

REPORT ON

CD Mini Project Carried out on

# DESIGN A COMPILER FOR HYPOTHETICAL PROGRAM TO FIND NUMBER OF DIGITS IN A BINARY NUMBER

*Submitted to*

# NMAM INSTITUTE OF TECHNOLOGY, NITTE

(An Autonomous Institution under VTU, Belagavi)

*In partial fulfillment of the requirements for the award of the*

Degree of Bachelor of Engineering in Computer Science and Engineering

*By*

# Suvith Kumar – 4NM20CS193

# Suvarna Siddhanth Vishwanath

# – 4NM20CS192

# Shreyas Prashant Kindalkar

# – 4NM20CS176

Submitted to,

**Ms. Minu P Abraham**

**Assistant Professor GD-II**

**Department of CSE, NMAMIT-Nitte**

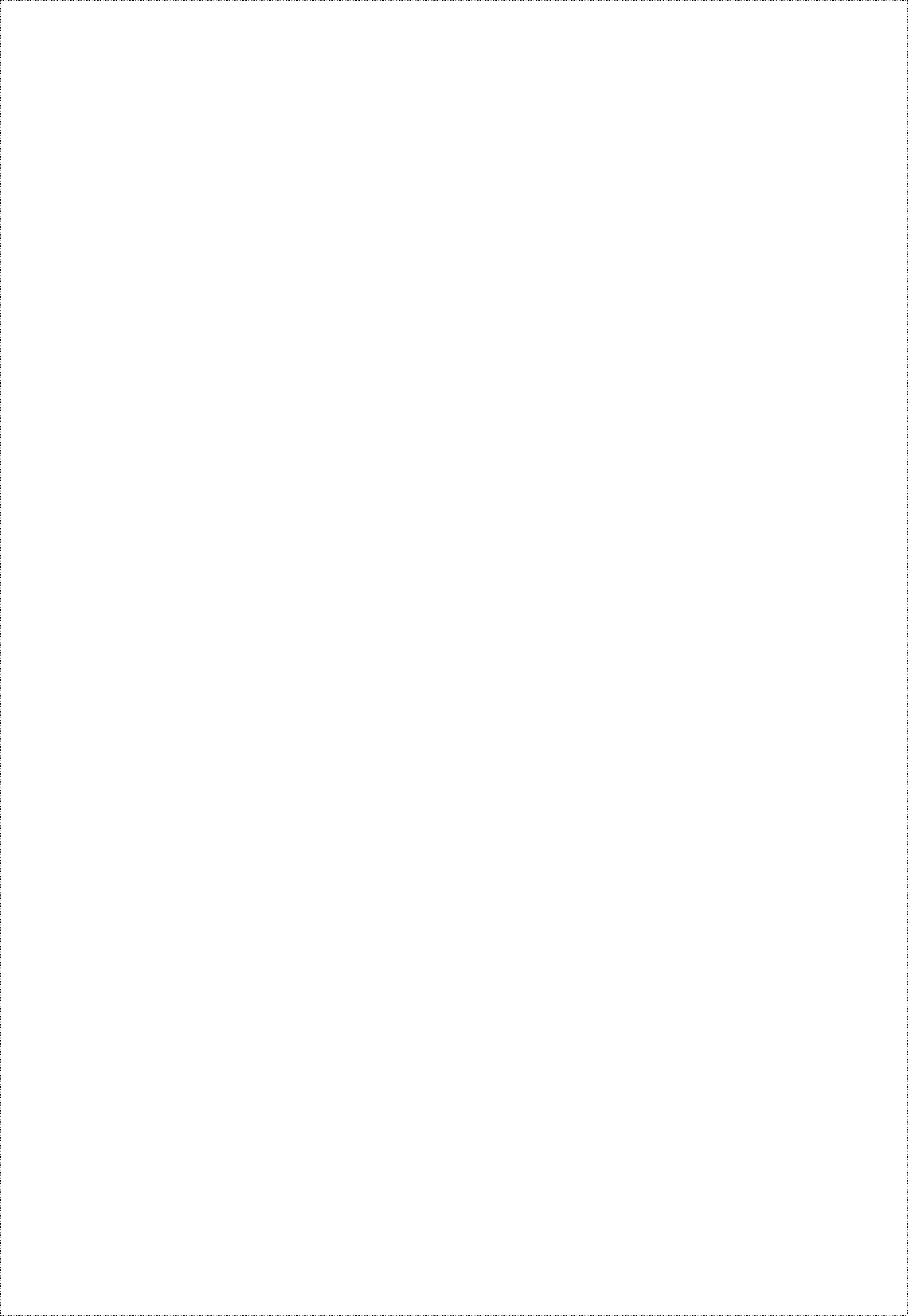
CERTIFICATE

*This is to certify that the Mr. Suvith Kumar bearing USN of 4NM20CS193, Mr. Suvarna Siddhanth Vishwanath bearing USN of 4NM20CS192 and Mr.Shreyas Prashant Kindalkar bearing USN of 4NM20CS176* *of VII semester B.E., bonafide student of NMAM Institute of Technology, Nitte, has completed CD mini project on*

*“COMPILER FOR THE GIVEN HYPOTHETICAL LANGUAGE”*

*during August 2023 -December 2023 fulfilling the partial requirements for the award of degree of Bachelor of Engineering in* ***Computer Science and Engineering*** *at NMAM Institute of Technology, Nitte****.***

*Name and Signature of Mentor Signature of HOD*



**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible because “Success is the abstract of hard work and perseverance, but steadfast of all is encouraging guidance.” So I acknowledge all those whose guidance and encouragement served as a beacon light and crowned my efforts with success.

I would like to thank our principal **Prof. Niranjan N. Chiplunkar** firstly, for providing us with this unique opportunity to do the mini project in the 7th semester of Computer Science and engineering.

I would like to thank my college administration for providing a conductive environment and also suitable facilities for this mini project. I would like to thank our HOD **Dr. Jyothi Shetty** for showing me the path and providing the inspiration required for taking the project to its completion. It is my great pleasure to thank my mentor **Ms. Minu P Abraham**

for her continuous encouragement, guidance, and support throughout this project.

Finally, thanks to staff members of the department of CSE, my parents and friends for their honest opinions and suggestions throughout the course of our mini project.

### Suvith Kumar(4NM20CS193) Suvarna |Siddhant Vishwanath(4NM21CS192)

### Shreyas Prashant Kindalkar(4NM21CS176)

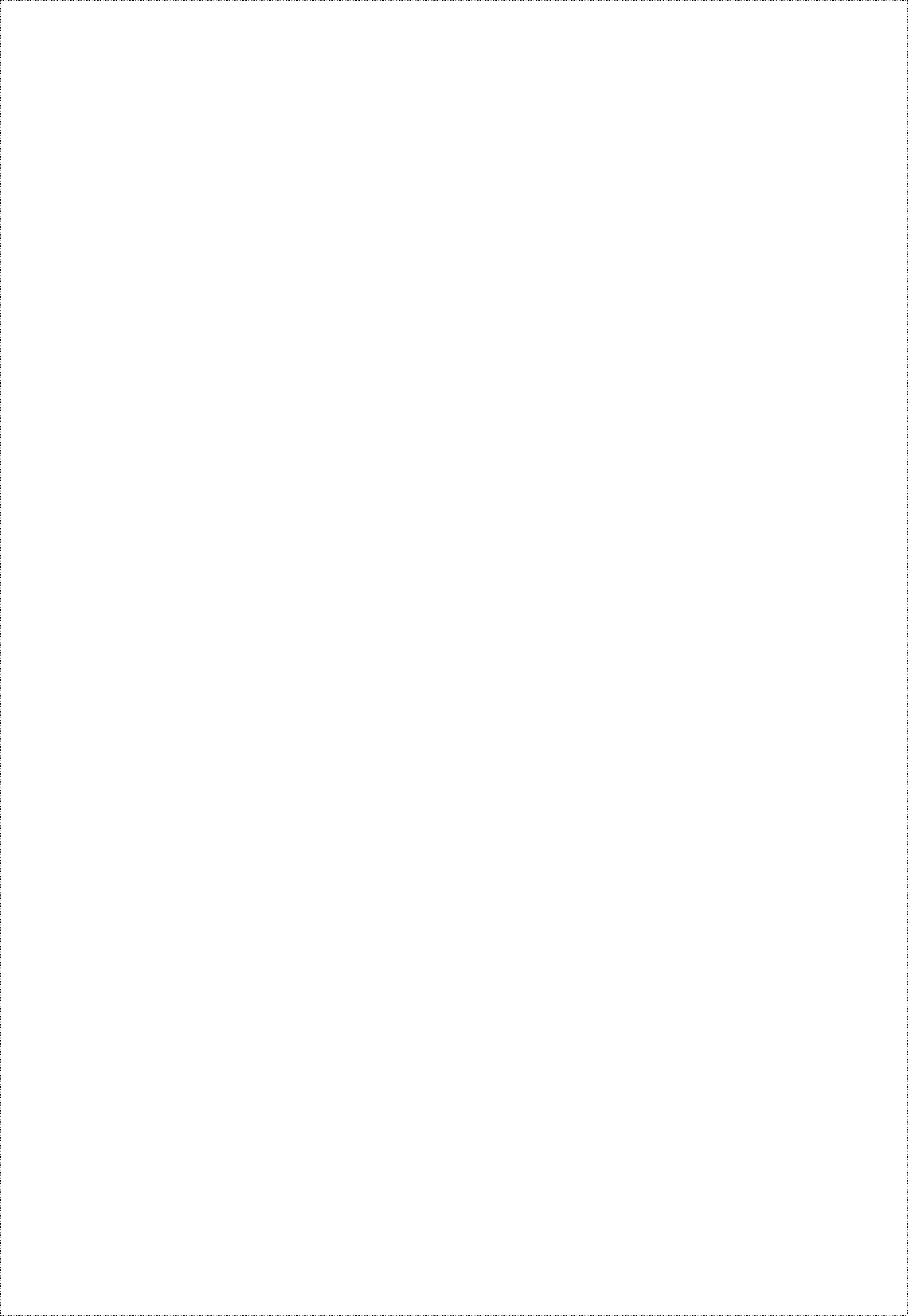
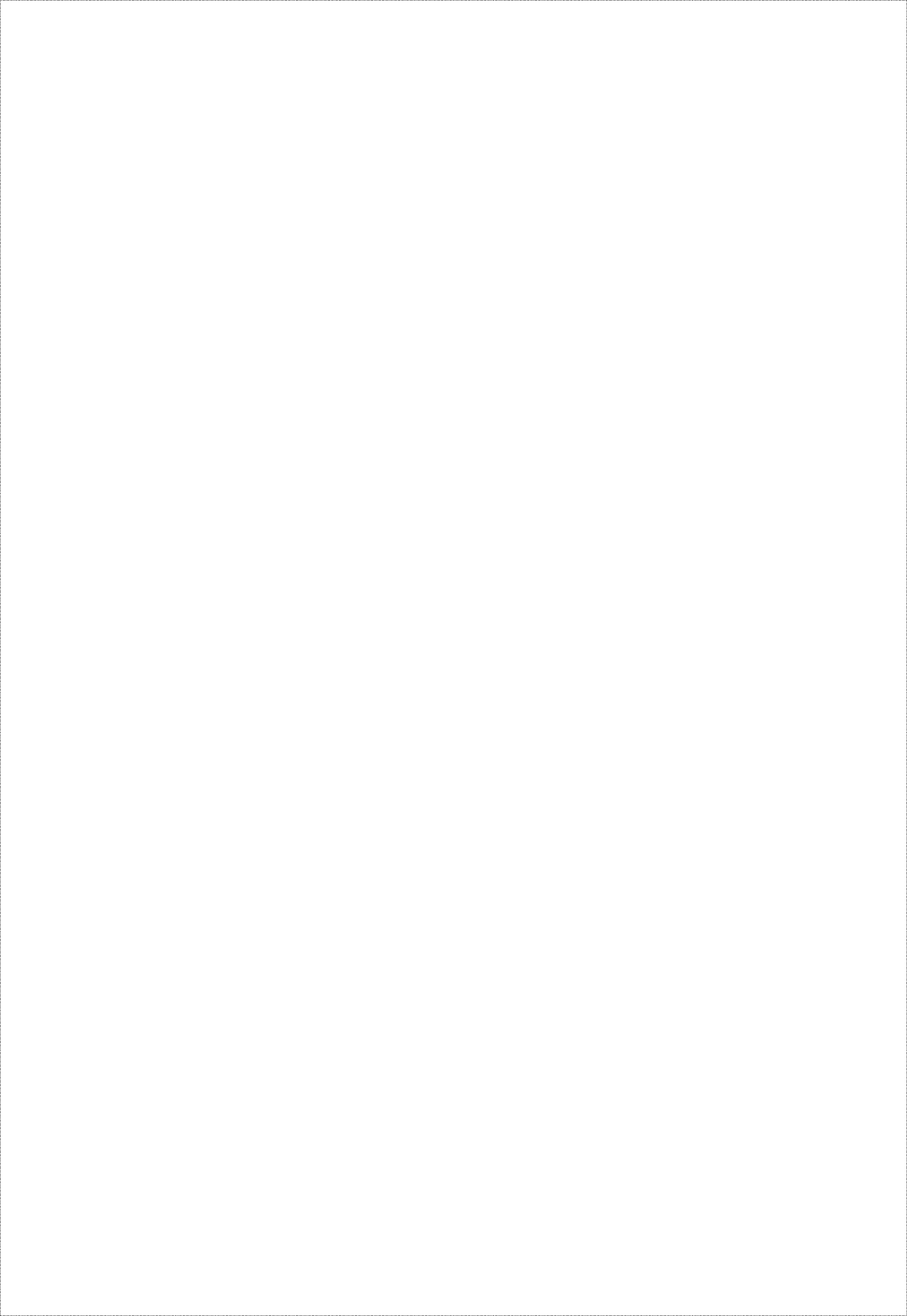


