COP4020
Programming Language Concepts
Spring 2017
Sample Final Exam Questions
SOLUTIONS

Disclaimer: This is a collection of sample exam questions, not a sample exam.
1. Add the unary postfix auto-increment operator ‘++’ to the RPAL Tiny interpreter. The AST is the form (‘post++’, ‘n’). To evaluate this operator, first evaluate the operand ‘n’. Then increment ‘n’ (changing memory). The resulting value is the value of ‘n’ before the increment.

2. Make the ‘else’ clause optional in the ‘if’ statement in the RPAL Tiny interpreter.

3. Add a ‘read’ statement to the RPAL Tiny interpreter. This is in addition to the existing ‘read’ expression. The AST is of the form (‘read’, ‘n’). To execute this statement, extract a value from the input, and store it in variable ‘n’ (changing memory).

4. Add the ‘repeat’ statement to the RPAL Tiny interpreter. The AST is of the form (‘repeat’, S, E). The statement S is executed at least once. After that, E is evaluated: if false, S is executed and E is re-evaluated; if true, the iteration stops.

5. Add the for statement (as in C, C++ or Java) to the RPAL Tiny interpreter. The AST is of the form (‘for’, S1, E2, S3, S4), where S1, S3, and S4 are statements, and E2 is an expression. Hint: (‘for’, S1, E2, S3, S4) is equivalent to (‘;’, S1, (‘while’, E2, (‘;’, S4, S3)))).

Note: On the final exam, for questions such as these (1-5), the Tiny RPAL interpreter will be provided.

THE FOLLOWING extended RPAL Tiny interpreter implements solutions to all five problems.
let EQ x y = Istruthvalue x & Istruthvalue y 
    -> (x & y) or (not x & not y) 
    | Isstring x & Isstring y 
    or Isinteger x & Isinteger y -> x eq y
    | false
in
let COMP f g x = let R = f x in R @EQ 'error' -> 'error' 
    | g R
in
let PIPE x f = x @EQ 'error' -> 'error' 
    | (f x)
in
let Return v s = (v,s)
in
let Check Dom (v,s) = Dom eq 'Num' -> Isinteger v -> (v,s) 
    | Dom eq 'Bool' -> Istruthvalue v -> (v,s) 
    | 'error'
in
let Dummy s = s
in
let Cond F1 F2 (v,s) = s @PIPE (v @EQ F1 | F2)
in
let Replace m i v x = x @EQ i 
    -> v | m x
in
let Head i = i 1
in
let Tail T = Rtail T (Order T) 
    where rec Rtail T N = 
    N eq 1 -> nil | 
    (Rtail T (N-1) aug (T N))
in
let rec EE E (m,i,o) = 
    Isinteger E -> Return E (m,i,o) 
    | Isstring E -> 
    ( E eq 'true' -> Return true (m,i,o) 
    | E eq 'false' -> Return false (m,i,o) 
    | E eq 'read' -> Null i -> 'error' | (Head i,(m,Tail i,o)) 
    | (let R = m E in R @EQ 'undef' -> 'error' 
    | (R,(m,i,o)))
    ) 
    | Istuple E -> 
    ( E 1) @EQ 'not' -> 
    (m,i,o) @PIPE EE(E 2) 
    @PIPE (Check 'Bool') 
    @PIPE (fn(v,s).(not v,s)) 
    | (E 1) @EQ '<=' -> 
    (m,i,o) @PIPE EE(E 2) 
    @PIPE (Check 'Num') 
    @PIPE (fn(v1,s1). s1 
    @PIPE EE(E 3) 
    @PIPE (Check 'Num') 
    @PIPE (fn(v2,s2).(v1 le v2,s2))
```ml
let rec CC C s =
  not (Istuple C) -> 'error'
| (C 1) @EQ ':=' -> s @PIPE EE (C 3)
  @PIPE (fn(v,s).(Replace (s 1) (C 2) v,s 2,s 3))
| (C 1) @EQ 'read' -> Null (s 2) -> 'error'
  | (Replace (s 1) (C 2) (Head (s 2)), Tail (s 2), s 3)
| (C 1) @EQ 'print' -> s @PIPE EE (C 2)
  @PIPE (fn(v,s).(s 1,s 2,s 3 aug v))
| (C 1) @EQ 'if' -> s @PIPE EE (C 2)
  @PIPE (Check 'Bool')
  @PIPE (Cond (CC(C 3)))
  (Order C eq 4 -> (CC(C 4)) | Dummy)
| (C 1) @EQ 'while' -> s @PIPE EE (C 2)
  @PIPE (Check 'Bool')
  @PIPE Cond (CC(';',C 3,C)) Dummy
| (C 1) @EQ 'repeat' -> s @PIPE CC (';', C 2, ('while', ('not', C 3), C 2))
| (C 1) @EQ 'for' -> s @PIPE CC (';', C 2, ('while', C 3, (';', C 5, C 4)))
| (C 1) @EQ ';' -> s @PIPE CC (C 2)
  @PIPE CC (C 3)
| 'error'
in
let PP P =
  not (Istuple P) -> (fn i. 'error')
| not ((P 1) @EQ 'program') -> (fn i. 'error')
| ((fn i. CC (P 2) ((fn i.'undef'),i,nil)) @COMP (fn s.{s 3}) )

in
Print (PP ('program'
  ,'
  ,('read', 'x'),
   (';', ('print', ('post++', 'x'))),
   (';', ('print', 'x')),
   (';', ('for', ':=', 'n', 1),
    ('<=', 'n', 5),
    (':=', 'n', ('+', 'n', 1))
))
```

