COMPILER DESIGN PROJECT REPORT

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.

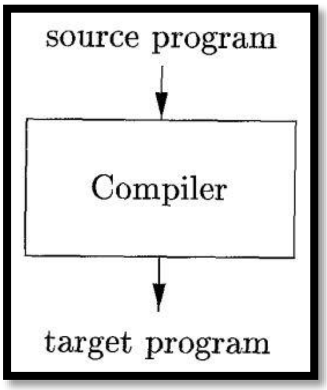
Compiler is a software which converts a program written in high level language (Source Language) to low level language (Object/Target/Machine Language). We know a computer is a logical assembly of Software and Hardware. The hardware knows a language, that is hard for us to grasp, consequently we tend to write programs in high-level language, that is much less complicated for us to comprehend and maintain in thoughts.

Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language in software programming. Binary language has only two alphabets, 0 and 1. To instruct, the hardware codes must be written in binary format, which is simply a series of 1s and 0s. It would be a difficult and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes. These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as Language Processing System.

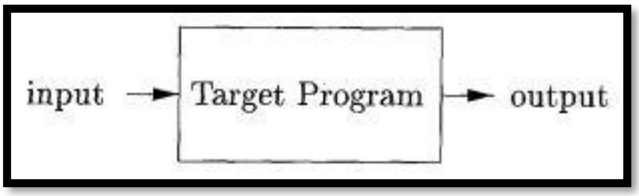
INDEX

Chapter 1: INTRODUCTION

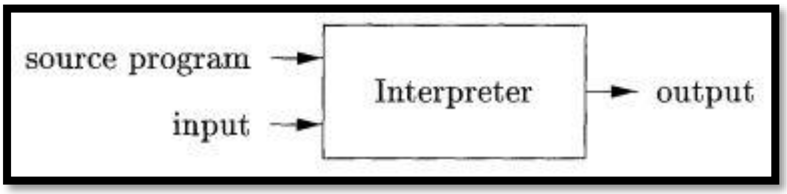
A compiler is a program that can read a program in one language - the source language - and translate it into an equivalent program in another language - the target language. An important role of the compiler is to report any errors in the source program that it detects during the translation process.



If the target program is an executable machine-language program, it can then be called by the user to process inputs and produce outputs.



An interpreter is another common kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.



The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

We know a computer is a logical assembly of Software and Hardware. The hardware knows a language, that is hard for us to grasp, consequently we tend to write programs in high-level language, that is much less complicated for us to comprehend and maintain in thoughts. Now these programs go through a series of transformation so that they can readily be used machines. This is where language processing systems come handy.

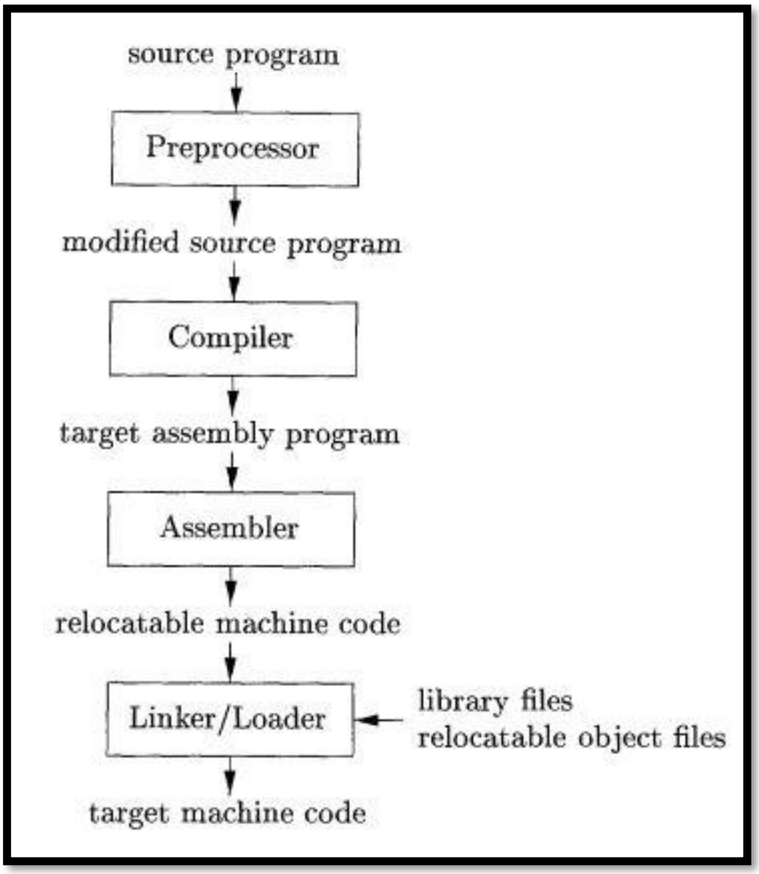


Figure – A language processing system

Phases of a Compiler

If we examine the compilation process in more detail, we see that it operates as a sequence of phases, each of which transforms one representation of the source program to another. A typical decomposition of a compiler into phases is shown in the figure below.

