



North Western University

Course Name: Compile Design (Course Code: CSE-4104)

Course Instructor: Md. Shymon Islam

Lecturer, Department Of CSE

Section: C, Spring 2023

Date of Submission: 5th June,2023

Submitted by:

1.Ringshina Akter Tithi

(20201085010)

2.S M Muhaimin

(20201108010)

3.Marium Akter Mithun

(20201152010)

Table of Contents

Abstract.....	3 (This report...)
Introduction.....	3 (The compilation...)
Objectives of the Report.....	3 (To provide...)
Compiler Architecture.....	4 (Lexical Analysis...)
Design & Implementation.....	5 (The lexer...)
i. Input Buffering.....	6 (Source code...)
ii. Tokenization	7 (The lexer...)
iii. Symbol table.....	8 (The symbol...)
Challenges and Future Directions.....	8 (Multi-language...)
Conclusion	9 (Compile design...)

Abstract:

This report provides an in-depth analysis of compile design, focusing on its strategies, techniques, and applications. Compile design plays a crucial role in software development by translating high-level programming languages into machine-readable code. The report explores various aspects of compile design, including compiler architecture, optimization techniques, and its applications in different domains. Additionally, it discusses challenges and future directions in the field of compile design.

Introduction:

The compilation process is an essential step in software development, where high-level programming languages are translated into executable code that a computer can understand and execute. The compile design refers to the methodology and techniques employed to accomplish this translation effectively and efficiently. Compile design, also known as compiler design, is a critical area of computer science that focuses on developing software tools known as compilers. Compilers play a crucial role in translating high-level programming languages into low-level machine code, enabling efficient execution on various hardware platforms. This report provides a broad overview of compile design, including its advancements, applications, and future prospects.

Objectives of the Report:

1. To provide a comprehensive understanding of compile design and its importance in software development.
2. To explore the different stages of compiler architecture, including lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization, and code generation.
3. To highlight various optimization techniques employed in compile design, such as constant folding, loop optimization, data flow analysis, register allocation, and instruction scheduling.
4. To discuss advanced techniques and approaches in compile design, such as Just-In-Time Compilation (JIT), profile-guided optimization (PGO), auto-vectorization, machine learning integration, and polyhedral compilation.

5. To examine the diverse applications of compile design in programming languages, embedded systems, high-performance computing, web development, and domain-specific languages.
6. To identify the challenges and future directions in compile design, including compilation for heterogeneous architectures, quantum computing, compile-time performance improvement, and security considerations.
7. To provide insights into the potential advancements and opportunities in compile design, exploring emerging technologies and integration of AI and ML techniques.

Compiler Architecture:

Lexical Analysis: Tokenization and scanning of the source code.

Syntax Analysis: Construction of parse trees and syntax validation.

Semantic Analysis: Type checking and symbol table generation.

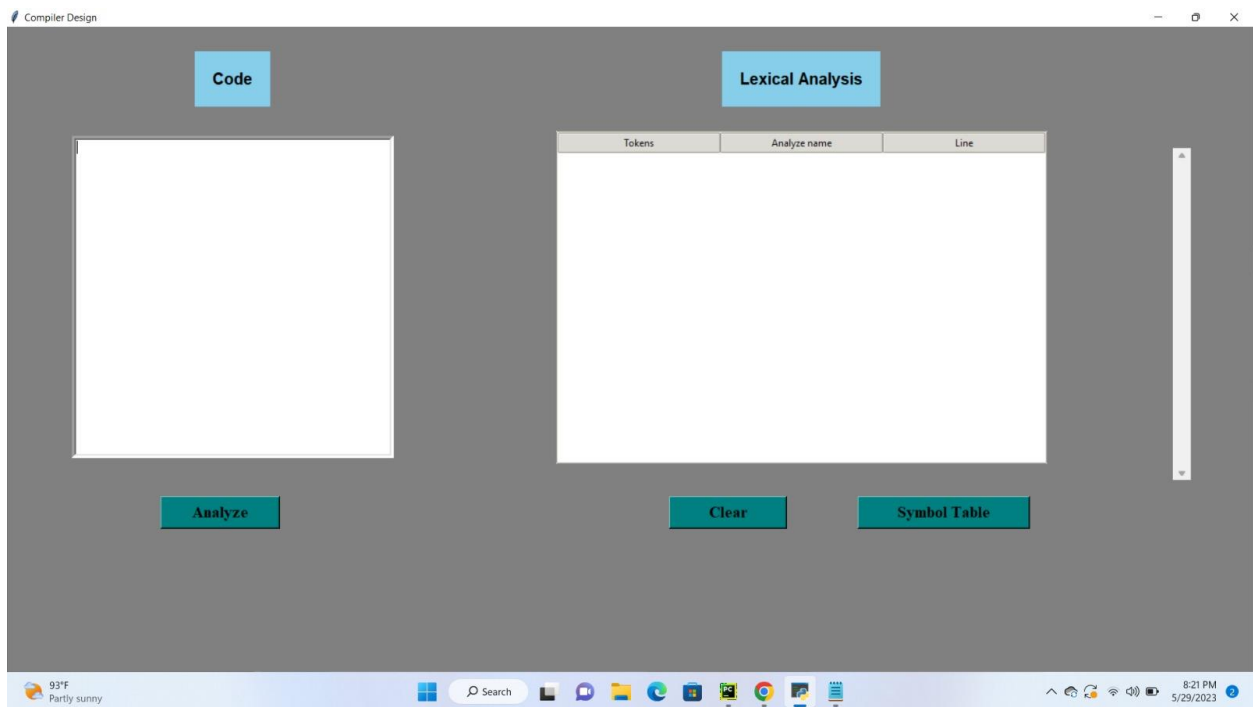
Intermediate Code Generation: Translation of source code into an intermediate representation.

