Programming Languages - Homework 1

Name: Peng-Yu Chen (pyc305)

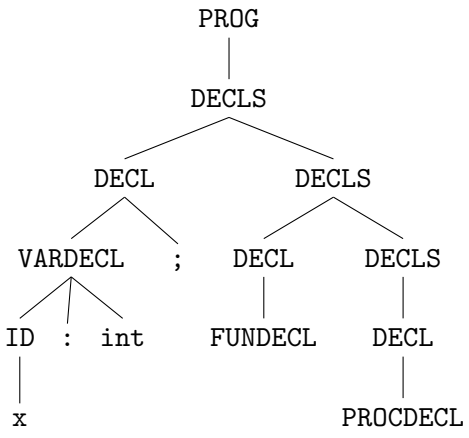
1. (a) `[a-z]*[A-Z][a-z]*[0-9][a-z]*[A-Z][a-z]*`

   (b) `(-|+|ϵ)[0-9][0-9]*.[0-9][0-9]*E(-|+|ϵ)[0-9][0-9]*`

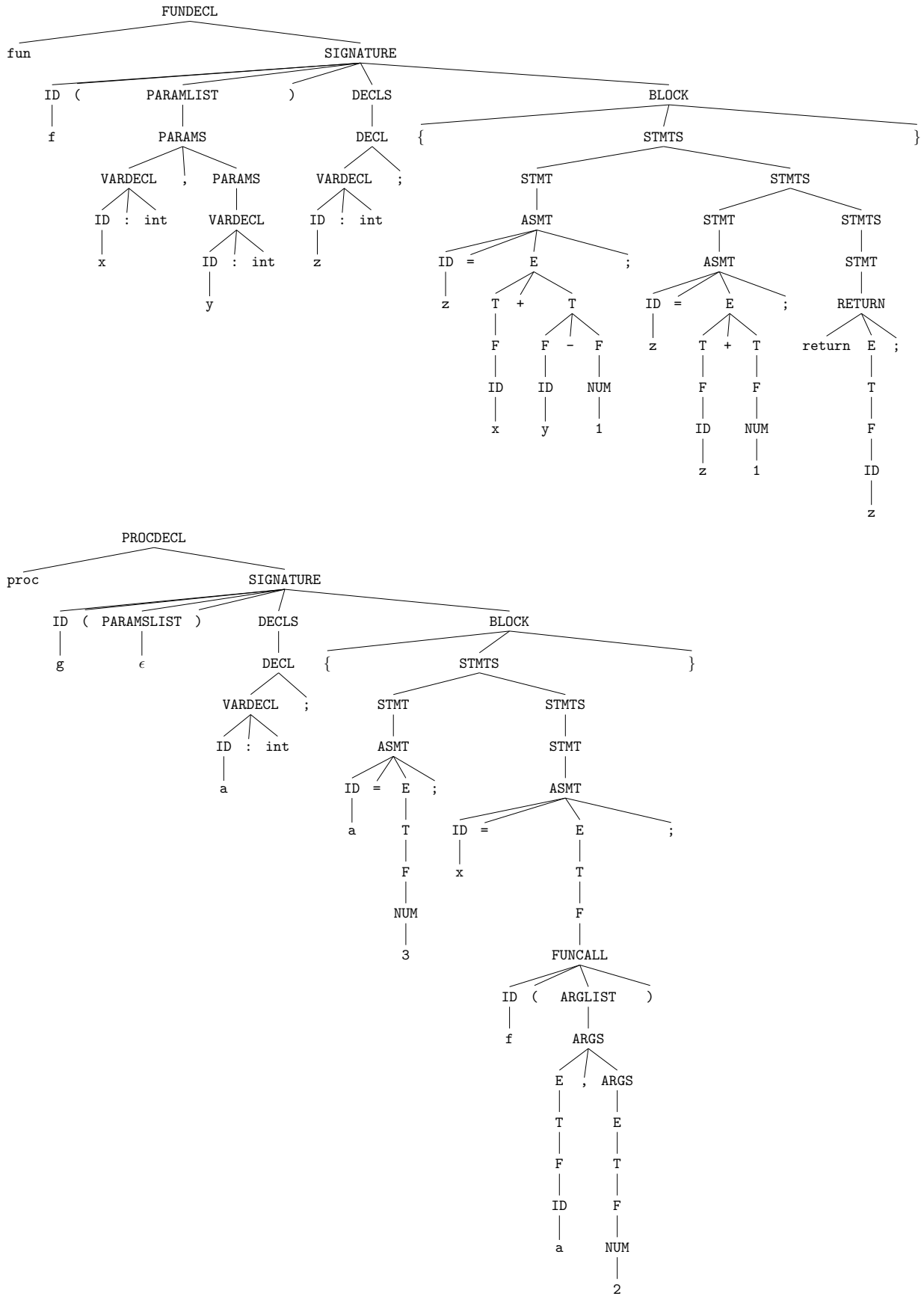
   (c) `[a-zA-Z]([a-zA-Z0-9_]|ϵ)([a-zA-Z0-9_]|ϵ) ([a-zA-Z0-9_]|ϵ)([a-zA-Z0-9_]|ϵ) ([a-zA-Z0-9_]|ϵ)([a-zA-Z0-9_]|ϵ)`