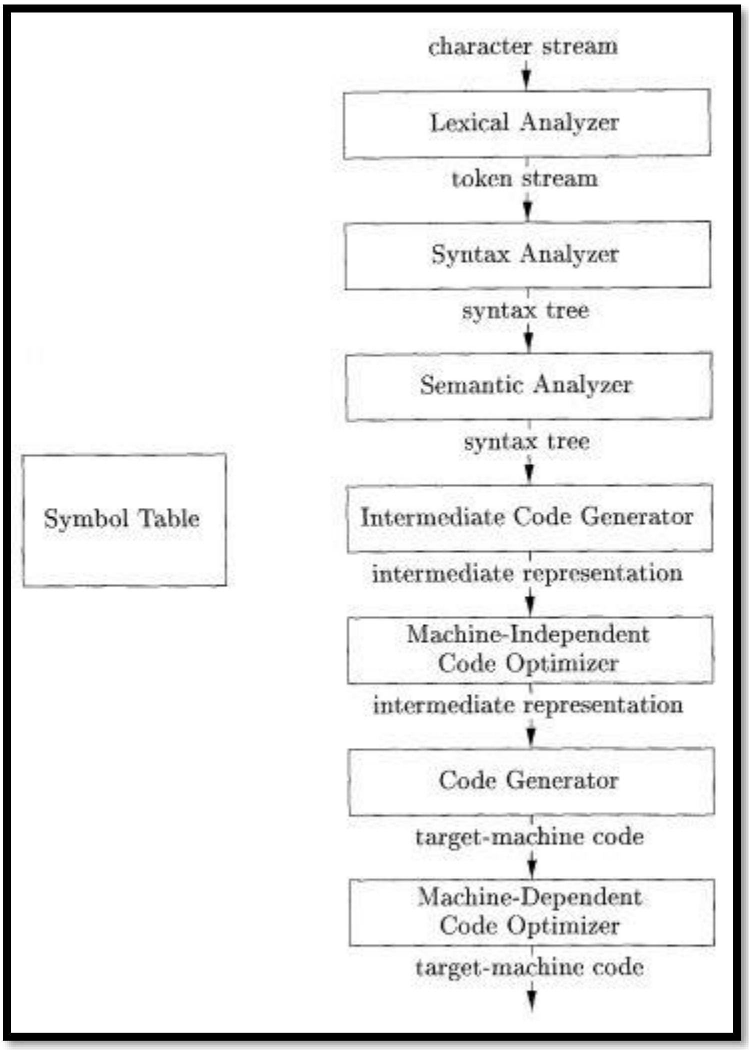


Figure – Phases of a Compiler

The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them. It then uses this structure to create an intermediate representation of the source program. If the analysis part detects that the source program is either syntactically ill formed or semantically unsound, then it must provide informative messages, so the user can take corrective action. The analysis part also collects information about the source program and stores it in a data structure called a symbol table, which is passed along with the intermediate representation to the synthesis part.

The synthesis part constructs the desired target program from the intermediate representation and the information in the symbol table. The analysis part is often called the front end of the compiler; the synthesis part is the back end.

Lexical Analyser

The first phase of a compiler is called lexical analysis or scanning. The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes. For each lexeme, the lexical analyzer produces as output a token of the form

(token-name, attribute-value)

Syntax Analyzer

The second phase of the compiler is syntax analysis or parsing. The parser uses the first components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream. A typical representation is a syntax tree in which each interior node represents an operation and the children of the node represent the arguments of the operation.

Semantic Analysis

The semantic analyzer uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition. It also gathers type information and saves it in either the syntax tree or the symbol table, for subsequent use during intermediate-code generation.

Intermediate Code Generation

In the process of translating a source program into target code, a compiler may construct one or more intermediate representations, which can have a variety of forms. Syntax trees are a form of intermediate representation; they are commonly used during syntax and semantic analysis.

Code Optimization

The machine-independent code-optimization phase attempts to improve the intermediate code so that better target code will result. Usually better means faster, but other objectives may be desired, such as shorter code, or target code that consumes less power.

Code Generation

The code generator takes as input an intermediate representation of the source program and maps it into the target language. If the target language is machine code, registers or memory locations are selected for each of the variables used by the program. Then, the intermediate instructions are translated into sequences of machine instructions that perform the same task.

Symbol Table

An essential function of a compiler is to record the variable names used in the source program and collect information about various attributes of each name.

Chapter 2: LEXICAL ANALYSIS

As the first phase of a compiler, the main task of the lexical analyzer is to read the input characters of the source program, group them into lexemes, and produce as output a sequence of tokens for each lexeme in the source program. The stream of tokens is sent to the parser for syntax analysis. It is common for the lexical analyzer to interact with the symbol table as well. When the lexical analyzer discovers a lexeme constituting an identifier, it needs to enter that lexeme into the symbol table. In some cases, information regarding the kind of identifier may be read from the symbol table by the lexical analyzer to assist it in determining the proper token it must pass to the parser.

These interactions are suggested in the figure below. Commonly, the interaction is implemented by having the parser call the lexical analyzer. The call, suggested by the *getNextToken* command, causes the lexical analyzer to read characters from its input until it can identify the next lexeme and produce for it the next token, which it returns to the parser.

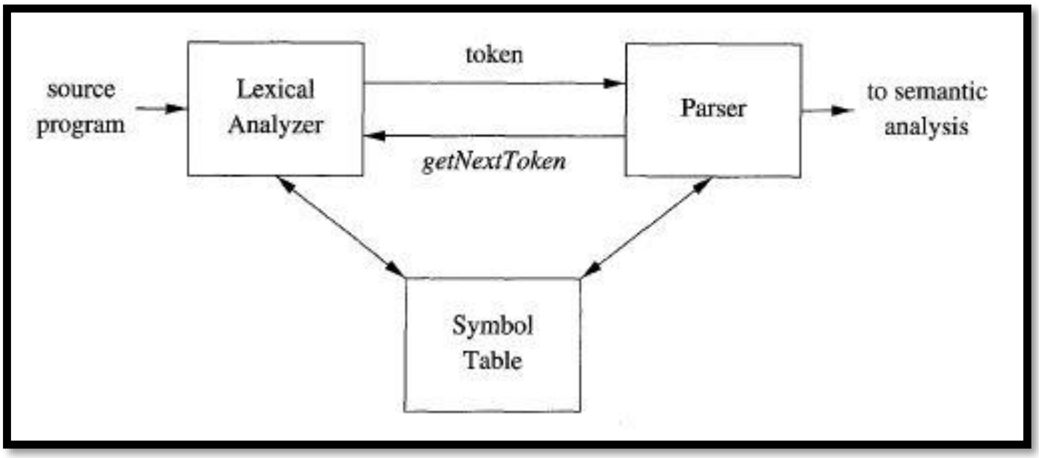


Figure - Interactions between the lexical analyzer and the parser

Since the lexical analyzer is the part of the compiler that reads the source text, it may perform certain other tasks besides identification of lexemes. One such task is stripping out comments and *whitespace* (blank, newline, tab, and perhaps other characters that are used to separate tokens in the input). Another task is correlating error messages generated by the compiler with the source program.

There are a number of reasons why the analysis portion of a compiler is normally separated into lexical analysis and parsing (syntax analysis) phases.

1. Simplicity of design is the most important consideration. The separation of lexical and syntactic analysis often allows us to simplify at least one of these tasks.
2. Compiler efficiency is improved. A separate lexical analyzer allows us to apply specialized techniques that serve only the lexical task, not the job of parsing.
3. Compiler portability is enhanced. Input-device-specific peculiarities can be restricted to the lexical analyzer.