TABLE OF CONTENTS

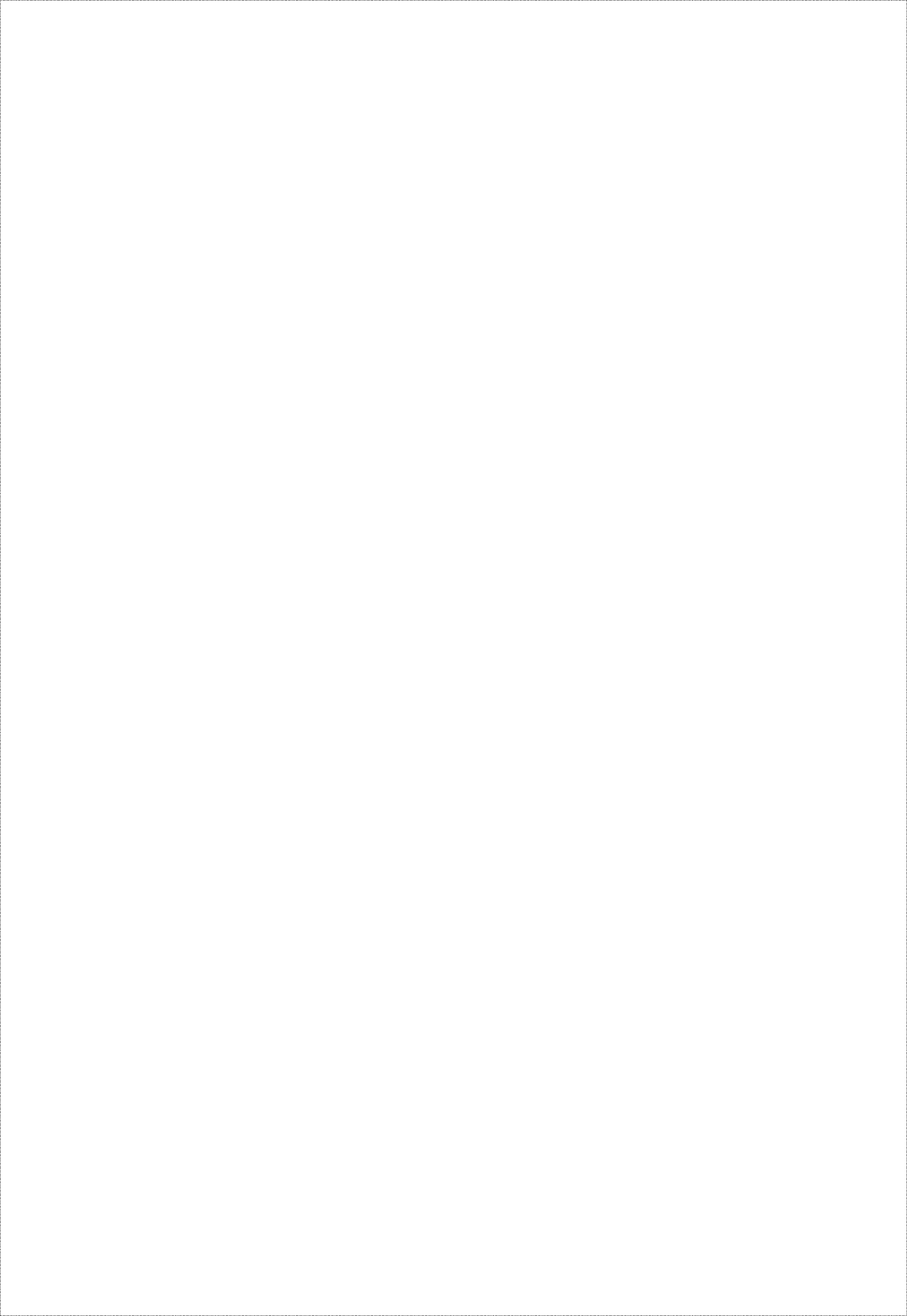
|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Contents** | **Page No.** |
| **1.** | **Abstract and Objectives** | 5-6 |
| **2.** | **Introduction** | 7-9 |
|  | * Compiler |  |
|  | * Phases of Compiler * Classification of compiler Phases |  |
| **3.** | **Implementation** | 10-11 |
| **4.** | **Screenshots** | 12 – 28 |
| **5.** | **Result & Conclusion** | 29-33 |



# ABSTRACT

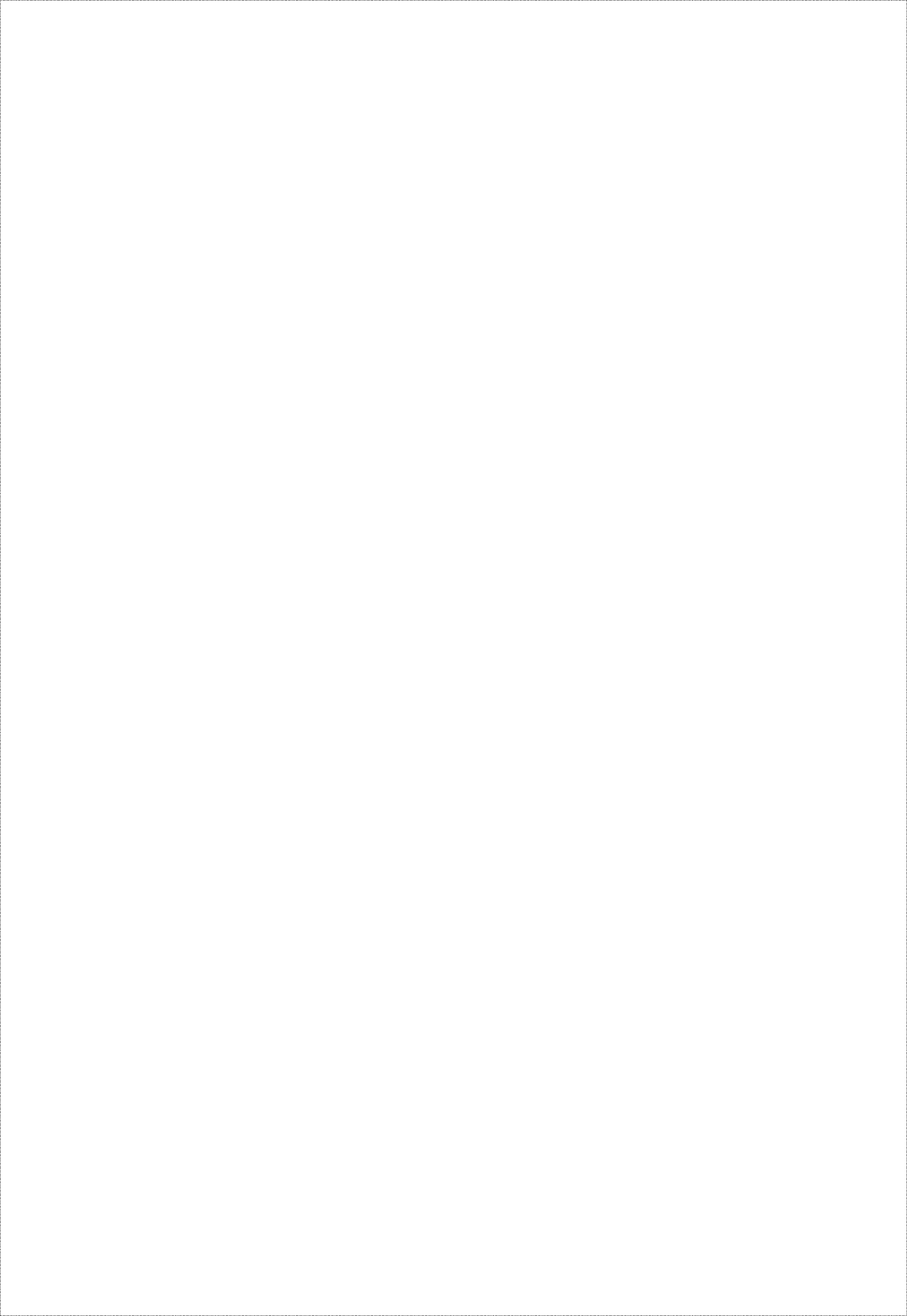
Programming languages are notations for describing computations to people and to machines. The world as we know it depends on programming languages, because all the software running on all the computers was written in some programming language. But, before a program can be run, it first must be translated into a form in which it can be executed by a computer. The software systems that do this translation are called compilers.

A compiler is a computer program that translates computer code written in one programming language into other language. We have designed a compiler for the given hypothetical program to find the number of digits in a binary number. This report contains the problem statement, the objectives and the methodology used to implement the project.



# OBJECTIVES

* Design a Compiler for the given problem statement.
* To develop a grammar for the given problem statement.
* To design and develop LL(1) parsing table for the grammar.
* To validate an input string against our grammar.



# INTRODUCTION

## COMPILER

A compiler is a software that takes a program written in a high-level language and translates it into an equivalent program in a target language. Most specifically a compiler takes a computer program and translates it into an object program. Some other tools associated with the compiler are responsible for making an object program into executable form.

**Source program** – It is normally written in a high-level programming language. It contains a set of rules, symbols and special words used to construct a computer program.

**Target program –** It is normally the equivalent program in machine code. It contains the binary representation of the instructions that the hardware of computer can perform.

