REPORT
ON
CD Mini Project
Carried out on

Compiler for Finding the Largest Number using IF statement

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

Abdeali (4NM19CS002)

Akash L M (4NM19CS012)

Aniket Ajit Pai (4NM19CS020)

Submitted to,

Dr. Anisha P Rodrigues

Associate Professor Dept Of CSE

NITTE EDUCATION TRUST | NMAM INSTITUTE OF TECHNOLOGY

CERTIFICATE

This is to certify that Mr. Abdeali bearing USN 4NM19CS002, Mr. Akash L M bearing USN 4NM19CS012, Mr. Aniket Ajit Pai bearing USN 4NM19CS020 *of seventh semester B.E., bonafide students of NMAM Institute of Technology, Nitte, have completed CD mini project on* " Designing the compiler for the hypothetical program to find the largest number using if statement " *during October 2022 -December 2022 fulfilling the partial requirements for the award of degree of Bachelor of Engineering in* ***Computer Science and Engineering*** *at NMAM Institute of Technology, Nitte.*

[illegible]

Name and Signature of Mentor

(Dr Anisha P Rodrigues)

[illegible]

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 conducive 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 **Dr. Anisha P Rodrigues** 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.

Abdeali (4NM19CS002)
Akash L M (4NM19CS012)
Aniket Ajit Pai (4NM19CS020)

TABLE OF CONTENTS

Pg No.

1. ABSTRACT

The purpose of this project is to design lexical analyzer and syntax analyzer for a Context Free Grammar. The two stages are the integral part of the Analysis phase of a compilation process which involves identifying the tokens of the given program and using these tokens to identify if each of them is syntactically proper based on given production rules.

The main program takes in two parts namely the source program which we need to process and the grammar rules to parse the program. The objective of the project is to generate the parsed sequence which can be further given for the later stages of the compiler.

2. 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 andspecial 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 the 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. An efficient compiler must preserve semantics of the source program and it should create an efficient version of the target language.

PHASES OF COMPILERS

Typically, a compiler includes several functional parts. For example, a conventional compiler may include a lexical analyzer 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.

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 given source program that contains:

- Lexical Analyzer
- Syntax Analyzer
- Semantic Analyzer

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

- Intermediate code Generator
- Code Optimization
- Code Generation

LEXICAL ANALYZER

Lexical analyzer 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. The process of generation and returning the tokens is called lexical analysis. Representation oflexemes in the form of tokens as:
<token-name, attribute-value>

SYNTAX ANALYSER

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

SEMANTIC ANALYSER

Semantic analyzer takes the output of syntax analyzer. Semantic analyzer 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 of the source code for the target machine. It represents a program for some abstract 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 optimization of code is in terms of time and space.

CODE GENERATION

This produces the target language in a specific architecture. The target program is normally 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.

LEXICAL ANALYSIS

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer 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
5) 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 is matched by the pattern for a token. We can implement lexical analyses using lex tools

3. PROBLEM STATEMENT & METHODOLOGY

Q. Find the largest number using if statement

```
int main()
begin
        int n1, n2, n3;
        if( expr relop expr )begin
                        printf( n1);
                end
                if ( expr relop expr )begin
                                printf( n2);
                end
                if( expr relop expr )begin
                                printf( n3);
                end
end
```

<u>Algorithm to construct LL(1) Parsing Table</u>:

Step 1: First check for left recursion in the grammar, if there isleft recursion in the grammar remove that and go to step 2.

Step 2: Calculate First() and Follow() for all non-terminals.

1. <u>First()</u>: If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.

2. <u>Follow()</u>: What is the Terminal Symbol which follows a variable in the process of derivation.

Step 3: For each production A –> α. (A tends to alpha)

1. Find First(α) and for each terminal in First(α), make entry A –>α in the table.

2. If First(α) contains ε (epsilon) as a terminal then, find the Follow(A) and for each terminal in Follow(A), make entry A –>α in the table.
3. If the First(α) contains ε and Follow(A) contains $ as terminal,then make entry A –> α in the table for the $.

