CS 415 Midterm Exam – Spring 2005 -
SOLUTION

Name _____

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	
Total	**/94**

Fortran and Algol 60 (28 points)

1. [2 points] Early Fortran had recursion. TRUE or **FALSE** (circle one)

2. [4 points] What effect did your answer to question #1 have on the implementation of Fortran?

No recursion meant that there would only be one instance of a subroutine/function active at a time. So local variables could be stored in a fixed location – did not need to use the stack to allocate local variables.

3. [4 points]
   a) Describe the scoping rules in Fortran.

   global scope and for variables local to a subroutine, local scope.

   b) Fortran had: **static scoping** or dynamic scoping (circle one)

4. [8 points]
   a) Describe how call by name worked.

```
foo(int x) {
  print x
  i = i +1
  print x
}

int i = 0;
call foo (A[i])
```

   Every time the formal parameter x is accessed inside of foo, the actual parameter, in this case A[i], is re-evaluated. In this example first A[0] will be printed, then A[1]. One way of thinking of this is as if the body of the subroutine foo was pasted into the call site with all the occurrences of x replaced by A[i].

   b) Give the name of AND describe the technique used to implement call by name.

   Call by name was not implemented with the cut and paste method described above, instead a thunk – an anonymous procedure, was used. When the parameter was needed inside of a procedure, the thunk was called to get the correct address.

5. [4 points]
a) Describe the scoping rules in Algol 60.

Algol 60 allowed nested scopes. You could nest procedures or blocks. You resolved a reference by looking for the closest nested scope. Nested scopes in Algol 60 were also an example of an open scope, variables did not have to be explicitly imported in from other scopes.

b) Algol had: **static scoping** or dynamic scoping (circle one)

6. [6 points] In Algol 60, when did binding of names to memory locations occur?

on entrance to a procedure/block

What effect does this have on memory management?

Arrays were allowed to have varying size, and thus their storage needs could not be known at compile time. Arrays (and other local variables) were allocated on entrance to a procedure. A stack was used to allocate local variables, which allowed efficient re-use of memory. (As opposed to the error-prone methods of re-use in Fortran – equivalence, and re-use of memory in common blocks for multiple purposes.)

Compilation [24 points]

1. [6 points] Describe what happens during the scanning phase of a compiler. (what is the input, what is the output, what happens inside).

   The scanning phase (lexical analysis) is the first phase of compilation. A stream of characters is provided as input, and a stream of tokens is produced as output, often a token at a time as the parser requests a token. The scanner attempts to recognize the characters passed in according to the regular expressions used to specify the tokens in the language. Other tasks performed during scanning include the removal of whitespace or comments and the recognition of keywords.

2. [2 points] What is the principle of longest match? Give an example?

   When scanning, recognize the longest sequence of characters that can be recognized as a token. For example, when seeing != in the code, the scanner should recognize this as a NOTEQUAL token as opposed to a NOT token followed by an EQUAL token.

3. [6 points] Why are compilers often separated into a front end and a back end?

   The front end is generally thought of as machine-independent. The back end is thought of as containing features that apply to a particular hardware platform. Thus by separating a compiler into a front end and a back end, you can retarget your compiler to a new platform by merely re-writing the back end. Similarly you could write a front end for a new language and have it use an already existing back end for a particular platform.

   In general what is the purpose of the front end?

   Take a stream of tokens and convert it to an intermediate format (IR). The front end does analysis, attempting to understand the structure and meaning of the program. Generally scanning, parsing, and semantic analysis/IR generation/machine independent optimization are considered part of the front end.

   In general what is the purpose of the back end?

