Cloud and Computer Architecture – Assignment 1

Creation of a simple parser using JavaCC

Table of Contents

[Background 2](#_Toc183381875)

[Purpose 2](#_Toc183381876)

[Discussion 2](#_Toc183381877)

[Compiler 2](#_Toc183381878)

[AI in compiler design 3](#_Toc183381879)

[Practical Implementation of the Parser 4](#_Toc183381880)

[Bibliography 6](#_Toc183381881)

[References 6](#_Toc183381882)

## Background

A compiler is software that translates code written in a high-level programming language (e.g. Java) into a lower-level representation (e.g. machine code) which can be executed by a computer. A Java compiler translates Java source code into bytecode, which can be executed in software using the Java Virtual Machine (JVM) or further compiled into machine code.

JavaCC (Java Compiler Compiler) is a software tool used in the front-end of compiler design. It helps automate the creation of programs used for lexical analysis and parsing, which are critical components of compiler usage. Java files are generated based on what is specified in the JavaCC file i.e. it defines the grammar of a language or a syntax that the parser needs to recognize. Since JavaCC source files generate Java source code from the grammar file, Java code can be included directly in the grammar rules. This allows for additional verification or functionality during parsing e.g. managing exceptions, generating error or debug information. Embedding Java allows integration between the generated parser and other Java-based components in the application.

## Purpose

The purpose of this report is to summarise activity relating to creation of a simple parser using JavaCC which will be used for syntactical and lexical analysis of key-value pairs. The technical aspects of using JavaCC and how it relates to creation of a parser, or a syntactical and lexical analyser will be discussed. Additionally, information on usage of the program developed for this assignment will also be included.

## Discussion

### Compiler

A compiler completes several stages when processing high-level language. Each stage of the process is required to perform a specific function to ensure the code is correct, efficient, and executable by a device.

**Lexical Analysis (Tokenisation):** The compiler reads source code as a sequence of characters and breaks these characters into tokens, which are the smallest purposeful units used by a compiler, such as operators, identifiers, delimiters, and keywords e.g. int, a, 1, +, =, {.

**Syntax Analysis (Parsing):** A programming language grammar is a set of rules that define valid structure of a program in that language. These rules specify how statements and expressions are formed, ensuring that code has correct syntax and is relevant to the programming language. The tokens generated during lexical analysis are organised according to the grammar rules in a process called parsing.

Parsing is the responsibility of the parser. This is a component of the compiler which performs syntax analysis. During syntax analysis, the compiler generates a Parse Tree (a syntax tree) which organises tokens into a logical tree structure of specific code elements - JavaCC uses JJTree for this purpose. If the tokens cannot be arranged into a valid tree, it indicates that the source code does not adhere to the grammar. This results in a syntax error.

**Semantic Analysis:** The compiler verifies that the code is logically valid and adheres to the semantic rules of the language. The compiler will check for logical errors, such as type mismatches, undeclared variables, and incorrect usage of operators.

**Intermediate code generation:** After lexical, syntactical and semantical analysis, the compiler converts the verified syntax tree into a platform-independent intermediate representation of the source code, which makes it easier to optimise and translate into machine code.

**Optimisation:** The compiler improves the intermediate code for better performance and resource utilisation. Actual optimisation completed will depend on the compiler being used but optimisation may result in reduced number of instructions, elimination of redundant calculations and rearrangement of code for better execution flow.

**Code Generation:** The optimised intermediate representation is translated into machine-readable code which is specific to the platform architecture e.g. x86, ARM.

The machine code and any libraries or other modules associated with it may be combined to produce an executable file e.g. .exe

### AI in compiler design

Artificial Intelligence (AI) could have an increasingly significant role in compiler design. AI employs techniques, such as machine learning (ML) and deep learning to various stages of the compilation process to enhance performance, optimise generated code, and improve error detection and recovery. However, we also have to accept the limitations of using AI, and the possible conflicts with human interaction.

**Code optimisation:** AI can enhance code optimisation by learning patterns and making decisions that are not possible when using traditional compilers in typical use cases. AI has hugely greater ability over human interaction when analysing patterns and determining the most effective methods for optimising compilers.

Rajwal, Swati & Chakraborty, Pinaki. (2023) [1] discussed how programmers designing and developing compilers could only analyse a few optimisation techniques. Using AI, optimisation techniques can be evaluated on a large number of sample programs in a very short timeframe. The best optimisation techniques can then be included in the compilers.