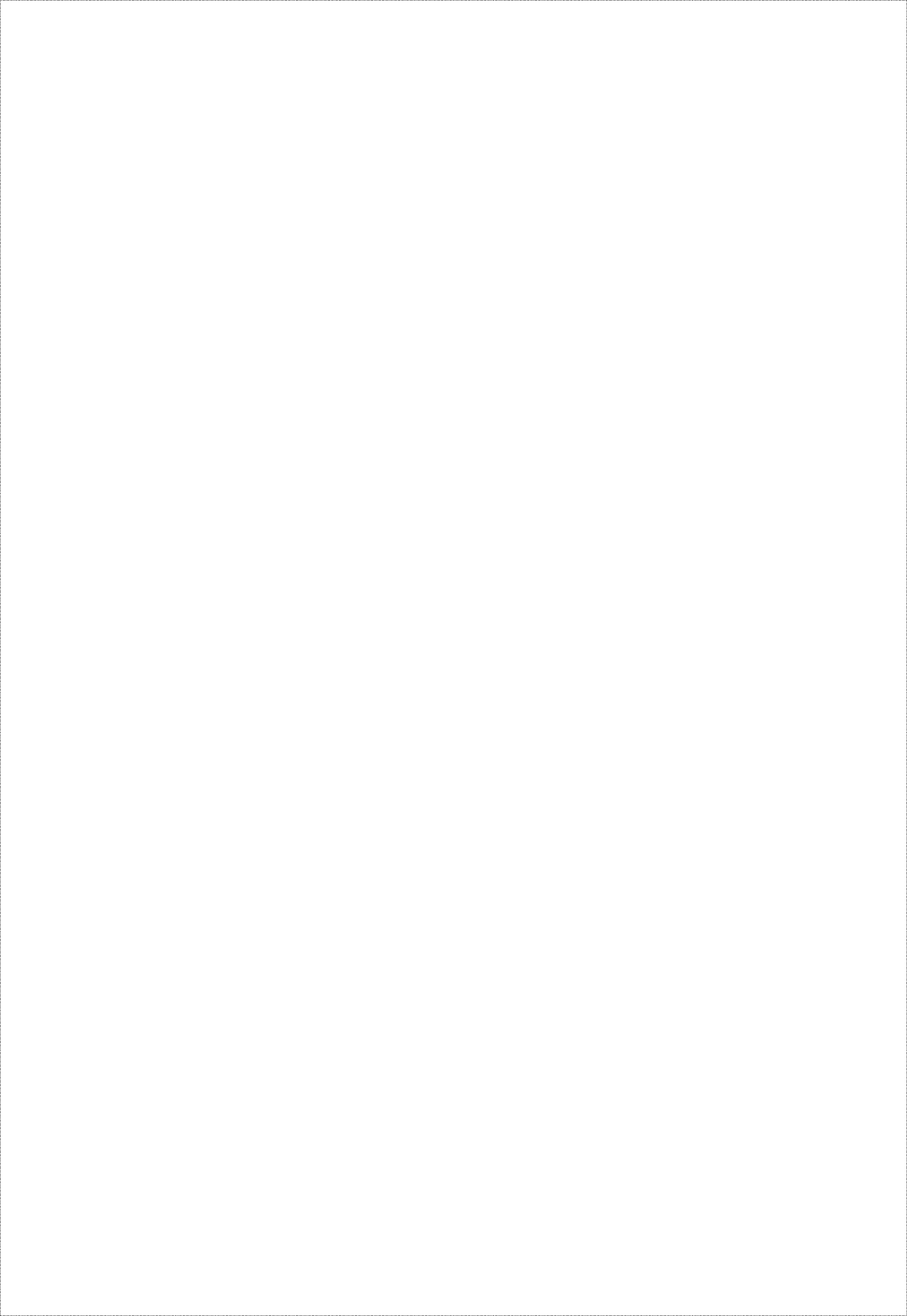
**Error Message** – A message issued by the compiler due to detection of syntax errors in the source program.

Compilation is a large process. It is often broken into stages. Many phases of compiler try and optimize by translating one form into a better (more efficient) form. Most of compiling is about “pattern matching” languages and tools that support pattern matching, are very useful. An efficient compiler must preserve semantics of the source program and it should create an efficientversion of the target language.

### PHASES OF COMPILERS

Typically, a compiler includes several functional parts. For example, a conventional compiler may include a lexical analyser that looks at the source program and identifies successive “tokens” in the source program. A conventional compiler also includes a parser or syntactical analyzer, which takes as an input a grammar defining the language being compiled and a series of actions associated with the grammar.

The syntactical analyzer builds a “parse tree” for the statements in the source program in accordance with the grammar productions and actions. For each statement in the input source program, the syntactical analyzer generates a parsetree of the source input in a recursive, “bottom-up” manner in accordance with relevant productions and actions in the grammar. Generation of the parse tree allows the syntactical analyzer to determine whether the parts of the source program comply with the grammar. If not, the syntactical analyzer generates an error



### CLASSIFICATION OF COMPILER PHASES

There are two major parts of a compiler phases: Analysis and Synthesis.

In analysis phase, an intermediate representation is created from the givensource program that contains:

* Lexical Analyser
* Syntax Analyser
* Semantic Analyser

In synthesis phase, the equivalent target program is created from this intermediate representation. This contains:

* Intermediate code Generator
* Code Optimization
* Code Generation

### LEXICAL ANALYZER

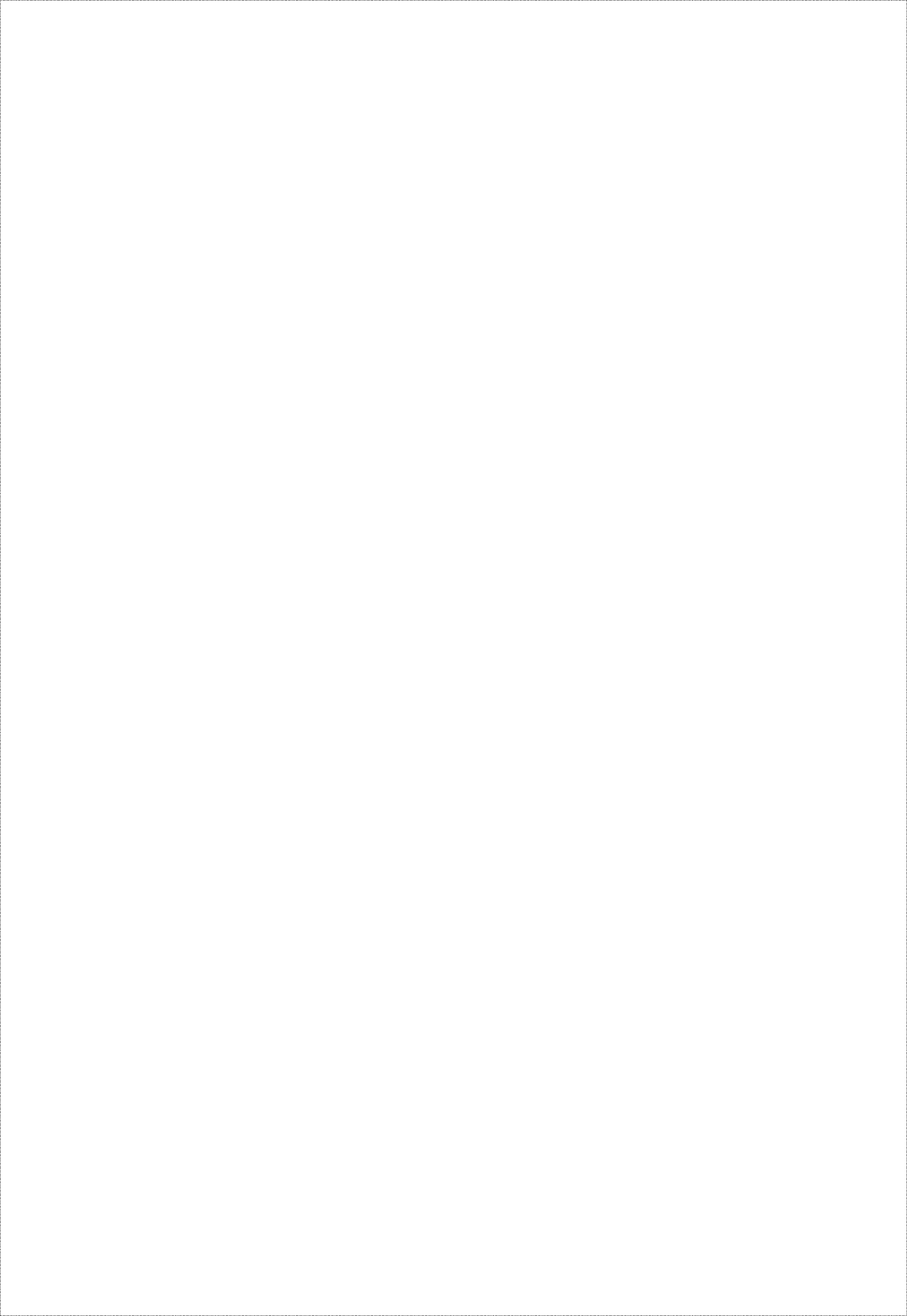
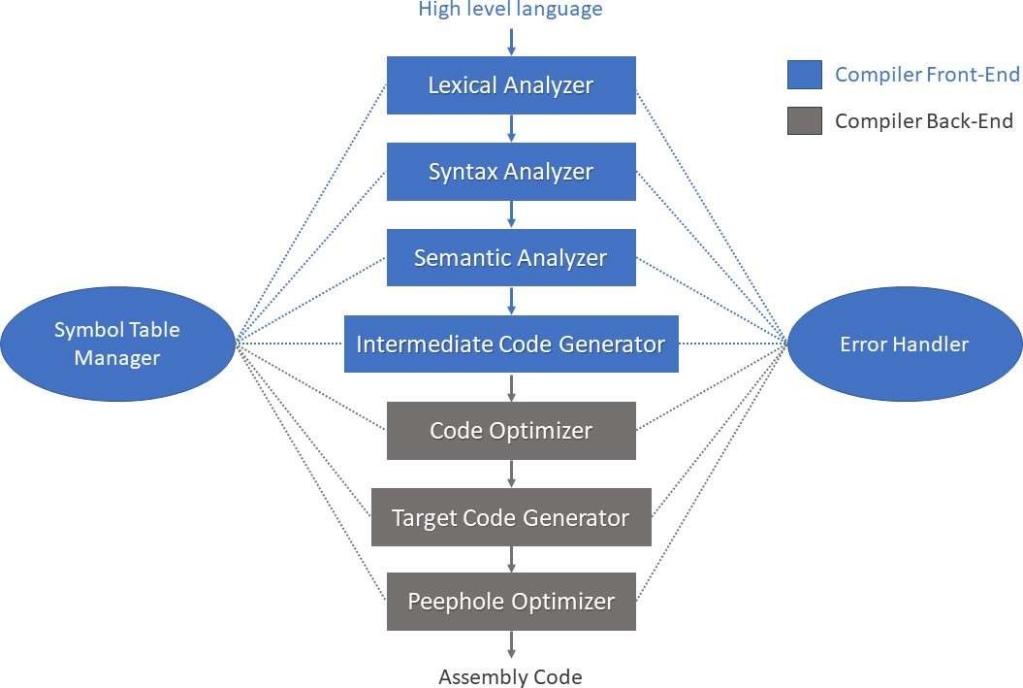
Lexical analyser takes the source program as an input and produces a string of tokens or lexemes. Lexical Analyzer reads the source program character by character and returns the tokens of the source program. Theprocess of generation and returning the tokens is called lexical analysis. Representation of lexemes in the form of tokens as:

### SYNTAX ANALYSER

A Syntax Analyzer creates the syntactic structure (generally a parse tree) of the given program. In other words, a Syntax Analyzer takes output of lexical analyser (list of tokens) and produces a parse tree. A syntaxanalyzer is also called as a parser. The parser checks if the expression made by the tokens is syntactically correct.

### SEMANTIC ANALYSER

Semantic analyser takes the output of syntax analyser. Semantic analyser checks a source program for semantic consistency with the language definition. It also gathers type information for use in intermediate-code generation.



### INTERMEDIATE CODE GENERATION

After semantic analysis, the compiler generates an intermediate code ofthe source code for the target machine. It represents a program for someabstract machine. It is in between the high-level language and the machine language.

### CODE OPTIMISER

The code optimizer takes the code produced by the intermediate code generator. The code optimizer reduces the code (if the code is not already optimized) without changing the meaning of the code. The optimizationof code is in terms of time and space.

