

CPTS 355 Midterm, Spring 2004

NAME _____

Directions: Please answer all of the questions. The test is open book and open lecture notes and other reference materials. You may **not** use a computer, PDA, calculator or phone during the test.

There are **9** major problems on **6** pages (3 sheets, front and back). Make sure that you have a complete exam **now**.

1) **[14 pts] Directions:** State whether each statement is TRUE or FALSE by writing TRUE or FALSE in the space provided. **(1 point each)**

- a) T F Scheme uses postfix notation for expressions.
- b) T F A statically allocated storage object is sometimes allocated on the stack.
- c) T F Names of C structs and typedefs are in different namespaces.
- d) T F Each function call in PostScript introduces a new namespace.
- e) T F PostScript uses postfix notation for expressions.
- f) T F PostScript has static typing.
- g) T F Scheme has dynamic typing.
- h) T F PostScript is compiled.
- i) T F In Postscript, the mul function (multiplication) has higher precedence than the add function (addition).
- j) T F In axiomatic semantics, post-conditions are related to pre-conditions to prove program correctness.
- k) T F Names in Scheme are explicitly bound to storage locations.
- l) T F In PostScript, the lifetime and scope of a name are equivalent.
- m) T F In C, the lifetime and scope of a name are equivalent.
- n) T F In C, each struct declaration introduces a new namespace that contains only the names of the struct's members.

Directions: The following questions are short answer questions. Please provide your answer in the space provided.

2) [4 points] PostScript and Scheme use similar notations (one uses prefix, the other postfix) for expressions. Scheme expressions are often parenthesized and PostScript expressions never are. What are the consequences of this language design decision for the readability and writability of these two languages.

Parentheses in Scheme enhance readability by visually delimiting the scope of each function call or special form. Because of parentheses it is possible for functions to take an arbitrary number of arguments which reduces the number of function calls that have to be written. On the other hand, the need to balance parentheses can make writing Scheme programs more difficult.

In Postscript the lack of parentheses means that each operator can take only a fixed number of arguments. It also means that you have to be very careful to track the contents of the operand stack while programming.

3) [8 points] Name two features of the C language that cause it not to be strongly typed. Give an example of how each can cause an undetected type error.

Unions, casting of pointer types, separate compilation are good examples. Coercions such as between ints and chars were acceptable if well-supported by your arguments.

4) [6 points] Determine the weakest precondition for the following program fragments, using the assignment axiom. Show your work.

a) $z = 3; \quad z = \text{foo} * 5 + z; \quad \{z == 23\}$
 $\text{wp}(z=3; z=\text{foo}*5+z, \{z==23\}) == \text{wp}(z=3, \{\text{foo}*5+z==23\}) ==$
 $\{\text{foo}*5+3==23\} == \{\text{foo}==4\}$

b) $z = 3; \quad y = \text{foo} * 5 + z; \quad \{z == 19\}$
 $\text{wp}(z=3; y=\text{foo}*5+z, \{z==19\}) == \text{wp}(z=3; \{z==19\}) ==$
 $\{3==19\} == \text{false}$

Notice that the variable y doesn't appear in $z==19$ so the $\text{wp}(y=\text{foo}*5+z, \{z==19\})$ is just $\{z==19\}$.

5) [3 points] List 3 of the stacks that are part of the virtual machine for executing PostScript programs.

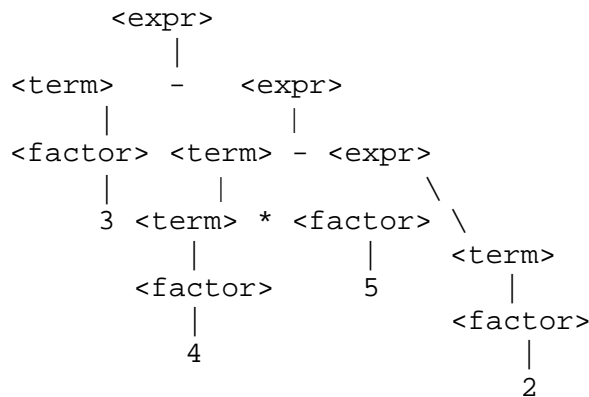
Operand stack (or just the stack)
 graphics context stack
 dictionary stack

6) [10 points] Draw the parse tree for the sentence

3 - 4 * 5 - 2

using the grammar given below. Assume that <expr> is the start symbol.

<expr> ? <term> - <expr> | <term>
 <term> ? <term> * <factor> | <factor>
 <factor> ? 3 | 4 | 5 | 2



What is the value of this expression when computed according to your parse tree? ____-15____
 3 - ((4*5) - 2)

For this grammar, does * have higher precedence than - ? __yes__

For *, is this grammar left associative, right associative, or neither? __left__

Is this grammar ambiguous? ____no____

7) [10 pts] Consider the following PostScript program.

```
1) /f1 { 1 dict begin
2)   /x exch def
3)   f2
4)   x
5)   end
6) } def
7) /f2 { /x x 3 mul pop (41*3) def } def
8) 4 f1
```

On line 7, what is the value of the boldfaced **x** during execution of this program? 4

True or false: the names f1 and f2 are defined in the same dictionary? T

True or false: the name x is defined in more than one dictionary? F

What type does the value at the top of the stack have when this program finishes? string

What value is at the top of the stack when this program finishes? the string 41*3

8) [10 pts] Consider the following Scheme program.

```
1) (define (f1 x)
2)   (define (f2)
3)     (let
4)       ((x (* x 3)))
5)       (set! x (* 41 3))
6)     )
7)   )
8) (f2)
9) x
10) )
11) (f1 4)
```

On line 4, what is the value of the boldfaced **x** during execution of this program? 4

On what line number is the boldfaced **x** on line 4 bound? 1 or 11

At what line number is the x used on line 5 bound? 4 or 5

At what line number is the x used on line 9 bound? 1 or 11

What is the result of evaluating (f1 4) on line 11? 4

9) [10 pts] Define a Scheme function named *altsum* that when given a list of numbers ($x_0 x_1 x_2 x_3 \dots x_n$) returns $x_0 - x_1 + x_2 - x_3 + \dots$. That is, even-numbered terms are added and odd-numbered terms are subtracted. If the list is empty return 0. Examples: `(altsum '(5 4 3 2 1 0))` is 3; `(altsum '(4))` is 4; `(altsum '(1 4))` is -3.

(Hint: define a second function, *altdiff*, that when given a list of numbers ($x_0 x_1 x_2 x_3 \dots x_n$) returns $-x_0 + x_1 - x_2 + x_3 \dots$. Again, if the list is empty return 0. Examples: `(altsum '(5 4 3 2 1 0))` is -3; `(altsum '(4))` is -4; `(altsum '(1 4))` is 3. Use *altdiff* in implementing *altsum* and *altsum* in implementing *altdiff*.)

```
(define (altsum L)
  (cond ((null? L) 0)
        (else (+ (car L) (altdiff (cdr L)))))
  )
)
```

```
(define (altdiff L)
  (cond ((null? L) 0)
        (else (- (altsum (cdr L)) (car L))))
  )
)
```

The specification does not say anything about whether the list elements are themselves odd or even so attempts to test `(car L) mod 2` were inappropriate.

You need to be careful about what is subtracted from what by `(-)`.