CSE305 Sample Prelim I Exam For Spring 2023

Closed book, no electronics, one notes sheet allowed but otherwise closed notes, closed neighbors, 75 minutes after 5-minute read-in period. *Show your work*, and explain your reasoning where it is naturally called for—doing so may help for partial credit.

(1) (30 pts.)

The following "EBNF fragment" could be part of a grammar for Java, although it omits access modifiers (like "public"), throws clauses, arrays, and qualified (i.e., dotted) class-or-interface names (CINAMEs). Literal commas and parens and `< >` are quoted to distinguish them from grammar notation, while `; & ?` are literal characters. The grammar defines a syntax for prototypes of possibly-generic methods appearing in interfaces.

```
IMETHOD ::=  ["<" TP{,TP} ">"] TYPE ID "(" [PARAM{"," PARAM}] ")" ;
TP      ::=  ID [extends CINAME{& CINAME}]
TA      ::=  CINAME  |  ? extends CINAME  |  ? super CINAME
CINAME  ::=  ID ["<" TA{,TA} ">"]          //real Java BNF allows dotted names
PARAM   ::=  [final] TYPE ID               //real Java BNF allows arrays too
TYPE    ::=  PRIMTYPE | CINAME | void      //and doesn't say "void" is a "type"
PRIMTYPE::=  int | long | short | float | double | char | byte | boolean
ID      ::=  ---any valid identifier---
```

(a) Taking `IMETHOD` as the start symbol, call the above grammar "G". For each of the following eight strings, say "yes" if it is derivable in G, and "no" if not. You need not show derivations or parse trees here—just the yes/no answer is enough—but scratchwork may help for partial credit if you're wrong. ($8 \times 3 = 24$ pts.)

(i) `void foo(int x, ? extends Bar y);`

(ii) `void foo(int x, Bar<? extends Star> y);`

(iii) `Bar foo(Bar x, Bar<? extends Bar> y);`

(iv) `void foo(Bar<int x, ? extends Star> y);`

(v) `void foo(int x, Bar<T, ? extends Star> y);`

(vi) `void foo(int x, Bar<T extends Star> y);`

(vii) `<T extends Star> Bar foo(int x, Bar y);`

(viii) `Bar<T extends Star> void foo(int x, Bar y);`

(b) It is not really proper to call `void` a "type" in Java, and method parameters cannot be `void`. Fix the "bug" by removing the option `TYPE ::= void`, and adding option(s) for different variable(s) to produce a "correct" grammer. (6 pts.)

(2) (6+9+3 = 21 pts.)

Consider the following expression in C/C++/Java. Note that these languages consider assignment to be an operator of lowest precedence and allow nested assignments.

```
x = y + (z = x + y) - z;
```

(a) Write an expression tree for this expression. You must follow the rules of precedence and associativity in C/C++/Java, including those for `=` as a binary operator.

(b) Now write a *parse tree* in the tiered grammar below, It resembles the the answer for HW2 problem (3) with assignment in place of rightshift, except that assignment is *right*-associative.

(c) If one removes the `(...)` around `(z = x + y)`, the code fails to compile. Why?

```
A  ::=  E  |  E = A
E  ::=  T  |  E+T  |  E-T
T  ::=  F  |  T*F  |  T/F  |  T%F
F  ::=  -F |  (A)  |  any-constant-or-variable.
```

(3) (12+6 = 18 pts.)

Suppose we have the following code with nested declarations inside different referencing environments:

```
class Bar {
   String x = "Bar.x";
   String y = "Bar.y";
   void foo1() {
      String x = "Foo1.x";
      y = x;
      foo2();
   }
   void foo2() {
      y = x;
   }
   ...
}
```

(a) For each occurrence of `x` and `y` in the two assignment statements `y = x;`, say which of the three declarations it refers to. You should have 4 separate answers.

(b) If `foo1()` is called, what is the final value of `y`?

(4) (31 pts. total)

Consider the following two OCaml functions. Note mod is the modulo function:

```
let rem a = a mod 2
let rec useTheForce1 han =
  match han with
    | [] -> []
    | h1::h2 -> if (rem h1) = 0
                then h1*h1::(useTheForce1 h2)
                else useTheForce1 h2

let useTheForce2 han =
  List.fold_left (fun acc h1 -> if (rem h1) != 0
                                then acc
                                else h1*h1::acc)
                  [] han
```

(i) Are useTheForce1 and useTheForce2 equivalent (by equivalent we mean for the same input they produce the same output)?

- A: Yes, both functions return the same lists with the same elements in the same order
- B: No, the list produced by useTheForce1 is the reverse of the list produced by useTheForce2
- C: Yes, both functions will return the same integer
- D: No, the two functions do not return the same type
- E: No, the list produced by useTheForce1 will contain more elements than the list produced by useTheForce2

(ii) What is the type of useTheForce1?

- A int list -¿ int list
- B: 'a list -¿ 'a list
- C: int list -¿ int * int list -¿ 'a list
- D: int list -¿ int
- E: int list -¿ int -¿ int list

(iii) What is the type of useTheForce2?

- A: int list -¿ int list
- B: int list -¿ ('a * 'b -¿ 'b) -¿ 'b -¿ 'a list -¿ 'b -¿ int list
- C: ('a * 'b -¿ 'b) -¿ 'b -¿ 'a list -¿ 'b
- D: 'a list -¿ 'a list
- E: 'a list -¿ ('a * 'b -¿ 'b) -¿ 'b -¿ 'a list -¿ 'b -¿ 'a list

End of Exam