PLC/PLCR ____________________________ **RIT Fall 2002-1**
Axel T. Schreiner print your name on every sheet Nov. 15, 2002 1/11

Please, solve **only four** of the five problems on this exam; each problem counts 25%. If you work on more then four problems, only the first four will be counted. Answer only in the space provided; try to limit the answer to an item to at most thirty words. Write legibly — if we cannot read your answer, it will not be counted.

Problem 1: Concepts [30 points, 2 each]

a. What is the difference between strong and dynamic typing?

strong: variables' types are declared and checked at compile time.
dynamic: variables get and change type through assignment at run time.

b. What is the difference between a function and a special form in Scheme?

function receives argument values, special form can decide to evaluate; special forms are built in, functions can be user-defined.

c. Why does `(let ((a '(1)) (b a)) (eq? a b))` not work?

`let` evaluates all initializers before binding any one. `a` is likely to be unbound.

d. Give an example of a successful test of two lists for equality and for identity in Scheme.

`(equal? '(1 2 3) '(1 2 3))` produces `#t`
`(eq? '(1 2 3) '(1 2 3))` produces `#f`
`(let* ((a '(1 2 3)) (b a)) (eq? a b))` produces `#t`

e. What is the difference when a variable is used as an rvalue and as an lvalue?

variable's value is used as rvalue, variable's value is changed when variable is used as lvalue. lvalue refers to variable's value's location.

f. How can you detect the difference between call by reference and call by value return?

assume that parameter refers to global variable and is assigned to; if argument is passed by reference, global will change immediately, if by value return, global will change after function ends.

g. Why does Java not support implicit comparison with zero?

conditions can be checked semantically, i.e., `if (x = 0)` is recognized as an error.

h. Why is defining EBNF in EBNF useful for *oops*?

oops converts a grammar into a recognizer; therefore, an EBNF grammar (after bootstrapping) provides a recognizer for EBNF.

i. Which operators are typically right-associative? Why?

assignment because `(x = y) = z` cannot be valid (no lvalue).
exponentiation because `(x ** y) ** z` would be `x ** (y * z).`

j. What's a literal?

a symbol that defines it's own value and type.

k. What kind of statement and what kind of expression is useful to deal with a hashtable in a scripting language?

```
for (index in table) ...
if (index in table) ...
```

l. What does row-major mean for an array?

rows are stored in consecutive memory locations, i.e., as memory addresses increase the rightmost array index changes fastest.

m. Give two reasons why an empty statement is useful.

`if` alternatives and (with embedded assignment) loop bodies might be empty; the question whether semicolon is a terminator or delimeter is mostly mute.

n. Why are operators like += useful and do more then just save typing?

the left argument is evaluated only once — this can be very important in the presence of side-effects like `a[i++]+=2`.

o. What is tail recursion? Why is it important?

a recursive call as the very last action in a subroutine; tail-recursive calls can be implemented without requiring stack space. Scheme permits unlimited tail recursion.

Problem 2: Languages [20 points, 2 each]

For the following program fragments name the programming language and answer the additional questions.

Only work on ten languages. If you work on more then ten only the first ten will be counted.

a. `/^title/ { ++ title[$2]; next }`

language? awk

purpose of fragment? count title (second field) frequency

additional code to produce a summary?

```
END { for (i in title) print i, title[i] }
```

b. `obase = 8; 17 * 3`

language? bc

purpose of first statement? set base for output

output? `63`

c.

```
class X {
  public: int f (int n) {
    return n<2 ? 1 : f(n-1) + f(n-2);
  }
};
```

language? C++

output of `(new X()) -> f(5)`? `8`

d.

```
   isum = 0
   do 10 j = 1.10
10 isum = isum + j
```

language? Fortran

actual effect of statement? sets `do10j` to `1.1`, `isum` is undetermined

correction to achieve intended effect? `do 10 j = 1,10`

e.
```
f n | n < 2 = [1]
    | otherwise = [n * first (f (n-1))] ++ f (n-1)
where first (x:xs) = x
```

