

CSE 6341, Programming project 1

Due Monday, September 14, 11:59 pm (10 points)

The goal of this project is to implement a type checker for a very simple language. The checking is based on material discussed in class (slides 45-63) but with a few modifications to the underlying context-free grammar. Your implementation will use the supplied code for parsing and AST building (AST = abstract syntax tree; a more compact version of the parse tree).

Step 1: Set up

Let's say you have created `/home/buckeye.8/6341` for this project and have built project 0 in it

```
cd /home/buckeye.8/6341/proj
```

Remove the old setup, as a few things have changed from project 0 to project 1

```
rm -r p1
wget web.cse.ohio-state.edu/~rountev.1/6341/project/p1.tar.gz
tar -xvzf p1.tar.gz
cd p1
make
./plan t1
```

Step 2: Understand the AST

The context-free grammar for the targeted language is as follows:

```
<program> ::= <unitList>
<unitList> ::= <unit><unitList> | <unit>
<unit> ::= <decl> | <stmt>
<decl> ::= <varDecl> ; | <varDecl> = <expr> ;
<varDecl> ::= int ident | float ident
<stmt> ::= ident = <expr> ; | print <expr> ;
<expr> ::= intconst | floatconst | ident | <expr> + <expr> | ( <expr> )
```

Terminal symbols are shown in blue. For example, **intconst** is a single terminal symbol representing an integer constant. If you want to see the details of how the terminal symbols are defined, look at `p1/parser/Scanner.jflex`. If you want to see the details of how the parser is defined, look at `p1/parser/Parser.cup`. You do not need to understand any of these details to complete the project successfully.

Read the code in `p1/ast` to see how the AST nodes are defined. The root of the AST is a node that is an instance of the `Program` class. Also see `p1/interpreter` to see the entry point of the whole project. It is important to do this reading early and to ask any clarification questions as soon as possible.

Step 3: Implement type checking

You need to implement a type checker to check for the following conditions:

- 1) Any variable appearing in an `<expr>` must have a declaration in some earlier `<decl>`. For example, `int x = 1; int y = x + w;` is not allowed because `w` is not declared. As another example, `int x = x+1;` is also not allowed (the `x` in `x+1` is not declared).
- 2) Each variable can be declared only once. For example, `int x; int y = 1; int x = y+1;` is not allowed.
- 3) In an assignment `ident = <expr>;` the variable on the left-hand side of the assignment must be already declared. For example, `int x; y = x+1;` is not allowed.
- 4) In an assignment `ident = <expr>;` or a declaration with initialization `<varDecl> = <expr>;` the type of the variable on the left-hand side of `=` must be the same as the type of the expression on the right-hand side. For example, `int x; float y = 1.; x = 3.14 + y;` is not allowed; neither is `int x = 3.14;`
- 5) Both operands of `+` must be of the same type. For example, `int x = 1; float y = 3.14; float z = y + 1.1; int w = x + z;` is not allowed because of `x + z`.

If the program violates any of these checks, call `Interpreter.fatalError` with exit code `EXIT_STATIC_CHECKING_ERROR`. The test script will check this exit code, so please make sure your implementation uses it. The text message associated with the error should be something simple that describes which specific check was violated.

Suggestions for your implementation (but feel free to ignore these completely):

- 1) A natural way to implement the checking is to add to each class in package `ast` a method `check` which takes as input a reference (i.e., a Java pointer) to a global table. The table maps identifiers to types. This is the “single global table” from slide 58.
- 2) To deal with checking violations, method `check` could throw some kind of exception. Class `Interpreter` can catch this exception and call `fatalError` as needed. See the handling of parse errors inside `Interpreter` as an example of this style of coding.
- 3) Feel free to add fields in classes from package `ast` to store the values of relevant attributes (e.g., attribute ‘type’)

Step 4: Testing

Write many test cases and test your checker with them. Submit at least 5 test cases with your submission. The test cases will not affect your score, but I will examine them to see how you approached testing. Put them in the same location as the provided file `t1` and name them `t2`, ...

Step 5: Submission

After completing your project, do

```
cd p1
make clean
cd ..
tar -cvzf p1.tar.gz p1
```

Then submit `p1.tar.gz` in Carmen.

General rules (copied from the course syllabus)

- 1) Your submissions must be submitted electronically via Carmen by midnight on the due date. The projects must compile and run on **stdlinux**. Some students prefer to implement the projects on a different machine, and then port them to stdlinux. If you decide to use a different machine, it is entirely your responsibility to make the code compile and run correctly on stdlinux before the deadline. In the past many students have tried to port to stdlinux too close to the deadline, leading to last-minute problems and missed deadlines.
- 2) Projects should be done independently. General high-level discussion of projects with other students in the class is allowed, but you must do all design, programming, testing, and debugging independently. Projects that show excessive similarities will be taken as evidence of cheating and dealt with accordingly. Code plagiarism tools may be used to detect cheating. See more details in the Syllabus under “Academic integrity”:
<http://web.cse.ohio-state.edu/~rountev.1/6341/pdf/CSE6341-Syllabus-Au20.pdf>
- 3) The projects are due by 11:59 pm on the due day. No exceptions will be made to this deadline: if you submit at 12:00 am, your submission will be late. Please plan your time carefully and do not submit in the last minute. You can submit up to 24 hours after the deadline; if you do so, your project score will be reduced by 10%. If you submit more than 24 hours after the deadline, the submission will not be accepted, and you will receive zero points for this project.