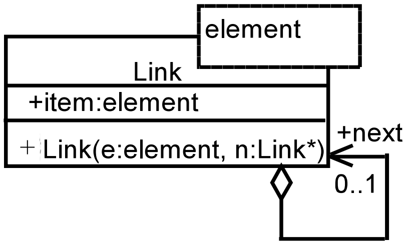
b.(6 points) Fill in the blanks in the following program so that it inputs some words and then outputs them *in the reversed order*. For example if I input "`Goodbye, Cruel world!`" it outputs

```
world!
Cruel
Goodbye,
```

Read the whole program and think before writing your answers. Note. This was a working program before I blanked out three things below. What were they?

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

#include <________>

int main( int argc, char* argv[] )
{  string word;

   __________ <string> words;

   while ( cin >> word ){

       words . push (word);

   }//While cin

   while ( ! words . empty() ){

       cout << words . ___________() << endl;
       words . pop ();

   }//while words not empty

   return 0;

}//main
```

7. (24 points). Here is the design for a parameterised class called a *Link*. It is used for implementing stacks, queues and other data structures.



Each *Link* contains an *item* element and a pointer to the *next Link* in a list. A *Link* is constructed by copying an element *e* into the *item* and copying a pointer to a Link into the *next* item in the *Link.*

a. (12points, 2 points max per blank, no answer=0) **Fill in blanks ______ in the code** below.

```
______________ < ______________ element>
class Link
{
   ______________:

        ____________ item;

        Link __ next;

        __________(element e, Link* n) {item = e; next = n; }
};//Link
```

b.(8 points). **Study** the main program below that uses the class above and **work out a diagram** of the computer's memory showing every pointer and object created. Invent addresses as needed. Erase nothing, cross out old values. No points for tidiness. You may add arrows if you wish.

```
#include <iostream>
int main()
{
 Link<int>*p1 = new Link<int>(1, NULL);
 Link<int>*p2 = new Link<int>(2, p1);
 Link<int>*p3 = new Link<int>(3, p2);
 Link<int>*p4 = p3;
 p3 = p2;
 p2 = p4;
 cout << ((*p2).item) <<endl;
 cout <<  p2 -> next -> item <<endl;
}
```

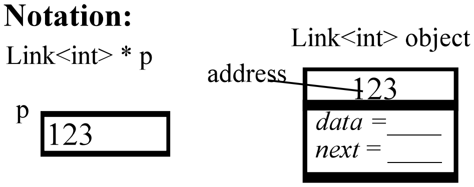
c.(4 points) **Write down** what appears on the screen when the above program is run:

____

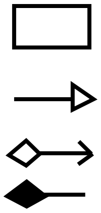
____

Draw answer to part b below.

Notation:



8. (24 points)
8a. (12 points). Draw a class diagram using the **UML** class generalization, aggregation, and composition symbols indicated on the right. Do not use other kinds of links/associations. Show no attributes or operations. Use composition and aggregation with roles and multiplicities instead.



Show the classes V, X, and Y and **all** the following relationships:

Y is a special kind of *X*.
Y has public pointer to a *V* that is called *vp*.
X contains a private vector *V*'s called *vs*.

8b. (12 points). Write C++ code that defines the **two** classes ***X*** and ***Y* described above**. Assume that *V* has already been defined. Assume that there are no operations/member functions.

9.(18 points) inheritance and polymorphism

a. (6 points) True or False? Circle the correct T|F choice (blank=wrong=0, correct T/F 1point)

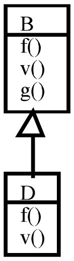
Assume that we declare a class of Students as follows: `class Student : public Person { ... };`

This means that:

Person is *derived* from Student. . . .. [T | F]
The *switch* operation converts a non-Student Person into a Student.. . . .. [T | F]
The Person's *constructors* are automatically inheritted by Student.. . . .. [T | F]
You can have the same function name defined in *both* Person and Student classes.. . . .. [T | F]
Every Student *is* automatically a Person with added properties.. . . .. [T | F]
A Student function is *virtual* if it is declared as *virtual* in Person . . . .. [T | F]

b. (12 points) Work out what is output by the following useless but correct program (1 point for each correct output, 0 if blank, missing, or wrong) **Put Any Working Below:**

```
#include <iostream>
class B{
 public:
          void f(){ cout << "Bf "; }
  virtual void v(){ cout << "Bv "; }
          void g(){ cout << "Bg "; }
};
class D : public B {
 public:
          void f(){ cout << "Df "; }
  virtual void v(){ cout << "Dv "; }
};
int main ()
{ B b; D d; B* p;

  b.f();  b.v();  b.g();  cout << endl;

  d.f();  d.v();  d.g();  cout << endl;

  p = &b;
  p->f(); p->v(); p->g(); cout << endl;

  p = &d;
  p->f(); p->v(); p->g(); cout << endl;
}
```



Put output below: