CS 415 Midterm Exam – Spring 2002

Name _____KEY _____

Email Address _____ Student ID # _____

Pledge:

This exam is closed note, closed book. Good Luck!

	Score
Fortran	
Algol 60	
Compilation	
Names, Bindings, Scope	
Functional Programming	
Total	**/100**

Fortran (10 points)

1. [5 points] Describe how common blocks allowed the sharing of information. What were the drawbacks of using common blocks. Was there any other way to share information in Fortran?

Common Blocks allowed subroutines to share access to the same block of memory. In this way values stored in that memory were shared between subroutines.

A drawback to the use of common blocks was that although each subroutine explicitly "imported" the common block, each subroutine was allowed to create its own names for that memory. This could cause aliases, as in the case below where the first location in /stuff/ is referred to as the int I in subroutine A, but as the real values(1) in subroutine B.

```
subroutine A(n)
common /stuff/ i, values(100), j

subroutine B(n)
common /stuff/ values(100), i, j
```

Another way of sharing variables is by passing parameters, as could be done using the parameter n and m above – the same value could be passed to each subroutine.

2. [1 point] Fortran was originally proposed by __John Backus at IBM ___

3. [4 points] Describe the design of Fortran. Why did it turn out like it did? What were the constraints? In your opinion was it a success or a failure? Why or why not?

Fortran was based on the assembly language of the IBM 704. As a result, many of the instructions in early Fortran were merely another way of saying a particular assembly instruction – rather than a language design focused on appropriate constructs for expressing algorithms. One of the main constraints was that a compiler for the language had to produce code that would be competitive with hand-coded assembly programs. Other constraints include small memory sizes at the time, which lead to space-saving techniques that re-used memory such as common blocks and equivalences.

Success – still used today, served as a proof of concept for other high level languages and compilers, with all its faults it was still an improvement over writing assembly code directly.

Algol 60 (10 points)

1. [3 points] What does BNF stand for? When specifically was it first introduced? What is it used for?

Backus-Naur Form was originally introduced in the Algol 60 Report. It is used to specify language syntax.

2. a) [3 points] List the three storage allocation areas where values are typically allocated in memory. (Another way of saying this is to list three storage management mechanisms.) (hint: you learned this in cs201)

stack, heap, static or global area

b) [3 points] Early versions of Fortran allocated all of its variables in only one of these areas. Algol 60 made use of an additional area that was not used in Fortran. List two new features of Algol 60 that forced it to allocate variables in this area. Which area was it?

Early versions of Fortran did not have recursion so variables were allocated statically. Algol 60 introduced recursion and dynamically-sized arrays which made use of the stack.

3. [1 point] Give a one-sentence description of **own** variables in Algol 60?

Local variables that retain their value between procedure calls, similar to static variables in C++.

Compilation (30 points)

1. [10 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. I would expect you to 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] Give two advantages ~~and two disadvantages~~ of interpretation (as compared to compilation). What is an advantage of compilation over interpretation?

Advantages of Interpretation: more flexible, better error messages, good for rapid response (no compilation)

Advantages of Compilation: better performance

3. [3 points] What would the tokens be in the following statement: if (a < 5) then b = a

IF LPAREN ID(A) LT NUM(5) RPAREN THEN ID(B) ASSIGN ID(A)

4. [4 points] What does lex take as input? What does it produce? What does yacc take as input? What does it produce?

regular expressions → lex → scanner

context free grammar → yacc → parser

5. [3 points] Write a regular expression for an unsigned integer. (It is o.k. for an unsigned integer to begin with one or more zeroes.)

$[0\text{-}9]^+$

6. [4 points] Do a top down leftmost derivation of the following string given the grammar listed below:

```
a = b * (c + a)
```

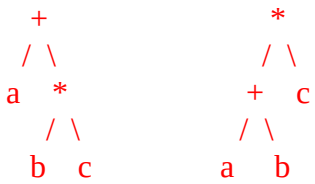
(For partial credit do any sort of derivation.) Write each step in the derivation on a separate line. Be careful not to skip any steps!

S → ID = E
ID → a | b | c
E → E + E
| E * E
| (E)
| ID

S → ID = E
→ a = E
→ a = E * E
→ a = ID * E
→ a = b * E
→ a = b * (E)
→ a = b * (E + E)
→ a = b * (ID + E)
→ a = b * (c + E)
→ a = b * (c + ID)
→ a = b * (c + a)

7. [3 points] Is the grammar from above ambiguous or not? Why or why not? If it is ambiguous, give an example.

Yes. An expression such as: a + b * c can generate two different parse trees:



Names, Scopes, and Bindings (20 points)

1. [1 point] Define scope

   the textual region of a program where a binding is active

2. [2 points] Define static scoping vs. dynamic scoping.

   static scoping – scope defined in terms of physical (lexical) structure of the program.

   dynamic scoping – bindings depend on flow of execution at runtime.

