Checkpoint One Cis*4650 Compilers

Ryan Taylor, Jackson McAvoy

Design Process

To design the parser, scanner and syntax tree we followed an incremental design as suggested. To begin with we set up version control so we could keep track of each-other's submissions and contributions. Following this we adapted the warmup assignment to act as the scanner for out .cm files. We had success here, however we found that the .flex file used to generate the Lexer would not integrate with our .cup file using many of the commands we had. As a result we decided to cut out losses and start fresh using professor Song's code as a base. We adapted tiny.flex to the cminus language with cup integration in mind from the get go.

From here we translated the grammar for the cminus language into our cminus.cup file. We focused on getting the grammar to the point of compiling and running without errors when passed a test file before moving onto abstract syntax tree generation. To ensure both members of the group received adequate experience, we split the work based on experience and time available.

We then worked on generating an AST for our grammar. We prioritized having optimized objects so that it would be easier to use when coding in our .cup file, however during this process we experienced hardware problems with the group member that had the most free time to work on this portion; their computer would no longer boot up, so they had to spend the majority of a day getting things installed on an emergency laptop. After getting things working

again, we realized that we hadn't left enough time for a contingency plan, and so the AST functionality was not finished after hitting a programming roadblock.

If we were able to reliably generate ASTs we would have moved onto error testing, giving the programs various syntactical and lexical errors. We created 5 files at this point: the first is a perfect file, and files 2-4 have a few errors each of them. Each file represents a different type of error: 2 has mismatched braces, 3 has invalid type declaration, 4 has a line filled with random garbage, and 5 was not a c- file at all. In all cases the goal is a full parse, or at very least a graceful exit. Not all of this could be completed however as we ran short on time towards the end.

Limitations

The cminus parser has a couple known limitations. The most notable is the dangling else problem. When generating sym.java and parser.java using cminus.cup, an additional command of -expect 1 must be added to the line. This tells the generator to expect a single conflict. As such it will decide on shift for the dangling else.

We found ourselves with very little time approaching the deadline to figure out the problems in our AST generation, perform error checking and ensure graceful failures. What we did implement was a graceful, non-segfault core-dumping exit from failures. We had a hard time getting the error checker to go through the entire file when an error was placed in the c- file. It never crashed per-say, but it would terminate on slight errors.

Improvements

The first improvement we would make is solving the dangling else problem. It's well documented so with some small amount of testing we could very likely solve the problem.

Additional error checking would have been the next step, starting with checking for matching braces, parenthesis and ending lines with semicolons. As is, we lacked the time to do any error checking. The process we would go through is generating simple and identifiable errors in a known to be working .cm file, such as the gcd.cm or fac.cm given for testing.

Of course, it would have been optimal to have the AST generation working properly. In the future to avoid this problem, it would be helpful to plan to have the assignment done a week in advance so that we can have breathing room for fixing all of our errors, rather than having the project mostly done and assuming that the last bit is going to go smoothly.

Lessons

The largest lesson gained from this assignment was how to integrate with external jar files, as well as learning from online documentations, especially when working on a Windows platform. Both flex and cup are well documented, but lack tutorials and extensive question and answers on sites like stack overflow. This project also provided insights on the implementation process of grammars and the intricacies of abstract syntax trees. In previous classes we learned about these ideas in an abstract way, but never had to chance to implement them.

We also learned that we should get an earlier start on the assignments for the compilers course. Our current method of motivation is inadequate, and we'll have to fix it for checkpoint

two. Our plan for the next checkpoint is to complete an iterative step every few days. Ideally, we can finish early and spend the remainder of the time perfecting and error checking our code to make it as elegant as possible. This approach will also allow both members to gain further expertise in the learning goals given by the assignment, as well as help us with a contingency plan in the event that another unexpected setback occurs. Having the extra time for us to discuss, teach each other and plan for disaster will be invaluable.

Work Allocation (Contributions)

Task in %	Jackson	Ryan
Parser	40	60
Scanner	70	30
Error Checking	20	80
Documentation	90	10