Implementation Project – Checkpoint Two

March 21, 2018

Jeremie Fraeys - 0892019 and Joel Klemens – 0895223

Review of implementation process:

To start off the implementation for checkpoint two we needed to go back and make some updates to what we had written in part one. These changes had to do with updating the way our syntax tree would handle instances of arrays. This entailed both array definitions as well as arrays being part of function parameters. The original way we had this implemented we would display an array expression such as:

X = a[i];

In the syntax tree:

Op: =

Id: x

Id: i

However, now we display:

Op: =

Id: x

Id: a

Id: i

This gives a better and more clear description of the expression with an array. We also changed how we display an array declaration, example:

Int a[10];

Original output:

Type: int

Id: a

Updated:

Array declaration: a, size: [10]

Type: int

To do this we implemented a new expression type called array to individually track the details of each array declaration, the name of the array, the type it was declared as and the size of that array are all stored. From checkpoint one we also had to change the way that we were reporting our errors, for checkpoint two we added better recovery on syntax errors so that the program would not stop when one is encountered. We also had one problem where depending on the operating system that a test file would be created in it could have different symbols used for spacing and for end of line / end of file representation. This would cause problems and display that an unknown symbol had been found. We removed this so that the problem is no longer an issue.

For this checkpoint we had to first implement a symbol table. To do this we started with looking at the examples that were provided in the lectures notes. In lecture "8 – Type checking" we were taught to understand that we were going to need to use a hash table, we started with creating a "symtable.c" and a "symtable.h" for the symbol table. We used the function that was outlined in the lecture to get an "add and fold" hashing function. We were also able to use an example of the "tiny compiler" from the text book to help us with the actual implementation and outline of how the symbol table would work. Using this we implemented the two new data structures in our global.h file. The line list structure and the bucket list structure. Together these nodes can be linked together to hold the line numbers as well as the name and memory locations of the variables needed. However, we not only needed to store

the name, location and line numbers but we also would need to keep track of the parent function name for the scope as well as the type of the variable. Do implement this feature we added a third data type structure called Scope, this is where we would save the hash table for that specific function as well as the name, what level of nesting it was at and the parent function name if it had one. Using the example provided in the text book it was easy to add the printing function with a secondary for loop that would handle the printing of the function scope information.

The final part of this assignment was to implement type checking, after we finally got the symbol table to work the type checking proved to not be extremely difficult. As outlined in the textbook we used a function called “typeCheck”, this function takes the syntax tree that is built from checkpoint one and then uses a recursive function called traverse to apply a preprocess and a post process to the “TreeNode” element. The preprocessor used to build the symbol table and looks for undeclared elements and elements that are declared multiple times. If there are no errors, the elements are then pushed onto the scope stack. The next step is to cover the semantic analysis. To do this we use a function called “checkNode”, this function accepts a tree node and then first looks for component nodes which are then popped off of the stack. Next it checks for while nodes to ensure that they have the correct typing. The return case will check to make sure that functions have proper return statements. Next it checks the expression to check for mismatched typing. Array nodes are checked to make sure that they have the correct type to define their size. Call nodes are checked for correct typing on parameter values and check to make sure they have the correct number of parameters. If any

errors are found in the process printed out to listing as defined in the textbook with the format "Symbol error on line: xx : error message".

What has been completed this checkpoint

During this check point the correction from check point one was finished. Also completed during this checkpoint was the implementation of the symbol table. The implementation of the symbol table included the use of a hash table and will output information about the type of variable, it shows entry and exit into different blocks showing the all of the scope levels for a given file. For the user to have the compiler output the symbol table they will need to user the command line flag "-s" which was a checkpoint requirement. This is in association with the implementation of type checking. Type checking will catch semantic errors such as mismatches, typing errors, void values in while loops, covers valid return statements and misuse of array indexing.

Possible Improvements:

When it comes to the implementation of the C minus compiler in this assignment there is always room to improve by adding more and more specific error checking functionality and recovery from more difficult situation. This would also include being able to pick up on other errors that would otherwise pass through the compiler without problems.

Individual Contributions:

For most of the major contributions of this checkpoint the team worked in person together implementing pair programming strategies to correct the implementation from checkpoint one and prepare for checkpoint two. Even though lots of pair programming was completed, for initial coding and set up of files for this checkpoint

Jeremie worked on the correction of problems from checkpoint one that are outlined in the review of the implementation process, this included addition of array handling and error recovery. Also, a majority of the type checking was completed by Jeremie.

Joel worked on the documentation for checkpoint two, as well as the implementation of the symbol table.