SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CHENNAI-602105

Building Compiler for functional programming language

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by

V. Mano Karthik Reddy (192225077)

D. Teja (192210626)

G. Ajay Kumar (192210590)

Under the Supervision of

Dr. G. MICHAEL

JUNE 2024

DECLARATION

We, **V. Mano Karthik Reddy, D. Teja, G. Ajay Kumar,** students of **'Bachelor of Engineering in Department of Computer Science**' in Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **Building Compiler for functional programming language** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

V. Mano Karthik Reddy (192225077)

D. Teja (192210626)

G. Ajay Kumar (192210590)

Date:

Place:

CERTIFICATE

This is to certify that the project entitled **"Building Compiler for functional programming language"** submitted by, **V. Mano Karthik Reddy, D. Teja, G. Ajay Kumar** has been carried out under our supervision. The project has been submitted as per the requirements in the current semester of B. Tech Information Technology.

Teacher-in-charge

Dr. G. MICHAEL

Table of Contents

ABSTRACT:

Creating compilers for functional programming languages presents a critical endeavor in advancing software development paradigms. This paper offers a comprehensive review of the process of building compilers for functional

programming languages, elucidating its significance, methodologies, and impact on programming language ecosystems. The abstract delves into the intricacies of compiler construction, emphasizing its role in translating high-level functional code into executable machine instructions.

It outlines the diverse components involved in compiler design, including lexical analysis, syntax parsing, semantic analysis, optimization techniques, and code generation strategies, elucidating their respective functionalities and interdependencies. The abstract also discusses various compiler construction tools and frameworks, such as Lex and Yacc, LLVM, and GCC, highlighting their suitability and effectiveness in building compilers for functional languages.

Moreover, the abstract evaluates the benefits and challenges inherent in developing compilers for functional programming languages, emphasizing their ability to promote code correctness, modularity, and abstraction while addressing issues like performance optimization and type inference complexities. It further examines emerging trends and advancements in compiler technology, such as Just-In-Time compilation and functional programming language extensions, underscoring their potential to revolutionize software development practices and enhance language expressiveness.

Introduction:

In the realm of modern software development, the construction of compilers for functional programming languages stands as a pivotal endeavor. These languages, renowned for their emphasis on abstraction, modularity, and correctness, offer

unique challenges and opportunities in the domain of compiler design. With the proliferation of functional programming paradigms in various software ecosystems, the need for robust compilers tailored to these languages has become increasingly apparent.

This investigation seeks to explore the intricate process of building compilers for functional programming languages, shedding light on the methodologies, tools, and innovations driving this transformative field of study. By delving into the complexities of compiler construction and examining emerging trends, this investigation aims to equip developers and language enthusiasts with the knowledge and insights necessary to navigate the terrain of functional language compilation effectively.

Problem Statement:

As functional programming languages gain prominence in software development, the demand for tailored compilers to support these languages grows. However, navigating the landscape of compiler construction presents a significant challenge. With a plethora of methodologies, tools, and innovations available, selecting the most suitable compiler framework for a given functional language becomes a daunting task for developers and organizations alike.

Proposed Design:

Requirements Gathering and Analysis: Engage in stakeholder interviews and surveys to ascertain the organization's needs regarding compiler functionality, language support, and performance requirements for the target functional

programming language.

Tool Selection Criteria: Compile a list of compiler construction tools, considering language compatibility, optimization capabilities, and community support. Evaluate tools based on project objectives using industry research and expert recommendations.

Scanning and Testing Methodology: Define a systematic approach to compiler design, covering lexical analysis, parsing, semantic analysis, optimization, and code generation. Implement best practices and methodologies to ensure the compiler's robustness and efficiency for the chosen functional programming language.

Functionality:

User Authentication and Role-Based Access Control:

- Implement user authentication measures to manage access to the compiler system.
- Define roles and permissions to control access based on user responsibilities and authorization levels, ensuring secure interaction with the compiler's functionalities.

Tool Inventory and Management:

- Maintain a centralized catalog of functional programming language development tools, including vendor information, version numbers, and

license status.

- Streamline tool management processes such as installation, configuration, and updates, ensuring seamless integration with the compiler development environment.

Security and Compliance Controls:

- To safeguard sensitive data, implement robust security measures such as encryption, access controls, and comprehensive audit trails to ensure compliance with relevant standards and regulations.

Architectural Design:

Presentation Layer:

- Develop a web-based user interface (UI) tailored for interacting with the assessment framework specific to the functional programming language.
- Implement role-based access control (RBAC) to manage user authentication and permissions within the compiler system.

Application Layer:

- The business logic layer is dedicated to processing user requests and orchestrating system functionality tailored for the functional programming language.
- The criterion management module is designed to define, store, and manage assessment criteria specifically relevant to the functional programming paradigm within the compiler system.

Monitoring and Management Layer:

- Integrate tools for real-time performance monitoring, log analysis, and system health checks optimized for the requirements of the functional programming language compiler.
- Utilize platforms for centralized and aggregated storage and analysis of system logs, ensuring seamless management and insight generation tailored to the compiler's specific needs.

UI Design:

Dashboard:

- Tiles/cards displaying key metrics about the compilation process, such as the number of source files compiled, errors encountered, and compilation time.
- System status indicators indicating the current state of the compiler e.g. idle, compiling, or error.

User Management:

- User account management interface allowing administrators to create, edit, and delete user accounts.
- Role assignment functionality enabling administrators to assign roles to users and define their permissions.

Help and Support:

- Help documentation section accessible from the dashboard, containing user manuals, guides, and FAQs.
- Support contact information displayed prominently, allowing users to reach out for assistance when needed.

Element Positioning and Functionality:

Real-time Monitoring:

- Positioned on the dashboard to provide real-time monitoring of the compilation process.
- Widgets or progress bars display live updates on compilation progress, including the number of files processed, errors encountered, and compilation speed.

Collaboration Features:

- Available within the compiler environment, allowing users to collaborate on source code files.
- Features such as comments, annotations, or version control support facilitate collaboration among compiler developers and testers.

Trend Analysis:

- Positioned in the reporting and analysis section, offering insights into the compiler's performance.
- Interactive charts or graphs visualize compilation metrics over time, such as compilation speed, error trends, and resource utilization.

Conclusion:

Overall, a well-designed UI for a compiler for a functional programming language streamlines the compilation process, supports collaboration and knowledge sharing, and empowers users with the tools and resources needed to effectively compile and manage code.