CSc 372, Spring 1997
Mid-term Examination
Thursday, March 13, 1997

READ THIS FIRST

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

This examination consists of 12 problems presented on 16 numbered 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.

If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

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.

On the C++ problems you may use any language constructs you desire.

On the ML problems you may use only language constructs covered in class. **You may not use the `hd` or `tl` functions. The #N operator to extract elements from a tuple (e.g. #2(t)) may not be used.**

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

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.

This is a seventy-five minute exam with a total of 100 points.

Name: _____

For the following ML-related problems you may assume you have at your disposal all the functions discussed in class, such as `m_to_n`, `reduce`, and `filter`. You may also use any of the built-in functions discussed in class such as `size`, `length`, `map`, etc.

If there is a function you would like to use but you are in doubt as to its suitability, please ask.

You are encouraged to use helper functions (either in `let`s or not) in your solutions.

As mentioned on the cover page, `hd`, `tl`, and the `#n` operator are off-limits.

Problem 1: (1 points each; 4 points total)

State the ***type*** of each of the following expressions, or if the expression is not valid, state why. For example, the type of the expression `3+4` is `int`.

```
([1, 2], [3.0, 4.0])
```

```
(map size) (explode "abcd")
```

```
("x", (2, "y"), ([3, "z"]))
```

```
(length, [size, length o explode])
```

Problem 2: (2 points)

Consider this ML function definition:

```
fun f(x) = [x 1, 2]
```

What type will be deduced for `f`?

Problem 3: (1 point each; 5 points total)

Consider these two ML function definitions:

```
fun f(_) = 1
fun g(a,b,c) =  if a = f(c) then b else [c,a]
```

State the type that will be deduced for each of the following:

`a`:

`b`:

`c`:

The return type of `f`:

The return type of `g`:

Problem 4: (3 points)

Write a function `f` that has the type `int -> int list -> bool`. You may use literals (constants) but you may not use any explicit type specifications such as `x:int`. The behavior of the function is not important—the task is simply to produce the desired type.

Problem 5: (9 points)

Write a function `tossbig(L,N)` of type `string list * int -> string list` that produces a list consisting of the elements in `L` that have a size less than `N`.

Examples:

```
- tossbig;
val it = fn : string list * int -> string list

- val SL = ["just", "a", "test", "right", "now"];
val SL = ["just","a","test","right","now"] : string list

- tossbig(SL, 4);
val it = ["a","now"] : string list

- tossbig(SL, 5);
val it = ["just","a","test","now"] : string list

- tossbig(explode "abc", 2);
val it = ["a","b","c"] : string list

- tossbig(explode "abc", 1);
val it = [] : string list
```

Problem 6: (9 points)

Write a function `samesums` of type `(int * int) list -> bool` that produces true if adding the two elements of each tuple in turn produce the same sum.

Examples:

```
- samesums;
val it = fn : (int * int) list -> bool

- samesums([(0,3)]);
val it = true : bool

- samesums([(0,3),(2,1),(4,~1)]);
val it = true : bool

- samesums([(1,2),(3,0),(4,0),(5,0),(1,2)]);
val it = false : bool

- samesums([(0,0),(1,~1),(2,~2),(3,~3)]);
val it = true : bool
```

Problem 7: (12 points)

Write a function `block(w,h)` of type `int * int -> unit` that prints a block of `x`'s having a width of `w` and height of `h`. You may assume that `w` and `h` are greater than or equal to 1.

Your solution must be NON-RECURSIVE in nature. You may use recursive routines such as `m_to_n` and `map`, but you may not invent any recursive routines yourself.

Examples:

```
- block;
val it = fn : int * int -> unit

- block(5,3);
xxxxx
xxxxx
xxxxx
val it = () : unit

- block(2,2);
xx
xx
val it = () : unit

- block(1,1);
x
val it = () : unit
```

Problem 8: (6 points)

Write a function `c_block` of type `int -> int -> unit` that behaves like `block`, but that allows partial application. You may use `block` in your solution.

Examples:

```
- c_block;
val it = fn : int -> int -> unit

- c_block 5;
val it = fn : int -> unit

- it 3;
xxxxx
xxxxx
xxxxx
val it = () : unit

- c_block 2 2;
xx
xx
val it = () : unit
```

Problem 9: (30 points)

In this problem you are to implement a C++ class that models a television set that has a list of favorite channels, each with a preferred volume. The television set has channels ranging from 2 to 83 and volume levels of 1, 2, 3, ..., 10.

```
class TV {
  public:
    TV();
       //
       // Initializes a TV, setting the channel to 2 and
       // the volume to 5

    void SetVol(int vol);
        //
        // Sets the volume to the level vol.  Attempts to set
        // the volume to less than 0 or more than 10 are ignored.

    void SetChan(int chan);
        //
        // Sets the current channel to chan.  Attempts to set the
        // channel to be less than 2 or more than 83 are ignored.

    void SetFav();
        //
        // Marks the current channel as a favorite channel and
        // records the current volume as the preferred volume
        // for the channel.

    void SetFav(int vol);
        //
        // Marks the current channel as a favorite channel and
        // establishes vol as the preferred volume for the
        // channel.  If the volume is less than 0 or more than
        // 10, SetFav is ignored.  SetFav does not change the
        // the current volume.

        // For both forms of SetFav, if the current channel has
        // already been selected as a favorite, the only effect
        // is to possibly change the preferred volume.  Note that
        // there is no way to indicate that a channel is no
        // longer a favorite.

    void NextFav();
        //
        // Goes to the next higher channel marked as a favorite
        // and sets the preferred volume.  After channel 83,
        // NextFav wraps around to 2 and continues, if necessary.
        //
        // If no favorites are marked, no change is made.

    void Status();
        //
        // Prints current channel and volume.
  };
```