**Adaptive Compilation:** Compilers could potentially adjust optimisationbased on the system being used and the resources available. AI models can predict runtime behaviour of code based on learned code patterns from historical data. Architecture on modern systems varies greatly, AI could determine the best optimisation depending on the system being used e.g. single core CPU, multi-core CPU, or specific GPU. Rajwal, Swati & Chakraborty, Pinaki. (2023) [1] discuss how cluster computers and supercomputers have multiple processing units which can execute different parts of the same program in parallel. AI techniques could be used to determine which parts of a program can be executed in parallel so as to maximise the speed of execution.

**Error Detection:** AI could potentially improve the error detection capabilities of compilers by suggesting fixes for syntax or semantic error based on evaluation of existing patterns in historical data. This could allow for prediction of complex logical errors and recommendations for resolution. DeepCode [2] and similar tools use AI to analyse source code for bugs and can offer possible resolution and recovery options. However, existing developers may not be in favour of this as a viable possibility. [3].

**Using trained AI models:** AI models are only as effective as the data on which they are trained. If an AI-based compiler is applied to a programming language it was not trained to optimise for, there may be little or no optimisation. Additionally, if the training data contained errors or irrelevant information, the resulting output could be effectively useless as error-prone training data can lead to biased or faulty models, which might produce ineffective compiler optimisations. Potentially, users of AI may be premature in deciding that AI is suitable for the purposes that they envisage. Unless AI optimisation of compilers is repeatedly shown to be possible and ultimately beneficial, adoption of AI optimised compilers may be slow.

**Use of AI for this assignment:** ChatGPT was used for this assignment while trying to further information on syntax trees as part of parsing. ChatGPT was again used to get examples of where AI has been used for compiler optimisation to determine if AI is actually being used in many real-world scenarios. Three examples were provided: 1. Meta (Facebook) Compiler Gym [4], 2. Low level virtual machine (LLVM) Machine Learning Optimization [5], and Google’s TensorFlow XLA Compiler [6].

I have found that using a GPT for acquiring information can be unreliable. It is up to the user to determine if the information is accurate and relevant. While trying to get more information on syntax trees, I wanted to verify that the information provided was correct. This required further online research. I would conclude that the information provided by the GPT was accurate and relevant, but it may not always be so. When requesting examples where AI has been used for compiler optimisation, I found that the examples are indeed real-world applications of AI for compiler optimisation. However, providing an example of Google using TensorFlow XLA is not entirely relevant as this does not relate to a programming language but to an open-source machine learning framework.

### Practical Implementation of the Parser

A simple parser was developed for this assignment which takes input from a user and determines if correct structure and syntax is used for assigning key-value pairs. Using Java, or when assigning values in configuration data, values are assigned to properties (keys) in the form “property = value”. If multiple key-value pairs are being used, a comma ‘,’ is included between key-value pairs e.g. property1=value1, property2=value2.

**Design**

The SimpleParser Java class was developed to prompt a user to input data and then parse the input according to predefined grammar rules. If the input adheres to the grammar rules, the system outputs a message indicating that parsing was successful. If the input contains syntax errors (i.e., does not conform to the grammar rules), a ParseException is thrown, and the user is notified with a relevant error message.

The parser allows key-value pairs where tokens used for both keys and values can consist of alphanumeric characters, underscores, and hyphens. These characters are valid for both keys and values, and no specific order or separate assignment of acceptable tokens for key and value is enforced. Whitespace and escape characters are ignored and are not included as tokens. An equals symbol (=) is required to assign values to properties, using the format “property=value”, making ‘=’ a necessary token in the grammar. A comma ‘,’ is also included to allow for assignment of multiple key-value pairs.

The grammar rules for assignment enforce the format “property=value” by method calls to separate methods for Property and Value. Separate methods were created to allow for error checking.

**Testing**

When the SimpleParser Java class is excuted, the user is prompted with a message to provide input for the Parser with example format: “Enter assignments (e.g., property1=value1, property2=value2):”

If the user enters acceptable syntax e.g. “property1 = value1” and presses enter, the parser accepts the input (no error is reported). Additional input can be provided until the user terminates the process using ctrl+c, at which point the system outputs a message “Parsing completed successfully!”. If unacceptable syntax is used e.g. “property1 = “ and the process is terminated before acceptable syntax is used, the system will output “Syntax error: Missing value after '=' in key-value pair.” as a ParseException has been thrown. Similarly, if “ = value1 “ is entered, a ParseException is thrown with a “Syntax error: Missing key before '=' in key-value pair.” message as output. Use of multiple assignment using a comma between key-value pairs generates “Parsing completed successfully!” once input is complete and the process is terminated.

**Mechanism**

When the main method is called and the user is prompted to input data, that data is passed to the Parser object (SimpleParser parser). The AssignmentList method is called to begin parsing input. This method processes one or more key-value pairs separated by commas.