When discussing lexical analysis, we use three related but distinct terms:

Token - A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit.
e.g keywords, identifiers, operators, special symbols, constants etc
The token names are the input symbols that the parser processes. In what follows, we shall generally write the name of a token in boldface. We will often refer to a token by its token name.

Patten - A pattern is a description of the form that the lexemes of a token may take. In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword. For identifiers and some other tokens, the pattern is a more complex structure that is *matched* by many strings.

Lexeme - A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Chapter 3: LEX CODE

Problem Statement:

Design a C/C++/Python compiler for the following pseudocode : (questions.txt)

```
int main()
begin
    int L[10];
    int maxval=L[0];
    for i=1 to n-1 do
        if L[i]>maxval
            maxval=L[i];
        endif
    endfor
    return(maxval)
end
```

Lexical program for the above problem statement: (gen.py)

```
keywords=['int','main','begin','for','to','do','if','endif','endfor','return','End'
operators=['(',')','[',']','=','-','>']
w=""
if os.stat("question.txt").st_size==0 :
    print("File is empty")
else:
    with open('question.txt','r') as f:
        for line in f:
            for word in line:
                for character in word:
                    if character.isspace() or character==';':
                        if w=="":
                            continue;
                        elif w in keywords:
                            print("%s : Keyword"%(w))
                        else:
                            print("%s : Identifier"%(w))
                        w=""
                    elif character in operators:
                        print("%s : Operator"%(character))
                        if w=="":
                            continue
                        elif w in keywords:
                            print("%s : Keyword"%(w))
                        else:
                            print("%s : Identifier"%(w))
                        w=""
                    else:
                        w=w+character
```

Chapter 4: SYNTAX ANALYSIS

In our compiler model, the parser obtains a string of tokens from the lexical analyzer, 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. In fact, the parse tree need not be constructed explicitly, since checking and translation actions can be interspersed with parsing, as we shall see. Thus, the parser and the rest of the front end could well be implemented by a single module.

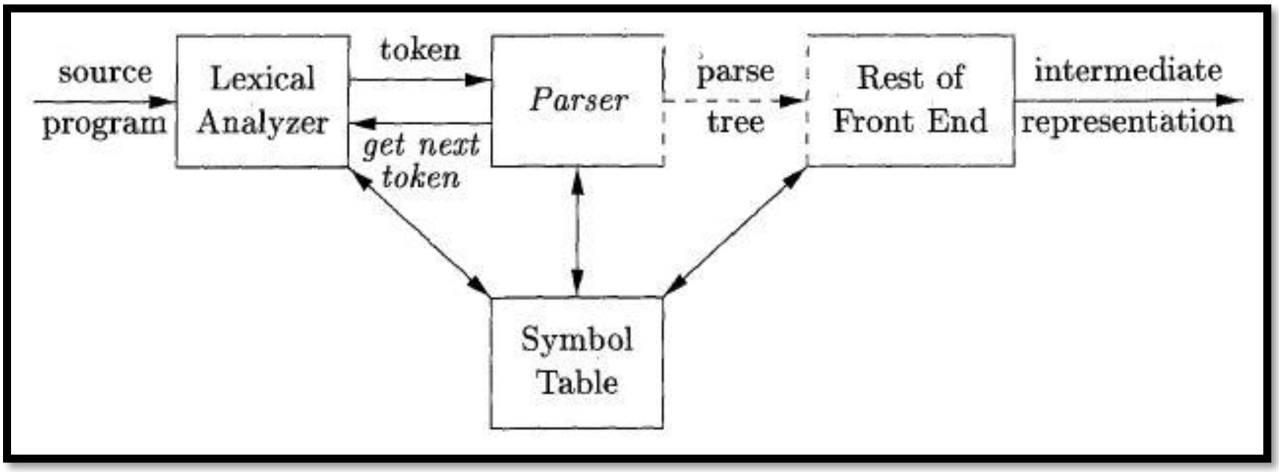


Figure – Position of parser in compiler mode

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 is also known as Syntax Analyzer. The parser obtains a string of tokens from the lexical analyzer 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 from which intermediate code can be generated.

Syntax Error Handling

Planning the error handling right from the start can both simplify the structure of a compiler and improve its handling of errors.

Common programming errors can occur at many different levels.

- Lexical errors include misspellings of identifiers, keywords, or operators.
- Syntactic errors include misplaced semicolons or extra or missing braces.
- Semantic errors include type mismatches between operators and operands.
- Logical errors can be incorrect reasoning on the part of the programmer.

Error-Recovery Strategies

There are many different general strategies that a parser can employ to recover from a syntactic error.

1. Panic-Mode Recovery

   With this method, on discovering an error, the parser discards input symbols one at a time until one of a designated set of synchronizing tokens is found. The synchronizing tokens are usually delimiters, such as semicolon or *3,* whose role in the source program is clear and unambiguous.

2. Phrase-Level Recovery

   On discovering an error, a parser may perform local correction on the remaining input; that is, it may replace a prefix of the remaining input by some string that allows the parser to continue.

3. Error Productions

   By anticipating common errors that might be encountered, we can augment the grammar for the language at hand with productions that generate the erroneous constructs.

4. Global Correction

   Ideally, we would like a compiler to make as few changes as possible in processing an incorrect input string. There are algorithms for choosing a minimal sequence of changes to obtain a globally least-cost correction.

Chapter 5: CONTEXT FREE GRAMMAR

A context-free grammar (grammar for short) consists of terminals, non-terminals, a start symbol, and productions.

$$stmt \rightarrow \textbf{if} \; (\; expr \;) \; stmt \; \textbf{else} \; stmt$$

Figure i)

1. **Terminals** are the basic symbols from which strings are formed. The term "token name" is a synonym for '"terminal" and frequently we will use the word "token" for terminal when it is clear that we are talking about just the token name. We assume that the terminals are the first components of the tokens output by the lexical analyzer. In Figure i), the terminals are the keywords if and else and the symbols "(" and ")".
2. **Non-terminals** are syntactic variables that denote sets of strings. In Figure i), stmt and expr are non-terminals. The sets of strings denoted by non-terminals help define the language generated by the grammar. Non-terminals impose a hierarchical structure on the language that is key to syntax analysis and translation.
3. In a grammar, one nonterminal is distinguished as the **start symbol**, and the set of strings it denotes is the language generated by the grammar. Conventionally, the productions for the start symbol are listed first.
4. The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each **production** consists of:

   (a) A nonterminal called the head or left side of the production; this production defines some of the strings denoted by the head.

