

Introduction

In Project 1, you wrote a scanner for the mad squirrel battle language. In this project, we will create a symbol table to store the variables and constants, and write a parser for constructing a parse tree from a language file.

Here again is the BNF for the grammar:

statements := statements statement | statement

statement := declaration | whileLoop | ifStatement | eatStatement | pushStatement | intAssign
| treeAssign | dirAssign

declaration := type varName ";"

type := INT | TREE | DIRECTION

VarName := [A-Za-z][A-Za-z0-9]*

whileLoop := WHILE boolExpr "{" statements "}"

ifStatement := IF boolExpr "{" statements "}" ELSE "{" statements "}"

eatStatement := EAT intExpr ";"

pushStatement := PUSH dirExpr ";"

intAssign := VarName "=" intExpr ";"

treeAssign := VarName "=" treeExpr ";"

dirAssign := VarName "=" dirExpr ";"

boolExpr := "(" boolExpr ")" | boolExpr "&&" boolExpr | boolExpr "||" boolExpr | intExpr "<" intExpr
| intExpr ">" intExpr | intExpr "==" intExpr | dirExpr "==" dirExpr | treeExpr "==" treeExpr

intExpr := "(" intExpr ")" | [0-9]+ | VarName | treeExpr "[X]" | treeExpr "[Y]" | intExpr "+" intExpr
| intExpr "-" intExpr | intExpr "*" intExpr | intExpr "/" intExpr | intExpr "%" intExpr | NUTS treeExpr

treeExpr := "(" treeExpr ")" | "[" intExpr "," intExpr "]" | VarName | TREE dirExpr | NULL

dirExpr := "(" dirExpr ")" | NORTH | SOUTH | WEST | EAST | HERE | VarName

Your Mission

A. Fix any remaining problems from Project 1 **(10 pts)**

B. Modify your scanner to add entries to a symbol table **(40 pts)**.

Your symbol table should keep track of four things:

Symbol id: a unique integer identifying each entry in the table.

Variable Name: a string storing the name of the variable, if any.

For constants, the name "constant" is fine.

Variable Type : Either "Tree", "Direction", or "Integer"

Variable Value : The current value of the variable. For constants, this should be set to the value of the constant. For variables, initialize this to "0" (null). When we do code generation, this will store the "relative address" of the variable in memory. It is probably best to use a C++ **string** for this field for now.

Example:

If your scanner encounters:

```
TREE curTree;
```

The symbol table entry would look like this:

```
1      curTree  TREE  0
```

If it then encountered:

```
curTree = [0,2];
```

The table would now look like this:

```
1      curTree      TREE  "0"  
2      constant    TREE  "0,2"
```

- C. Use Bison to create a parser for MSBPL files. You should use your scanner from Project 1 to provide the "yylex" function. **(40 pts)**

Your parser should:

1. Construct a parse tree using C++ objects and pointers **(20 pts)**.
 2. Print out an in-order traversal of the parse tree **(20 pts)**.
- D. Create a main function (either in your bison file or in a separate C file) that accepts input from the standard input (cin) and parses it, sending output to the standard output (cout). **(5 pts)**

- E. Glitter **(5 pts)**

Glitter points are assigned for extending the project assignment in a creative way.

Some suggestions for glitter are:

1. Extend the language by adding new keywords.
2. Use a C++ union for the "Variable Value" column of the Symbol Table.
3. Write a Makefile for compiling the project more easily.
4. Add internal documentation
5. Add external documentation
6. Your own, original, idea.

Coming up with your own creative glitter idea is worth more than simply selecting one of these alternatives (except for type checking, obviously), as long as your idea makes a significant and creative contribution to how the project works, looks, or behaves.

As usual, submission is through the course interface at <http://narnia.homeunix.com/~robert/submit>