Compiler Log-sheet:

For ELEC50010 Instruction Set Architectures and Compilers CW 2.

Contributors: Aryan Ghobadi, Ani Hazarika

Plan: The idea is to focus on the generalised stages of the Compiler process, starting from lexing, parsing and then code generation to be more general and then adding further detail with things such as type checking, symbol tables etc in order to create a realistic compiler system.

Timing: Work on this Compiler started on 21/02/2021 (coinciding with end of midterm exams and both team members completion of the labs).

Ideal Plan:

-1.5 week to implement the lexing and parsing process for the basic requirements

-1 week to implement the code generation process for the code-generation and general back-end MIPS work, while performing tests.

-1.5 weeks to add further complexity to our complete design and complete more tests.

One thing to note is that this is an ideal worst case timing. This way gives us the most time to focus and take time for each stage while still finishing with almost a full week to spare, giving us even more leniency, however one thing to note is that this is a worst case projection, and given both team members' experience in the labs with related compiler content, and by re-using resources and concepts from the aforementioned labs, this time could be cut shorter by many days, meaning that the finishing time could be even earlier than this, however it is better to give more leniency than set unrealistic expectations.

Phase 1: Lexing/Parsing:

21/02/2021: Midterms finished, labs finished and can now fully start with the project.

Aryan designed a plan for how we would implement it over time, Aryan has also started work on the lexer that we will need. Ani started work implementing the Parser with an AST file.

Week Beginning 22nd Feb: Week Objective:

> -Produce a lexer that could lex the basic requirement syntax from C, in accordance with the spec.
>
> -Implement a basic parsing process which could convert the tokens lexed into a 3-address-code representation.

23/02/2021: Ani added more to parser primitive hpp, as well as adding more considerations to lexer with regards to misc chars like semicolons etc, Aryan prepared makefile, testing executable and testcases to debug lexer.

Lexer v1 should be finished by this Thurs, will begin to focus on Parser.

25/02/2021: Lexer v1 finished, working on Parser now, test cases for lexer have been finished, makefile etc generated. The lexer is now able to lex the individual constructs and decipher the values. Later on we will implement a symbol table in order to do proper type checking, however this means we can move on to the parser.

26/02/2021: First operator (Add) was implemented on the parser, developed in the AST style. This implementation converts the read tokens and parsed symbols and converts it into a 3 address-code format. So z = x + y would be read in the intermediate ADD [Z X Y], although conversion to MIPS could've been more directly performed in the parser to code-gen phase, having a dedicated intermediate stage allowed for easier debugging, and made more functional sense by following the pathway defined in lectures, from lexing->parsing->intermediate generation->code gen and addressing.

Between these two dates the same logic was being applied to produce 3AC representations for more relevant functions (basic reqs) in the AST-based parser.

Weekly Review:

Q- Have the weekly requirements been achieved?

A- Not only were they achieved, but infact had been exceeded, lexing and parsing of the compiler was meant to be 1.5 weeks however the team have finished it within 1, putting us ahead of schedule for the code generation phase.

Q-How can the team improve based on this week's performance, so that the next week is better?

A- There aren't really any criticisms given that the team finished ahead of time, but perhaps instead of starting codegen right away, we could have instead tried to expand upon our existent parser and lexer and tried to start on the improvements earlier.

Phase 2: Code Generation:

Week beginning 1st March: Weekly objective:

-Add further implementation of 3AC representation for the compiler, with a debugging apparatus to help us fix issues.

-Begin conversion from 3AC to MIPS code with a separate file.

-Develop test apparatus in order to test the newly compiled code with QEMU-MIPS generated code.

02/03/2021: 3 address-code testing has been enhanced by producing print_canonical, which is a script that enables us to debug code and to validate 3AC, as this was a crucial checkpoint, once code has been converted to 3AC, this is what the MIPS code will be based on, team working on translate.hpp which enables us to convert each 3AC line into a corresponding MIPS instruction, for now, we have not taken into consideration more efficient register allocation methods or steps such as dead-code elimination which could make memory management and code more efficient, as we are aiming for a functionally correct compiler, and making improvements and tweaks after this point, at the current pace, Phase 2 has finished faster than anticipated and Phase 3 will now be worked on now.

