Homework 1

Problem 1

$\langle\text{lstring}\rangle ::= \langle s\rangle_1\langle s\rangle_2$ Cond: $\neg(\text{Start}(\langle s\rangle_1) = A) \land \neg(\text{End}(\langle s\rangle_2) = A) \land$
$(\text{End}(\langle s\rangle_1) = A) \rightarrow (\text{Start}(\langle s\rangle_2) = C) \land$
$(\text{Start}(\langle s\rangle_2) = A) \rightarrow (\text{End}(\langle s\rangle_1) = B)$

$\langle s\rangle ::=$ a
$\text{Start}(\langle s\rangle) \leftarrow A$
$\text{End}(\langle s\rangle) \leftarrow A$

| b
$\text{Start}(\langle s\rangle) \leftarrow B$
$\text{End}(\langle s\rangle) \leftarrow B$

| c
$\text{Start}(\langle s\rangle) \leftarrow C$
$\text{End}(\langle s\rangle) \leftarrow C$

| $a\langle s\rangle_1$
$\text{Start}(\langle s\rangle) \leftarrow A$
$\text{End}(\langle s\rangle) \leftarrow \text{End}(\langle s\rangle_1)$
Cond: $\text{Start}(\langle s\rangle_1) = C$

| $b\langle s\rangle_1$
$\text{Start}(\langle s\rangle) \leftarrow B$
$\text{End}(\langle s\rangle) \leftarrow \text{End}(\langle s\rangle_1)$

| $c\langle s\rangle_1$
$\text{Start}(\langle s\rangle) \leftarrow C$
$\text{End}(\langle s\rangle) \leftarrow \text{End}(\langle s\rangle_1)$
Cond: $\neg\text{Start}(\langle s\rangle_1) = A$

Problem 2

$\langle\text{exp}\rangle ::= \langle\text{simple}\rangle$
$\text{PlusOp}(\langle\text{exp}\rangle) \leftarrow F$

| $\langle\text{exp}\rangle_1 + \langle\text{exp}\rangle_2$
$\text{PlusOp}(\langle\text{exp}\rangle) \leftarrow T$

| $\langle\text{exp}\rangle_1 * \langle\text{exp}\rangle_2$
$\text{PlusOp}(\langle\text{exp}\rangle) \leftarrow F$
Cond: $\neg\text{PlusOp}(\langle\text{exp}\rangle_1) \land$
$\neg\text{PlusOp}(\langle\text{exp}\rangle_2)$

$\langle\text{simple}\rangle ::= \langle\text{number}\rangle \mid \langle\text{variable}\rangle$

*Note: F = False, T = True

18

Problem 3

```
<prog> ::= <decsNStmts>
           Decs(<decsNStmts>) ← LIFOStack()

<decsNStmts> ::= <decl>      | <stmt>
                               Decs(<stmt>) ← Decs(<decsNStmts>)
                 | <decl> <decsNStmts>_1
                 Decs(<decsNStmts>_1) ← Push(Var(<decl>),Decs(<decsNStmts>))

                 | <stmt> <decsNStmts>_1
                 Decs(<stmt>) ← Decs(<decsNStmts>)
                 Decs(<decsNStmts>_1) ← Decs(<decsNStmts>)

<decl> ::= int <id>;
           Var(<decl>) ← (Name(<id>), int)

           | bool <id>;
           Var(<decl>) ← (Name(<id>), bool)

<stmt> ::= <assign>
           Decs(<assign>) ← Decs(<stmt>)

           | if <id> then <decsNStmts>_1 else <decsNStmts>_2 end;
           Decs(<decsNStmts>_1) ← Decs(<stmt>)
           Decs(<decsNStmts>_2) ← Decs(<stmt>)
           Cond: Declared(Name(<id>),Decs(<stmt>))
           Cond: LastType(Name(<id>),Decs(<stmt>)) = bool

           | while <id> do <decsNStmts> end;
           Decs(<decsNStmts>) ← Decs(<stmt>)
           Cond: Declared(Name(<id>),Decs(<stmt>))
           Cond: LastType(Name(<id>),Decs(<stmt>)) = bool

<assign> ::= <id> = <boolExp>;
             Cond: Declared(Name(<id>),Decs(<assign>))
             Cond: LastType(Name(<id>),Decs(<assign>)) = bool

             | <id> = <intExp>;
             Cond: Declared(Name(<id>),Decs(<assign>))
             Cond: LastType(Name(<id>),Decs(<assign>)) = int
```

EXPLANATION:

The attribute 'Decs' is a list of variables declared up until a particular point in the program. 'Decs' is represented by a LIFO Stack and is initialized to empty in the definition of <prog>.

The attribute 'Var' represents a declared variable. It is a tuple of a name and a type and is created in the definition of <decl>.

Push(var,decs) takes a variable and a stack of declared variables, and it produces a copy of the stack with the variable inserted at the beginning. Since Declared and LastType will iterate over the Stack from most recently added to least recently added, no removal or overwriting is necessary.

Declared(name,decs) checks whether a variable has been declared by searching for 'name' in 'decs'. It iterates through 'decs', starting at the front (most recently added), and checks the first element of each tuple in 'decs'. If it finds a match with 'name', it returns True (else False).

LastType(name,decs) returns the type of the most recent declaration of a variable in 'decs'. It iterates through 'decs', starting at the front (most recently added), and checks the first element of each tuple in 'decs'. When it finds a match with 'name', it returns the second element of the tuple.

When a variable X is used as the <id> of an if-statement or while-statement, there is a condition that it should have been declared previously and a condition that the most recent declaration should be a <bool> (see <stmt> alternatives #2 and #3), and there are similar conditions for assignment (<assign>) of ints and bools.

This grammar ensures that if a variable X is declared at any point in a program, X is accessible to statements which follow the declaration, but not those that precede it. 'Decs' is built from the top down, passed down to all branches of the tree that need to access it, and when 'Decs' is added to (in a declaration), it can only be passed to statements following the declaration (see <decsNStmts> alternative #3).

Declarations inside if-statements and loops are not accessible outside because 'Decs' is only passed down in the tree and never passed up outside of those portions.