<u>Construction of SLR parsing table</u> :

1. Construct C = { I0, I1, ... In}, the collection of sets of LR(0) items for G'.
2. State i is constructed from Ii. The parsing actions for state i are determined as
follow :
• If [A -> ?.a?] is in Ii and GOTO(Ii , a) = Ij , then set ACTION[i, a] to "shift j".
Here a must be terminal.
• If [A -> ?.] is in Ii, then set ACTION[i, a] to "reduce A -> ?" for all a in FOLLOW(A); here A may not be S'.
• Is [S -> S.] is in Ii, then set action [i, $] to "accept". If any conflicting actions are generated by the above rules, we say that the grammar is not SLR.
3. The goto transitions for state i are constructed for all nonterminals A using the rule:
if GOTO(Ii , A) = Ij then GOTO [i, A] = j.
4. All entries not defined by rules 2 and 3 are made error.

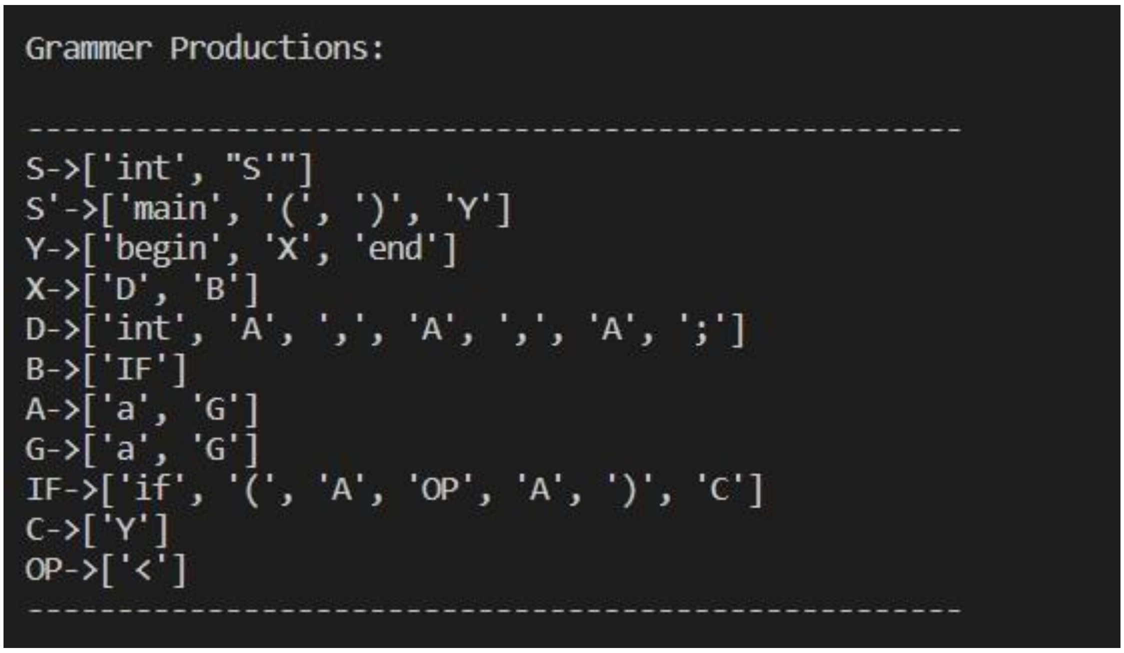
4. RESULT

Code to generate tokens:

```
import re
output = [ ]
def isValidDelimiter(ch):
   if( ch == ' ' or ch == '+' or ch == '-' or ch == '*' or ch == '/' or ch == ',' or ch == ';'
or ch == '>' or ch == '<' or ch == '=' or ch == '(' or ch == ')' or ch == '[' or ch == ']'):
      return True
   else:
      return False

def isValidOperator(ch):
   if(ch == '+' or ch == '-' or ch == '*' or ch == '/' or ch == '>' or ch == '<' or ch ==
'='):
      return True
   else:
      return False
   " " "regex = '^[+-/\*><=]$()\{\}[ ]'
   print(str)
   if(re.search(regex, str)):
      return True
   else:
      return False" " "

def isvalidIdentifier(str):
   " " "if(str[0] == '0' or str[0] == '1' or str[0] == '2' or str[0] == '3' or str[0] == '4' or
str[0] == '5' or str[0] == '6' or str[0] == '7' or str[0] == '8' or str[0] == '9' or
isValidDelimiter(str[0]) == True):
      return False
   else:
      return True" " "
   regex = '^[A-Za-z_][A-Za-z0-9_]*'
   if(re.search(regex, str)):
      return True
   else:
      return False

def isValidKeyword(str):
   lst = ["if", "else", "while", "break", "begin", "end", "printf", "int", "main"]
   if str in lst:
      return True
   else:
      return False

def isValidInteger(str):
    return bool(re.search(r"^\d+$", str))

def isRealNumber(str):
    return bool(re.search(r"^\d+\.\d+$", str))

def isRealNumber(str):
    return bool(re.search(r"^\d+\.\d+$", str))

def takeSub(str, left, right):
    return str[left:right]
    """print(str)
    lst = list(str)
    newlst = []
    for i in range(left, right+1):
        newlst.append(lst[i])
    #sub[right-left+1] = '\0'
    return str(lst)"""

