CSc 372, Fall 1996
Mid-Term Examination
Monday, October 21, 1996

READ THIS FIRST

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

This examination consists of nine problems and an extra credit section presented on twelve 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 fifty minute exam with a total of 100 points. You should therefore average two points of completed problems per minute in order to finish in the allotted time.

Name: _____

For the following ML-related problems you may assume you have at your disposal the `m_to_n`, `map`, `reduce`, and `filter` functions as discussed in class. 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: (2 points each; 4 points total)

State the type of each of the two following expressions, or if the expression is not valid, state why.

```
(1,2,(1=2,1+2),"x")
```

```
[[],[1],[1,2]]
```

Problem 2: (2 points each; 8 points total)

Consider this ML function definition:

```
fun f(a,b,c) = (b, a(b), c(a)) = ("x", 2, [1])
```

State the type that will be deduced for each of the following: (2 points each)

`a`:

`b`:

`c`:

The type of the result produced by `f`:

Problem 3: (4 points)

Write a function `f` that has the type `int -> int -> int -> int -> bool`. You may use literals (constants) but you may not use any explicit type specifications such as `x:int`.

Problem 4: (12 points)

Write a function `ints_to_strs(L,c)` that for the `int list L` and the `string c` produces a list of strings where the *i*th element in the resulting list is a string composed of *N* replications of `c` where *N* is the *i*th element in the input list.

Examples:

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

- ints_to_strs([2,1,3], "x");
val it = ["xx", "x", "xxx"] : string list

- ints_to_strs([2,1,3], "ab");
val it = ["abab", "ab", "ababab"] : string list
- ints_to_strs(m_to_n(1,5), "a");
val it = ["a", "aa", "aaa", "aaaa", "aaaaa"] : string list
- ints_to_strs([1,2], "");
val it = ["", ""] : string list
- ints_to_strs([], "x");
val it = [] : string list
```

You may assume that all the integers in the given list are greater than zero.

Problem 5: (12 points)

Write a function `len(L)` of type `string list list -> int` that produces the total number of characters in all the strings in the lists contained in `L`. (12 points)

Examples:

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

- len([["a","b"], ["xx", "yyy"]]);
val it = 7 : int

- len([["1"], ["2"], ["3"], ["4","5","6"]]);
val it = 6 : int

- len([]);
val it = 0 : int

- len([[]]);
val it = 0 : int
```

Problem 6: (10 points)

Write a function `pair(L)` that accepts an `'a list` as an argument and produces a list of tuples containing consecutive pairs of elements from the list `L`. That is, the first tuple in the result list should contain the first and second elements from the list. The second tuple should contain the third and fourth elements from the list, etc.

If the list is of an odd length, e.g., three elements, the exception `OddLength` should be raised.

Examples:

```
- pair;
val it = fn : 'a list -> ('a * 'a) list

- pair([1,2,3,4,5,6]);
val it = [(1, 2), (3, 4), (5, 6)] : (int * int) list

- pair(["a","A","two","too","up","down"]);
val it = [("a", "A"), ("two", "too"), ("up", "down")]
    : (string * string) list

- pair([1,2,3,4,5,6,7]);
Uncaught exception: OddLength

- pair([]);
val it = [] : ('a * 'a) list
```

Problem 7: (26 points)

In this problem you are to implement a class named `Eater`. Instances of `Eater` are initialized with an integer that is the `Eater's` capacity expressed in food units. This creates an `Eater` with a capacity of ten food units:

```
Eater e1(10);
```

An `Eater` can be told to eat some number of food units:

```
e1.Eat(7);
e1.Eat(5);
```

If an `Eater` consumes more than its capacity, it burps. In the above case, the second invocation of `Eat` would produce this output as a side effect:

```
burp!
```

Each burp reduces the volume currently retained in the `Eater` by the capacity of the `Eater`. After the burp, `e1` would then be holding two units.

Continuing the example, a large feeding may produce several burps:

```
e1.Eat(29);
```

Output:

```
burp!
burp!
burp!
```

Note that three burps are produced because the `Eater` had two units remaining after the first burp.

An `Eater` may be inserted into an output stream:

```
cout << e1 << endl;
```

Output: (note that `0x5eb10` is the memory address of the `Eater`)

```
Eater at 0x5eb10 has burped 4 times
```

Another small example:

```
Eater a(1); a.Eat(3);
```

Output:

```
burp!
burp!
burp!
```

A longer example:

```
Eater E(10);
```

```
E.Eat(5);
cout << "..A.." << endl;
E.Eat(12);
cout << "..B.." << endl;
E.Eat(2);
cout << "..C.." << endl;
E.Eat(12);
cout << "..D.." << endl;
cout << E << endl;
```

Output:

```
..A..
burp!
..B..
..C..
burp!
burp!
..D..
Eater at 0x5eaf8 has burped 3 times
```

You may assume the specified capacity of an `Eater` will be greater than zero. You may assume that a given amount to eat will be non-negative.

Problem 8: (10 points)

Create an abstract base class named `Food` and create two derived classes named `Burger` and `Fries`. Any instance of a subclass of `Food` can be queried for the

number of food units it is:

```
Burger b;
Fries f;

cout << "A burger has " << b.GetUnits() << " units" << endl;
cout << "An order of fries has " << f.GetUnits() << " units"
     << endl;
```

Output:

```
A burger has 25 units
An order of fries has 15 units
```

Note: Every `Burger` has 25 units and every `Fries` has 15 units.

Extend `Eater` so that an `Eater` can be told to eat an instance of any subclass of `Food`.

```
Burger b;
Fries f;
Eater e2(20);

cout << "..1.." << endl;
e2.Eat(&b);
cout << "..2.." << endl;
e2.Eat(&f);
cout << "..3.." << endl;
```

Output:

```
..1..
burp!
..2..
burp!
..3..
```

Operation summary for `Eater`, `Food`, `Burger`, and `Fries`—be sure that your implementations support all these operations:

```
Eater e(10);
e.Eat(100);
cout << e << endl;
Burger b;
Fries f;
e.Eat(&b);
e.Eat(&f);
int x = b.GetUnits(), y = f.GetUnits();
```

[Space for solution for problem 8]

Problem 9: (2 points each; 14 points total)

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

What is the difference between a class and an object?

What is the purpose of a constructor?

What is the purpose of a destructor?

If a class has no member functions, how many data members should it have?

Challenge or defend this statement: In C++, one possible reason to have a public data member is to avoid the overhead of calling a function to access that data.

Challenge or defend this statement: The purpose of member functions is to provide well-controlled access to data members.

Describe a benefit provided by using inheritance.

Optional Extra Credit Problems:

What is a practical reason to prefer pattern matching rather than using the `hd` and `tl` functions in SML? (2 points)

Write the minimum amount of code necessary to compile and run this code: (4 points)

```
X x1, xa[10], *xp;
X x2 = x1;
x1 = xa[0];
cout << &*xp << endl;
```

Could C++ be used effectively for functional programming? Present an argument to support your answer—don't just say "yes" or "no". (5 points)

Write in C a version of `int strcmp(char *, char *)` that uses no assignment operators of any form, nor the `++` or `--` operators. Recall that `strcmp` returns 0 if the strings are equal and a non-zero value if the strings are not equal. (3 points)

[Additional work space]