CS 415 Midterm Exam – Fall 2003

Name _____**KEY**_____

Email Address _____ **Student ID #** _____

Pledge:

This exam is closed note, closed book. Questions will be graded on ***quality*** of answer. Please supply the best answer you can to each question. Good Luck!

	Score
Fortran & Algol 60	
Compilation	
Names, Bindings, Scope	
Functional Programming	
Logic Programming	
Control Flow	
Total	**/72**

Fortran and Algol 60 (11 points)

1. [1 point] Fortran uses pass by __**reference**_____ for parameters.

2. [1 point] Algol was designed by _**a committee of 8, in Zurich in 1958** _____.

3. [5 points] List 3 significant differences between Fortran and Algol 60.

   (Many different answers possible here)

4. [4 points] Algol 60 introduced blocks of statements. List three reasons why these were useful in Algol 60.

   - **programs easier to read (compared to gotos in Fortran)**
   - **supported structured programming**
   - **can use in place of a single statement**
   - **allows nested scopes**
   - **efficient storage management (allocate on entrance to a block and de-allocate on exit)**
   - **allows a fix to the dangling else problem**

Compilation [15 points]

1. [5 points] List the phases a typical compiler uses in converting a program into assembly language. Indicate what is passed from one phase to the next. A full credit answer would have at least 5 phases.

character stream -> Scanner (Lexical Analysis) -> token stream

-> Parser (Syntax Analysis) -> parse tree (concrete syntax tree)

-> Semantic Analysis & intermediate code generation -> abstract syntax tree or other intermediate form

-> Machine-Independent Code Improvement (optional) -> modified intermediate form

-> Target Code Generation -> assembly code or other target language

-> Machine-Specific Code Improvement (optional) -> modified target language

2. [3 points] Why might a program written in a compiled language execute faster than one written in a language that is interpreted?

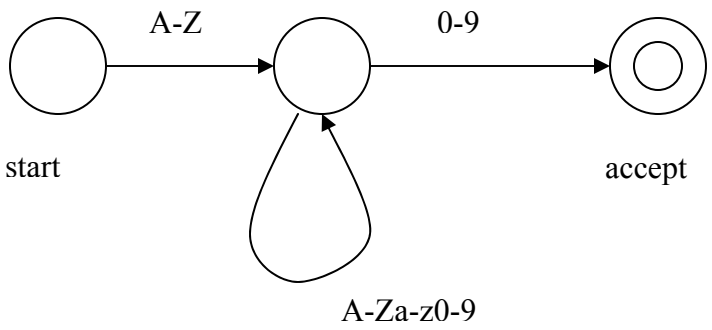
Interpreted – must read and execute each statement as it is running, incurs overhead of analysis at run-time.

Compiled – compiler does analysis at compile-time, so this overhead is not incurred at runtime. In addition, a compiler can afford to take more time to do optimizations and may have access to more information at compile time than can typically be done in an interpreter.

3. [2 points] Write a regular expression for an identifier in the cs415 language. Identifiers can consists of only letters (upper and lowercase) and digits, and must start with a capital letter and end with a digit.

[A-Z] [A-Za-z0-9]* [0-9]

4. [2 points] Draw a finite automaton for the regular expression you gave in question 3 above. Be sure to properly denote the accepting states.



5. [3 points] Describe the dangling else problem.

In the example below, it can be unclear which if stmt the else clause matches with. This can be solved in a number of ways, either by having an else should match the closest unmatched then, or by requiring explicit end markers to show the end of an IF statement.

```
if cond then
    if cond then
            stuff
 else
    other stuff
```

Names, Scopes, and Bindings (10 points)

1. [4 points] What does the function **`test`** print if the language uses static scoping? What does it print with dynamic scoping? (otherwise assume C++ syntax and semantics).

```
int  n = 100;  // global

print_plus_n(int x) {
     cout << x + n;
}
increment_n() {
     n = n + 1;
}

test() {
     int n;
     n = 1;
     print_plus_n(25);

     n = 33;
     print_plus_n(n);

     increment_n();
     cout << n;

     print_plus_n(n);
}
```

With Static Scoping:

125 133 33 134

With Dynamic Scoping:

26 66 34 68

2. [4 points] Support the claim: "Early binding generally leads to greater efficiency." First, define binding, and then give evidence why this might be true.

binding – an association between two things, such as a name and the thing it names (examples: a variable bound to a location in memory, a function name bound to a specific piece of code)

Reasons why early binding can lead to greater efficiency:
- compilation generally binds things earlier than interpretation. This can allow the compiler to do more optimization at compile time, or to avoid some indirection at run-time.
- binding variables to memory locations early can avoid indirection at run-time (we can use a particular address rather than an indirect access).
- binding variables to types ahead of time can eliminate some run-time type checking or can bind function addresses to call sites (rather than needing to consult a vtable at run-time to determine which function to call as in dynamic dispatch of functions).

