# CSc 372, Spring 1996
# Final Examination
Tuesday, May 7, 1996

## READ THIS FIRST

**Do not turn this page until you are told to begin.**

This examination consists of 14 problems and an extra credit section presented on 15 numbered pages. When you are told to begin, first check to be sure you have all the pages.

On problems that ask you only to write code, you need not include any explanation in your answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

Please feel free to ask the instructor questions during the course of the exam. If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are especially encouraged to ask the instructor about that construct.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit. If you can offer any sort of pertinent response, it may earn you partial credit.

Except as otherwise noted, you may use any language elements you desire.

Please do not write on the back of any page—ask for additional sheets of paper if you run out of room.

When you have completed the exam, give it to the instructor and enter your name on the exam sign-out log.

Course grades are due at 5pm on Thursday, May 9. Shortly after that, if not before, you will be notified of your final grade via e-mail. Graded exams with solutions will be available in the Computer Science main office by Monday, May 13.

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

Name: ________________________________________

Problem 1 (10 points):

What is the difference between a class and an object?

What is the proper way to view the relationship between function members and data members in a C++ class?

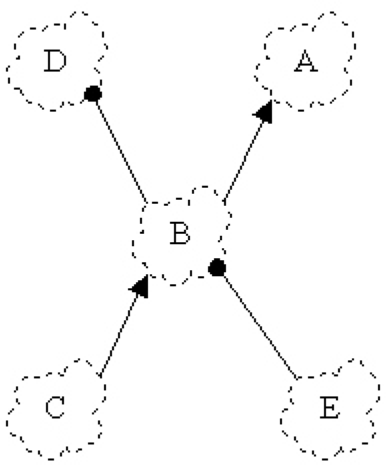
Under what circumstances should a data member in a C++ class be public?

What is the difference between the class relationships of inheritance and containment?

As described by the instructor, what is the primary benefit of inheritance?

Problem 2 (5 points):

Write a set of C++ class declarations that capture the relationships shown in this Booch Notation class diagram:



Here's a start—a complete declaration for `A`:

```
class A { };
```

Problem 3 (20 points):

In this problem you are to fully implement a C++ class named `ReplStr`. The abstraction represented by `ReplStr` is a character string that consists of a "base" string repeated (replicated) a specific number of times. For example, the string "`ababab`" might be represented with `ReplStr("ab", 3)` or `ReplStr("ababab", 1).`

`ReplStr` should have this interface:

```
class ReplStr {
    public:
     ReplStr();
          //
          // Creates a string of length zero

     ReplStr(String s, int n);
          //
          // Creates a string consisting of n
          // concatenations of s.

     int Length();
          //
          // Produces the length of the replicated String.

     String GetString();
          //
          // Produces the replicated string as an
          // instance of the String class.
     };
```

The binary operations of equality (==), inequality (`!=`), and concatenation (+) can be applied to pairs of `ReplStr`s.

Examples of use:

```
ReplStr r1("abc", 2);
int l1 = r1.Length();       // should be 6
String s1 = r1.GetString(); // should be "abcabc"

ReplStr ab1("ab",3), ab2("ababab",1);
ReplStr x6("x",6);

int t1 = ab1 == ab2;    // Should be 1
int t2 = x6 == ab1;     // Should be 0

int t3 = ab1 != ab2;    // Should be 0
int t4 = x6 != ab1;     // Should be 1

ReplStr s2("abc",2), s3("xy",3), s4;
```

```
s4 = s2 + s3;   // s4 should represent "abcabcxyxyxy"
                // s2 and s3 should be unchanged
```

`ReplStr`s can be inserted in `ostream`s. The lines:

```
cout << "ab1 = '" << ab1 << "', ab2 = '" << ab2
     << "', x6 = '" << x6 << "'" << endl;

cout << "s4 = '" << s4 << "'" << endl;
```

should produce the output:

```
ab1 = 'ababab', ab2 = 'ababab', x6 = 'xxxxxx'
s4 = 'abcabcxyxyxy'
```

Tasks in this problem:

(1) Define a set of private data members for `ReplStr`.

(2) Define implementations for the two constructors and the `Length` and `GetString` methods.

(3) Define implementations for the overloaded ==, !=, +, and << operators. You may add additional public and/or private methods to `ReplStr`.

(4) If needed, define implementations for the assignment operator and the copy constructor. If either or both are not needed, explain why not.