def detectTokens(str):
    left = 0
    right = 0
    length = len(str)

    # print(length)
    try:
        while (right<= length and left <= right):
            #print(left)
            if isValidDelimiter(str[right])==False :
                right=right+1

            if (isValidDelimiter(str[right])==True and left == right):
                if (isValidOperator(str[right]) ==True):
                    print(("|     Operator : "+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    output.append(str[right])
                if str[right]==")" or str[right]=="(" or str[right] == '+' or str[right] == '-' or
str[right] == '*' or str[right] == '/' or str[right] == ',' or str[right] == ';':
                   # print("Here ", str[right])
                    output.append(str[right])
                right=right+1
                left = right
            elif (isValidDelimiter(str[right]) == True and left != right or (right == length
and left != right)):
                sub =  takeSub(str, left, right)
                # print("sub",sub)
                # right=right+1
                if isValidKeyword(sub)==True:
                    print(("|     Keyword :"+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    output.append(sub)
                elif isValidInteger(sub) == True:
                    print(("|     Integer :"+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    output.append(sub)
                elif (isRealNumber(sub) == True):
```

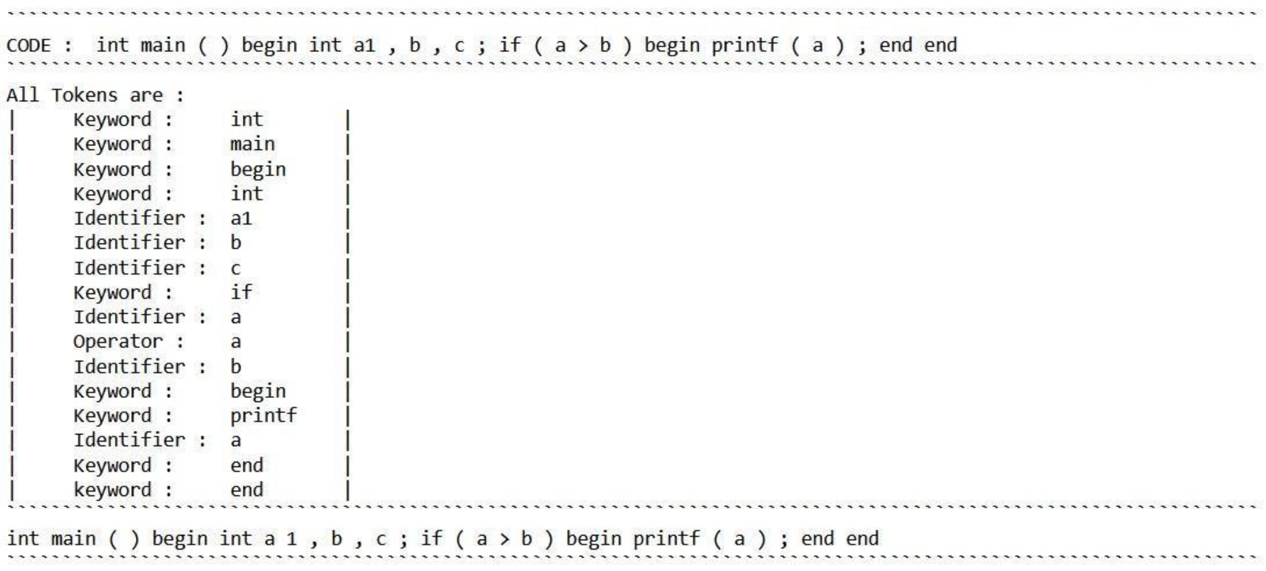
~~~
                    print(("|     Real Number : "+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    output.append(sub)
                elif (isvalidIdentifier(sub) == True and isValidDelimiter(str[right - 1]) ==
False):
                    print(("|     Identifier : "+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    for i in range(len(sub)):
                        output.append(sub[i])
                elif (isvalidIdentifier(sub) == False and isValidDelimiter(str[right - 1]) ==
False):
                    print(("Invalid Identifier : "+ '\t'+ sub + '\t' + "|").expandtabs(10))
                    output.append(sub)
                left = right


    except:
        print(("|     keyword : "+ '\t'+ "end" + '\t' + "|").expandtabs(10))
        output.append("end")

f  = open("InputProg.txt", "r")
str = f.readline()

print("\n
\n```````````````````````````````````````````````````````````````````````````````````````````````````
```````")