3. [2 points] What does this program print if the language uses static scoping? What does it print with dynamic scoping?

```
x: integer      -- global

proc set_x(n:integer)
     x := n

proc print_x
     write_integer(x)

proc first
     set_x(1)
     print_x

proc second
     x: integer
     set_x(2)
     print_x

set_x(0)
first()
print_x
second()
print_x
```

Static: 1 1 2 2
Dynamic: 1 1 2 1

4. [1 point] What is a binding?

An association between two things, such as a name and the thing (function, value) it names.

5. [2 points] What is the advantage of binding things as early as possible? Is there any advantage to delaying binding?

Early binding generally leads to greater efficiency (compilation approach)

Late binding general leads to greater flexibility (interpreter approach)

6. [2 points] Give an example in C++ of an object whose lifetime has exceeded the lifetime of any binding to that object. What is the name we give to this sort of thing?

```
Object *ptr = new Object;        // line 1
ptr = new Object;                // line 2
```

Now the object created on line 1 is garbage or a memory leak.
The object lives on, but we no longer have a binding to that object (assuming there are no other pointers to it).

7. [2 points] Give two pieces of information normally found in a stack frame in C++.

parameters, local variables, return address, saved registers

8. [1 point] Give the name of a language that has garbage collection.

Java, Scheme, Lisp, Python, Modula-3, ML

9. [4 points] What is an alias? Give an example (in pseudo code or a clear description). Why are aliases considered bad?

An alias is an extra name for something that already has a name in the current scope. "Two names for the same object."

Example: Fortran common blocks and equivalences can create aliases.
In C++, passing a reference to an object to a subroutine that already references that object directly. (see p. 126 in Scott)

Aliases make a program more confusing and can prevent inhibit optimizations.

10. [3 points] What is the purpose of a symbol table in a compiler? Give two sample symbol table entries.

A symbol table maps each identifier to the information that is known about that identifier, such as its type, internal structure, scope, memory address. It is used in semantic analysis and in the debugger. It can be used to tell if an identifier is declared before it is used, to check whether something is a subroutine or a variable, check that the correct type and number of parameters is being passed, etc.

Example:	Index	Symbol	Type
	1	Circle	type
	2	A	1
	3	foo	subroutine
	4	B	integer

Functional programming (30 points)

1. [2 points] What does it mean for something to be a first class citizen in a programming language? Give an example of a first class citizen in some programming language.

   A first class citizen can be passed as a parameter, returned from a subroutine, or assigned into a variable. Integers are first class in most programming languages. Functions are first class in functional languages

2. [5 points] What is the difference between normal-order and applicative order evaluation? What is lazy evaluation?

   Normal Order – evaluate arguments only when needed (sometimes this means you never have to evaluate them)

   Applicative Order – evaluate arguments before passing them to function

   Lazy evaluation is a way of implementing normal order where you *memo-ize* results – keep them around if you compute them so you do not have to compute them over again.

3. [2 points] What is a higher order function?

   a function that takes a function as a parameter or returns a function as a result.

4. [4 points] What is currying? Give an example of a curried function in Scheme.

   (Named for Haskell Curry) Replacing a multi-argument function with a function that takes a single argument and returns a function that expects the remaining arguments.

Example: (define curried-plus (lambda (a) (lambda (b) (+ a b))))
((curried-plus 3) 4)

5. [4 points] Define side effect. Give three examples of side effects.

   A side effect is something that influences future computation in any way other than by returning a value. Purely functional languages do not have side effects.

   Examples: reading or writing values, raising an exception, setting global variables

6. [3 points] Write a lambda function in Scheme that takes one argument and returns the value of that argument plus itself. Write this same function in Lambda Calculus.

   (lambda (x) (+ x x))

   Lx. x + x

7. [3 points] What is the result of the following in Scheme:

```
(map (lambda (x) (+ x 5)) '(4 6 2))
```

   (9 11 7)

8. [2 points] Assuming that the following definitions are executed in this order:

```
(define a 6)
(define b '(3 14 27))
(define c (cons a (cdr b)))
```

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

```
c              => ???    ( 6  14   27 )

(car (cdr c))  => ???    14
```

9. [5 points] Write a recursive Scheme function, nth (alist, n) that returns the n-th element of the list alist. You can assume that you will have well-formed input to the function. That is, you will always be passed a list and a value n such that the value n is not out of range.

Example: **(nth '(4 5 6) 1) => 4**

```
(define (nth alist n)
     (if (= 1 n) (car alist)
          (nth (cdr alist)  (- n 1) ) ) )
```