04-05/03/2021: Initial implementation of 3AC to MIPS translation was complete, a few errors to correct was incorrect parsing of the tokens by using CString functions on the 3AC output files, but by

adjusting the list of delimiters, as well as adding more space to the linearray to process the entire line, it started as an initial separate cpp file but it was implemented into a header file in order to enable make based compilation. Can now be run as bin/compilealt -S compiler_tests/default/test_RETURN.c -o compiler_tests/default/test_RETURN.o

Phase 3 is now complete; it took approx. 12 days to finish. This was done more than half a week earlier than what was originally planned, which is good pace. This milestone means we have produced a functional and working compiler, which means that the original goal we had set out has now been achieved, although it is worth noting that this is only supporting the basic requirements. The rest of the time will now be dedicated simply to adding more tweaks to the now functional compiler to support more requirements.

06/03/2021: Development of the bash script that runs through all test-cases complete, develops the executables and uses the drivers and cross compiles the files with the QEMU-MIPS based equivalent and compares return codes, currently 18/87 test cases pass, but with more 3AC to MIPS translation of operators this can go up.

07/03/2021: Further operators added, intermediate requirements started: 34/87 test cases passed. Switch, enums and function calls (4 parameters max) delegated.

Weekly Review:

Q- Have the weekly requirements been achieved?

A- They had been achieved in accordance with the original plan, taking about a week to develop the complete codegen apparatus as defined in the weekly objective.

Q-How can the team improve based on this week's performance, so that the next week is better?

B- Keep a record of remaining implementations in order to make it easier to quickly tick off.

Phase 3: Extra Requirements:

Week beginning 8th March: Weekly objective:

-Given that the main deliverable has been initially complete, there is now more flexibility in our plan, however we have decided to dedicate this week to implementing things from the intermediate requirements of the specification.

-Specific additions: Function calls (generally and up to 4 arguments), enums, switch statements and structs.

09/03/2021: Function calls implemented getting 43/87, working on function calls for up to 4 parameters.

10/03/2021: Function calls for 4 arguments implemented raising testcases to 43/87, now working on break and continue statements as well as debugging other implementations.

12/03/2021: Raised to 52/87 after debugging and other implementations.

14/03/2021: Structs implemented, raised to 59/87.

Weekly Review:

Q- Have the weekly requirements been achieved?

A- While not all the intermediate requirements were achieved (due to Aryan especially taking time to finish Info Processing and other coursework, as well as Ani also having other coursework commitments), a good amount of intermediate requirement functionality was implemented.

Q-How can the team improve based on this week's performance, so that the next week is better?

C- Given that there was a conflict of other coursework at the time, perhaps making either adjusted expectations of weekly requirements or dedicating specific days of the week specifically for the requirements instead of slow drip.

Week beginning March 15th: Weekly objective:

-Finish work on Intermediate Requirements (Enums and other datatypes, switch statements), slowly add some Advanced Requirements features. Slower additions due to the commitments from other projects. This would be the last week of major implementations, as other coursework commitments needed work, and since we had already passed quite a lot of testcases, only minor tweaks were needed to increase the number.

18/03/2021: Enums implemented, raising to 61/87.

21/03/2021: Pointers implemented, raising to 65/87.

21/03/2021: Floats and Double support for datatypes added, raising overall testcases to 71/87. This was where we stopped.

Weekly Review:

Q- Have the weekly requirements been achieved?

A- Yes, barring Switch statements, the other Intermediate Requirements have been completely implemented and applied, along with some Advanced Requirements.

Q-How can the team improve based on this week's performance, so that the next week is better?

D- There could have been more dedication to the compiler this week, however due to the other time commitments simply there was nothing further to be added.

By this point, the compiler at 71/87 was to be considered the 'final' state of the compiler and it's C support, however for this final week, extra last minute additions would be piped in, especially on the final deadline day, in order to increase the support and pass number by just a small amount.

Week beginning March 22nd: Weekly objective:

-The compiler was considered 'done' for now, the team were not planning any major implementations due to the other coursework and lab reports required, however last-minute tweaks and additions were to be made.

24-26/03/2021: Last minute tweaks to lexer, parser functions and clean up of repo was undertaken, make files were verified and test-scripts re-ran.

Overall, the initial set of goals that we set as a team were met, we were aiming to produce a functioning compiler that was able to handle all the phases of compilation for basic requirements, as this was intended to be the most difficult part of the development process, once that was complete, we dedicated the rest of the time to implementing further types and functionality that C is included with. While the final two weeks of development could have had more additions, since we had already produced a functioning compiler and had passed a steadily increasing number of test-cases already, we felt that it would be better to delegate time to other module coursework which were more pressing at the time, towards the end further implementation was added to bring it to 71/87, we considered this to be a good stopping point, as by this point most of the general functionality for C had been compiled successfully. In a future product development cycle, it would be better to scale the goals and expand upon them as soon as the major requirements were met, that way organised goal setting and achievements can remain to be tracked, rather than adding them sporadically after the general development of the basic compiler had been completed.