print("CODE : ", str)
print("``````````````````````````````````````````````````````````````````````````````````````````
````````````")

print("All Tokens are :")
try:
    detectTokens(str)
except :
    pass

def listToString(instr):
    emptystr=""
    for ele in instr:
        emptystr += ele
        emptystr += " "
    return emptystr

temp =listToString(output)

print("``````````````````````````````````````````````````````````````````````````````````````````
````````````")
print(listToString(output))
print("```````````````````````````````````````````````````````````````````````````````````````````
````````````")

f2 = open("intermediate.txt", "w")
f2.write(temp)
~~~

Productions:



```
Grammer Productions:

----------------------------------------------------
S->['int', "S'"]
S'->['main', '(', ')', 'Y']
Y->['begin', 'X', 'end']
X->['D', 'B']
D->['int', 'A', ',', 'A', ',', 'A', ';']
B->['IF']
A->['a', 'G']
G->['a', 'G']
IF->['if', '(', 'A', 'OP', 'A', ')', 'C']
C->['Y']
OP->['<']
----------------------------------------------------
```

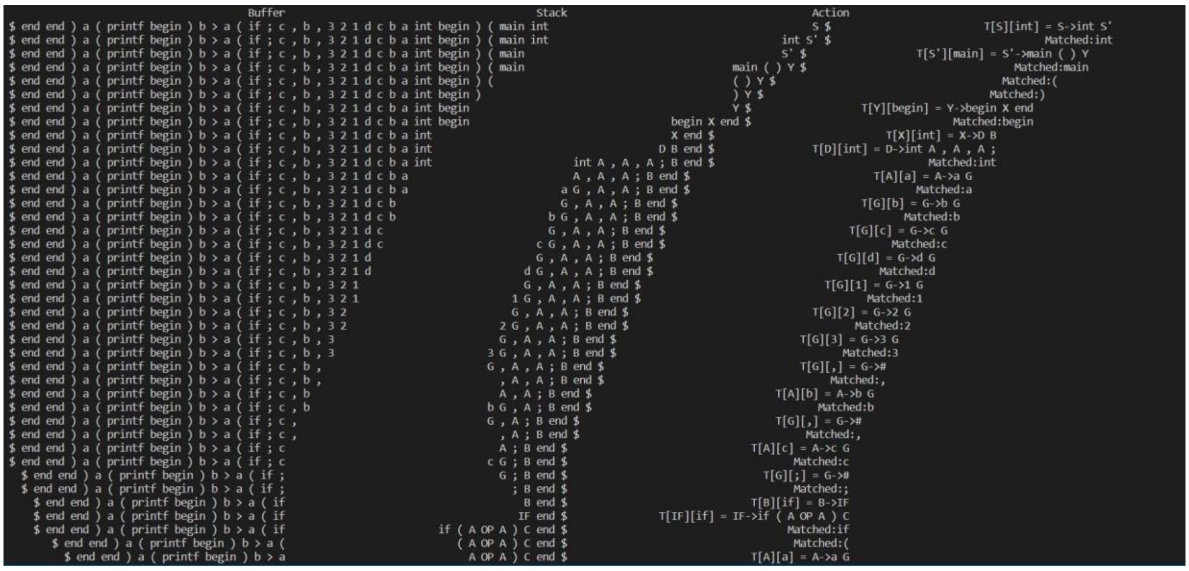
Output of tokenizer:



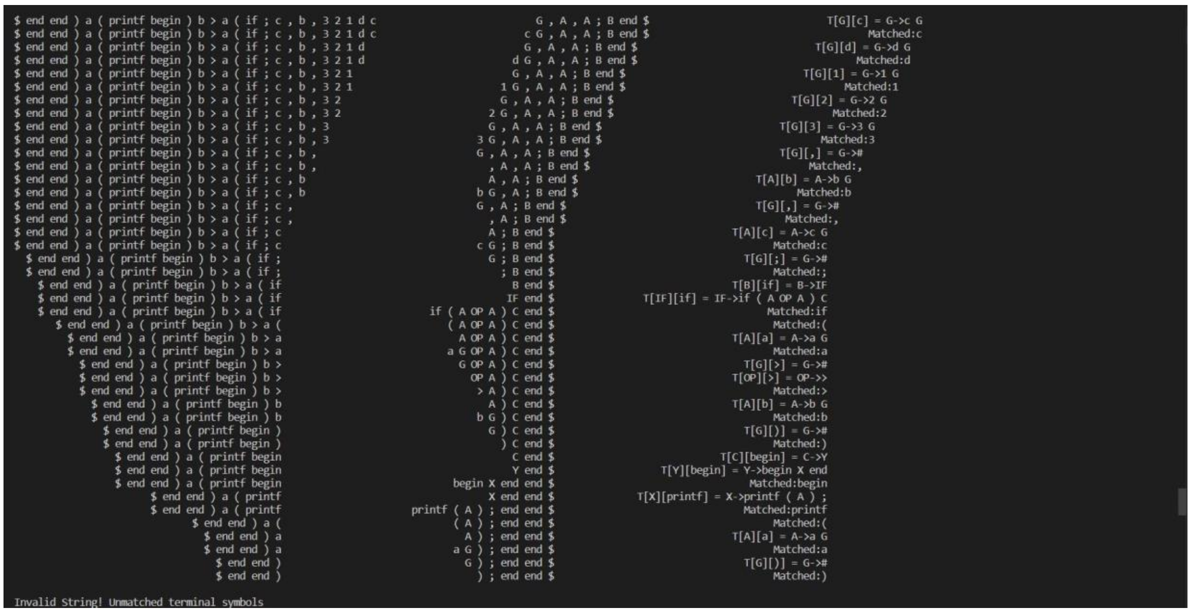
```
CODE :  int main ( ) begin int a1 , b , c ; if ( a > b ) begin printf ( a ) ; end end

All Tokens are :
|      Keyword :      int       |
|      Keyword :      main      |
|      Keyword :      begin     |
|      Keyword :      int       |
|      Identifier :   a1        |
|      Identifier :   b         |
|      Identifier :   c         |
|      Keyword :      if        |
|      Identifier :   a         |
|      Operator :     a         |
|      Identifier :   b         |
|      Keyword :      begin     |
|      Keyword :      printf    |
|      Identifier :   a         |
|      Keyword :      end       |
|      keyword :      end       |

int main ( ) begin int a 1 , b , c ; if ( a > b ) begin printf ( a ) ; end end
```

Output for invalid string:

Code -> int main ( ) begin int abcd1213 , b , c ; if (a > b) begin printf (a) end end