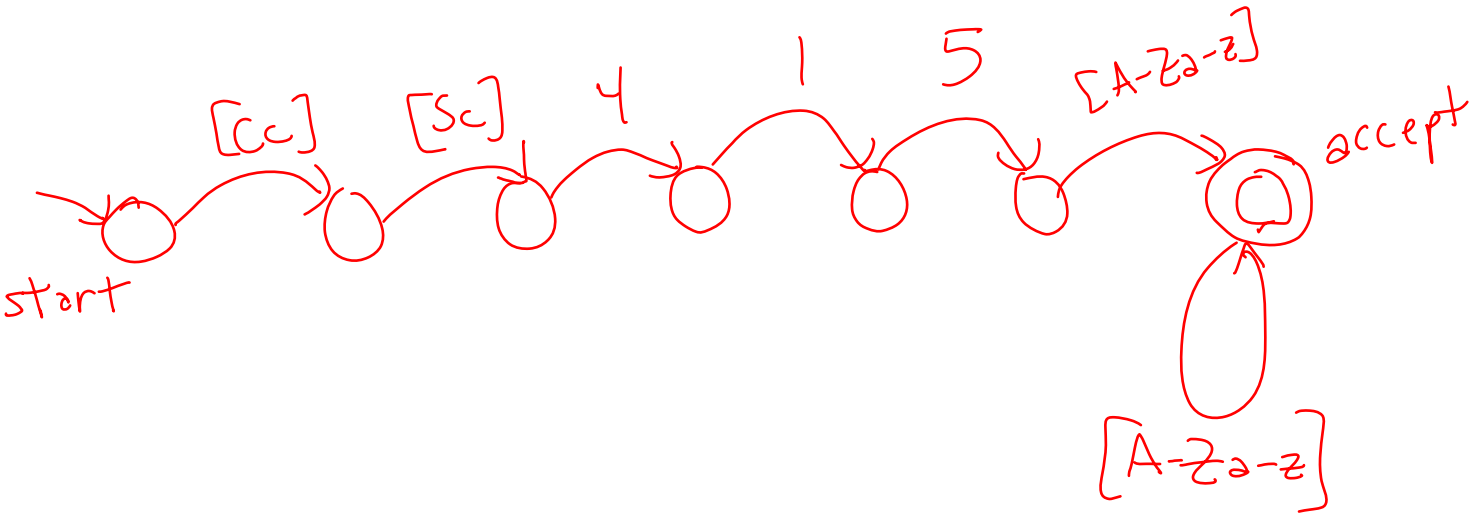
   Take a program in intermediate format and convert it into the target language (often this is the assembly language of a particular machine). Machine specific optimization and code generation are considered part of the back end.

4. [2 points] A pushdown automata is to a _**context free grammar**_ as a finite automata is to a regular expression.

5. [4 points] Write a regular expression for an identifier in the cs415 language. Identifiers must start with cs415, followed by one or more upper or lowercase letters. The c and the s in cs415 can be either upper or lowercase. Examples of valid identifiers include: “CS415rocks” and “cS415isEARLY”.

`[Cc][Ss]415[A-Za-z]+`

6. [4 points] Draw a finite automaton for the regular expression you gave in the previous question above. Be sure to properly denote the accepting states.



Names, Scopes, and Bindings (18 points)

1. [6 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, e.g. call by value).

```
int  n = 1;     // global

print_plus_n(int x) {
     cout << x + n;
}
increment_n() {
     n = n + 2;
     print_plus_n(n);
}

test() {
     int n;
     n = 200;
     print_plus_n(7);

     n = 50;
     increment_n();
     cout << n;

}
```

With Static Scoping:

8 6 50

With Dynamic Scoping:

207 104 52

2. [2 poinst] Define: referencing environment

   The set of bindings in effect at a given point in a program.

3. [2 points] Define: scope

   The textual region of the program where a binding is active.

4. [4 points] What is the static link? What is it used for?

   The static link is a pointer to the stack frame of the lexically enclosing scope. It is used for static scoping to allow the compiler to generate code that can find a variable declared in a lexically enclosing scope.

5. [4 points] In some languages, the user manages the heap. Why is this potentially dangerous?

   For example in C++ the user manages the heap by explicitly requesting memory from the heap via new and releasing it by delete. Two common problems that occur are:

   dangling pointers: the user releases memory back to the heap "too early" or rather while there are still references to that memory in their program that may be accessed in the future.

   memory leaks: when the user neglects to return memory to the heap when it is no longer needed, may cause you to eventually run out of memory on the heap.

Functional programming (24 points)

1. [4 points]
   a) What is a first class citizen in a programming language?

   Something that can be passed to or returned from a function and also have a value assigned into it.

   b) Give an example of a first class citizen in scheme.

   int, and real are first class citizens in Scheme and many other programming languages. Functions are also first class citizens in Scheme.

2. [4 points] What is programming in a "purely functional style"?

Programming without side effect, using only the composition of functions to accomplish things.

3. [2 points] What is the result of the following in Scheme:

```
(map (lambda (x) (+ x 50)) '(1 2 3 4))
```

```
(51 52 53 54)
```

4. [4 points] Assuming that the following definitions are executed in this order:

```
(define x '(3 28 400))
(define y (cons (cdr x) '(6 15 77)))
```

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

```
y                => ???     ((28 400) 6 15 77)

(cons 'x (cdr (cdr x)))   => ???     (x 400)
```

5. [10 points] Write a recursive Scheme function, **merge_sorted** that takes two sorted lists as parameters and returns a single list that contains all of the elements of both lists in sorted order. You can assume that the two lists: both contain only integer values > 0, and are sorted from smallest to largest. The two lists may not be of the same length.

   Example:
   (merge_sorted '(4 8 26) '(6 200)) => (4 6 8 26 200)

```
(define (merge_sorted x y)
  (cond ((null? x) y)
        ((null? y) x)
        ((< (car x) (car y))
           (cons (car x) (merge_sorted (cdr x) y)))
        (else (cons (car y ) (merge_sorted (cdr y) x)))))
```