March 18th 2024

Megan Tibbles

1091728

CIS4650 Checkpoint 2 Report

Work Completed:

This assignment was a continuation of checkpoint 1, building off of the C minus parser. In checkpoint 1, my program reads a C minus file as input and outputs an abstract syntax tree. For checkpoint 2, my program reads a C minus file as input but outputs a symbol table as output. You can choose whether you want both to print out or for it to be written to a file. The symbol table shows all the declarations made in the file, this includes variable and function declarations. Type checking is also implemented. The program sees if the data types match for assignments, returns, arguments, operations, etc. If they do not match, an error will be printed and sometimes an action will be taken to correct it.

Implementation:

My first step was to understand the purpose of checkpoint 2. I read the assignment description, the marking scheme, and the notes from lecture 8. Using the information and tips provided, I decided to illustrate what I thought the symbol table should look like. I did this on paper, constructing a hashmap and drawing arrows to make connections, similar to what is shown in the lecture notes. Once I had a good grasp of what I wanted to do, I started implementing the symbol table. I wrote the insert, delete, and lookup functions. The first file I tried it on contained a simpleDec such as “int x;”. This helped me figure out how to insert an element into the table and how to display it. Gradually, I added more and more declarations such as arrayDec and FunctionDec. Next, I worked on the different scopes. I

used the level to indicate a change in scope, so anytime a new block is entered, the level increases and then decreases when the block is left. Different blocks can be functions, if and else statements, while loops, etc. Next, I added error checking for undefined and redefined variables.

Eventually I started working on the type checking. I found this to be challenging because the dec dtype did not make sense to me. Once I read the discussion boards and asked the TA, I had a better understanding of what to do and was able to start on the error checking. This was a long process because there are so many cases to consider such as variable assignments, function return types, function parameters, operations, if conditions, etc. As I was doing this I would create test files to try out the specific error I was trying to check.

Once I was pleased with my error checking, I added the odds and ends. I added the command line arguments, added the check for the main function and the input and output functions. Throughout the whole process I added many helper functions as well.

My last step was to create the testing files 1-5.cm in the symTest folder. The files should have the following output:

- 1.cm : no errors. Shows many different structures the program can handle. It shows nested scopes. Shows a function prototype before a function declaration. Shows an operation and shows a function call.
- 2.cm: 3 errors. Redefines a variable that has already been declared, it does not add it again. Next, the code tries to assign a value to a variable that has not been defined so a declaration is created for it. There is also no main function which causes an error. You can see that these actions have taken place by examining the symbol table. The redefined variable is only included once and the undefined variable has been added.
- 3.cm: 2 errors. Shows a variable declared as void, this is not allowed so it displays an error and redeclares it as an int. The next error is in the return statement. The value

being returned is an int but the function expects a bool to be returned. An error message is displayed.

- 4.cm: 2 errors: The first error is in the if condition, the condition should be Boolean but it is not. The second error is an index error. Indices should be integers but in this case it is a boolean, this causes an error.
- 5.cm: 5 errors: The first error is a type mismatch for the arguments passed to a function. The argument types do not match the function declaration. The second error is a type mismatch for a variable, it has been declared a bool but is then assigned an int value. The variable is redeclared as an int. The third error is a type mismatch in an operation, the program attempts to add a boolean and an int, an error message is printed. The fourth error is in the same line because the invalid operation was being assigned to the int variable. This causes a type mismatch error. The main is also not the last function in the file so a message is displayed.

Lessons learned:

Checkpoint 1 taught me to take time in the beginning stages of the project to clearly understand what to do. I applied that lesson to checkpoint 2, I read all the documentation available and drew out a plan before starting. I think this helped a lot, even though I did run into confusion at one point, I was able to move more smoothly throughout the implementation process. One thing that I could change and that I will try to implement next time is to organize my code and helper functions better. I created a lot of helper functions for this assignment because there are a lot of actions that you are doing multiple times such as checking types, retrieving an element from an arraylist or hashmap or nodetype, displaying structures, etc. By the end of the assignment, I ended up with a bunch of functions that are not the most efficient, there are definitely some that could be combined into one. If I needed a

function that worked slightly differently than another, I would just add new ones instead of refactoring the old ones. Now I am left with inefficient code and not a lot of time to fix it up. Therefore, the lesson I learned at this checkpoint was to refactor the code as I am working on it. Or to leave more time at the end to refactor.

Assumptions/Improvements:

One assumption I made with the command line arguments was that if no command line argument is used, both the abstract syntax tree and the symbol table are displayed to the terminal. I was not sure if I should have only displayed the symbol table. When -a is used, the AST is output to a file and no symbol table is generated. When -s is used, the AST is not created and the symbol table is output to a file. Therefore, you cannot use both -a and -s at the same time. This is because I redirect the output to a file at the beginning of the program when reading the arguments, if both -a and -s are specified, the file will not be correct.

I could improve the extensiveness of my error checking for the next checkpoint. Although I think my error checking is thorough now, there are most likely cases I will catch in the future that I have not thought of now. These will come up while working on the next checkpoint because it is code generation. If there is a mistake in the code now, it will be revealed during that stage.

Conclusions:

Overall, this checkpoint went smoother than checkpoint 1. It still took a while but I spent less time stuck, wondering how to move forward. As I mentioned in the last report, I am doing this project alone which is a lot of work but allows me to understand all sides of this assignment. For the next checkpoint, I will try to apply what I learned this time and refactor as I work to keep the code more organized.