language? Haskell

output of `f 5`? `[120,24,6,2,1]`

f.
```
var image = new Image();
image.src = "http://www/picture.gif";
```

language? JavaScript

purpose of fragment? (pre)load image from Web

g.
```
(cons 1 (cdr '(1 2 3)))
```

language? Lisp/Scheme

output? `(1 2 3)`

h.
```
sensor lightSensor on 2
lightSensor is light as percent
```

language? MindScript

purpose of fragment? declare light sensor on port 2, show value in 0..100

i.
```
task cry () { PlaySound(SOUND_FAST_UP); }
```

language? Not Quite C

purpose of fragment? make system sound when task is started

additional code to execute fragment? `task main () { start cry; }`

j.
```
Dim LED As New oDio1
Sub Main()
  LED.IOLine = 3
  LED.Direction = cvOutput
  LED = cvOn
End Sub
```

language? Visual Basic/oopic

purpose of fragment? set line 3 for output, set signal to high

k.
```
: Hi ." Hello World" ;
: Say-Hi Hi ;
: Hi ." Greetings" ;
Say-Hi
```

language? pbForth

effect of fragment? output `Hello World`

l.
```
/f { 2 copy lt {pop} {exch pop} ifelse } def
36 54 f =
```

language? Postscript

result of fragment? output 36, the minimum

m.
```
younger(X, Y) :- child(X, Y); child(X, Z), child(Z, Y).
```

language? Prolog

purpose of fragment? rule, relating children and grandchildren to age

two facts that only together make `younger(a,b)` true?

`child(a,c). child(c,b).`

n.
```
while x != y:
    x, y = min(x,y), max(x,y)-min(x,y)
else: print x
```

language? Python

purpose of fragment?

Euclid's algorithm for greatest common divisor; only prints if `x` equals `y` to begin with

initialization code so that output is `18`? `x, y = 18, 18`

o.
```
[ x ~= y ] whileTrue: [ x > y ifTrue: [ x := x - y ]
                              ifFalse: [ y := y - x ]].
```

language? SmallTalk

purpose of fragment? Euclid's algorithm

Problem 3: Language Design [30 points, 2 each]

This problem asks for different designs. The answers must differ in a significant way; e.g., `do-while` and `repeat-until` would *not* be considered significantly different designs for a loop.

a. The need to efficiently pass arrays as arguments has driven the design of parameter calling conventions. Describe four different mechanisms.

Java: arrays are objects. Objects are referenced and the reference values are passed.

C/C++: array name is address of first element and therefore passed as pointer value.

Fortran, call by reference: variable names, and therefore array names, are passed by address.

Algol, call by name: Jensen's device: `sub(array[i], i)`, by changing `i` one can access entire `array`.

[The question was slightly ambiguous. Describing the four calling conventions also got full credit.]

b. If a language supports functions, it needs a way to specify the result of a function. Describe three designs and one advantage of each.

Java: set result value with `return`; can leave function early.

Pascal: assign result to function name as a variable; provides implicit result variable.

Scheme: result is value of last form; needs no special syntax, but requires that everything has a value.

[The question was slightly ambiguous. Describing the typing of functions got credit, too.]

c. Selection and iteration are two basic paradigms to control program flow. Describe two designs for selection and five designs for iteration.

Selection:

`if` condition `then` statements `else` statements `fi`
`case` index `in` constant`:` statement ... `else` statement

Iteration:

`while` condition `do` statements `end`, test before any loop
`do` statements `while` condition, test after first loop
`for (`initialize`;` condition`;` step`)` statement
`for (`index `in` hashtable`)` statement
tail recursion

d. What is the essential design objective for control structures in modern programming languages?

one entry — one exit

Problem 4: Language Processing [26 points]

a. Use EBNF to specify an unambiguous grammar for arithmetic expressions with parentheses and numbers where, however, addition and subtraction take precedence over multiplication and division. [6 points]

```
product: sum ( ('*' | '/') sum )*;
sum: term ( ('+' | '-') term )*;
term: 'Number' | '(' product ')';
```

b. Assume that `condition` and `expression` have been defined already. Use EBNF to specify a LL(1) grammar starting with `statement`, which includes a statement type for performing arithmetic, for selection from two alternatives, for an infinite loop, and for a way to terminate the loop only from within a loop. It must be possible to nest the statements. [8 points]

```
statement: 'expression' ';'
         | 'if' 'condition' 'then' statement*
                            'else' statement* 'fi'
         | 'loop' exitStatement* 'end';

exitStatement: 'exit'
             | 'expression' ';'
             | 'if' 'condition' 'then' exitStatement*
                                'else' exitStatement* 'fi'
             | 'loop' exitStatement* 'end';
```

c. Does your design need a statement type for grouping multiple statements? Why or why not? [2 points]

no, because both, `if` and `loop`, accept groups.

