

# Midterm

CSC 4101, Fall 2004

11 October 2004

Read the whole exam first (there are a total of 6 pages) and plan your time. You have 50 minutes to complete all the questions. There are a total of 100 points. The exam is open book, open notes, and closed neighbors. Good luck!

**Name:**

1. (25 pts)

Show the result of evaluating each of the following Scheme expressions:

(a) `(car (car '((a) (b) (c))))`

(b) `(cdr (cdr '((a) (b) (c))))`

(c) `(cons (cons '(a) '(b)) '(c))`

(d) `(car '(car '((a) (b) (c))))`

(e) `(car (cons (car '((a) (b) (c))) '(d)))`

(f) `(cdr (cons '((a) (b)) (cons '(c) '(d))))`

(g) `(car (cdr (car (cdr '((a) (b)) ((c) (d))))))`

(h) `(car (car (cons '(quote ((a) (b))) '((c) (d)))))`

Hint: remember that `(car (cons a d)) = a` and `(cdr (cons a d)) = d`.

2. (25 pts)

The following Scheme functions take lists of integers (no nested lists) as arguments.

```
(define (a l)
  (if (null? l) 0
      (+ 1 (a (cdr l)))))

(define (b l)
  (if (null? l) 0
      (+ (car l) (b (cdr l)))))

(define (c l)
  (if (null? l) 1
      (* (car l) (c (cdr l)))))

(define (d l)
  (if (null? l) 0
      (max (car l) (d (cdr l)))))

(define (e l)
  (if (null? l) 0
      (+ (if (= (car l) 42) 1 0)
         (e (cdr l)))))
```

Describe in English what these functions compute. Use short descriptive phrases, don't simply restate the algorithms in English.

3. (25 pts)

Consider the following grammar for variable and class declarations in Java:

```
<Decl>      -> <VarDecl>
             | <ClassDecl>

<VarDecl>   -> <Modifiers> <Type> <VarDec> SEM

<ClassDecl> -> <Modifiers> CLASS ID LBRACE <DeclList> RBRACE

<DeclList>  -> <Decl>
             | <DeclList> <Decl>

<VarDec>    -> ID
             | ID ASSIGN <Exp>
             | <VarDec> COMMA ID
             | <VarDec> COMMA ID ASSIGN <Exp>
```

Indicate any problems in this grammar that prevent it from being parsed by a recursive-descent parser with one token lookahead. You can simply circle the offending parts of the grammar above.

Transform the rules for <Decl> and <DeclList> so they can be parsed by a recursive-descent parser with one token lookahead (you don't need to transform the rules for <VarDec>). I.e., remove any left-recursion and left-factor the grammar. Make as few changes to the grammar as possible, but do not use Extended BNF. The nonterminals <Decl> and <DeclList> of the modified grammar should describe the same language as the original nonterminals.

4. (25 pts)

Given the following C++ class declarations:

```
class B {
private:
    int x;
public:
    virtual int foo() { return this->bar(); }
    virtual int bar() { return 1; }
};
```

```
class C {
private:
    int x;
public:
    virtual int foo() { return 2; }
};
```

```
class D : public B, public C {
private:
    int y;
public:
    virtual int foo() { return B::foo(); }
};
```

```
class E : public D {
private:
    int z;
public:
    virtual int bar() { return 3; }
};
```

Show the layout of objects of class E and the contents of the virtual function tables (you can ignore the this-offsets). What are the values assigned to variables i, j, k, and l when the following code is executed?

```
B * p = new D();
C * q = new D();
B * r = new E();
C * s = new E();
int i = p->bar();
int j = q->foo();
int k = r->bar();
int l = s->foo();
```

(Additional space for answering question 4.)