Cloud and Computer Architecture – Assignment 1

Creation of a simple parser using JavaCC

Table of Contents

[Background 2](#_Toc183211031)

[Purpose 2](#_Toc183211032)

[Discussion 2](#_Toc183211033)

[Compiler 2](#_Toc183211034)

[AI in compiler design 3](#_Toc183211035)

[Practical Implementation of the Parser 3](#_Toc183211036)

[Bibliography 4](#_Toc183211037)

[References 4](#_Toc183211038)

## 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.

JavaCC source files (.jj extension) define the rules and grammar for analysing and parsing text. This file is provided as input to JavaCC, which generates Java source code. Since JavaCC source files generate Java source code from the .jj 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. The technical aspects of using JavaCC and how it relates to creation of a parser, or a lexical analyser will be discussed. Additionally, usage of the program developed for this assignment will also be outlined.

## 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) is playing 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.

**Code optimisation:** AI can enhance code optimisation by learning patterns and making decisions that are typically not possible when using traditional compilers in typical use cases.

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 can 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].

### Practical Implementation of the Parser

## 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>.

‌

‌

‌

‌

‌

‌