   (b) The symbol +. Sometimes : : = has been used in place of the arrow.

   (c) A body or right side consisting of zero or more terminals and non-terminals.

The components of the body describe one way in which strings of the nonterminal at the head can be constructed.

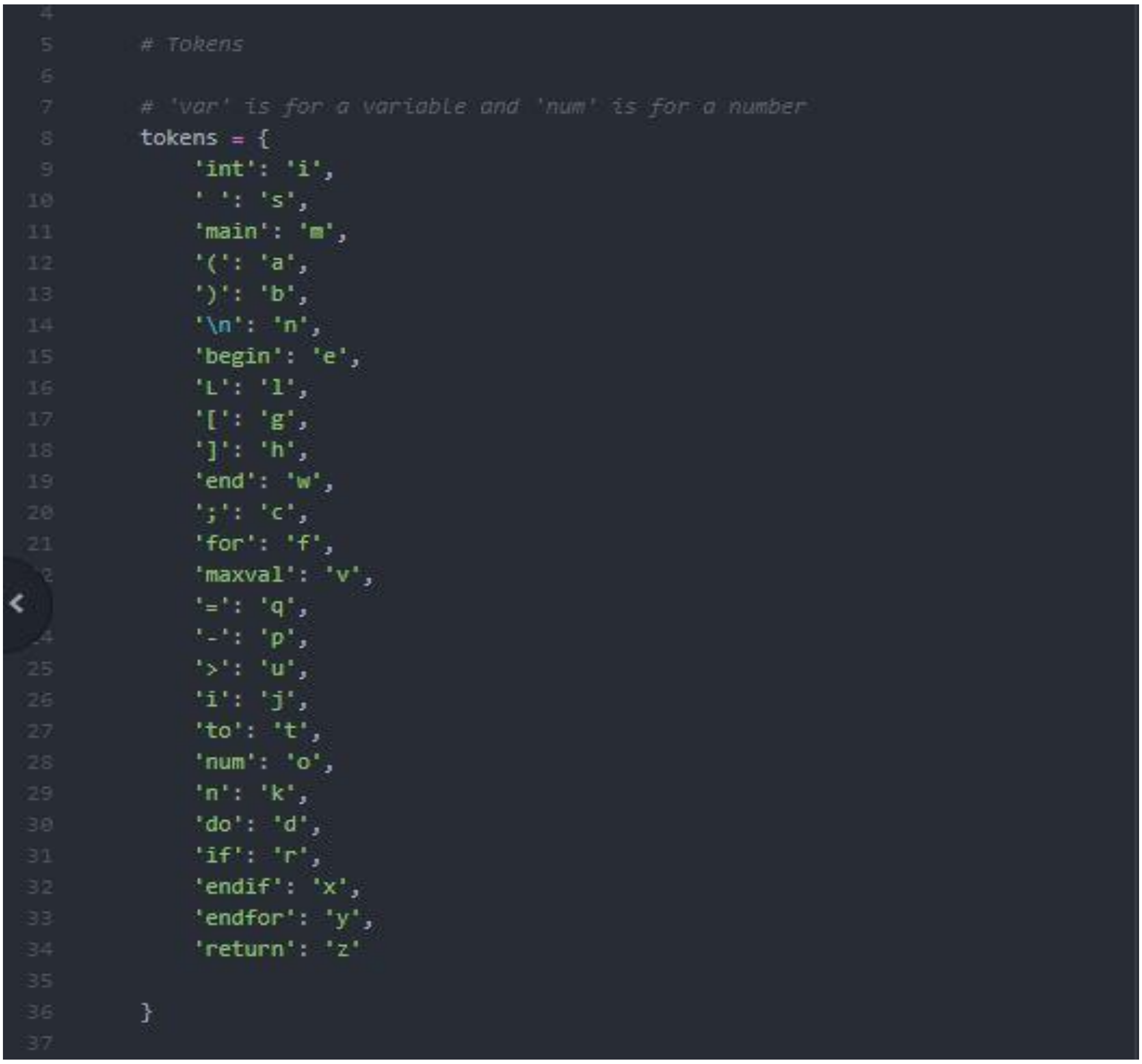
Grammar used

S’ -> S
S -> i s m a b n T
T -> e n i s l g o h c n U
U -> I s v q l g o h c n V
V -> f s j q o s t s k p o s d n W
W -> r s l g j h u v n X
X -> v q l g j h c n x n Y
Y -> y n z a v b n w n

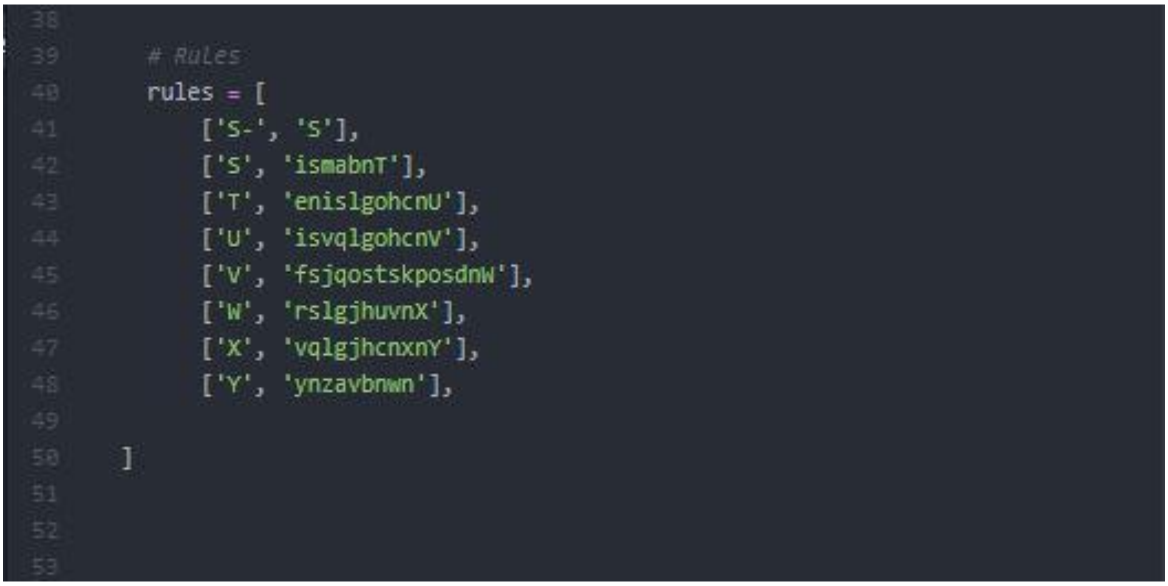
Tokens used

i -> int	t -> to
s -> ‘ ‘ (space)	k -> n
m -> main	p -> -
a -> (	d -> do
b ->)	r -> if
n -> ‘\n’ (newline)	u -> >
e -> begin	x -> endif
l -> L	y -> endfor
g -> [	z -> return
h ->]	w -> end
o -> num	
c -> ;	
v -> maxval	
q -> =	
f -> for	
j -> i	

