

Spring 2003 - Programming Languages Qualifier

Computer Science Department
University of Massachusetts Lowell
Lowell, MA 01854

March 1, 2003.

Each problem is worth 10 points. Attempt any 10.

Please be brief: long discussions with little content will receive **negative** points.

1. Module Systems.

- (a) (5 pts) What are *Structures*, *Signatures* and *Functors* in SML? What is their use?
- (b) (5 pts) Describe the analogous (same function) entities - at least the ones that exist - in two of Ada, C, C++, Modula-3 or Java. You may use only one language from each of the two classes: imperative and object-oriented.

2. First Class Entities.

An entity in a programming language is called first-class if

- (a) it can be passed as an argument;
- (b) it can be evaluated;
- (c) it can be assigned;
- (d) it can be used as a component in a composite value;
- (e) it can be returned as a value;

What entities are first-class in

- i) SML
- ii) C
- iii) Scheme
- iv) C++

Explain - some entities are "almost first-class" in that, for example, they may not be returned by a function but may fulfill all other requirements: word your answers to reflect these restrictions.

3. **Parameter Passing.** Many parameter-passing mechanisms have been implemented in programming languages. Given the declarations

```
a = array[1..5] of integer;

procedure Swap(x, y : integer);
  var temp: integer;
  begin
    x := y;
    y := temp;
  end;
```

If $a = [2,3,4,5,6]$ and $i = 3$ at the time of the call `Swap(i, a[i])`, what are the values of i and a at the end of the procedure execution

- (a) (2 pts) If parameters are passed by value
- (b) (2 pts) If parameters are passed by reference
- (c) (2 pts) If parameters are passed by value-result
- (d) (4 pts) If parameters are passed by name

Explain.

4. **Static and Dynamic Binding.** Discuss the distinction between static and dynamic binding. Furthermore, given the definitions

```
(define a 10)
(define add-a
  (lambda (x)
    (cond ((= x 0) (+ a x))
          (else (add-a (- x 1))))))
(define trick-me
  (lambda (f y)
    (let ((a 17)
          (+ -)
          (add-a (lambda (x) (+ a x))))
      (f y))))
```

what would the result of the call `(trick-me add-a 3)` be under static binding? Under dynamic binding? Of `(trick-me add-a -3)`? Explain. In particular, SHOW - via environment diagrams - the difference in the environments in which the body of `add-a` is evaluated in each instance.

5. **Polymorphism.** Compare and contrast - with examples associated with specific languages -

- (a) ad-hoc polymorphism,
- (b) parametric polymorphism,
- (c) inclusion polymorphism.

6. **Fixed Points.** Find the least fixed points of

- (a) $f = \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } f(n + 1)$
- (b) $f = \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } f(n - 1)$

The domain over which the functions are (partially) defined is the non-negative integers. Make sure you show the construction of each fixed point. What makes the fixed point you constructed the *least fixed point*?

7. **Reduction Strategies.**

- (a) State the Church-Rosser Theorem and explain why it is important.
- (b) Show what the following expression reduces to using *i*) normal order reduction and *ii*) applicative order reduction:

```
((lambda (y) 4)
  ((lambda(x) (x x)) (lambda (x) (x x))))
```

Make sure you go through enough steps so that your conclusions are justified.

8. **Exceptions.** Several languages support the notion of an *exception*.

- (a) What is an exception?
- (b) Different languages use different mechanisms for the handling and definition of exceptions. Give examples in two of Ada, Modula-3, SML and Java.

9. **Scheme and Type Inference.** A function f with domain A and range B is said to be of type $A \rightarrow B$, usually denoted by $f : A \rightarrow B$. A function f of two arguments would have its type denoted by $f : A \times B \rightarrow C$. A , B , and C are also types (e.g., Characters, Integers, Reals, etc...). Given that the type of $+$ is $Integer \times Integer \rightarrow Integer$, infer the type of the Scheme functions:

```
(define sum1 (lambda (l)
  (cond ((null? l) 0)
        (else (+ (car l) (sum1 (cdr l)))))))
(define insert (lambda (z f l)
  (cond ((null? l) z)
        (else (f (car l) (insert z f (cdr l)))))))
(define sum2 (lambda (l) (insert 0 + l)))
```

Hints: let Integer-List denote the type of a list of Integers. If X denotes an unspecified type, X -List denotes a list of items of type X . Some of the functions above may have no restrictions on the types of the underlying items being manipulated. Assume, for definiteness, that each list is composed of objects of the same type - no lists of mixed types are allowed.

10. Prolog and Logic Programming.

- (a) What is a *clause*?
- (b) What is the *general resolution principle*?
- (c) What is a *Horn Clause*?

11. **Prolog Programming.** Consider the Prolog assertions (using standard Edinburgh syntax - lower case initiated strings denote constants, upper case initiated ones denote variables):

```
sentence(Whole,Suffix) :-
    noun_phrase(Whole,Suffix1), verb_phrase(Suffix1,Suffix).
noun_phrase(Whole,Suffix) :-
    noun(Whole,Suffix).
noun_phrase(Whole,Suffix) :-
    adjective(Whole,Suffix1), noun_phrase(Suffix1,Suffix).
verb_phrase(Whole,Suffix) :-
    verb(Whole,Suffix).
verb_phrase(Whole,Suffix) :-
    verb(Whole,Suffix1), adverb(Suffix1,Suffix).
verb_phrase(Whole,Suffix) :-
    verb(Whole,Suffix1), verb(Suffix1,Suffix).
verb_phrase(Whole,Suffix) :-
    verb(Whole,Suffix1), verb(Suffix1,Suffix2), adverb(Suffix2,Suffix).
noun([planes|L],L).
adjective([flying|L],L).
verb([can|L],L).
verb([be|L],L).
adverb([dangerous|L],L).
```

- (a) Given this database, what will the query `sentence([flying, planes, can, be, dangerous], X)` return? Careful, because there is more than one way to satisfy the query and Prolog will allow the user to exhaust all answers.
- (b) What two assertions would you have to add for the program to also recognize "dangerous planes can be flying" as a sentence? This may do some violence to the English language, but we are doing programming, not linguistics.

12. **Denotational Semantics.** Assume the language of Regular Expressions has

Syntactic Categories

$A \in \mathbf{Alpha}$ the alphabet

$R \in \mathbf{RE}$ regular expressions

Where the regular expressions have BNF definition

$$R ::= A \mid \emptyset \mid (R + R) \mid (R \cdot R) \mid R^*$$

Value Domains

Lang = formal languages (sets of strings over **Alpha**)

Semantic Functions

$$\mathcal{A} : \mathbf{Alpha} \rightarrow \mathbf{Lang}$$

$$\mathcal{D} : \mathbf{RE} \rightarrow \mathbf{Lang}$$

Starting with the semantic function definitions:

$$\mathcal{A} \parallel A \parallel = \{A\}$$

$$\mathcal{D} \parallel A \parallel = \mathcal{A} \parallel A \parallel$$

where $\{A\}$ is the language containing the single character string, denoted by $\mathcal{A} \parallel A \parallel$,

(a) (5 pts) extend the semantic functions to provide the denotational semantics for regular expressions;

(b) (5 pts) apply the semantic functions to obtain the denotation of the regular expression $A \cdot B \cdot (A^* + B^*)$, over the alphabet $\{A, B\}$. Show the details.