

University of Toronto

Department of Computer Science

CSC 326H1F — Fall 2003

Midterm test

No aids allowed.

Time: 50 minutes

name: _____

student number: _____

tutor: _____

1	
2	
3	
Total	

1. Scheme: 12 marks = 4 x 3

Write Scheme functions to perform the tasks listed below. You may use let (and its variants) but you may not define variables. You may also write helper functions, if you wish.

(a) sumList: given a list of numbers, sum the elements of the list. For example, (sum '(1 2 3)) is 6. You may use any standard predefined Scheme functions you like.

```
(define (sumList ls) (apply + ls))
```

(b) swapFirstTwo: given a list, swap the first two elements of the list. For example, (swapFirstTwo '(a b c d)) is (b a c d). Your function is permitted to fail if there are not enough elements in the list.

```
(define (swapFirstTwo ls) (cons (cadr ls) (cons (car ls) (cddr ls))))
```

1. (continued)

- (c) countNums: Given a parameter that may be an atom or a list of arbitrary structure containing any kind of elements, count the numbers. (That is, count the atoms that are numbers.) For example, this list contains 3 numbers:

```
((1 a) 2 (((3)) b))
```

```
(define (atom? s) (not (pair? s)))
(define (countNums ls)
  (cond ((atom? ls) (if (number? ls) 1 0))
        ((null? ls) 0)
        (else (+ (countNums (car ls)) (countNums (cdr ls))))))
```

- (d) altSigns: Takes one parameter, a list of lists of numbers, like this:

```
((1 2 3) (-1 2 -3) (4 5 6 7))
```

The return value is the sum of the first, third, fifth ... lists, minus the sum of the second, fourth, sixth ... lists. In our example:

$$+ (1 + 2 + 3) - (-1 + 2 + (-3)) + (4 + 5 + 6 + 7) = 30$$

You are permitted to write helper functions, if you wish.

```
(define (altSigns ls) (alt_helper ls 1 0))
(define (alt_helper ls sign sum)
  (cond ((null? ls) sum)
        (else (alt_helper (cdr ls) (- sign)
                          (+ sum (* sign (apply + (car ls))))))))
```

2. Class hierarchies in Java: 12 marks

The following three classes compile without errors.

```
public class Animal {
    private int numFeet;
    public Animal(int nf) {
        numFeet = nf;
    }
    public int getNumFeet() {
        return numFeet;
    }
}

public class Bug extends Animal {
    private double lifetime;
    public Bug(int nf, double lt) {
        super(nf);
        lifetime = lt;
    }
    public double getLifetime() {
        return lifetime;
    }
}

public class Dog extends Animal {
    private int numTeeth;
    public Dog(int nt) {
        super(4);
        numTeeth = nt;
    }
    public int getNumTeeth() {
        return numTeeth;
    }
}
```

The statements on the next page use the classes Animal, Bug, and Dog. These statements might appear in a “main” method. Some of the statements contain errors. The order of the statements matters, but errors in earlier statements don’t invalidate later ones.

Every statement has one of four possibilities:

1. This statement can be compiled without producing any error messages, and runs without error.
2. This statement can be compiled without error messages, but crashes when running.
3. This statement contains compile errors, and can be repaired by either adding or removing a cast.
4. This statement contains compile errors, and cannot be repaired by either adding or removing a cast.

To the right of each line, place a checkmark in the appropriate column. The first three lines are done for you. Every incorrect checkmark is worth –0.5 mark.

2. (continued)

```
Animal a1 = new Animal(3);
```

```
System.out.println(a1.getNumFeet());
```

```
System.out.println(((Dog) a1).getNumTeeth());
```

```
Dog d1 = new Dog(23);
```

```
System.out.println(d1.getNumFeet());
```

```
System.out.println(d1.getNumTeeth());
```

```
System.out.println(((Bug) d1).getLifetime());
```

```
Bug b1 = new Bug(1000, 0.76);
```

```
System.out.println(b1.getLifetime());
```

```
System.out.println(((Animal) b1).getNumFeet());
```

```
Animal a2 = new Dog(73);
```

```
System.out.println(((Dog) a2).getNumFeet());
```

```
System.out.println(a2.getNumTeeth());
```

```
System.out.println(((Bug) a2).getLifetime());
```

```
Dog d2 = new Bug(100, 13.8);
```

1	2	3	4
√			
√			
	√		
X	X	X	X
√			
√			
√			
			√
X	X	X	X
√			
√			
√			
X	X	X	X
√			
√			
		√	
	√		
X	X	X	X
			√

3. Languages: 6 marks = 3 + 3

In this question, we use a language “Lang” with a syntax like C's but with different meanings for statements. Specifically, the language has:

- dynamic binding of variable names to types and values, so that a name refers to the thing most recently assigned to it, whatever type or value that thing may have;
- dynamic scoping, so that the meaning of a variable name is given by the declaration or assignment statement most recently executed, and not by the nearest declaration or assignment in the surrounding program text.

However, parameter-passing works as in C: a copy is made of the parameter, and the called function works on the copy, not on the original parameter.

There are also a couple of syntactic differences between Lang and C:

- If a function called "func" is defined as "int func(void) {...}", so that it takes no parameters, then when func is called the parentheses are omitted: we say just "func" instead of "func()" as in C. This means that you have to be aware that a plain name may ask for the value of a variable or for a call of a function.
- Functions may be defined inside other functions.

Here is a program written in Lang.

```
int f(x)
{
    int g(void)
    {
        return 2*b;
    }

    if (x == 0) {
        a = g;
        return 3*x;
    }
    else if (x == 1) {
        return 4*a;
    }
    else
        return f(x - 1); // You can assume this works the way it does in C.
}

int h()
{
    return 0;
}

main(void)
{
    a = 5;
    b = 7;

    printf("first %d\n", f(a));
    printf("second %d\n", f(h));
    printf("third %d\n", a);
}
```

3. (continued)

(a) What is the output from the program?

```
first 20
second 0
third 14
```

(b) Suppose that we change the definition of Lang so that variable types are static — that is, a variable's type cannot be changed after it is first established.

i) Does the language now require C-style variable declarations and definitions? (Circle your answer.)

yes

no

ii) BRIEFLY (in not more than 25 words) explain your answer to (i).

because if you can still figure out the type from the usage, then you don't need a declaration.