Code Optimization: Transformations to enhance code efficiency.

Code Generation: Translation of the intermediate code into executable machine code.

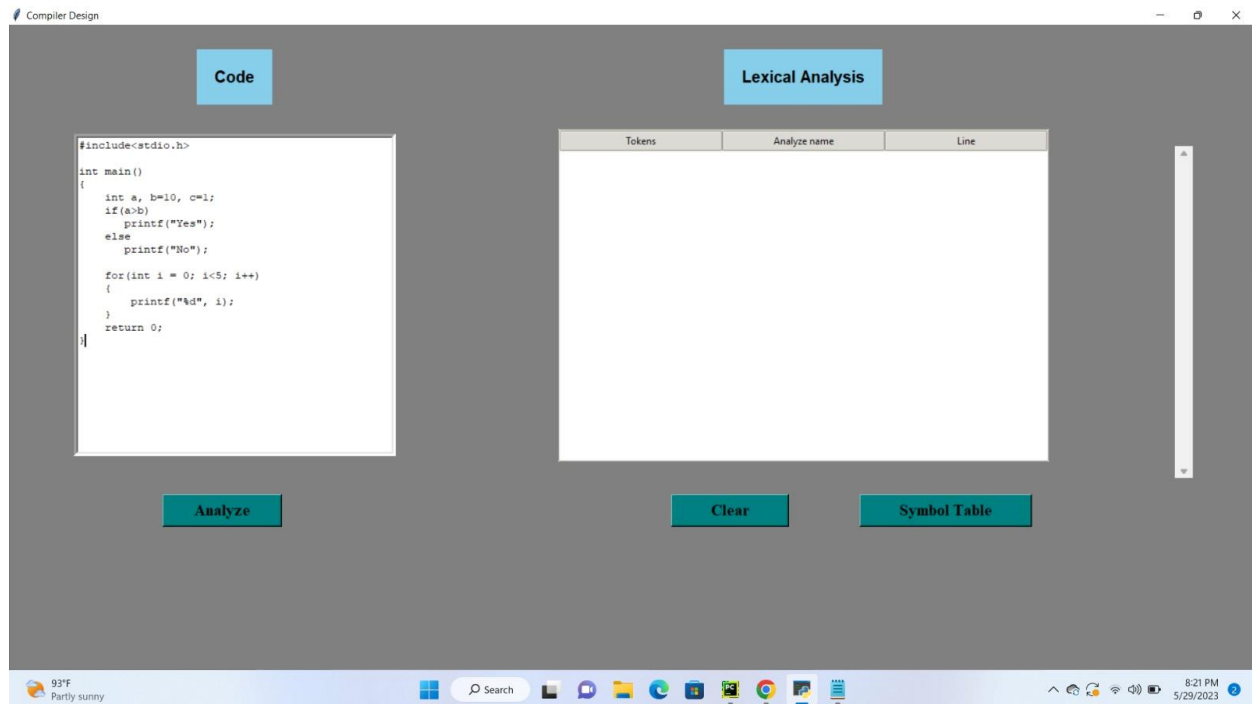
Design & Implementation:

The lexer iterated over the input buffer character by character, recognizing and categorizing each token it encountered. Source code were utilized to define the patterns for various tokens, such as keywords, identifiers, number, strings, symbols,function etc. Whole process handled after clicking analyze button. And there is clear tokenize button for clear the tokenize table. The lexer maintained a symbol table to store and manage identifiers encountered during tokenization.



Project Interface

Input Buffering:



Source code is provided for input field .Only C language code can be used. The clear source button is used to clear the input field.

Tokenization:

The lexer iterated over the input buffer character by character, recognizing and categorizing each token it encountered. Source code were utilized to define the patterns for various tokens, such as keywords, identifiers, number, strings, symbols,function etc. Whole process handled after clicking analyze button. And there is clear tokenize button for clear the tokenize table. The lexer maintained a symbol table to store and manage identifiers encountered during tokenization.

