Compiler Class Reference

**AST node reference**
Assume AST node subclasses have been defined as in class and the on-line readings (and as provided on the programming assignments) and have the following methods (as appropriate to their subclass):

```java
void nameAnalysis(SymTable symTab)
Type typeCheck()
void codeGen()
void genJumpCode(String trueLabel, String falseLabel)
```

Selected subclasses of the ASTNode class and their fields

**DotAccessNode**
- ExpNode myLoc // LHS of LHS.RHS
- IdNode myId // RHS of LHS.RHS
- Sym mySym // if LHS is a struct type, this is a link to LHS's sym

**FnDeclNode**
- TypeNode myType
- IdNode myId
- FormalsListNode myFormalsList
- FnBodyNode myBody

**IdNode**
- String myStrVal
- Sym mySym

**StructDeclNode**
- IdNode myId
- DeclListNode myDeclList

**StructNode** (extends TypeNode)
- IdNode myId

**TypeNode**
- Type myType

**VarDeclNode**
- TypeNode myType
- IdNode myId
- int mySize

**Symbols**
Methods of the Sym class

```java
String getName()
Type getType()
void setType(Type t)
int getOffset()
void setOffset(int offset)
boolean isGlobal() // true if offset == 1
```

Selected subclasses of the Sym class with additional methods

**StructSym class** (for ID corresponding to variables declared to be of a struct type)
- IdNode getStructType()

**StructDefSym class** (for ID corresponding to the name of a struct type)
- SymTable getSymTable()
Types

Methods of the Type class

- boolean isErrorType()
- boolean isIntType()
- boolean isBoolType()
- boolean isVoidType()
- boolean isStringType()
- boolean isFnType()
- boolean isStructType() // variable declared of a struct type
- boolean isStructDefType() // name of struct definition (declaration)

Subclasses of Type:
- ErrorType
- IntType
- BoolType
- VoidType
- FnType
- StructType
- StructDefType

Error Message Generation

Assume that you have an error method that takes one String argument (representing the error message to display). For example:

```java
error("invalid type");
```

Note: you can call the error method directly; you do not need to worry about line or character numbers.

Code Generation

Assume that you have the auxiliary methods for code generation (you can just call them directly, i.e., you don’t need to put Codegen in front of them):

- generate – write the given op code and arguments, nicely formatted, to the output file
- generateIndexed – the arguments are: an op code, a register R1, another register R2, and an offset; generate code of the form: op R1, offset(R2)
- genPush – generate code to push the value of the given register onto the stack
- genPop – generate code to pop the top-of-stack value into the given register
- nextLabel – return a string to be used as a label
- genLabel – given a label L, generate: L:

and the register constants: SP, FP, RA, V0, A0, T0, T1 as well as the boolean constants TRUE and FALSE.