Homework 3 . Semantic Analysis

Submit your home work to blackboard by **11:59PM, Wed Mar 10**. Bonus: submission will be still acceptable until **11:59 AM Thur Mar 11** without any penalty. Your submission must be in PDF format. If your PDF file is from an image, make sure it is easily readable.

1. (5) a) the name(s) of people who have contributed to your solution of this homework, and b) their contribution (briefly). If you worked by yourself, the answer of this question would be "N.A."

   b) (5) Please make your writing for the homework easy to read. Acknowledge this.

2. (20) a) Give two examples of semantic rules (NOTE: semantic rules are NOT the assignment expressions used in the attribute grammar. See slides for what are semantic rules.). b) Why do we need semantic rules?

3. (40) Consider the following context free grammar for an arithmetic expression containing operation – only

   ```
   <expr> -> number <expr_tail>
   <expr_tail> -> - number <expr_tail> | ϵ
   ```

   (a) The following is an attribute grammar to define the value of any arithmetic expression specified by the grammar above.

   ```
   <expr> -> number <expr_tail>
          ▷ <expr>.value := <expr_tail>.value
          ▷ <expr_tail>.st := number.value
   <expr_tail> -> - number <expr_tail>
          ▷ <expr_tail>2.st := <expr_tail>1.st - number.value
          ▷ <expr_tail>1.value := <expr_tail>2.value
   <expr_tail> -> ϵ
          ▷ <expr_tail>.value := <expr_tail>.st
   ```

   Draw a decorated parse tree for `1-2-3-4` using the attribute grammar.

   (b) Instead of the value of an arithmetic expression, we would like to know the number of "numbers" in an arithmetic expression. For example, in `1-2-3-4`, we have four numbers.

   Write an attribute grammar to define the number of "numbers" of any arithmetic expression following the grammar in part a).

4. (30) Consider programs consisting of two parts

(a) The first part has only one function declaration: *functionName(parameterList)* where *functionName* is an identifier (i.e., a sequence of letters) and *parameterList* is of the form *type id1, type id2, ⋯ , type idn*, where *type* is **int** or **bool**, and *idi*'s are identifiers. An example of a function declaration is

```
f(int x, bool y)
```

(b) The second part has only a function call of the form *functionName(argumentlist)* where *argumentlist* is of the form *id1, id2, ..., idn* where *idi*'s are identifiers. For example,

```
f(x, z)
```

An example program is

```
f(int x, bool y)
f(x, z)
```

Questions.

(a) Write a context free grammar (CFG) for such programs.

(b) We would like to introduce two attributes – `name` and `number` – for the non-terminal symbol in your CFG that defines the function declaration. The former has a value of the function name and the latter means the number of arguments of the function. Similarly we introduce `name` and `number` for the non-terminal symbol in your CFG that defines the function call. The former takes, as its value, the name of the function and the latter takes, as its value, the number of arguments in the function call.

Write an attribute grammar for such programs using the given attributes (and auxiliary attributes you may need).

(c) Given a program

```
f(int x, bool y)
f(x)
```

Explain how the semantic rule of "The number of parameters of a function call has to have the same number of parameters as that of the declared function" can be checked using the decoration of the parse tree.