

Final Exam

CSC 4101, Fall 2004

7 December 2004

Scan the whole exam first (there are a total of 7 pages) and plan your time. You have 1 hour, 50 minutes to complete all five questions. There are a total of 100 points. The exam is open book, open notes, and closed neighbors. Good Luck and Have a Cool Yule!

Name:

1. (20 pts)

The following grammar

```
<qualname> -> IDENT
            | <qualname> DOT IDENT

<expr>      -> INTCONST
            | <qualname>
            | <expr> DOT IDENT
            | <expr> LBRACK <expr> RBRACK
```

defines a subset of Java expressions. E.g., it allows expressions of the form

```
a.b.c[1].x[2].y.z[3][4][5]
```

(a) (5 pts)

What are the problems (if any) with this grammar that might prevent us writing a recursive descent parser for this grammar.

(b) (5 pts)

Rewrite `<qualname>` so it can be parsed with a recursive descent parser. Do not use EBNF.

(c) (5 pts)

Rewrite `<expr>` so it can be parsed with a recursive descent parser assuming that `<qualname>` is a terminal symbol. Do not use EBNF.

(d) (5 pts)

Is the above grammar ambiguous? If yes, provide an input string that can be parsed in more than one way. If no, explain why not. Is your modified grammar ambiguous?

2. (20 pts)

We have seen that different languages require different run-time data structures for representing activation records of functions. Assuming that the language allows recursive functions (and ignoring the use of registers for optimization purposes), we have the following design choices:

- use a static link or don't use a static link,
- use a dynamic link or don't use a dynamic link,
- allocate activation records on the stack or on the heap,
- represent functions as simple function pointers or as closures.

For the following languages, state which implementation mechanisms you need. Note that when an exception is thrown, we need to find the exception handler along the call chain.

(a) (5 pts)

Pascal: static scoping, nested functions, no functions as return values, no exceptions

(b) (5 pts)

C++: static scoping, no nested functions, functions as return values, exceptions

(c) (5 pts)

Scheme: static scoping, nested functions, functions as return values, no exceptions

(d) (5 pts)

ML: static scoping, nested functions, functions as return values, exceptions

3. (20 pts)

Consider the following program fragment in C-style syntax, but with static scoping:

```
void test() {
    int a[5];
    int i;

    void f(int x) {
        a[i] = 9;
        i++;
        i = x;
    }

    i = 1;
    a[1] = 7;
    a[2] = 4;
    f(a[i]);
    // print i and a[1]
}
```

What are the values of `i` and `a[1]` after function `f` returns if the parameter is passed by

	<u>i</u>	<u>a[1]</u>
value	_____	_____
reference	_____	_____
copy-in-copy-out	_____	_____
need	_____	_____

4. (20 pts)

For each of the following two ML functions, state in English what the function does and translate the function into Scheme. The types of the ML functions are provided in comments for reference.

(a) (10 pts)

The function `foo` takes a function and a list as argument and returns a list.

```
(* val foo = fn : ('a -> 'b) -> 'a list -> 'b list *)
fun foo f nil      = nil
  | foo f (h::t) = f h :: foo f t
```

(b) (10 pts)

The function `bar` takes a binary function, a value, and a list as arguments and returns a value.

```
(* val bar = fn : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b *)
fun bar f b nil      = b
  | bar f b (h::t) = f (h, bar f b t)
```

The operator `::` corresponds to `cons` in Scheme.

Hint: for finding out what the functions do, take `'a` and `'b` to be `int`, pick some simple argument values that fit the types, and trace the functions.

5. (20 pts)

Translate the following Scheme code into a C++ or Java class hierarchy.

```
;; Tree data structure:
;; (leaf integer-value)
;; (node left-subtree right-subtree)
;; E.g.,
;; (node (leaf 1) (node (leaf 2) (leaf 3)))

;; Add all leaf values in the tree
(define (sum t)
  (if (eq? (car t) leaf)
      (cadr t)
      (let ((left (caddr t))
            (right (caddr t)))
          (+ (sum left) (sum right)))))
```

Define the class hierarchy with classes `Tree`, `Leaf`, and `Node`, such that the following code works (in C++ syntax):

```
Tree * left  = new Leaf(1);
Tree * right = new Node(new Leaf(2), new Leaf(3));
Tree * root  = new Node(left, right);
int   h      = root->sum();
```

where `sum()` is a virtual function. Do not use an if-then-else.

(Additional space for answering question 5.)