An annotated example of operation follows. The only output produced by the program are the boldface lines beginning "`Channel is ...`".

```
     void main()
     {
          TV tv;

          tv.Status();
Channel is 2, volume is 5
 (Initial conditions)

          tv.SetChan(9);
          tv.SetFav(10);

          tv.Status();
Channel is 9, volume is 5
 (Channel 9 was established as a favorite with preferred volume of 5)

          tv.SetChan(12);
          tv.NextFav();
          tv.Status();
Channel is 9, volume is 10
 (Starting from channel 12, just marked as a favorite, the search for
  the next favorite channel goes through 83 and back around to channel
  9, with a preferred volume of 10.)

          tv.SetChan(13);
          tv.SetVol(3);
          tv.SetFav();
          tv.Status();
Channel is 13, volume is 3

 (At this point, channels 9 and 13 have been marked as favorites, with
     volumes of 10 and 3, respectively.)

          tv.NextFav();
          tv.Status();
Channel is 9, volume is 10

          tv.NextFav();
          tv.Status();
Channel is 13, volume is 3

          tv.SetChan(8);
          tv.SetFav(0);
          tv.SetVol(4);
          tv.SetChan(50);

          tv.NextFav();    tv.Status();
Channel is 8, volume is 0
          tv.NextFav();    tv.Status();
Channel is 9, volume is 10
          tv.NextFav();    tv.Status();
Channel is 13, volume is 3
          tv.NextFav();    tv.Status();
Channel is 8, volume is 0
     }
```

Your task in this problem is to specify the private portion of the `TV` class and implementations for each method. You need not recopy the public portion shown above. Hints: (1) For a central data structure my solution uses an 84 element `int` array with the Nth element holding a preferred volume if channel N is a favorite channel. (2) The purpose of this problem is to determine if you can put together a C++ class. With that purpose in mind, I don't intend to deduct for minor programming problems such as off-by-one errors.

[Additional space for solution for problem 9]

Problem 10: (5 points)

With our `String` class in mind, overload the `%` operator so that an expression such as `s % c` produces an `int` that is the zero-based position of the first occurrence of the character `c` in the `String` `s`. If `s` contains no occurrences of `c`, negative one should be produced. You may choose to implement the operator as a member function or a free-standing function. Recall that the C library function `char *strchr(const char *s, char c)` returns the *address* of the first occurrence of `c` in `s`.

Your solution should simply consist of the function or method definition itself.

Example:

```
String str = "testing";
int p1 = str % 's';
int p2 = str % 'x';

cout << "p1 = " << p1 << ", p2 = " << p2 << endl;
```

Output:

```
p1 = 2, p2 = -1
```

Problem 11: (2 points each; 6 points total)

Briefly answer each of the following questions related to object-oriented programming in general and C++ in particular.

(a) What is the difference between a class and an object?

(b) As you know, the destructor for a class `X` is a method named `~X`. The rationale for this choice of name is that "destruction is the complement of construction". However, the instructor has contended on several occasions that the relationship between constructors and destructors is in fact somewhat asymmetrical. Describe that asymmetry.

(c) What is the key benefit provided by in-line functions in C++?

Problem 12: (9 points)

In this problem you are to implement a class called `Ring` that is a subclass of the `Shape` class described in the slides. A ring can be thought of as a disk with a circular center region removed. In the following diagram, the black region is a `Ring`:

An instance of `Ring` is created by specifying the radius of the inner circle and the radius of the outer circle. A ring having the proportions of the one shown above might be created in this way:

```
Ring r(0.75, 1.0);
```

The radius of the inner circle is specified first and may be assumed to be non-negative and not greater than the radius of the outer circle. The area of a ring is the difference of the areas of the two circles. The perimeter of a ring is the perimeter of the outer circle.

Your first task in this problem is to specify a full class definition and member implementations for `Ring` as a subclass of `Shape`. You are to work with this definition of `Shape`:

```
class Shape {
  public:
    virtual double Area() = 0;
    virtual double Perimeter() = 0;
    virtual void Print(ostream& o) = 0;
    };
```

If your implementation of `Ring` requires changes in `Shape`, show those changes.

Here is an example of `Ring` in operation:

```
void main()
{
    Ring r(.75, 1.0);
    r.Print(cout);

    Ring r1(0.0, 1.0);
    Ring r2(0.0, 2.0);
    Ring r3(1.0, 2.0);

    r1.Print(cout);
    r2.Print(cout);
    r3.Print(cout);
}
```

Output:

```
Ring; area = 1.37445, perim = 6.28319
Ring; area = 3.14159, perim = 6.28319
Ring; area = 12.5664, perim = 12.5664
Ring; area = 9.42478, perim = 12.5664
```

Recall the `SumOfAreas` and `Biggest` functions presented in the slides, which follow below. Your second task in this problem is to indicate what changes that are required to these routines in order for them to accommodate `Rings` in addition to `Circles` and `Rectangles`, which the routines already handle.

```
double SumOfAreas(Shape *shapes[])
{
        double area = 0.0;

        for (int i = 0; shapes[i] != 0; i++) {
                Shape *sp = shapes[i];
                area += sp->Area();
                }

        return area;
}

Shape* Biggest(Shape *shapes[])
{
     Shape *bigp = shapes[0];

     for (int i = 0; shapes[i] != 0; i++) {
          Shape *sp = shapes[i];
          if (sp->Area() > bigp->Area())
               bigp = shapes[i];
          }
     return bigp;
}
```

If you wish, you may use the `Circle` class in your implementation of `Ring`.

[Space for solution for problem 12]