2. (a)
```
PROG → DECLS
DECLS → DECL DECLS | DECL
DECL → VARDECL ; | FUNDECL | PROCDECL

VARDECL → ID : int
FUNDECL → fun SIGNATURE
PROCDECL → proc SIGNATURE
SIGNATURE → ID ( PARAMLIST ) DECLS BLOCK

PARAMLIST → PARAMS | ϵ
PARAMS → VARDECL | VARDECL , PARAMS
BLOCK → { STMTS }

STMTS → STMT STMTS | STMT
STMT → ASMT | RETURN
ASMT → ID = E ;
RETURN → return E ;

E → T | T + T
T → F | F - F
F → ID | NUM | FUNCALL

FUNCALL → ID ( ARGLIST )
ARGLIST → ARGS | ϵ
ARGS → E | E, ARGS
```

   (b)



(see next page for detailed FUNDECL and PROCDECL)



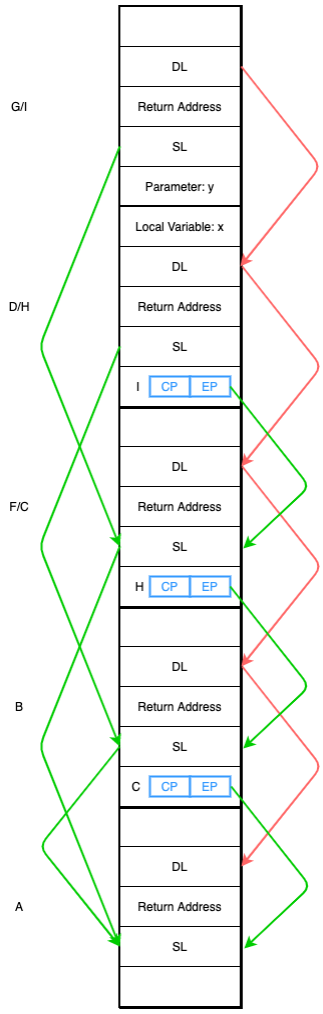
3. (a)
   - *static scoping*: the scope of a variable is evaluated in the environment of where it's defined.
   - *dynamic scoping*: the scope of a variable is evaluated in the environment of where it's called.

(b)
```
void A() {
  int x;   // static scoping will update this x

  void B() {
    x = 1;
  }

  void C() {
    int x; // dynamic scoping will update this x
    B();
  }

  C();
}
```

(c)
- Compile-time: to find the declaration of a variable in a statically scoped language, if we can find the declaration locally, then we use it; otherwise, it is a non-local variable, so we have to look at the scope in the same level of the definition of the function/procedure.
- Run time: we can find the reference of a variable by traversing the static link for pre-computed jumps in languages that support nested procedures.

(d) To find the declaration of a variable in a dynamically scoped language, if we can find the declaration locally, then we use it; otherwise, is a non-local variable so we have to look at the scope in the same level of the calling of the function/procedure by traversing the dynamic link and checking each stack frame until we find the variable.

4.



5. (a) `2 4 6 8 10`

   (b) `2 11 6 8 10`

   (c) `2 7 6 8 10`

   (d) `2 4 6 8 11`

6. (a)
```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Print_Numbers is
  package Int_IO is new Ada.Text_IO.Integer_IO(Integer);
  use Int_IO;

  task T1 is
    entry Start;
  end T1;

  task T2 is
    entry Start;
  end T2;

  task body T1 is
  begin
    for I in 1..100 loop
      Put(I);
      if I mod 10 = 0 then
        T2.Start;
        accept Start;
      end if;
    end loop;
  end T1;

  task body T2 is
  begin
    for I in 201..300 loop
      if I mod 10 = 1 then
        accept Start;
      end if;
      Put(I);
      if I mod 10 = 0 then
        T1.Start;
      end if;
    end loop;
  end T2;

begin
  null;
end Print_Numbers;
```

   (b) No, the printing of the numbers is not occurring concurrently. The order in which numbers should be printed is already determined, while concurrency means that we don't know which event will occur first.