Homework 2

Problem 1

```
<assign>    ::=   <id> := <ae>;
                  Code(<assign>) ← append(Code(<ae>), (“POP” Name(<id>)))

<ae>        ::=   <int>
                  Code(<ae>) ← <(“PUSH” Value(<int>))>
                  | <id>
                  Code(<ae>) ← <(“PUSH” Name(<id>))>
                  | <ae>_1 + <ae>_2
                  Code(<ae>) ← append(Code(<ae>_1), Code(<ae>_2), (“ADD”))
                  // Should be pretty clear what other arithmetic and
                  // boolean expressions would look like based on this
```

Problem 2

```
<prog>      ::=   <decl seq> <stmt>
                  SymTab(<stmt>) ← SymTab(<decl seq>)
                  // Symbol table is built up in <decl seq> and then
                  // passed down to be used in <stmt>

// Building symbol table

<decl seq> ::=    <decl>
                  SymTab(<decl seq>) ← Syms(<decl>)
                  | <decl> <decl seq>_1
                  SymTab(<decl seq>) ← union(Syms(<decl>),
                                             SymTab(<decl seq>_1))

<decl>      ::=   int <id list>;
                  Syms(<decl>) ← Syms(<id list>)

<id list>   ::=   <id>
                  Syms(<id list>) ← {(Name(<id>),nxtAddr())}
                  // Next memory address could be assigned similarly to
                  // how Labels are assigned (Labin,Labout) if we wanted
                  | <id>, <id list>_1
                  Syms(<id list>) ← union({(Name(<id>),nxtAddr())},
                                          Syms(<id list>_1))
```

```
// Using symbol table

// Assume that SymTab gets passed down to all non-terminals that
// come from <stmt>
<assign>   ::=   <id> := <ae>;
                 SymTab(<ae>) ← SymTab(<assign>)
                 Temp(<ae>) ← 1
                 Code(<assign>) ← append(Code(<ae>),
                                        ("STO" getAddr(Name(<id>),
                                                       SymTab(<assign>))))
                 // getAddr(name,symtab) searches symtab for tuple whose
                 // first element is name and returns the second element
                 // of that tuple

<ae>       ::=   <int>
                 Code(<ae>) ← <("LOAD" Value(<int>))>
                 | <id>
                 Code(<ae>) ← <("LOAD" getAddr(Name(<id>),
                                               SymTab(<ae>))))>
                 // Would need to somehow make it clear what's a literal
                 // and what is a memory address
                 | <ae>_1 + <ae>_2
                 Code(<ae>) ← append(Code(<ae>_1),("STO" temp(Temp(<ae>))),
                                   Code(<ae>_2),("ADD" temp(Temp(<ae>))))
                 Temp(<ae>_1) ← Temp(<ae>)
                 Temp(<ae>_2) ← Temp(<ae>) + 1
```

Problem 3

```
a.   (4 . NIL)                        =    (4)

b.   ((3 . NIL) . (4 . NIL))          =    ((3) 4)

c.   (3 . ((4 . NIL) . (5 . NIL)))    =    (3 (4) 5)

d.   (3 . (4 . 5))                         (3 4 ERROR
                                           '5' is not a binary tree
                                           nor the atom NIL
```