3. [2 points] What does reference counting refer to? Give one problem with this approach.

As discussed in class, this is an approach to garbage collection that keeps track of the number of references to an object. When the count goes to zero, then the object can be reclaimed.

One problem with this approach is that it does not work on circularly linked structures (like a circularly-linked list) as each node will always be pointed to even if the structure is unreachable from the program.

Functional programming (14 points)

1. [5 points] At several schools (MIT, Berkley, Rice) Scheme is the programming language used in introductory programming courses. Give at least 3 reasons why Scheme could be argued to be well-suited for this purpose?

   A few reasons:
   - garbage collection (don't need to worry about memory management)
   - simple syntax makes it easy to learn
   - if used in a purely functional style, no side effects will be used, this tends to avoid bugs related to un-expected side effects and makes it easier to reason about programs
   - good environments/interpreters are available free of charge (Dr. Scheme)
   - benefits of using an interpreter – fast response (don't have to wait to recompile the whole program each time you change something, makes it easy to experiment), good error messages
   - closely related to math functions – draws on students' math reasoning abilities.

2. [2 points] What is the result of the following in Scheme. (Be specific)

```
(define y (lambda (x y) (+ x y)))
```

   y is bound to a function that takes two parameters and returns their sum

3. [2 points] Assuming that the following definitions are executed in this order:

```
(define b '(3 14 27))
(define c (cons (car (cdr b)) (list 'a 'b 'c))))
```

   What is the result of typing the following into the Scheme interpreter:

```
c                        =>   (14 a b c)

(car (cdr (cdr c)))  =>   b
```

4. [5 points] Write a recursive Scheme function, **`double_each`** that takes a single parameter that is a list and returns a list that contains all of the elements of that list doubled. You can assume that you will always be passed a list, and one that contains only integer values.

   Example: **`(double_each '(4 5 6)) => (8 10 12)`**

```
(define (double-each x)
  (if (null? x) ()
      (cons (* 2 (car x)) (double-each (cdr x)))))
```

Prolog and Logic Programming (12 points)

1. [2 points] What is the scope of a variable in Prolog?

   The clause in which it appears.

2. [2 points] Define "backward chaining".

   Start with the goal and work backward, attempting to un-resolve it into a set of pre-existing clauses. (In contrast to forward chaining, that would start with the set of existing clauses in the database and work forward from that, attempting to derive the goal.)

3. [5 points] Write a function **`find_nth`** that will find the Nth element in a list. For example: **`find_nth([11, 12, 13, 14, 15], 4, X)`** returns **`X=14`**.

```
find_nth([H|_], 1, H).
find_nth([_|T], N, H2):- N2 is N - 1, find_nth(T, N2, H2).
```

4. [3 points] More Prolog:

```
likes(mary,marlins).
likes(fred,dracula).
likes(fred,bats).
likes(sue,yankees).
likes(sue,bats).
likes(fred,baseball).
likes(fred,screaming).
likes(fred,marlins).
likes(mary,ghosts).
likes(bob,cubs).
likes(bob,pumpkins).
likes(bob,dracula).
likes(mary,bats).

friends(X,Y, Something) :-
    likes(X, Something),
    likes(Y, Something),
    not(X = Y).
```

What does this query produce? Give the complete set of responses obtained after typing ; until a "No" response is given.

```
?- friends(fred,X,Y).
```

```
X = bob
Y = dracula ;

X = sue
Y = bats ;

X = mary
Y = bats ;

X = mary
Y = marlins ;

No
```

Control Flow (10 points)

1. [5 points] Define side effect and short circuiting. Give an example where short circuiting and side effects could lead to unexpected results (if you didn't know that short circuiting was being used, for example).

   side effect – influencing subsequent computation in any way other than returning a value for use in the surrounding context.

   short circuiting – (of Boolean expressions) skips over the second half of an AND if the first part is false, skips over the second part of an OR if the first part if true. (Since in both cases the value of the expression can already be determined based on the value of the first part of the expression.)

   Generic example: (false AND side effect-producing code)
   If short circuiting is used, then the side effect-producing code would never be executed when the first condition is false.
   (cond1 AND i++)
   If i is meant to count the number of times the condition is tested, it will not be incremented whenever the cond1 is false.

2. [4 points] Give an example of a tail recursive function. (This can be in any language, and does not have to be perfect syntax, or to actually do anything useful.) What is an optimization that can be applied to tail recursive functions?

```
int func (int x) {
    if (x == 0)
        return 1;
    else
        … stuff …
        return func (x – 1);
}
```

   An optimization is to convert the recursive call to a loop. In this way we avoid the overhead of function call and return and can re-use the stack frame from the original call (we don't need a new stack frame for the recursive calls anymore).

3. [1 point] Macros in C++ are an example of __**Normal**_____ Order evaluation.