GROUP 15 - CHECKPOINT 2 DOCUMENTATION

Sam Engel, Affan Khan, Wisam Dayoub

CHECKPOINT OVERVIEW

For this checkpoint, we added to the parser-scanner we built from the last checkpoint. This time around, we added symbol table construction and display functionality as well as type checking to the project. Our program can now detect/report syntactic errors with the parser-scanner and can now detect and report semantic errors such as mismatched types and redefined/undefined variables. To accomplish this, we added a new visitor class *SemanticAnalyzer.java* handling all the functionality from creating the symbol table to performing type checking throughout.

DESIGN PROCESS

Following the tips provided in Lecture 8 for the implementation, we followed this design process:

Sam Engel's contributions

1. Created new classes: SemanticAnalyzer and NodeType. Implemented symbol table insertions at Dec nodes, implemented the displaying of and the deletion of symbols as the tree leaves each scope.

2. Implemented support for “-a” and “-s” command line arguments for saving abstract syntax trees and symbol tables to file respectively.
3. For cspec.cup, set boolean *valid* variable to false whenever the error reporting function is called. When this is false, syntax trees and symbol tables will neither be displayed or saved; only the errors detected by the parser will appear in the console.

Affan Khan’s contributions

4. Added in functionality that checks if the return value within a function matches the return type of a function. If a function: *int foo()* returns a variable of anything other than type int, it will throw an error to STDOUT and terminate the program providing the user at which line the error occurs.
5. Added in functionality for undefined/redefined variables error handling. Produces an error to STDOUT providing the line number the error occurs before terminating the program. Ensured that the redefined variables only throw errors in the same scope or scopes before it encircling the latest scope. Example. Global variables won’t be repeated in a scope within a function, but the same variable can be repeated in two different functions.
6. Added functionality for Type checking the test conditions for if and while statements. Made sure the test conditions within were either bool or int, throwing an error to STDOUT otherwise, producing the line number and terminating the program.
7. Further testing and debugging and documentation updates.

Wisam Dayoub’s contributions

8. Created functionality for type checking that the array index must be int. made sure that a function cannot be inside an array if it's not compatible. Made sure it throws an error when it does.
9. Created functionality for type checking two sides of an assignment. Therefore if the program encounters an assignment that isn't compatible it would throw an error indicating which line it is at.
10. Create functionality for type checking operands of binary/unary operations to make sure that stuff like y = x + foo() throws an error since the types are incompatible.
11. Created the [1-5].cm test files to test for various errors and indicate where they are happening in the file.

Further contributions from Sam Engel

12. Fixed errors and oversights in type checking, scanner, and parser (accounted for function return types in CallExp checks within WhileExp and IfExp nodes, expanded getTypeOfExpression to check for IntExps, BoolExps, and OpExps, changed comment regex, modified actions for signed factors in parser, made TRUTH a string that would be converted into a boolean in parser, etc.)
13. Added column information to error messages and wrote in any missing exit commands afterward.

ASSUMPTIONS

- For redefined variables, variables in the scope outside of it count as variables that must not be redefined. For example, if there is a variable globally declared, the variable can’t be redefined anywhere else.
- Type checking errors thrown result in program terminating and the .sym file not being created. In other words, up to one semantic error will be reported before termination – unlike the parser which will try to report as many errors as possible.
- getSimpleDecType does not check for level as it is assumed the variable for the current scope will be the first name match in the key arraylist.
- It is assumed that boolean variables cannot be added, subtracted, divided, or multiplied, but comparative operators between them are valid.
- The level variable is used in SemanticAnalyzer to differentiate between scopes.
- OpExps are only valid as assignments if the operands are all integers

LIMITATIONS

- Currently, the “-a” and “-s” flags are mutually exclusive. When running the program, you have to choose one or the other.
- The deletion method for the symbol table will currently delete by a certain key at a certain level, not a specific node. The deleteContents method uses this to clear out all keys at a certain level when leaving a scope.

- Since error checking in the parser has its limitations, sometimes an invalid syntax tree/symbol table might be displayed and saved to file in circumstances not accounted for.
- Type checking does not account for case scenario if a function's return value is not void and a variable is not returned. It continues the program and does not see this as an error.
- Limitation: fully nested comments like /* /* comment */ */ are not supported, but – following the example from C proper, comments like /* /* comment */ are supported: there can be infinite openings, but there can only be one closing.