DETAILED PROJECT REPORT

On

Theoretical Practice and implementation of a basic Compiler

Submitted to

University of Petroleum and Energy Studies

In Partial Fulfilment for the award of the degree of

BACHELORS IN TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING (with specialization in Cloud Computing and Virtualization Technologies)

By

Name: Gautam Vijan
SAP ID: 500083392

Under the guidance of

Prof. Saurabh Shanu

Assistant Professor — Senior Scale

University of Petroleum and Energy Studies

Dehradun - India

DECEMBER 2022

CANDIDATE'S DECLARATION

I hereby certify that the project work entitled "Theoretical Practice and implementation of a Basic Compiler" in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Cloud Computing and Virtualization Technology and submitted to the Department of Systematic at School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from Aug 2022 to Nov, 2022 under the supervision of Mr. Saurabh Shanu, Assistant Professor – Senior Scale, Systemic Cluster. The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University. This is to certify that the above statement made by the candidate is correct to the best of my knowledge. Date: 06 December 2022

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University. This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 06 December 2022

Mr. Saurabh Shanu

Project Guide

ACKNOWLEDGEMENT

I would wish to impart our guide Mr. Saurabh Shanu, for all the advice, encouragement and constant support he gave throughout our project work. This work wouldn't be doable without his support and valuable suggestions. I would like to thank my batchmates for testing my compiler and founding some major bugs at early stage. I wish to impart all our friends for his or her facilitate and constructive criticism whereas acting on our project and also our sincere gratitude to our parents for every support they gave us.

ABSTRACT

In this report, I will present implementation of a basic compiler that can perform Tokenization and Parsing. The generation of Binary Code has been implemented in very basic way. These all can be performed with a GUI based Compiler. The programming language used in this project is Java. The project has been uploaded to GitHub for anyone to use it and bring changes to it. This Project comes under GNU Public License version 3 so any one can use it for private use.

Table of Contents

INTRODUCTION

1.1. BACKGROUND

Without altering the program's meaning, a compiler converts code written in one language to another. Additionally, it is assumed that a compiler will make the target code effective and space and time optimal. There are several stages in the compilation process. Each step of the compiler receives input from the stage before it, has its own representation of the source code, and feeds its output to the stage after. The compiler reads the source code, separates it into its component components, and then does a lexical, grammatical, and syntax check. Assembly Language Code, which is also known as intermediate code, is produced during the analysis step.

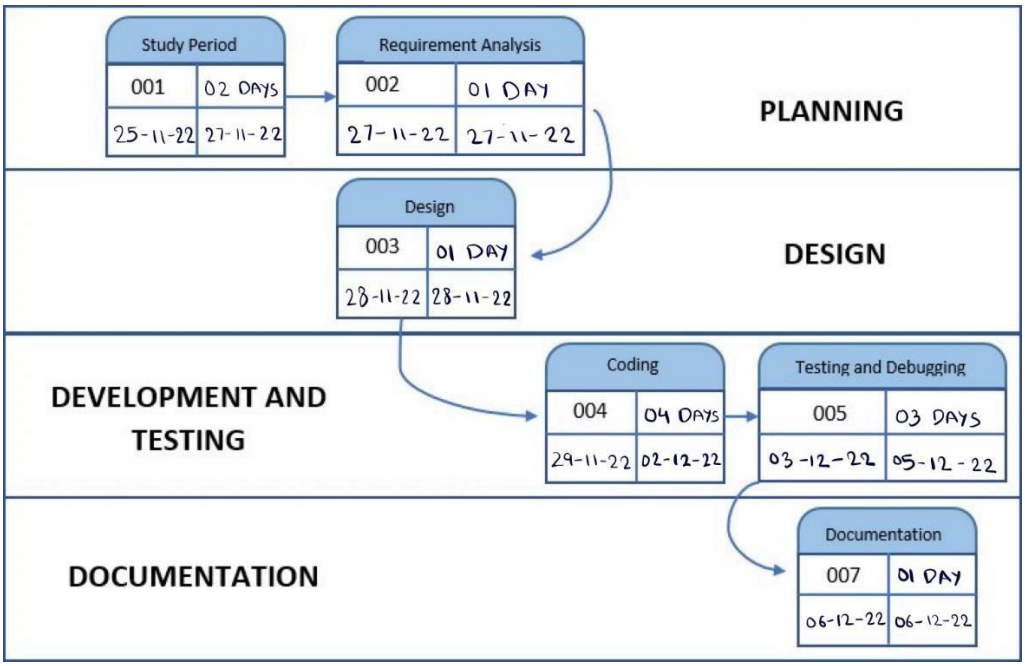
1.2. OBJECTIVE OF THE PROJECT

Our goal is to translate a language into intermediate code. Front end is the component of compiler design. The Code is to convert into an Abstract Syntax Tree using parsing. The binary code that was generated is not machine-dependent. Later, this code is improved and transformed into machine code.

1.3. PROBLEM STATEMENT

Compiler is a very complex subject and studying it with books or other materials can only make us pass our exams but if we want to understand the whole and shole of a compiler then we need to see its implementation. But there is no way to do this. So, I came up with this idea which can make anyone understand these concept just by visualizing it.

1.4. PERT CHART



SOFTWARE REQUIREMENTS SPECIFICATION

3.1. GENERAL DESCRIPTION

3.1.1. Purpose of the project

The main purpose of my project is building a compiler that can perform Lexical Analysis i.e., Tokenization and Parsing.