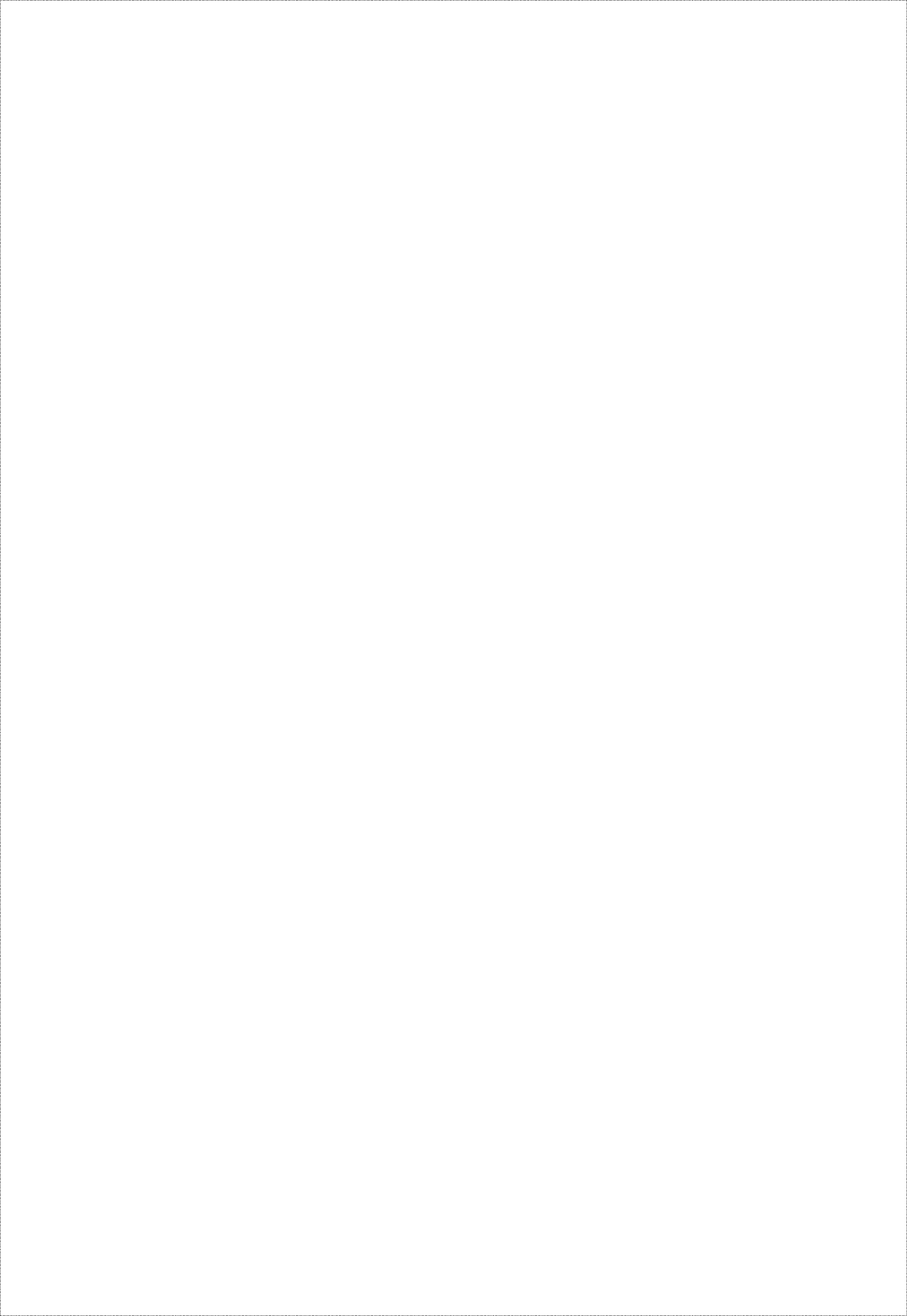
### CODE GENERATION

This produces the target language in a specific architecture. The target program is normally is an object file containing the machine codes. Memory locations are selected for each of the variables used by the program.

### SYMBOL TABLE

It is a data-structure maintained throughout all the phases of a compiler. All the identifiers’ names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

Figure 1: Phases of Compiler



# IMPLEMENTATION

**Problem Statement:**

int main() begin

int count=1; while(n>1)

count=count+1; n=n/2;

end while return count end

## PHASE -I

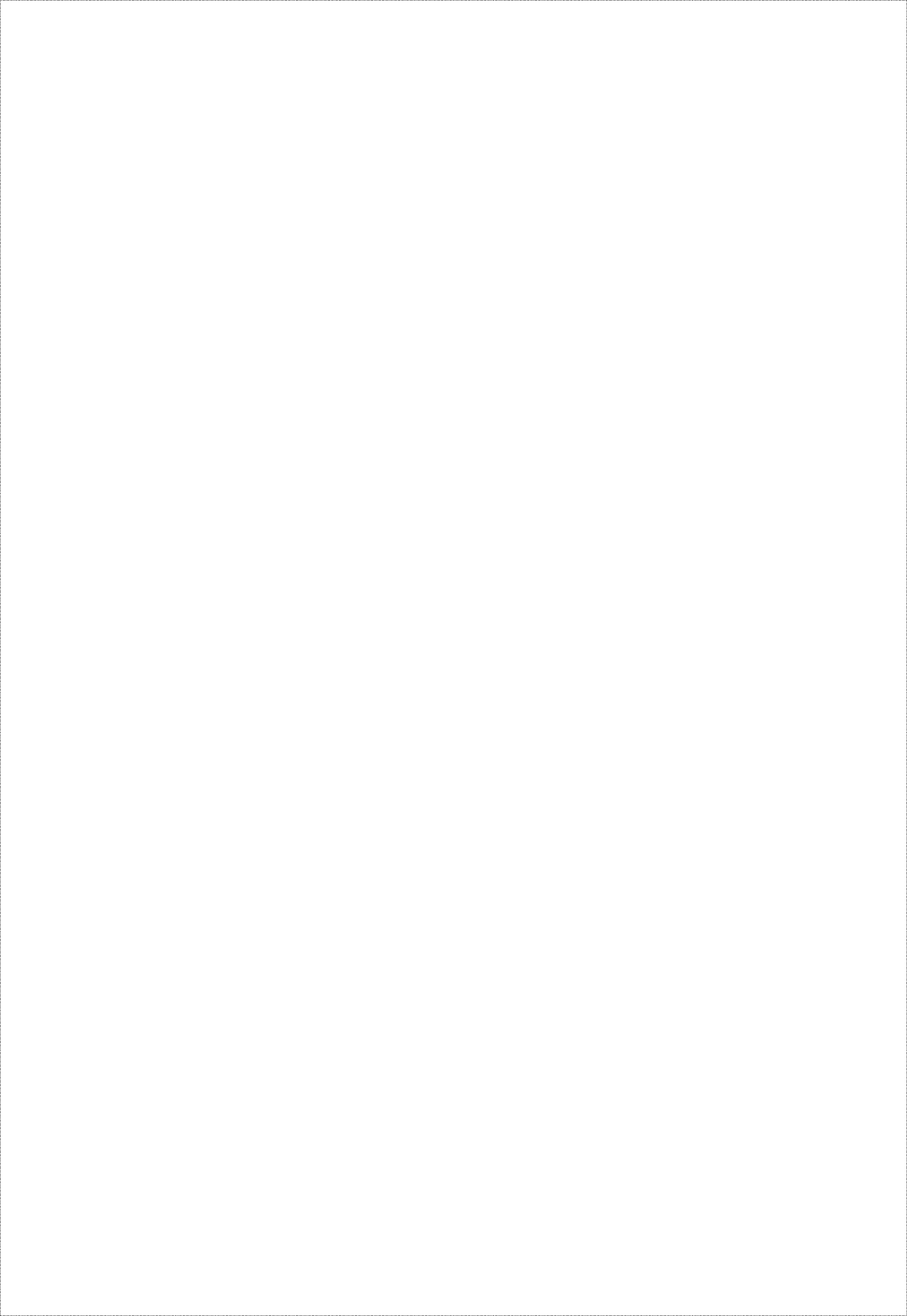
### LEXICAL ANALYSIS

Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors that are written in the form of sentences.The lexical analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with the syntax analyser. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyser when it demands. The main purpose of lexical analysis is to make life easier for the subsequent syntax analysis phase.

### Token:

Token is a sequence of characters that can be treated as a single logical entity.Typical tokens are:

1. Identifiers
2. keywords
3. operators
4. special symbols



1. constants

### Pattern:

A set of strings in the input for which the same token is produced as output.This set of strings is described by a rule called a pattern associated with the token.