6. Write, test, and debug a complete RPAL program that defines and uses a function to merge two strings. Assume the characters in each string are in ascending order. For example,

```
merge ‘acegh’ ‘bdfi’ = ‘abcdefghi’
```

```
let merge s1 s2 =
  not Isstring s1 or not Isstring s2 -> 'error'
| rmerge s1 s2 ''
  where rec rmerge s1 s2 R =
    s1 eq '' -> R @Conc s2
  | s2 eq '' -> R @Conc s1
  | Stem s1 > Stem s2 ->
    rmerge s1 (Stern s2) (R @Conc (Stem s2))
  | rmerge (Stern s1) s2 (R @Conc (Stem s1))

in Print(merge '137' '246',
  merge '13' '456',
  merge '1234' '34',
  merge '12' 3,
  merge 2 '134' )
```

7. Write, test, and debug a complete RPAL program that defines and uses a function to remove the last n characters from a given string, if possible. For example,

```
remove ‘abcdef’ 3 = ‘abc’
```

```
let rec length s = s eq '' -> 0 | 1+length(Stern s)
```
let remove s n =
   not Isstring s -> 'error'
   | not Isinteger n -> 'error'
   | n > length s -> 'error'
   | rrem s (length s - n) ''
   where rec rrem s m R =
      m eq 0 -> R
      | rrem (Stern s) (m-1) (R @Conc Stem s)
in
Print (remove 'abcdefg' 3,
       remove 'ab' 3,
       remove 3 2,
       remove 'abc' 'a')

8. A severe system crash takes place on your computer system, leaving you with an implementation of a Pascal-like language in executable form. The source code is gone forever. You remember that before the crash there were three executable compilers. One used copy-in,copy-out as the parameter passing mechanism; another used pass by reference, and the third used pass-by-value. You don't know which compiler survived. The only difference between the three compilers was the parameter passing mechanism. Write a (short) program in Pascal/C-like pseudo-code that will allow you to determine which compiler survived the crash. Specifically, write a program whose output will be 0 if copy-in,copy-out is used, 1 if pass by reference is used, and 2 if pass-by-value is used. Show a legible trace of the program's execution.
9. For the program shown below, draw a picture of the run-time environment, at the point marked "HERE", i.e., at the point when the constant value “1” has been placed on the stack. Show all temporaries, local variables, parameters, locations reserved for return values, and return addresses, as well as the base pointer, frame pointer(s), and the stack pointer. Use the frame organization illustrated in Lecture 12.2.

```c
function fact(int n) {
    int t;
    if n = 0 then return 1  // HERE
    else {
        t=fact(n-1);
        return n*t;
    }
}
main()
begin
    write(fact(3))
end;
```
10. Given the “family relation” facts shown in the next-to-last example in Lecture 14.3 (defining FatherOf and MotherOf predicates), write a predicate CousinOf(X,Y). Trace the search process through the database, to answer the query ?-CousinOf(ken,X). Explain why other family relations, such as UncleOf(X,Y) and Niece(X,Y), are unrealistic with the given database.
Predicate:  CousinOf(X,Y) :- PO(A,B), PO(A,C), PO(B,X), PO(C,Y), not PO(C,X).

The following diagram illustrates the tree of possibilities for CousinOf(ken,Z). CousinOf(ken,Z) depends on all five predicates being true: PO(A,B), PO(A,C), PO(B,ken), PO(C,Z), and not PO(C,ken). Listed under PO(A,B) are all the possibilities in the database; each one is the root of the tree of possibilities for the other sub-predicates. Not shown: the breakdown of PO (ParentOf) into MotherOf and FatherOf. Processing a tree branch stops as soon as a false predicate is encountered.

Predicates such as UncleOf(X,Y) and Niece(X,Y) are unrealistic because they would require gender.