Code Snapshots



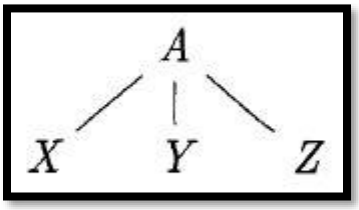
```
# Tokens

# 'var' is for a variable and 'num' is for a number
tokens = {
    'int': 'i',
    ' ': 's',
    'main': 'm',
    '(': 'a',
    ')': 'b',
    '\n': 'n',
    'begin': 'e',
    'L': 'l',
    '[': 'g',
    ']': 'h',
    'end': 'w',
    ';': 'c',
    'for': 'f',
    'maxval': 'v',
    '=': 'q',
    '-': 'p',
    '>': 'u',
    'i': 'j',
    'to': 't',
    'num': 'o',
    'n': 'k',
    'do': 'd',
    'if': 'r',
    'endif': 'x',
    'endfor': 'y',
    'return': 'z'

}
```



```
# Rules
rules = [
    ['S-', 'S'],
    ['S', 'ismabnT'],
    ['T', 'enislgohcnU'],
    ['U', 'isvqlgohcnV'],
    ['V', 'fsjqostskposdnW'],
    ['W', 'rslgjhuvnX'],
    ['X', 'vqlgjhcnxnY'],
    ['Y', 'ynzavbnwn'],

]
```

Chapter 6: PARSE TREE

A parse tree pictorially shows how the start symbol of a grammar derives a string in the language. If nonterminal A has a production A -+ XYZ, then a parse tree may have an interior node labelled A with three children labelled X, Y, and Z, from left to right:

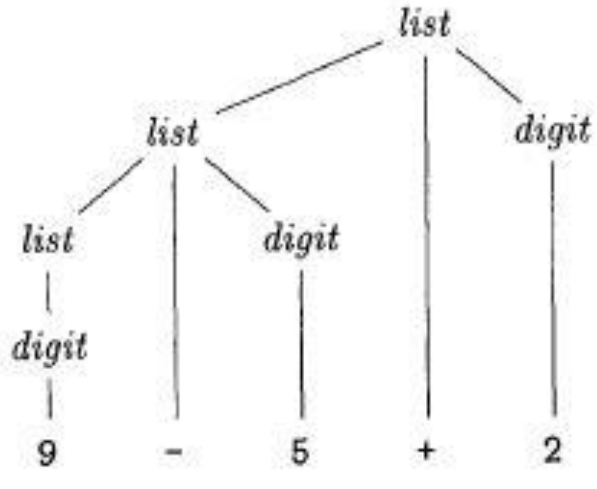


Formally, given a context-free grammar, a *parse tree* according to the grammar is a tree with the following properties:

1. The root is labelled by the start symbol.
2. Each leaf is labelled by a terminal or by e.
3. Each interior node is labelled by a nonterminal.
4. If A is the nonterminal labelling some interior node and XI, Xz, . . . , Xn are the labels of the children of that node from left to right, then there must be a production A -+ X1X2 . . Xn. Here, X1, X2, . . . , X, each stand for a symbol that is either a terminal or a nonterminal. As a special case, if A -+ **c** is a production, then a node labelled A may have a single child labelled ***E.***

e.g – Let us consider the grammar $list \rightarrow list + digit$

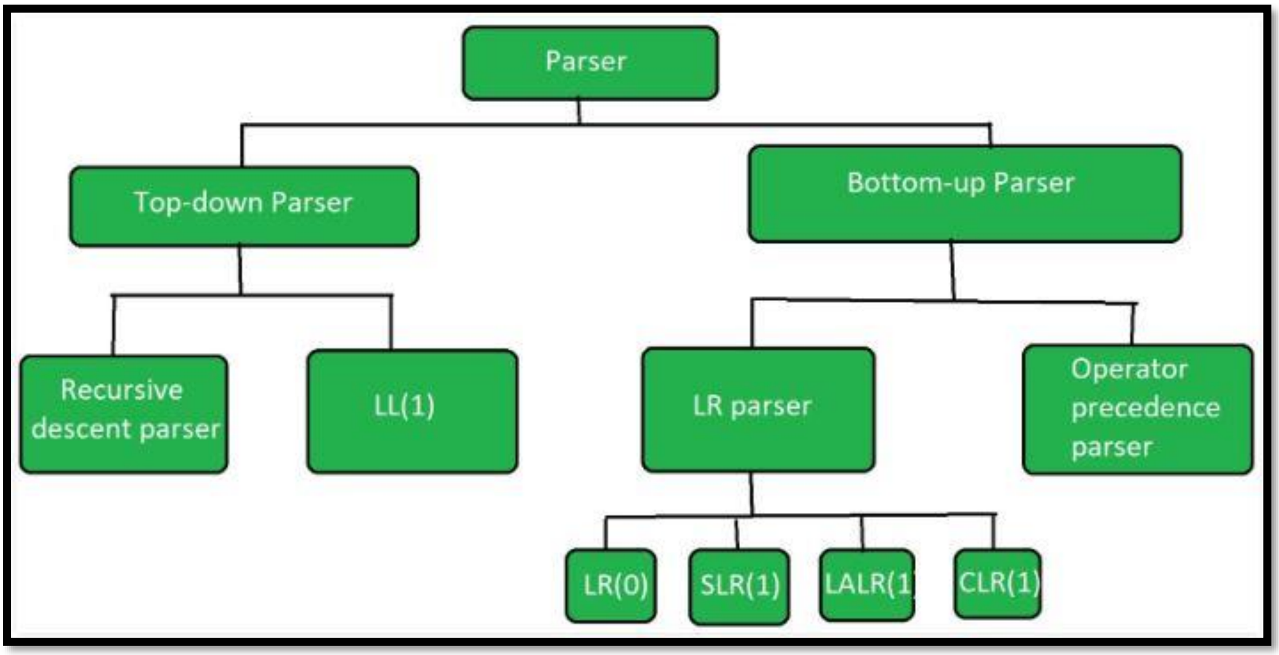
Parse tree for 9-5+2 according to the above grammar is



Chapter 7: TYPES OF PARSING

Types of Parsers in Compiler Design

Parser is mainly classified into 2 categories: Top-down Parser, and Bottom-up Parser. These are explained as following below.