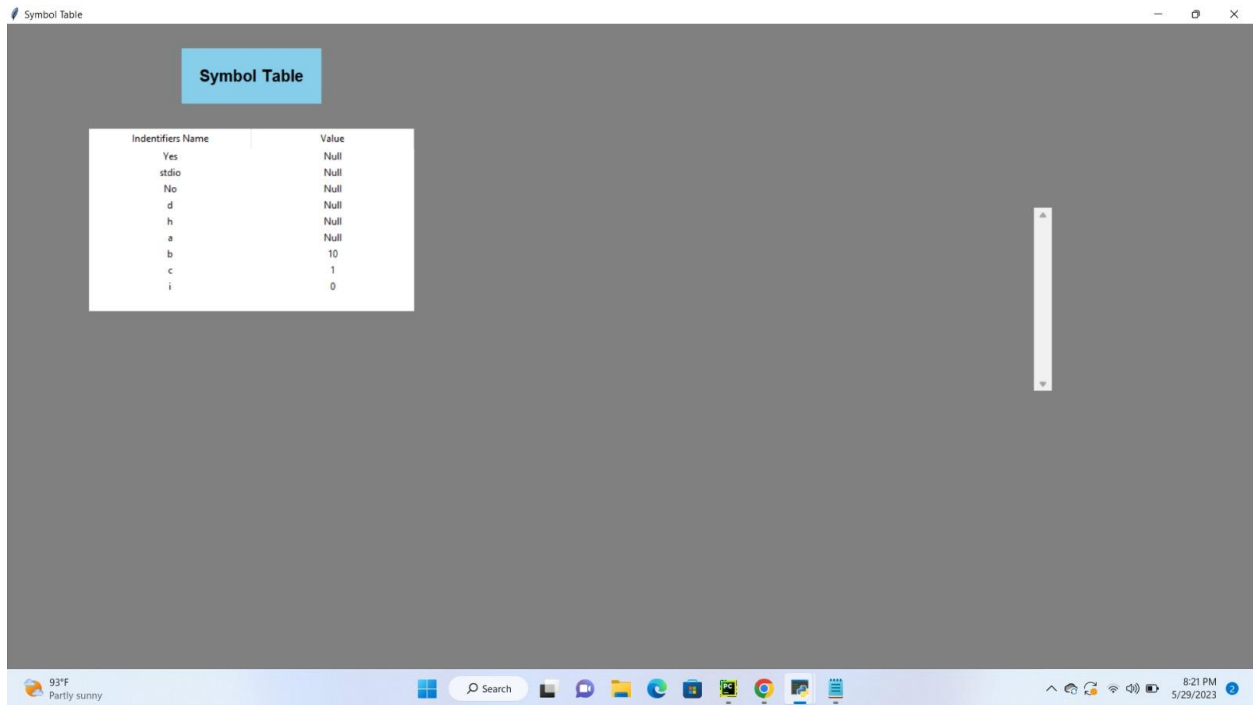
The screenshot displays a software application titled "Compiler Design". It features two main panels: "Code" on the left and "Lexical Analysis" on the right. The "Code" panel contains a C program snippet. Below the code editor is an "Analyze" button. The "Lexical Analysis" panel shows a table of tokens with columns "Tokens", "Analyze name", and "Line". Below this table are "Clear" and "Symbol Table" buttons. The application is running on a Windows desktop, with the taskbar at the bottom showing the date and time as 8:21 PM on 5/29/2023.

```
#include<stdio.h>
int main()
{
    int a, b=10, c=1;
    if(a>b)
        printf("Yes");
    else
        printf("No");

    for(int i = 0; i<5; i++)
    {
        printf("%d", i);
    }
    return 0;
}
```

Tokens	Analyze name	Line
#	Special Character/Symbol	0
include	Keyword	0
<	Operator	0
stdio	Identifiers	0
.	Special Character/Symbol	0
h	Identifiers	0
>	Operator	0
int	Keyword	0
main	main function	0
()	Special Character/Symbol	0

Symbol Table:



Identifiers Name	Value
Yes	Null
stdio	Null
No	Null
d	Null
h	Null
a	Null
b	10
c	1
i	0

The symbol table is used to hold the identifiers of the source code.

Challenges and Future Directions:

1. Multi-language Compilation: Handling multiple programming languages and their interoperability.
2. Optimization for Emerging Architectures: Adapting compile design for GPUs, quantum computers, and other novel architectures.
3. Compile-Time Performance: Reducing compilation time and memory requirements.
4. Integration of Machine Learning: Exploring the use of machine learning techniques in compile design.

Conclusion:

Compile design plays a pivotal role in enabling efficient software development, optimizing code execution, and supporting a wide range of computing platforms. Advancements in optimization techniques, parallelization, language support, and JIT compilation have transformed the field. As emerging technologies such as machine learning, quantum computing, and hardware-software co-design gain prominence, the role of compilers will evolve, providing new opportunities and challenges for compiler designers in the future.