**Lexeme**: A lexeme is a sequence of characters in the source program that ismatched by the pattern for a token.

### SYNTAX ANALYSIS

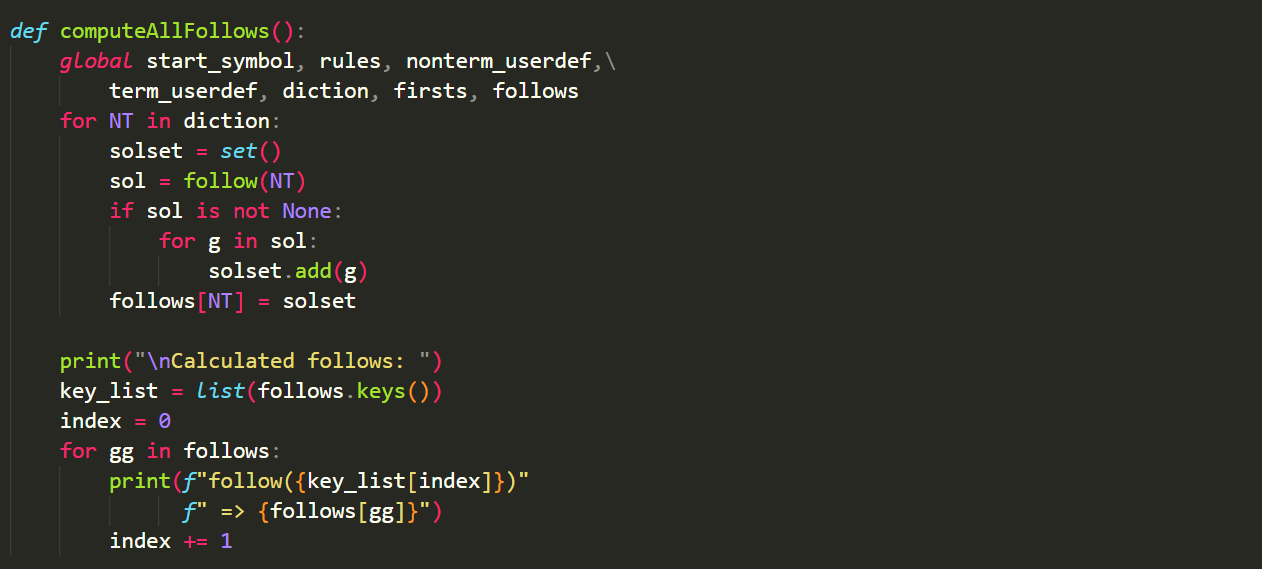
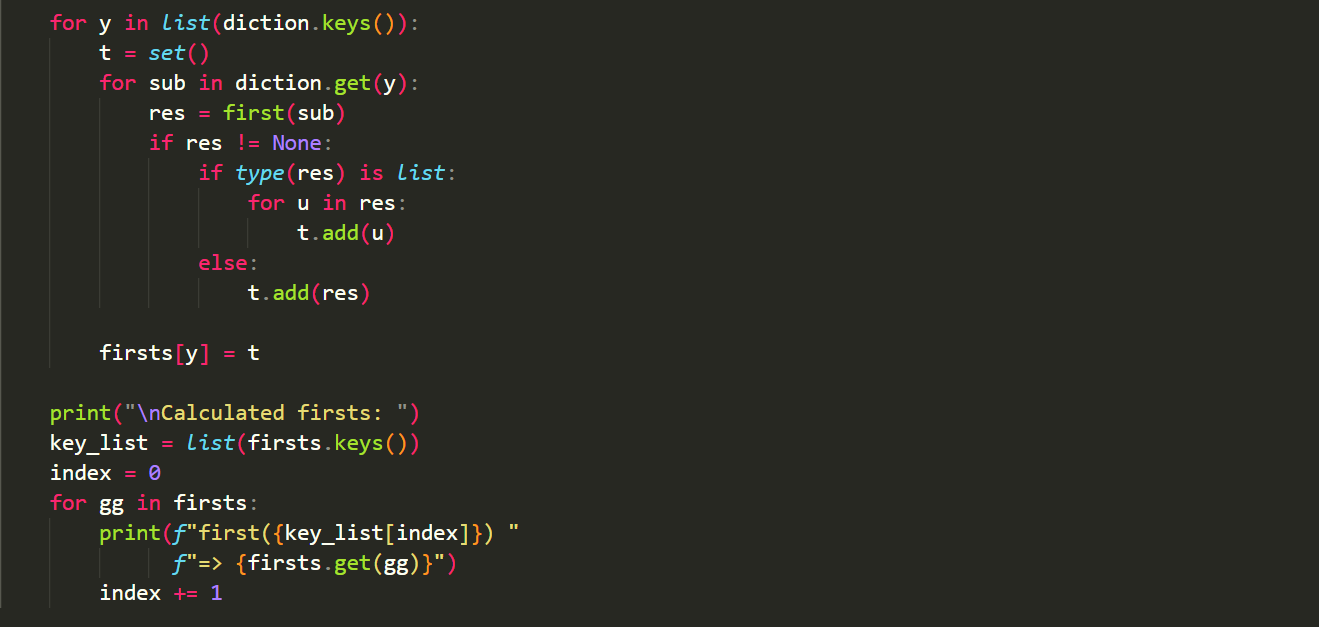
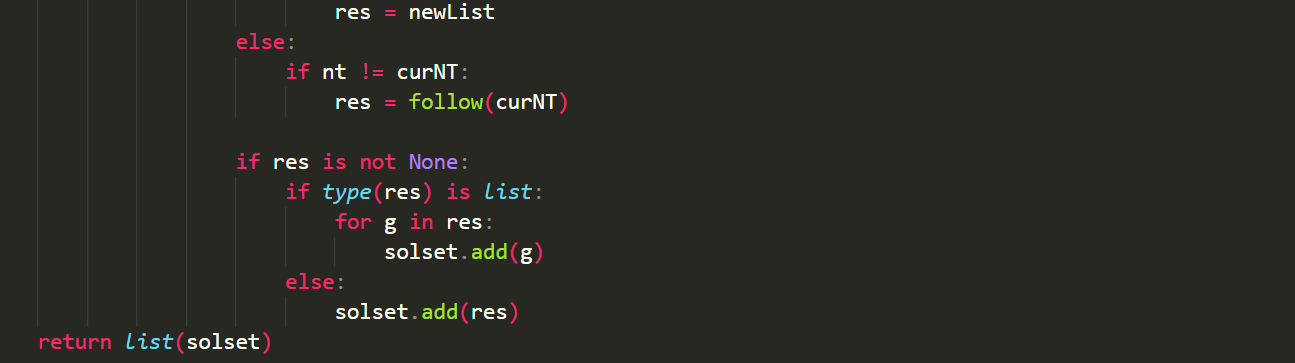
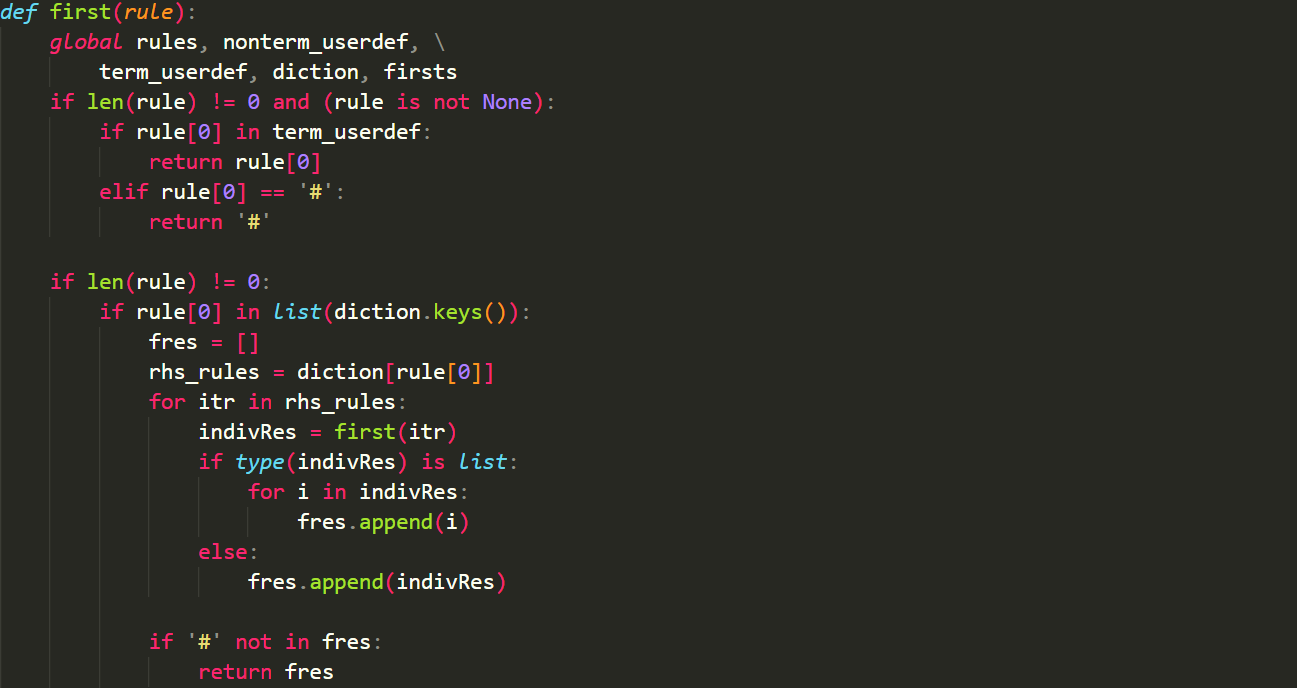
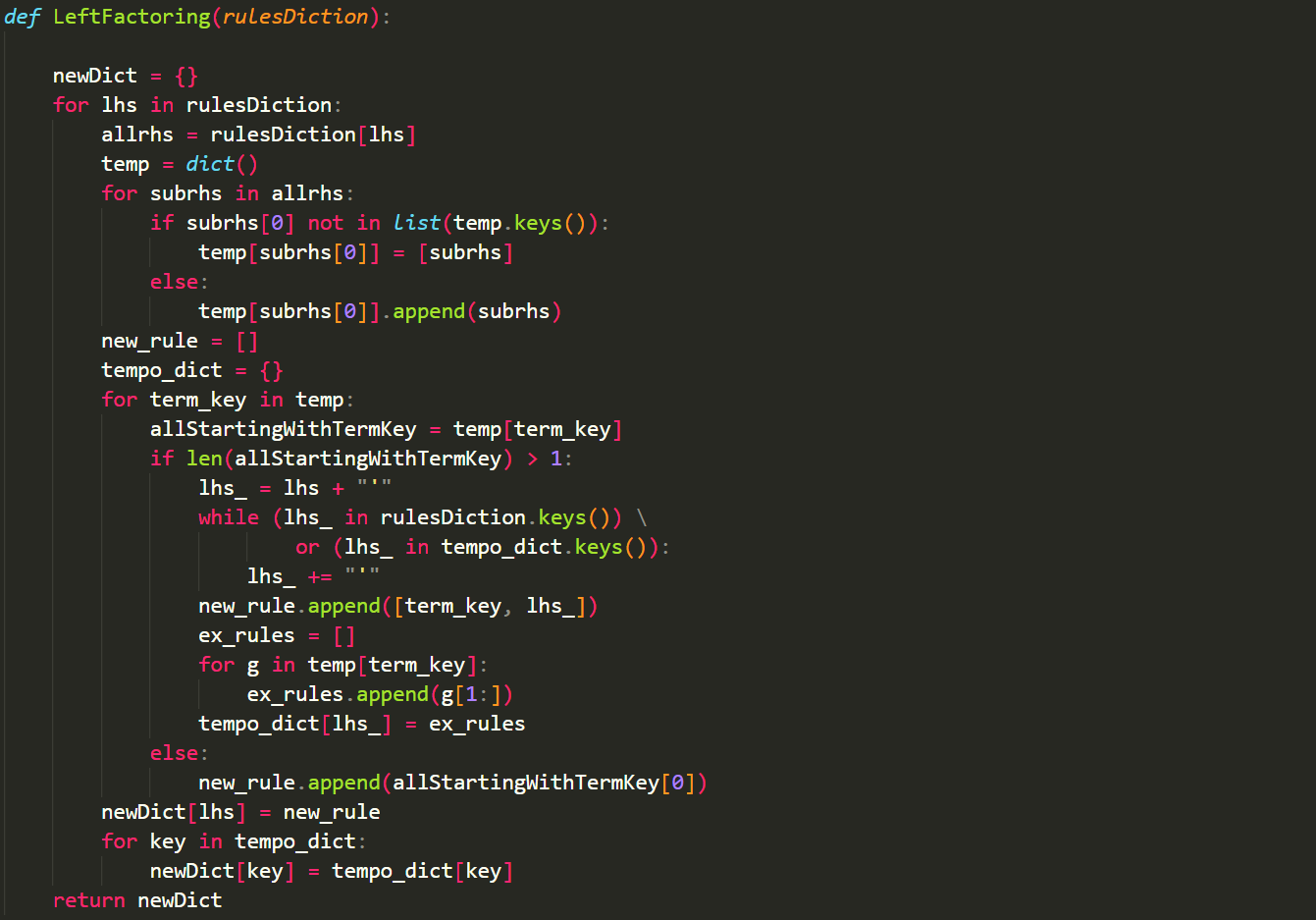
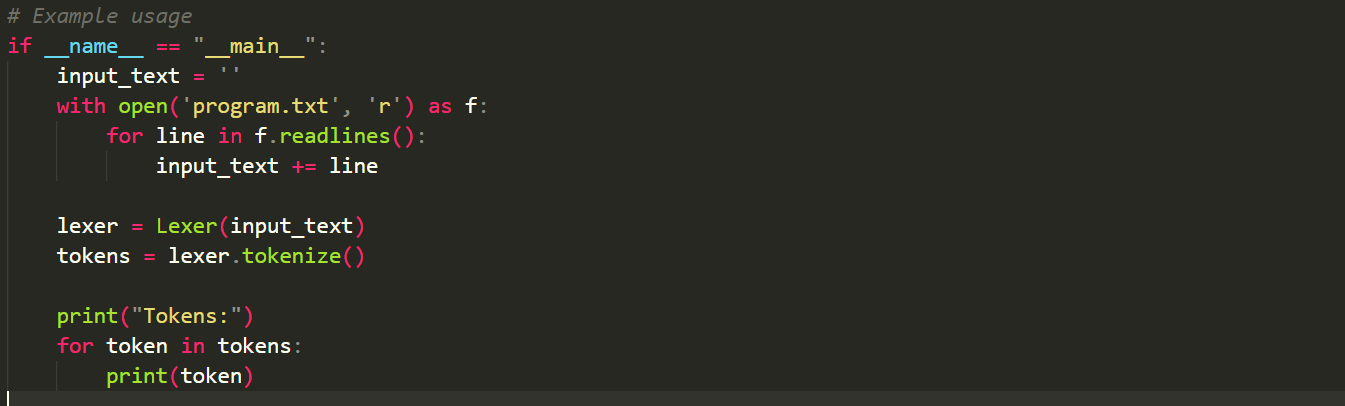
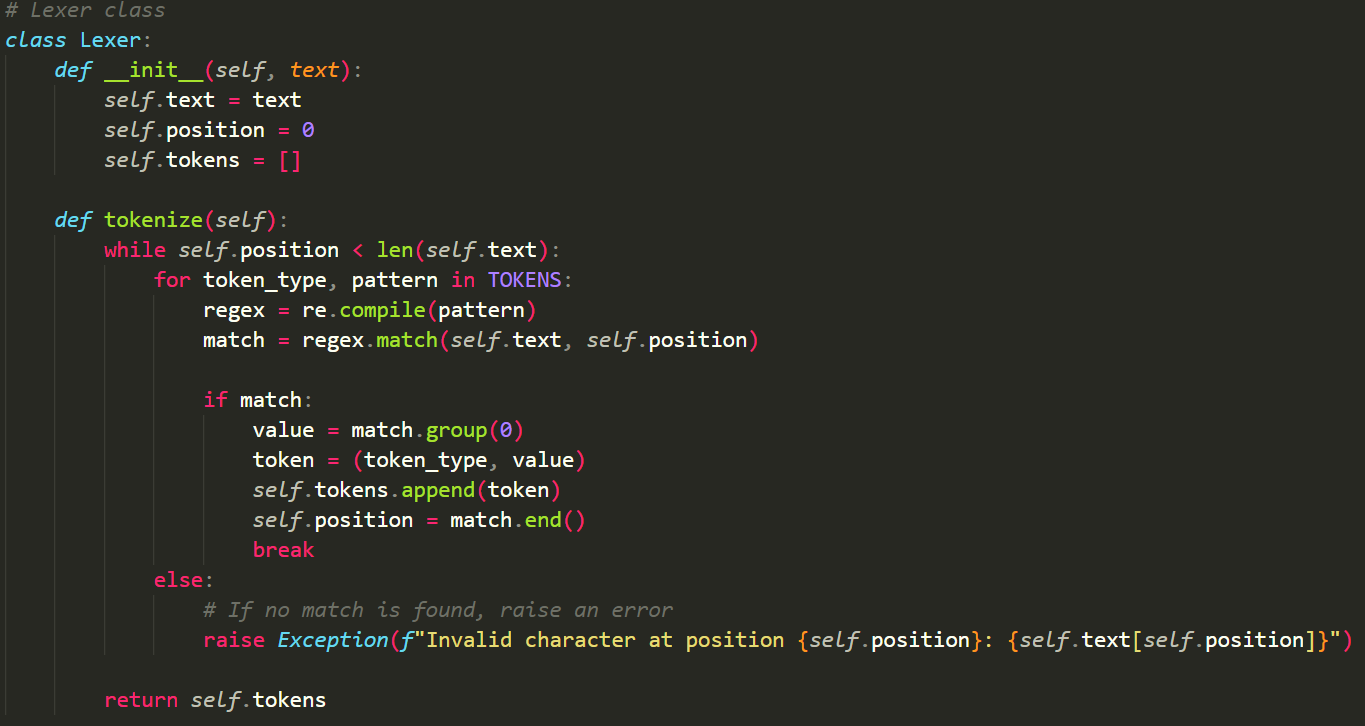
In our compiler model, the parser obtains a string of tokens from the lexical analyser, as shown in the figure below, and verifies that the string of token names can be generated by the grammar for the source language. We expect the parser to report any syntax errors in an intelligible fashion and to recover from commonly occurring errors to continue processing the remainder of the program. Conceptually, for well-formed programs, the parser constructs a parse tree and passes it to the rest of the compiler for further processing