1. Top-down Parser:

   Top-down parser is the parser which generates parse for the given input string with the help of grammar productions by expanding the non-terminals i.e. it starts from the start symbol and ends on the terminals. It uses left most derivation. Further Top-down parser is classified into 2 types: Recursive descent parser, and Non-recursive descent parser.

   i. Recursive descent parser:

      It is also known as Brute force parser or the with backtracking parser. It basically generates the parse tree by using brute force and backtracking.

   ii. Non-recursive descent parser:

      It is also known as LL(1) parser or predictive parser or without backtracking parser or dynamic parser. It uses parsing table to generate the parse tree instead of backtracking.

2. Bottom-up Parser:

   Bottom-up Parser is the parser which generates the parse tree for the given input string with the help of grammar productions by compressing the non-terminals i.e. it starts from non-terminals and ends on the stat symbol. It uses reverse of the right most derivation. We have made use of this parser in our project.

   Further Bottom-up parser is classified into 2 types: LR parser, and Operator precedence parser.

   i. LR parser:

      LR parser is the bottom-up parser which generates the parse tree for the given string by using unambiguous grammar. It follows reverse of right most derivation.

      LR parser is of 4 types:

      a. LR(0)

      b. SLR(1)

      c. LALR(1)

      d. CLR(1)

   ii. Operator precedence parser:

      It generates the parse tree form given grammar and string but the only condition is two consecutive non-terminals and epsilon never appear in the right-hand side of any production.

Chapter 8: PARSING TABLE

```
parse_table = {
        '0': {'i': 'S2', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': '1', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '1': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'A', 'S-': 'E', 'S': '1', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '2': {'i': 'E', 's': 'S3', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '3': {'i': 'E', 's': 'E', 'm': 'S4', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '4': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'S5', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '5': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'S6', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '6': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S7', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '7': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'S9', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': '8', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '8': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'R1', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '9': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S10', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '10': {'i': 'S11', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '11': {'i': 'E', 's': 'S12', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '12': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'S13', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '13': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'S14', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '14': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'S15', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '15': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'S16', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '16': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'S17',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '17': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S18', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '18': {'i': 'S20', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': '19', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '19': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'R2', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '20': {'i': 'E', 's': 'S21', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '21': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'S22', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '22': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'S23', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '23': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'S24', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '24': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'S25', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
```

'25': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'S26', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'26': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'S27', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'27': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'S28', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'28': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S29', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'29': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'S31', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': '30', 'W': 'E', 'X': 'E', 'Y': 'E'},

'30': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'R3', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'31': {'i': 'E', 's': 'S32', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'32': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'S33', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'33': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'S34', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'34': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'S35', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'35': {'i': 'E', 's': 'S36', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'36': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'S37', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

'37': {'i': 'E', 's': 'S38', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E', 'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w': 'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

```
        '38': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'S39', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '39': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'S40', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '40': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'S41', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '41': {'i': 'E', 's': 'S42', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '42': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'S43', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '43': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S44', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '44': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'S46', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': '45', 'X': 'E', 'Y': 'E'},

        '45': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'R4', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '46': {'i': 'E', 's': 'S47', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '47': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'S48', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '48': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'S49', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '49': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'S50', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '50': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'S51', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '51': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'S52', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '52': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'S53', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '53': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S54', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '54': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'S56', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': '55', 'Y': 'E'},

        '55': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'R5', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '56': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'S57', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '57': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'S58', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '58': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'S59', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '59': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'S60', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '60': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'S61', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '61': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'S62',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '62': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S63', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '63': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'S64', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},
        '64': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S65', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '65': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'S67', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': '66'},

        '66': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'R6', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '67': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S68', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '68': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'S69', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '69': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'S70', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '70': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'S71', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '71': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'S72', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '72': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S73', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '73': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'S74', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

        '74': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'S75', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'E', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

            '75': {'i': 'E', 's': 'E', 'm': 'E', 'a': 'E', 'b': 'E', 'n': 'E', 'e': 'E', 'l': 'E', 'g': 'E', 'o': 'E', 'h': 'E', 'c': 'E',
'v': 'E', 'q': 'E', 'f': 'E', 'j': 'E', 't': 'E', 'k': 'E', 'p': 'E', 'd': 'E', 'r': 'E', 'u': 'E', 'x': 'E', 'y': 'E', 'z': 'E', 'w':
'E', '$': 'R7', 'S-': 'E', 'S': 'E', 'T': 'E', 'U': 'E', 'V': 'E', 'W': 'E', 'X': 'E', 'Y': 'E'},

}
```

Chapter 9: PARSER CODE

```
def tokenizer(program_string):
    tokens = Consts.tokens
    ip = 0
    program_string += '$'
    token_string = ""
    while program_string[ip] != '$':
        current_token = ""
        if program_string[ip].isalpha() or program_string[ip] == '_':
            current_token += program_string[ip]
            ip += 1
            while program_string[ip].isalnum() or program_string[ip] == '_':
                current_token += program_string[ip]
                ip += 1
            if current_token in tokens.keys():
                token_string += tokens[current_token]
            else:
                token_string += tokens['var']
        elif program_string[ip].isnumeric():
            current_token += program_string[ip]
            ip += 1
            while program_string[ip].isnumeric() or program_string[ip] == '.':
                current_token += program_string[ip]
                ip += 1
            token_string += tokens['num']
        else:
            if program_string[ip] in tokens:
                token_string += tokens[program_string[ip]]
                ip += 1
            else:
                nl_count = 1
                pointer_count = 0
                for _ in range(ip):
                    if program_string[_] == '\n':
                        nl_count += 1
                        pointer_count = 0
                    else:
                        pointer_count += 1
                print("Tokenizer: Error on line " + str(nl_count) + " on column " + str(pointer_count))
                exit()
    return token_string
```