3.1.2. Target Beneficiary

The people who will profit from your project are known as the project beneficiaries, often known as the target group or the project beneficiaries. They are the individuals whose circumstances you intend to alter by putting your idea into practice. The project may have an impact on them directly or indirectly.

My target beneficiary is every student who find compiler a very complex subject. It can help them organize file instantly.

3.1.3. Project Scope

To give users a practical user interface so they can use it to see how tokens are created and how Abstract Syntax Tree is generated.

3.2. PROJECT DESCRIPTION

3.2.1. Data/Data Structure

The form of a variable to which a value can be assigned is the data type. It specifies that only values of the specified data type shall be assigned to the specific variable.

A data structure is a grouping of several types of data. The application can use an object to represent the complete set of data and use that object throughout.

Data Structures I have used in my project are:

- Array
- ArrayList
- Stack
- LinkedList

3.2.2. SWOT Analysis

Strength: Implementation is very basic. Anyone can use it with a very basic GUI.

Weakness: The project is complex. Use of very complex concepts like Java Collections. GUI making in Java is very complex.

Opportunities: Learned how to implement theoretical concepts into real life project.

Threats: Most of the bugs has been solved but it the project is prone to some undetermined errors.

3.2.3. Project Features

The main focus of my project is to implement Lexical Analysis and Parsing.

3.2.4. Design and Implementation Constraints

The limits placed on the design solution—in this case, the ESS design—are referred to as design constraints. These restrictions are often enforced by the client, the production team, or by laws from outside sources. Our Design and Implementation Constraints includes:

- Using many and different Java Collections •
  Creating a GUI for the Compiler.

3.2.6. System Requirements

My Compiler doesn’t require some high system requirements. It can run on any basic computer with less RAM, CPU and Storage. It doesn’t require any high end GPUs.

SYSTEM TESTING

7.1. INTRODUCTION

System testing verifies that an application performs tasks as designed. The testing step was critical in the development life cycle since it was here that any faults that lingered from previous phases were discovered. As a result, testing is crucial in ensuring software package stability and quality assurance. Following installation, a test strategy should be developed and executed using a specified set of test data. Each test has a specific aim, but they are all designed to ensure that all system parts are correctly connected and are doing their assigned roles. The testing technique is in reality used to guarantee that the product operates precisely as intended. The final internal phase of the organization.

7.2. UNIT TESTING

Although secret writing and unit testing are sometimes administered as distinct stages, unit testing is frequently performed as part of a combined code and unit test component of the software package lifecycle. Examine your strategy and plan. Manual field testing will be performed, as well as complete practical tests.

For testing of my Compiler, any String can be passed consisting of any basic Mathematical expression like, “((2+3)*(2-4)/4)”.

7.2. OUTPUT

Sample Input :- ((2+3)*(2-4)/4)

The compiler checks whether the Brackets are Balanced or not. If the Brackets are not balanced then the Syntax tree is not printed.

```
INPUT:
((2+3)*(2-4)/4)

------Breaking your code into Tokens------

Token   Type
 (     Parentheses
 (     Parentheses
 2     Integer
 +     Operator
 3     Integer
 )     Parentheses
 *     Operator
 (     Parentheses
 2     Integer
 -     Operator
 4     Integer
 )     Parentheses
 /     Operator
 4     Integer
 )     Parentheses
```

```
------Checking for Balanced Brackets------
Number of Brackets are Balanced
```

```
Abstract Syntax Tree for '((2+3)*(2-4)/4)':
{
      type: 'Program',
      body: [{

            {

                  {

                        type: 'number'
                        Value: '2'
                        type: 'number'
                        Value: '3'
                        type: 'operator'
                        Operation: 'add'
                  }

                  {

                        type: 'number'
                        Value: '2'
                        type: 'operator'
                        Operation: 'multiply'
                        type: 'number'
                        Value: '4'
                        type: 'operator'
                        Operation: 'subtract'
                  }

                  type: 'number'
                  Value: '4'
            }

            type: 'operator'
            Operation: 'divide'
      }]
}
```

```
}

------Binary Code------
00101000 00101000 00110010 00101011 00110011 00101001 00101010 00101000 00110010 00101101 00110100 00101001 00101111 00110100 00101001
```

Sample Input :- (((4-5)*9)

```
INPUT:
(((4-5)*9)

------Breaking your code into Tokens------

Token   Type
 (     Parentheses
 (     Parentheses
 (     Parentheses
 4     Integer
 -     Operator
 5     Integer
 )     Parentheses
 *     Operator
 9     Integer
 )     Parentheses
```

```
------Checking for Balanced Brackets------
Number of Brackets are Not Balanced
So Abstract Syntax Tree cannot be Created

------Binary Code------
00101000 00101000 00101000 00110100 00101101 00110101 00101001 00101010 00111001 00101001
```

FEASIBILITY AND FUTURE SCOPE

In future I want to implement Code Optimization in this Compiler. Even the GUI of the compiler is very basic that has to be improved in future. Also, I want this compiler to compile different programming languages in future.

CONCLUSION

In my compiler the process of tokens creation and parsing is independent of the system and can be done using basic GUI. The Output is generated and stored in 'Output.txt' file and Binary file created is stored in 'Binary.bin' file.