Assignment() ( <COMMA> Assignment() )\*

The parser uses the token definitions in the JavaCC file to split the input into smaller pieces. These pieces are what are identified as tokens. Each token corresponds to a specific type of input (e.g. <PROPERTY\_OR\_VALUE>, <EQUALS>, <COMMA>).

<PROPERTY\_OR\_VALUE> can be alphanumeric, hyphen, or underscore for Key or Value characters.

<EQUALS> must be ‘=’ symbol, and this symbol only.

<COMMA> must be ‘,’ symbol, and this symbol only.

Whitespace and escape characters are ignored by the parser.

The parser evaluates an assignment using the Assignment method.

Property() <EQUALS> Value()

If a comma is encountered, another assignment is expected. If no comma is encountered, the single assignment is evaluated: the Property() method is called to validate the sequence of characters used for the key. The parser then determines if there is a character following which matches ‘=’ , and Value() is called to validate the sequence of characters used for the value.

The Property() and Value() methods both evaluate if the input consists of valid characters as specified in < PROPERTY\_OR\_VALUE >, the Assignment method determines if correct assignment (structure) is used. The SimpleParser class is set up to generate a message if parsing is successful. If not, the Parser determines that there is an error in syntax and a ParseException is thrown. Similarly, if there are no characters before or after the ‘=’, or if there is no ‘=’, a ParseException is thrown notifying the user of the relevant syntax error.

## Bibliography

JavaCC. (2024). *JavaCC*. [online] Available at: <https://javacc.github.io/javacc/documentation/jjtree.html>.

‌Braga Rodrigues, T. (2021). *Cloud and Computer Architecture JavaCC*. [online] Cloud and Computer Architecture JavaCC. Available at: [TUS - Moodle](https://ait-main-mdl-euwest1.s3.eu-west-1.amazonaws.com/c0/65/c06553fd5c14ad271b41dc5425470c9cb47bf1df?response-content-disposition=inline%3B%20filename%3D%22JavaCC%20Lab1.pdf%22&response-content-type=application%2Fpdf&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAWRN6GJFLQXT45LXM%2F20241122%2Feu-west-1%2Fs3%2Faws4_request&X-Amz-Date=20241122T212113Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21587&X-Amz-Signature=8c8f16f4dab5f5dd78f2958bef4405452c0df00c5af9267cf6b8c8403a947ad7).

Sheldon, R. (2022). *What is a compiler?* [online] WhatIs.com. Available at: <https://www.techtarget.com/whatis/definition/compiler>.

GeeksforGeeks. (2015). *Introduction to Syntax Analysis in Compiler Design*. [online] Available at: <https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/>.

GeeksforGeeks. (2020). *Parse Tree in Compiler Design*. [online] Available at: <https://www.geeksforgeeks.org/parse-tree-in-compiler-design/>.

Oracle.com. (2024). Available at: <https://blogs.oracle.com/javamagazine/post/introducing-the-javacc-21-parser-generator-for-java>.

Wikipedia. (2023). *Optimizing compiler*. [online] Available at: <https://en.wikipedia.org/wiki/Optimizing_compiler>.

‌

## References

[1] Rajwal, Swati & Chakraborty, Pinaki. (2023). Application of artificial intelligence in compiler design. Available at: <https://www.researchgate.net/publication/376375064_Application_of_artificial_intelligence_in_compiler_design>

[2] snyk (n.d.). *Snyk Powered by DeepCode AI*. [online] Snyk. Available at: <https://snyk.io/platform/deepcode-ai/>.

[3] Quora. (2019). *Will we ever have a compiler that will fix the error instead of just stating what it is?* [online] Available at: <https://www.quora.com/Will-we-ever-have-a-compiler-that-will-fix-the-error-instead-of-just-stating-what-it-is>.

[4] Meta Research. (2023). *CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research - Meta Research | Meta Research*. [online] Available at: <https://research.facebook.com/publications/compilergym-robust-performant-compiler-optimization-environments-for-ai-research/>.

[5] Doerfert, J., Hosek, P., Cummins, C., Grossman, A., Trofin, M., Qian, Y., Sykora, O., Das, D., Ashouri, A., Elhoushi, M. and Venkatakeerthy, S. (n.d.). Machine Learning Guided Optimizations (MLGO) in LLVM. [online] Available at: <https://llvm.org/devmtg/2022-11/slides/Panel1-MLGO.pdf>

[6] Googleblog.com. (2017). *XLA - TensorFlow, compiled*. [online] Available at: <https://developers.googleblog.com/en/xla-tensorflow-compiled/?utm_source=chatgpt.com>.