d. Consider an *oops* observer for the rule

```
try: 'try' statements `catch` "Id";
```

Explain which observer methods you would define and what information they could use or provide. [8 points, need at least *]

`void shift (Terminal sender)` would see `try`, could install handler.

`void shift (BTerminal sender)` would see `catch`, could provide jump.

`Observer init (int rule, String name)` needs to create observer for `statements`.*

`void shift (Observer sender)` sees completed `statements`, can ask `sender.value()`, e.g., for a tree representation.*

`void shift (DTerminal sender)` would see `Id`, can ask `scanner.value()` for actual name.*

`void reduce()` called next, could wrap up.

`Object value()` could provide tree representation.*

e. What is the essential idea in *oops*' observer pattern and what is it's advantage? [2 points]

one `Observer` for each rule activation, `Observer` methods can be tightly related to the symbols in the rule.

Problem 5: Scheme Programming [22 points]

a. Define a function `bsearch` that accepts a key and a inorder sorted binary tree represented as nested lists of zero or three elements and returns `#t` exactly if the key is in the tree. [5 points]

```
(bsearch 5 '(((() 1 ()) 2 (() 3 ()))
             4
             ((() 5 ()) 6 (() 7 ())))
```
`)` produces `#t`

```
(define (bsearch key tree)
  (cond
    ((null? tree) #f)
    ((equal? key (cadr tree)) #t)
    ((< key (cadr tree)) (bsearch key (car tree)))
    (else (bsearch key (caddr tree)))
  )
)
```

b. Define a function `pairwise` that accepts a function `f` and returns a function that accepts an even number of arguments, applies `f` to successive pairs of arguments, and returns the list of results; an empty list shoud be returned if the number of arguments is odd. [5 points]

`((pairwise +) 1 2 3 4 5 6)` produces `(3 7 11)`

```
(define (pairwise f)
  (define (result list) ; recursively for list argument
    (if (or (odd? (length list)) (< (length list) 2))
      ()
      (cons (f (car list) (cadr list))
            (result (cddr list))
      )
    )
  )
  (lambda list (result list)) ; turns arguments into list
)
```

Briefly describe a recursive idea on which each of the following Scheme functions can be based — no need to write the code itself. [3 points each]

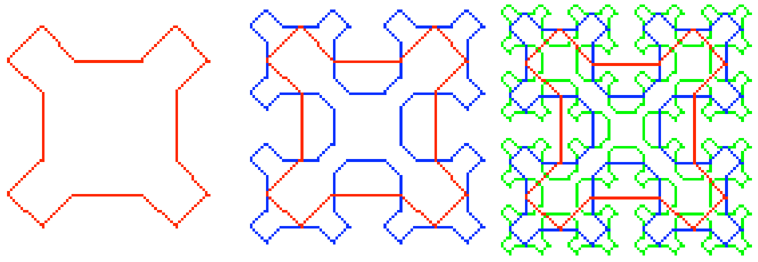
c. `(define (permute n) ; return all permutations of n elements`

insert `n` everywhere in every permutation of `n-1` elements.

d. `(define (quicksort list) ; return a sorted list`

concatenate `quicksort` of elements in `list` less than first element, list with first element, and `quicksort` of elements in `list` greater then or equal to first element.

e. `(define (sierpinski n) ; plot a space-filling curve`



starting at the bottom and proceeding counterclockwise:

```
A: A NE B E D SE A
B: B NW C N A NE B
C: C SW D W B NW C
D: D SE A S C SW D
```

f. `(define (queens n) ; solve the n Queens Problem`

try to place a queen in a row and a column: if possible, recurse to next column. If not possible, or if recursion does not succeed, recurse to next row. Fail if row gets too big, succeed if column gets too big.