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):

```
     void nameAnalysis(SymTable symTab)
     Type typeCheck()
     void codeGen()
```

Selected subclasses of the `ASTNode` class and their fields

```
   TupleAccessNode
      ExpNode myLoc // LHS of LHS.RHS
      IdNode myId   // RHS of LHS.RHS
      Sym mySym     // if LHS is a tuple type, this is a link to LHS's sym

   FctnDeclNode
      TypeNode myType
      IdNode myId
      FormalsListNode myFormalsList
      FctnBodyNode myBody

   IdNode
      String myStrVal
      Sym mySym

   TupleDeclNode
      IdNode myId
      DeclListNode myDeclList

   TupleNode (extends TypeNode)
      IdNode myId

   TypeNode
      Type myType

   VarDeclNode
      TypeNode myType
      IdNode myId
      int mySize
```

Symbols

Methods of the `Sym` class

```
     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

`TupleSym` class (for ID corresponding to *variables* declared to be of a `tuple` type)

```
      IdNode getTupleType()
```

`TupleDefSym` class (for ID corresponding to the *name* of a `tuple` type)

```
      SymTable getSymTable()
```

Types

Methods of the Type class

```
      boolean isErrorType()
      boolean isIntegerType()
      boolean isLogicalType()
      boolean isVoidType()
      boolean isStringType()
      boolean isFctnType()
      boolean isTupleType()     // variable declared of a tuple type
      boolean isTupleDefType()  // name of tuple definition (declaration)
```

Subclasses of Type:

```
      ErrorType
      IntegerType
      LogicalType
      VoidType
      FctnType
      TupleType
      TupleDefType
```

Error Message Generation

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

```
      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 logical constants TRUE and FALSE.