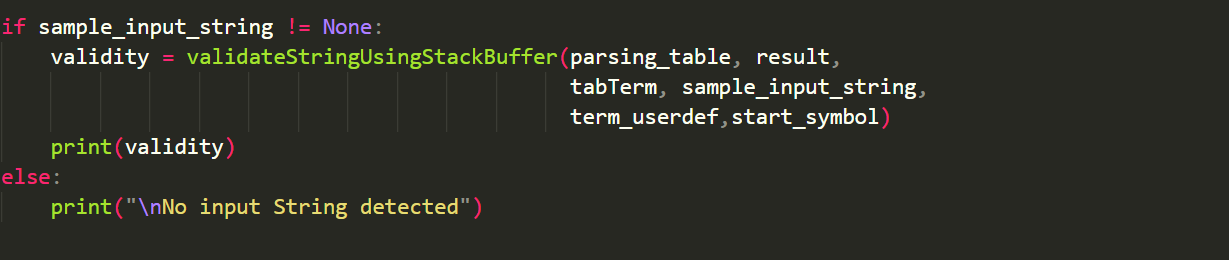
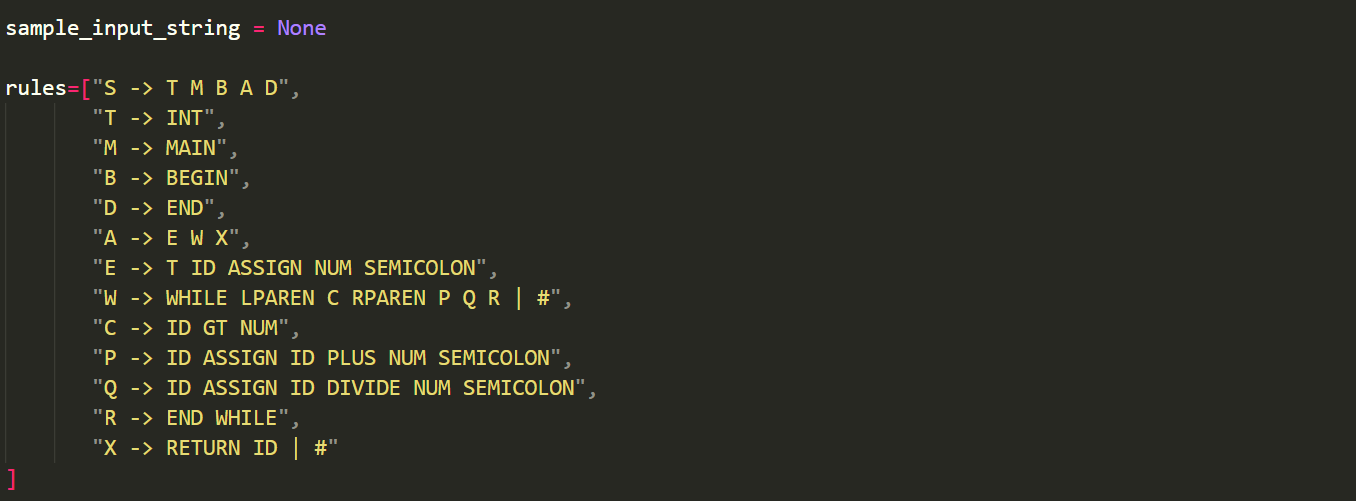
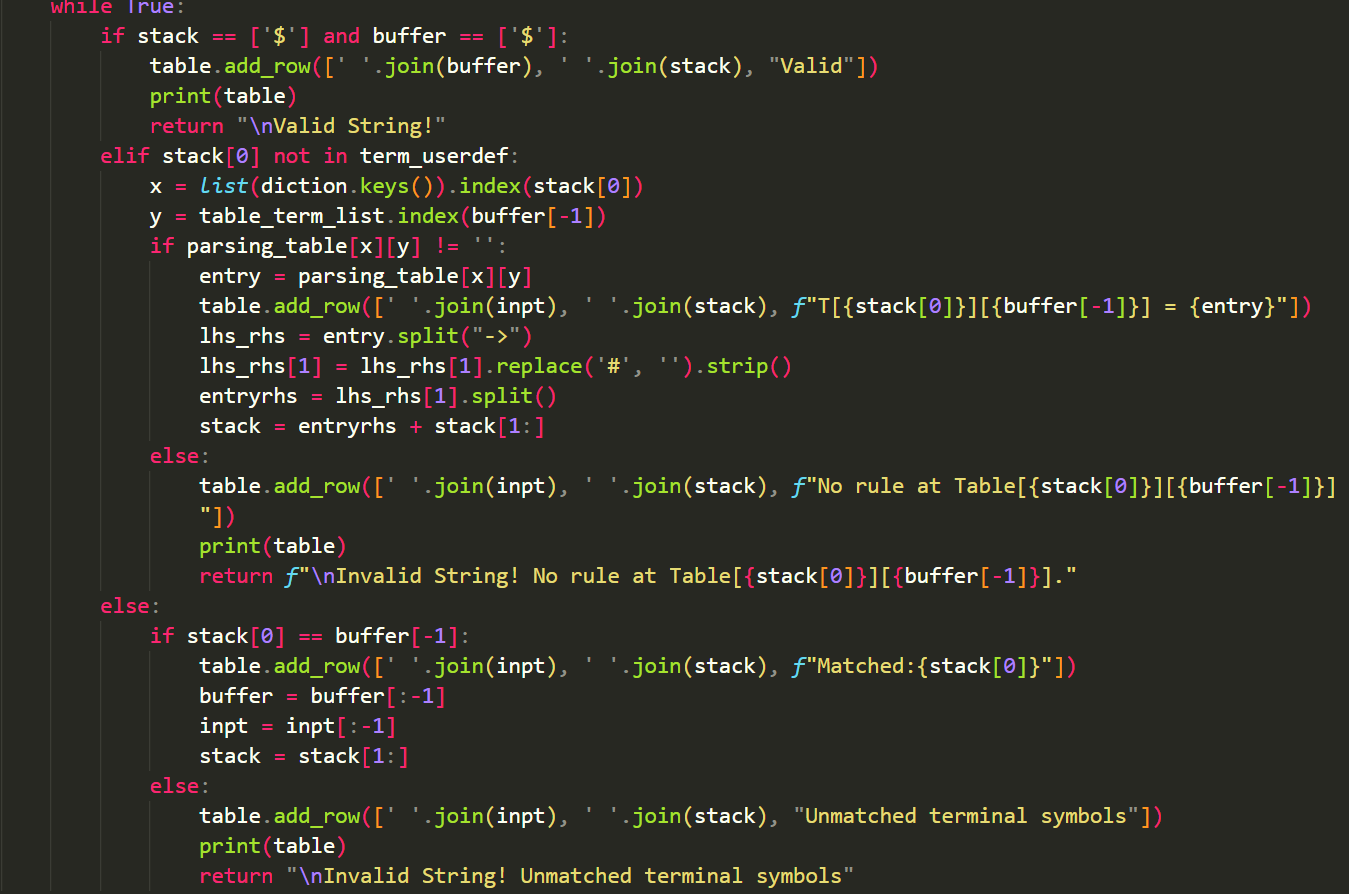
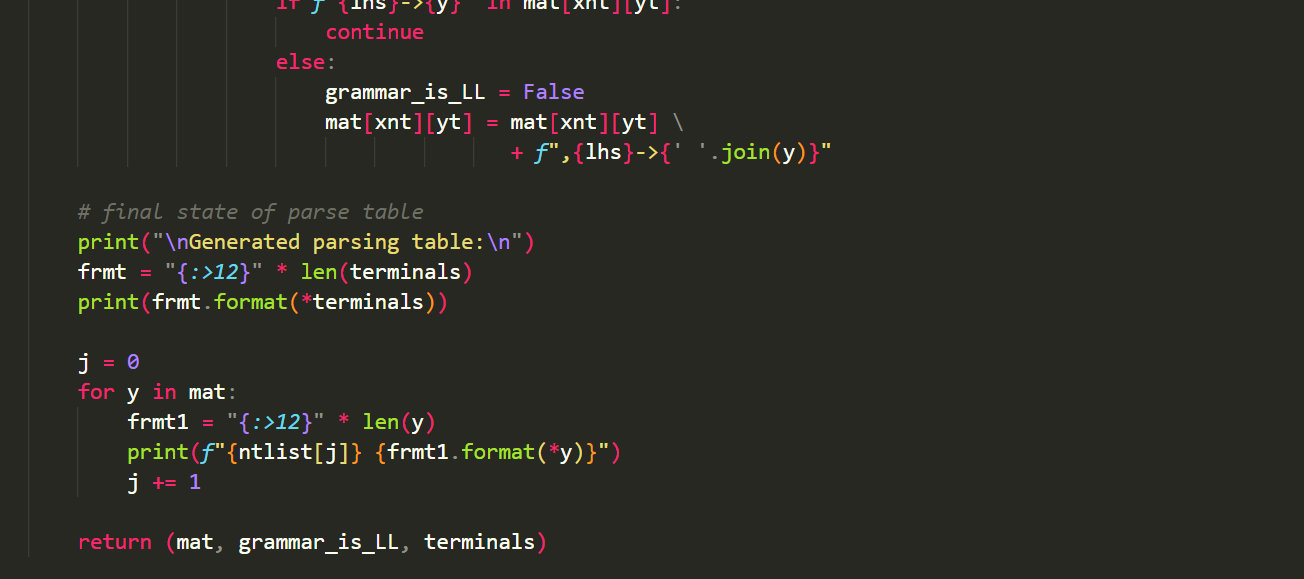
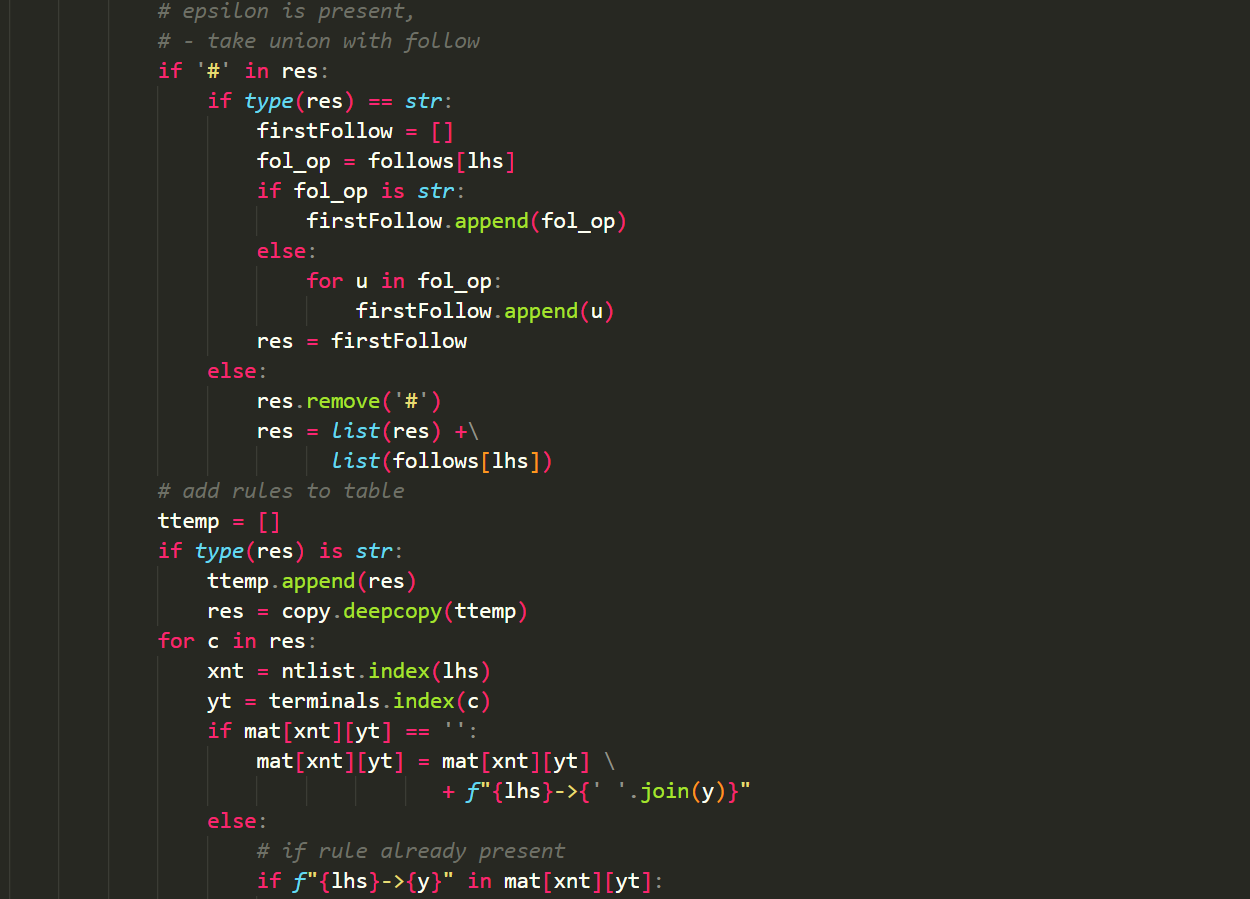
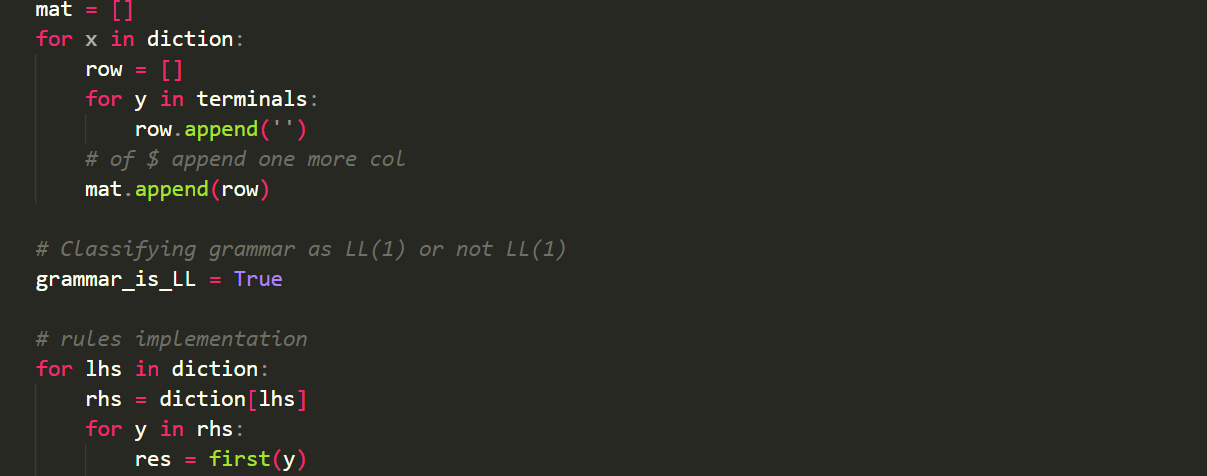
### Parser:

Parser is that phase of compiler which takes token string as input and with the help of existing grammar, converts it into the corresponding parse tree. Parser isalso known as Syntax Analyzer. The parser obtains a string of tokens from the lexical analyser and verifies that the string can be the grammar for the source language. It detects and reports any syntax errors and produces a parse tree fromwhich intermediate code can be generated.

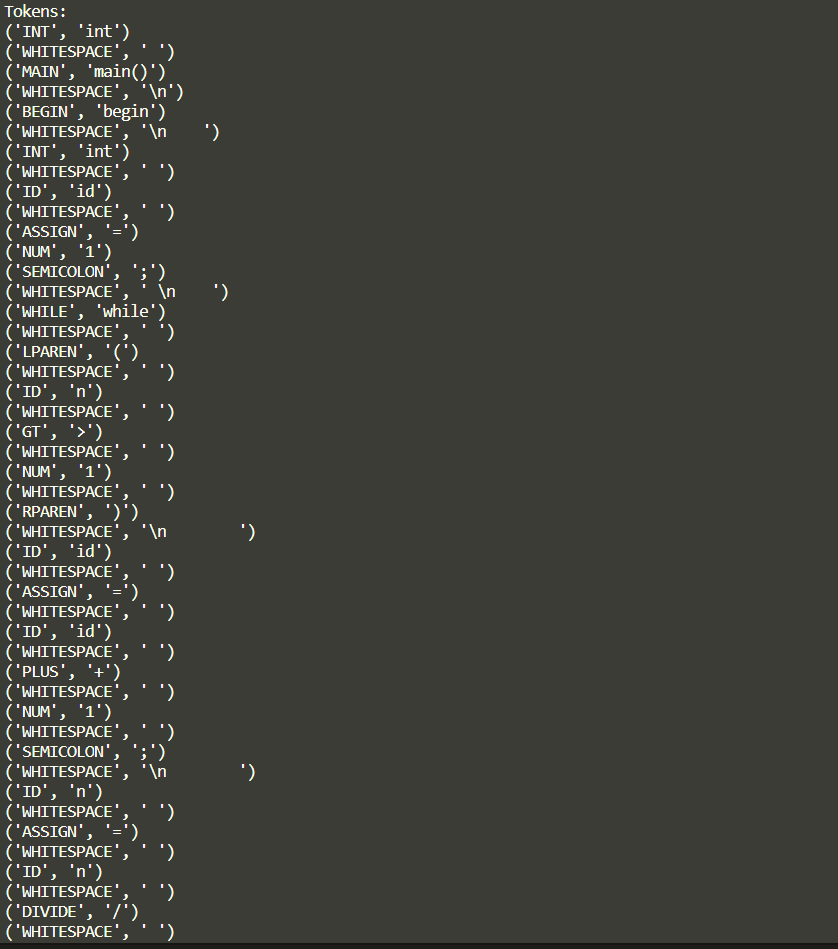
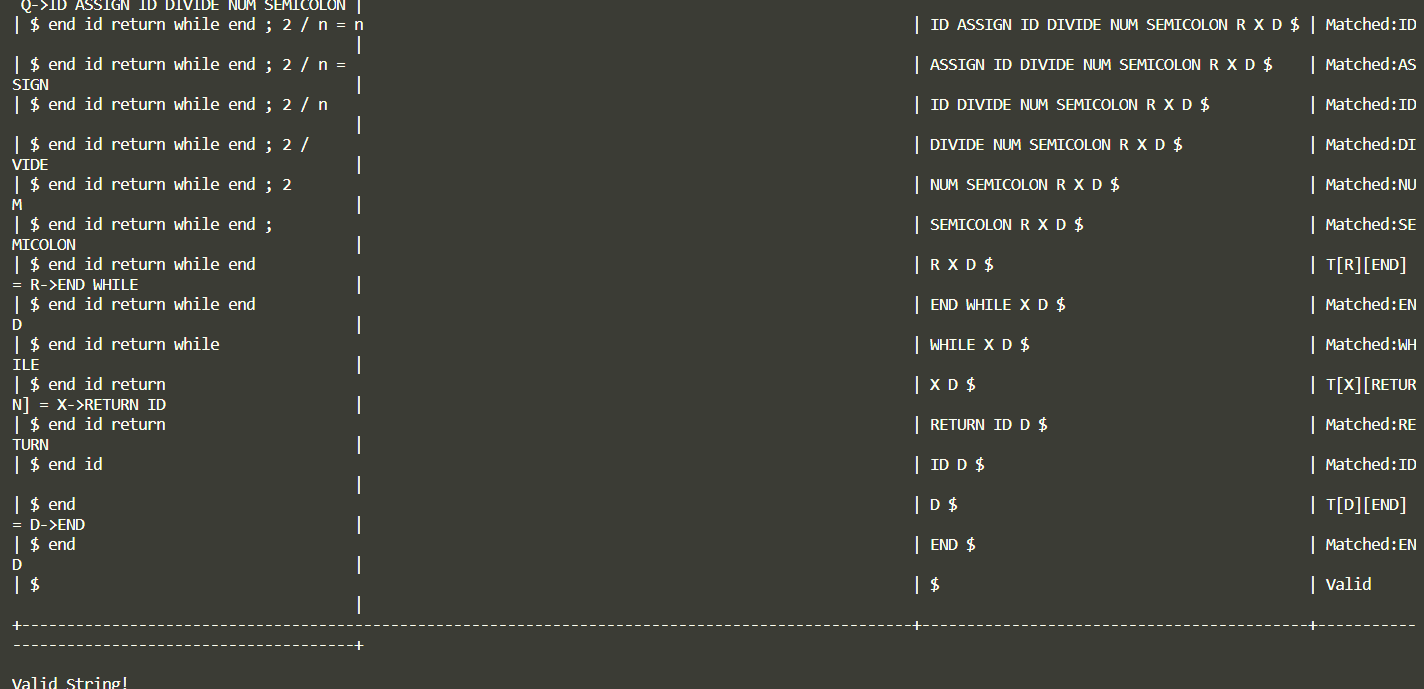
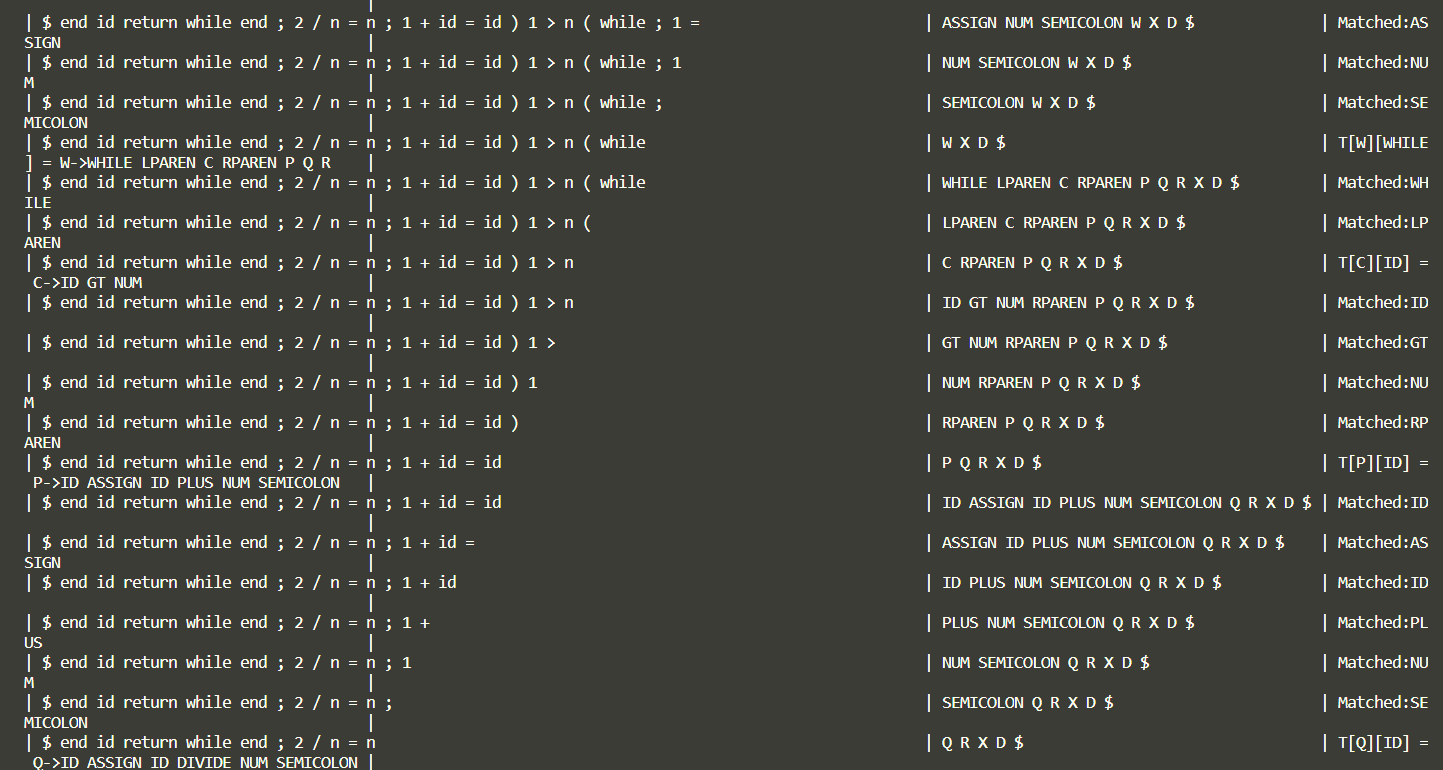
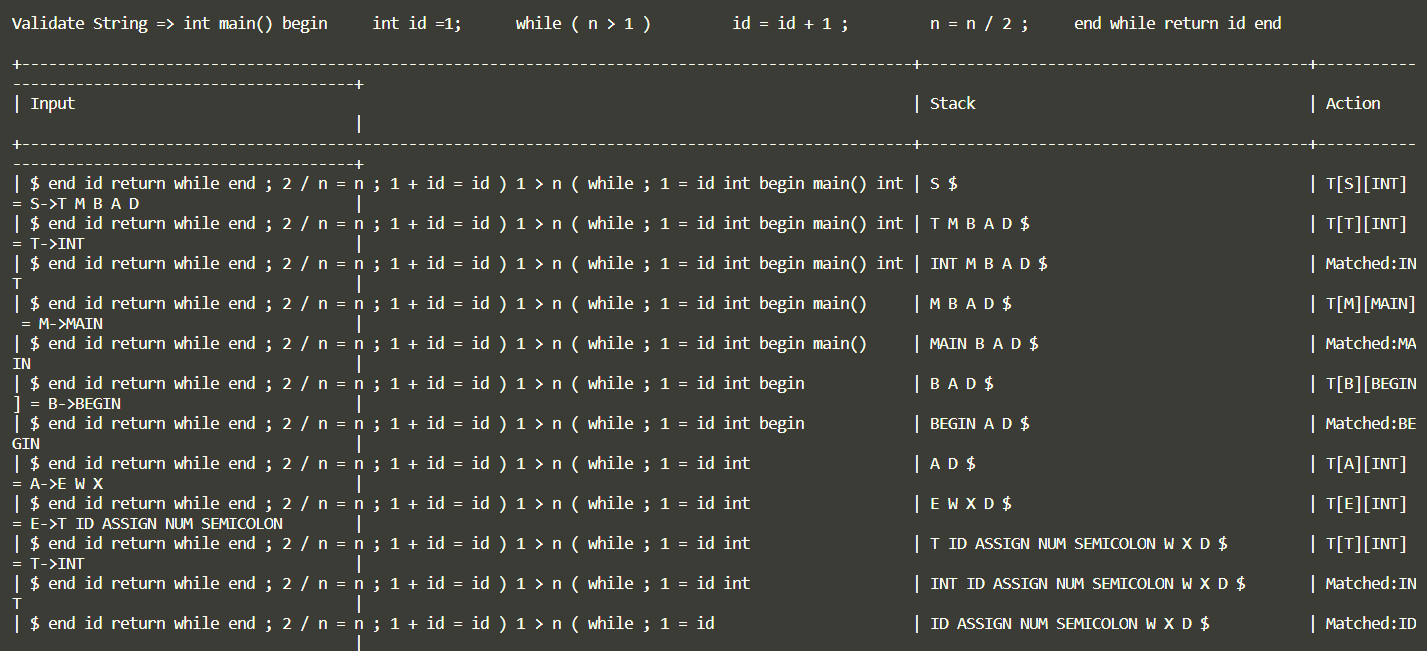
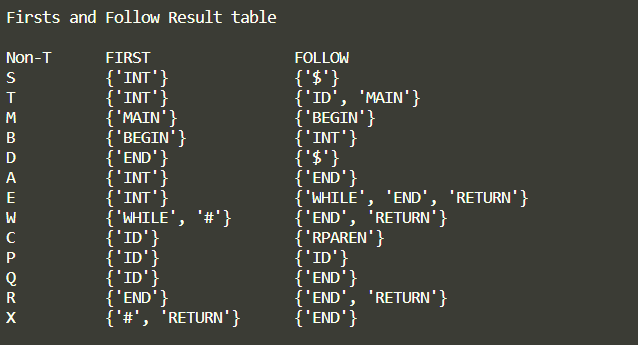
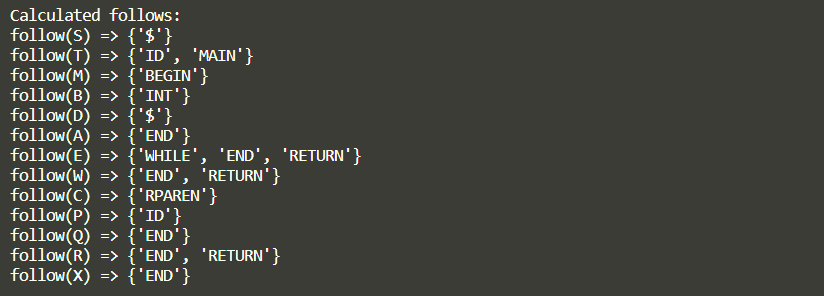
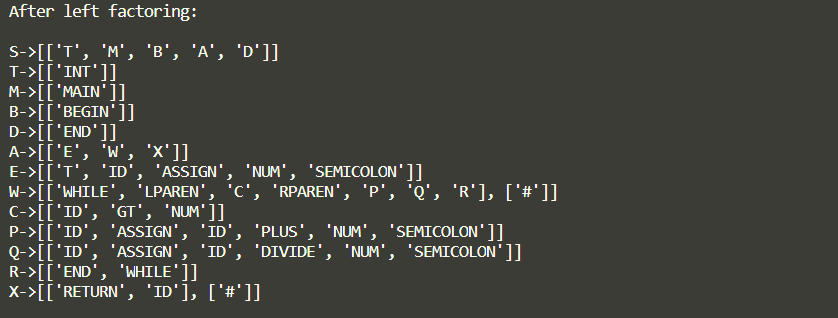
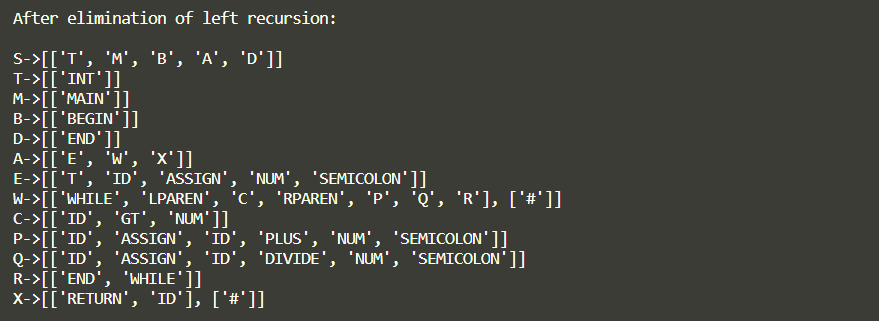
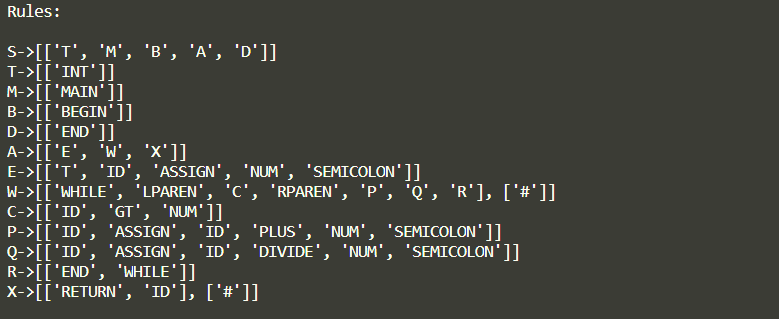
# SCREENSHOTS:

****





**RESULTS AND CONCLUSION**

****

# CONCLUSION

We have designed a compiler for the given problem statement. This is done by first creating a grammar for the given problem statement and creating a parse tree for the grammar using LL(1) parsing technique. We stored a string to be validated in an input file and used that file as input for string validation. If the string matches with the grammar, then the stack will be empty at the end of operation indicates that the string is a valid string. Else, it indicates that the string is invalid.