Syntax Analyser

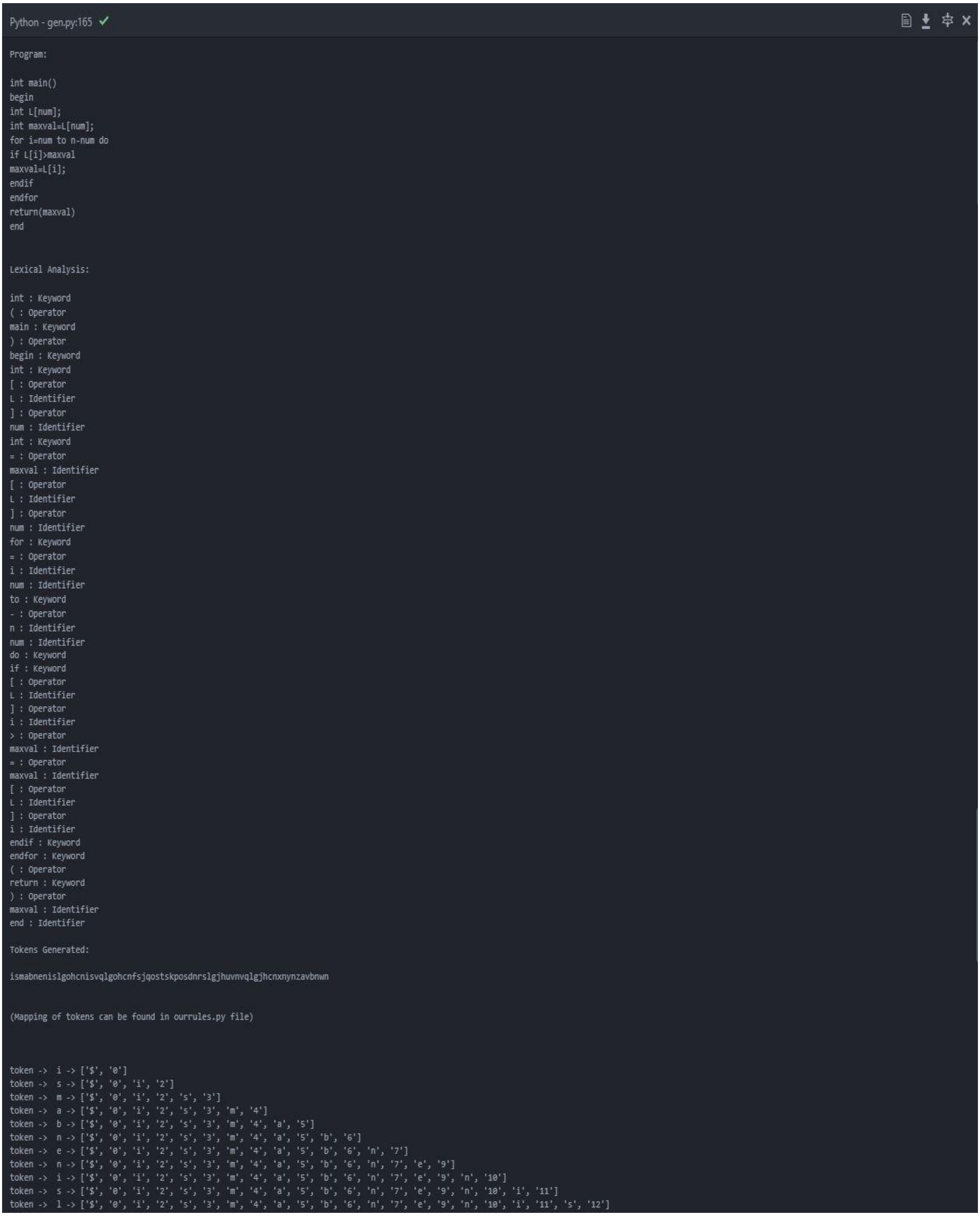
This function analyses the correctness of the program in terms of the syntax

```
def syntax_analyser(token_string, new_line):
    # The rules
    rules = Consts.rules

    # Parse Table
    parse_table = Consts.parse_table
    token_string += "$"

    # Stack that is used for parsing
    stack = ['$', '0']

    # Parsing happens here
    ip = 0
    while True:
        print("token -> ", token_string[ip], "-> ", end="")
        print(stack)
        pivot = parse_table[stack[-1]][token_string[ip]]
        if pivot[0] == 'S':
            stack.append(token_string[ip])
            ip += 1
            stack.append(pivot[1:])
            continue
        elif pivot[0] == 'R':
            rule = rules[int(pivot[1:])]
            for _ in range(2*len(rule[1])):
                stack.pop()
            stack.append(rule[0])
            new_pivot = parse_table[stack[-2]][stack[-1]]
            if new_pivot != 'E':
                stack.append(new_pivot)
                continue
            else:
                break
        elif pivot[0] == 'A':
            print("Parsing Completed Successfully!!!\n No errors")
            exit()
        else:
            break
    line_count = 1
    for _ in range(ip):
        if token_string[_] == new_line:
            line_count += 1
    print("Parser: Error in line " + str(line_count))
```