(5) Describe a pattern of usage for which your implementation would have poor performance characteristics and explain the difficulty. If you believe your implementation has uniformly good performance characteristics, make an argument to support that claim.

Note that `ReplStr` utilizes the `String` class studied during the course. In addition to the `String` methods studied in class and implemented in assignment 6, you may assume that an insertor for `String` exists and that the operations of == and != are defined for pairs of `String`s.

Problem 3—space for solution

Problem 4 (5 points):

Fully implement classes `Bicycle` and `Tricycle` as follows.

A `Bicycle` or `Tricycle` can only be created given an owner's name represented by a `char *`. Examples:

```
Bicycle b1("Jimmy"), b2("Susan");
Tricycle t3("Mary"), t4("Bobby"), t5("Joey");
```

A `Bicycle` or `Tricycle` can be asked for the name of its owner or the number of wheels it has:

```
char* o1 = b1.GetOwner(); // Should produce "Jimmy"
int w1 = b1.GetNumWheels(); // w1 should be 2
int w2 = t3.GetNumWheels();  // w2 should be 3
```

Write a function `CountWheels` that can be used to count the number of wheels in a collection of bicycles or tricycles. Two possible uses of it:

```
Bicycle *bikes[] = { &b1, &b2, 0 };
Tricycle *trikes[] = ( &t1, &t2, &t3, 0 };
int c1 = CountWheels(bikes); // c1 should be 4
int c2 = CountWheels(trikes); // c2 should be 9
```

Your implementation of `CountWheels` should be able to accommodate possible future classes such as `Unicycle` or `TandemBicycle`, assuming that they also have a `GetNumWheels` method.

If you wish, you may introduce an additional class or classes to simplify your solution.

Note that you must specify all the code necessary to compile and run the examples given.

Problem 5 (5 points):

Write a Prolog predicate `flip(L1,L2)` that if given a list such as `[a,b,c,d,e,f]` as `L1` it will instantiate `L2` to be the list `[b,a,d,c,f,e]`. That is, it flips the position of each element in a list on a pair-wise basis. `flip` should fail if given a list with an odd length.

Examples of usage (blank lines have been elided):

```
| ?- flip([1,2,3,4], L).
L = [2,1,4,3] ? ;
no
| ?- flip([1,2,3], L).
no
| ?- flip([a,b], L).
L = [b,a] ? ;
no
| ?- flip([],L).
L = [] ? ;
no
```

Problem 6 (5 points):

State in your own words the relationship expressed by each of the rules of the `append` predicate:

```
append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

For reference, here are some examples of usage:

```
| ?- append([1,2],[3,4],L).
L = [1,2,3,4] ? ;
no
| ?- append([1,2],L,[1,2,3,4]).
L = [3,4] ? ;
no
```

Problem 7 (5 points):

Write a Prolog predicate `trim(L, SL, NL)` that describes the relation that the list `NL` is the list `L` with the list `SL` removed if `SL` is a prefix or suffix of `L`. Note that if `SL` is a both a prefix and a suffix, two alternatives should be generated.

Examples of usage (blank lines have been elided):

```
| ?- trim([a,b,c,d],[a,b],L).
L = [c,d] ? ;
no
| ?- trim([a,b,c,d],[b,c,d],L).
L = [a] ? ;
no
| ?- trim([a,b,c,d,a],[a],L).
L = [b,c,d,a] ? ;
L = [a,b,c,d] ? ;
no
| ?- trim([a,b,c],[a,b,c],L).
L = [] ? ;
L = [] ? ;
no
| ?- trim([a,b,c],[],L).
L = [a,b,c] ? ;
L = [a,b,c] ? ;
no
| ?- trim([a,a,b,c,a,a],[a],L).
L = [a,b,c,a,a] ? ;
L = [a,a,b,c,a] ? ;
no
| ?- trim([a,a,b,c,a,a],[a,a],L).
L = [b,c,a,a] ? ;
L = [a,a,b,c] ? ;
no
```

Problem 8 (5 points):

**Do EITHER part (8a) or part (8b), but not both. If you work on both, CLEARLY mark which solution you wish to have graded.**

(8a) Write a predicate `maxint/2` that if given a list of integers will find the largest element in the list. `maxint` should fail if given an empty list.

Examples of usage:

```
| ?- maxint([5,10,15],M).
M = 15 ? ;
no
| ?- maxint([10,5,15],M).
M = 15 ? ;
no
| ?- maxint([10],M).
M = 10 ? ;
no
| ?- maxint([],M).
no
```

(8b) Write a predicate `sum_check/2` that determines if a given integer is the sum of a list of integers.

Examples of usage:

```
| ?- sum_check(10, [1,2,3,4]).
yes
| ?- sum_check(10, [1,2,3,4,5]).
no
| ?- sum_check(0, []).
yes
```

Hint: Note that a query such as '`4 is 2+2`' is valid and succeeds.

Problem 9 (5 points):

Write a Prolog predicate `find_missing/3` that can be called with any two arguments instantiated and if the two supplied arguments are equal, the uninstantiated argument is instantiated to the value of the other two.

Examples of usage:

```
| ?- find_missing(X,1,1).
X = 1 ? ;
no
| ?- find_missing(1,2,X).
no
| ?- find_missing(3,X,3).
X = 3 ? ;
no
| ?- find_missing([a],X,[a|[]]).
X = [a] ? ;
no
```

Problem 10 (5 points):

Consider two predicates that express relationships about employees in a factory. The first is `knows/2`:

```
knows(A,B)
```

This expresses the relationship that `A` knows `B`. For example, if Bob knows Mary that might be expressed with this fact:

```
knows(bob,mary).
```

The second predicate is `shift/2`:

```
shift(bob,day).
shift(mary,night).
```

The above two facts indicate that Bob works on the day shift and Mary works on the night shift.

Using the two predicates `knows` and `shift`, do the following:

(1) Write a Prolog query that expresses *Is there is a person who knows both Bob and Mary?*:

(2) Write a Prolog query that expresses *Does Mary know anyone who works on a different shift than she?*:

(3) Define a new clause for `knows/2` that expresses *A person knows everyone who works on the same shift.*

(4) Define a predicate `by_shift` that prints a list of employees by shift. Usage:

```
| ?- by_shift.
Day:
   bob
   mary
   joe
Night:
   bill
   jim
   sam
   susan
```

At your discretion, `by_shift` may either succeed or fail.

Problem 11 (3 points):

Write an ML function `allsame(L)` of type `''a list -> bool` that returns true if all the elements in a list are equal and false otherwise. `allsame([])` should return false.

Examples of usage:

```
- allsame;
val it = fn : ''a list -> bool

- allsame([1]);
val it = true : bool

- allsame([1,2,3]);
val it = false : bool

- allsame([1,1,1,1]);
val it = true : bool

- allsame([1,1,2]);
val it = false : bool

- allsame([]);
val it = false : bool
```

Problem 12 (2 points):

Consider this function definition:

```
fun f(x,g) = x::g(x-1)
```

What types would ML infer for:

`f?`

`g?`

`x?`

Problem 13 (5 points):

Write an Icon program `tac` to read a text file on standard input and print on standard output the lines of the file in reverse order—last line first; first line last.

For the input file:

```
just
a
test
right here
```

The output should be:

```
right here
test
a
just
```

Write your solution to the right of the above example.

Problem 14 (20 points):

Write a letter to a friend that discusses each of the four languages studied in this class. You may assume that your friend has had exposure to languages like C and Pascal. You may also assume that your friend has knowledge of data structures such as linked lists and trees.

For each language be sure to:

(1) Describe in broad terms the type of problems that the language is well-suited for.

(2) Describe at least three distinctive elements of the language and give an example of the use of each. Choose elements that are not closely related to each other—for example, don't simply cite three different data types, or three different control structures.

(3) Show a situation where the language presents a clear advantage over doing the same task in C.

And, mention which of the languages is your favorite, and why.

Problem 14—additional space for answer

Extra Credit Problems

(1 point) Andrew Koenig's paper, *An Anecdote About ML Type Inference*, describes an incident in which ML's type inferencing system produced a suprising result. What was that result?

(2 points) Name five programming languages that originated before 1980.

(2 points) In C, write a recursive version of the library function `strlen(char*)`. You may not use any library functions, or perform any assignment, augmented assignment, or increment/decrement operations. In other words, write `strlen` using a functional style.

(1 point) Who was the person that first described the notion of partially evaluated functions, such as that provided with currying in ML?

(1 point) Name a popular operating system in which Prolog is utilized.

(5 points) Do a great job on question 14.