T/F

There is a one to one correspondence between instructions in assembly and instructions in a high level language.

The scanner analyzes the syntactic structure of a program, while the parser analyzes its lexical structure.

A local variable in C can be automatically initialized with a default value at compile time.

Subroutine closures are used in languages with shallow binding.

Interpreters offer better efficiency, while compilers offer better flexibility.

C belongs to the class of declarative programming languages.

Every case statement can also be written as one or more if...then...else statements.

Every logically-controlled loop can also be written as an enumeration controlled loop.

MC

Which of the following best characterizes the difference between declarative and imperative programming languages?

-Declarative requires that all variables be declared explicitly and imperative does not.

-Declarative is usually compiled; imperative is usually interpreted.

-Declarative describes what to do; imperative describes how to do it.

-Declarative emphasizes iteration, imperative emphasizes recursion.

Given a language and its compiler, what is the formal language accepted by the scanner?

-Set of all valid variable names in the language

-Set of all valid programs in the language

-Set of all valid tokens in the language

-Set of all valid arithmetic expressions in the language

Which of the following is an error reported by the scanner in C?

-A variable name that begins with a digit

-A missing }

-Dereferencing null pointer

-The use of a variable that has not been declared

Why are tail recursive functions useful?

-They compile faster.

-They produce faster code, since tail-recursive calls may re use space on the stack, and hence do not involve push and pop operations.

-Because they are easier to read and write.

-Because they produce faster code, since their parameters are evaluated only when needed.

FR

1. Assume two versions of the same program- one using macros, and the other using functions. In general, which will run faster? Which one will produce a larger compiler-generated code? Why?

2. Is short circuiting useful just because it is more efficient, or can it also change the program behavior (ie by computing different results or by avoiding runtime crashes)? If it can, write a short program whose behavior is different depending on whether or not the language uses short-circuiting. If not, explain why not.

3. Write a regular expression that describes the following language: the set of strings that contain an odd number of as, all adjacent, over alphabet {a,b}

4. Write a context free grammar that describes simple function headers in C syntax. Assume that the return type and the type of formal parameters are either int or float. The following are legal strings according to the grammar:

int f();

float g (int x);

int h (float a, int b, float c);

You do not need to describe the identifiers for function names and parameter names, consider them given by the scanner as ID.

5. Consider the following grammar for a Scheme expression:

expr -> ATOM | list

list -> ( exprs )

exprs -> expr exprs | E

Using this grammar, show a parse tree for the expression ( lambda (a) (\* a a))

6. Indicate the value returned by the following Scheme calls:

( car ( cdr ‘( 1 5 7 ) ) )

( cons ‘( a b ) ‘( c d ) )

( list ‘( a b ) ‘( c d ) )

( append ‘( a b ) ‘( c d ) )

( define x ‘( a b c ) )

( set-cdr! x ( cons ‘m ( cdr x ) ) )

x

7. Write a recursive function (tally V L) which counts and returns the number of occurrences of the element V in list L, as per this example:

> ( tally ‘a ‘(b a 7 c a a 3 a ) )

4

8. Consider the Scheme function f:

( define ( f V L )

( cond

( ( null? ) L ‘() )

( ( equal? V ( car L ) ) ( cdr L ) )

( else ( cons ( car L ) ( f v ( cdr L )))

Indicate the value returned by the following calls:

( f ‘a ‘() )

( f 4 ‘( 2 3 4 3 9 4 ) )

( f ‘a ‘( b ( a c ) a d a ) )

Is function f tail-recursive? Justify your answer.

9. Consider the following program fragment, in no particular language:

string tag = “b” //global

function print\_formatted\_string ( string s )

print “<”, tag, “>”, s, “</”, tag, “>”

function emphasize ( string s )

string tag = “i”

print\_formatted\_strings (s)

begin // main program

emphasize (“hi, mom”)

end

What does this program print using static scoping? Dynamic scoping?

10. Write recursive Scheme function (make-set L).

( define (make-set L )

( cond

( (null? L ) ‘() )

( (member (car L) (cdr L)) ( make-set (cdr L) )

( else ( cons ( car L ) (make-set (cdr L )))