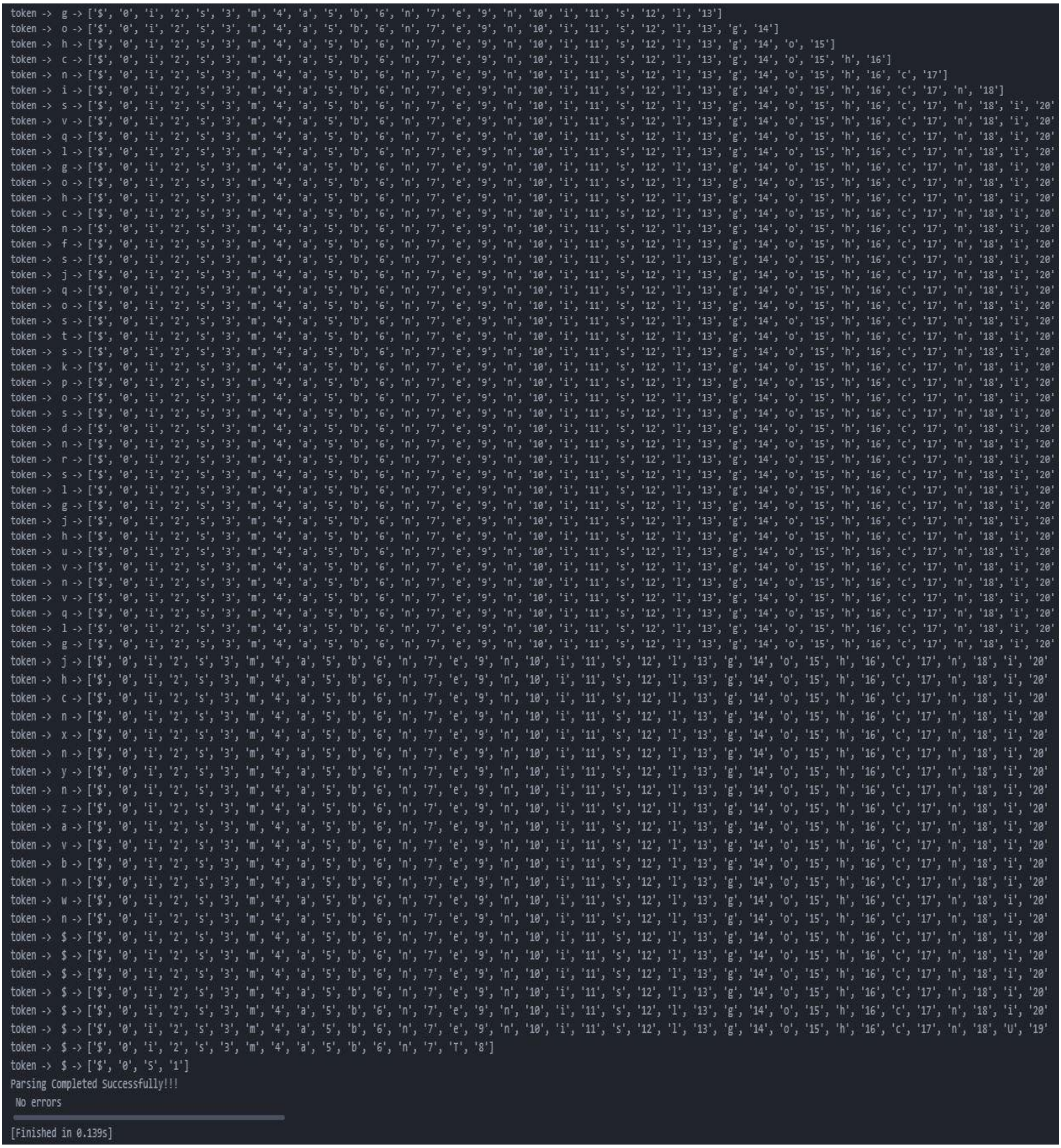
```
# Main function
def main():
    file = open("question.txt", "r")
    program_string = file.read()
    print("\nProgram:\n")
    print(program_string)
    lexical_analyser()
    print("\nTokens Generated:\n")

    # Tokenizing
    token_string = tokenizer(program_string)
    print(token_string)
    print("\n")
    print("(Mapping of tokens can be found in ourrules.py file)")
    print("\n\n")

    # Parsing
    syntax_analyser(token_string, Consts.tokens['\n'])


# Call to main() function
try:
    main()
except Exception as e:
    print("Compiler: Unknown compiler time Exception occured...")
```

Chapter 10: RESULT AND CONCLUSION

Output Snapshots



```
Program:

int main()
begin
int L[num];
int maxval=L[num];
for i=num to n-num do
if L[i]>maxval
maxval=L[i];
endif
endfor
return(maxval)
end


Lexical Analysis:

int : Keyword
( : Operator
main : Keyword
) : Operator
begin : Keyword
int : Keyword
[ : Operator
L : Identifier
] : Operator
num : Identifier
int : Keyword
= : Operator
maxval : Identifier
[ : Operator
L : Identifier
] : Operator
num : Identifier
for : Keyword
= : Operator
i : Identifier
num : Identifier
to : Keyword
- : Operator
n : Identifier
num : Identifier
do : Keyword
if : Keyword
[ : Operator
L : Identifier
] : Operator
i : Identifier
> : Operator
maxval : Identifier
= : Operator
maxval : Identifier
[ : Operator
L : Identifier
] : Operator
i : Identifier
endif : Keyword
endfor : Keyword
( : Operator
return : Keyword
) : Operator
maxval : Identifier
end : Identifier

Tokens Generated:

ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn


(Mapping of tokens can be found in ourrules.py file)



token ->  i -> ['$', '0']
token ->  s -> ['$', '0', 'i', '2']
token ->  m -> ['$', '0', 'i', '2', 's', '3']
token ->  a -> ['$', '0', 'i', '2', 's', '3', 'm', '4']
token ->  b -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5']
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6']
token ->  e -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7']
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9']
token ->  i -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10']
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11']
token ->  l -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12']
```



```
token ->  g -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13']
token ->  o -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14']
token ->  h -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15']
token ->  c -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16']
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17']
token ->  i -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18']
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  v -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  q -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  l -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  g -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  o -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  h -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  c -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  f -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  j -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  q -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  o -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  t -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  k -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  p -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  o -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  d -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  r -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  s -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  l -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  g -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  j -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  h -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  u -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  v -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  v -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  q -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  l -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  g -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  j -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  h -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  c -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  x -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  y -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  z -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  a -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  v -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  b -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  w -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  n -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'i', '20'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'e', '9', 'n', '10', 'i', '11', 's', '12', 'l', '13', 'g', '14', 'o', '15', 'h', '16', 'c', '17', 'n', '18', 'U', '19'
token ->  $ -> ['$', '0', 'i', '2', 's', '3', 'm', '4', 'a', '5', 'b', '6', 'n', '7', 'T', '8']
token ->  $ -> ['$', '0', 'S', '1']
Parsing Completed Successfully!!!
 No errors

[Finished in 0.139s]
```

Conclusion

The project mainly aimed at implementing Lexical and Syntax Analyser which are the first two stages in Compiler Analysis Phase. We were successful in doing the same. Similarly, we can also develop remaining stages of the compiler such that our final outcome would be an assembly level language, which can be easily understood by the computer. The two phases of the compiler are separated to increase simplicity, improve the efficiency and increase the portability.