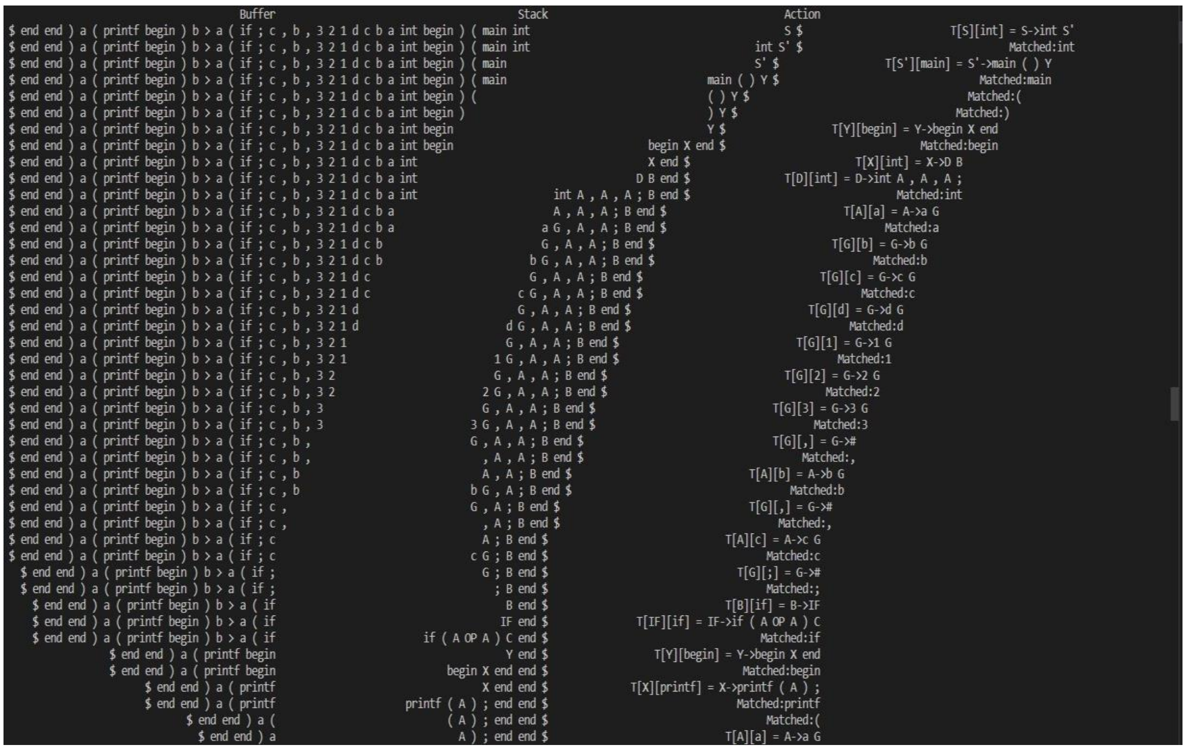
```
                                         Buffer                                            Stack                         Action
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main int                       S $            T[S][int] = S->int S'
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main int                  int S' $                    Matched:int
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main                      S' $          T[S'][main] = S'->main ( ) Y
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main              main ( ) Y $                   Matched:main
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) (                        ( ) Y $                   Matched:(
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin )                            ) Y $                   Matched:)
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin                                Y $        T[Y][begin] = Y->begin X end
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin                      begin X end $              Matched:begin
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                                X end $             T[X][int] = X->D B
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                              D B end $      T[D][int] = D->int A , A , A ;
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                  int A , A , A ; B end $              Matched:int
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a                        A , A , A ; B end $          T[A][a] = A->a G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a                      a G , A , A ; B end $              Matched:a
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                          G , A , A ; B end $          T[G][b] = G->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                        b G , A , A ; B end $              Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                            G , A , A ; B end $          T[G][c] = G->c G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                          c G , A , A ; B end $              Matched:c
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                              G , A , A ; B end $          T[G][d] = G->d G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                            d G , A , A ; B end $              Matched:d
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                                G , A , A ; B end $          T[G][1] = G->1 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                              1 G , A , A ; B end $              Matched:1
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                  G , A , A ; B end $          T[G][2] = G->2 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                2 G , A , A ; B end $              Matched:2
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                    G , A , A ; B end $          T[G][3] = G->3 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                  3 G , A , A ; B end $              Matched:3
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                      G , A , A ; B end $            T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                        , A , A ; B end $              Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c , b                                            A , A ; B end $          T[A][b] = A->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b                                          b G , A ; B end $              Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c ,                                              G , A ; B end $            T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c ,                                                , A ; B end $              Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c                                                    A ; B end $          T[A][c] = A->c G
$ end end ) a ( printf begin ) b > a ( if ; c                                                  c G ; B end $              Matched:c
  $ end end ) a ( printf begin ) b > a ( if ;                                                    G ; B end $            T[G][;] = G->#
  $ end end ) a ( printf begin ) b > a ( if ;                                                      ; B end $              Matched:;
    $ end end ) a ( printf begin ) b > a ( if                                                        B end $           T[B][if] = B->IF
    $ end end ) a ( printf begin ) b > a ( if                                                       IF end $  T[IF][if] = IF->if ( A OP A ) C
    $ end end ) a ( printf begin ) b > a ( if                                       if ( A OP A ) C end $             Matched:if
       $ end end ) a ( printf begin ) b > a (                                          ( A OP A ) C end $              Matched:(
         $ end end ) a ( printf begin ) b > a                                            A OP A ) C end $          T[A][a] = A->a G
```



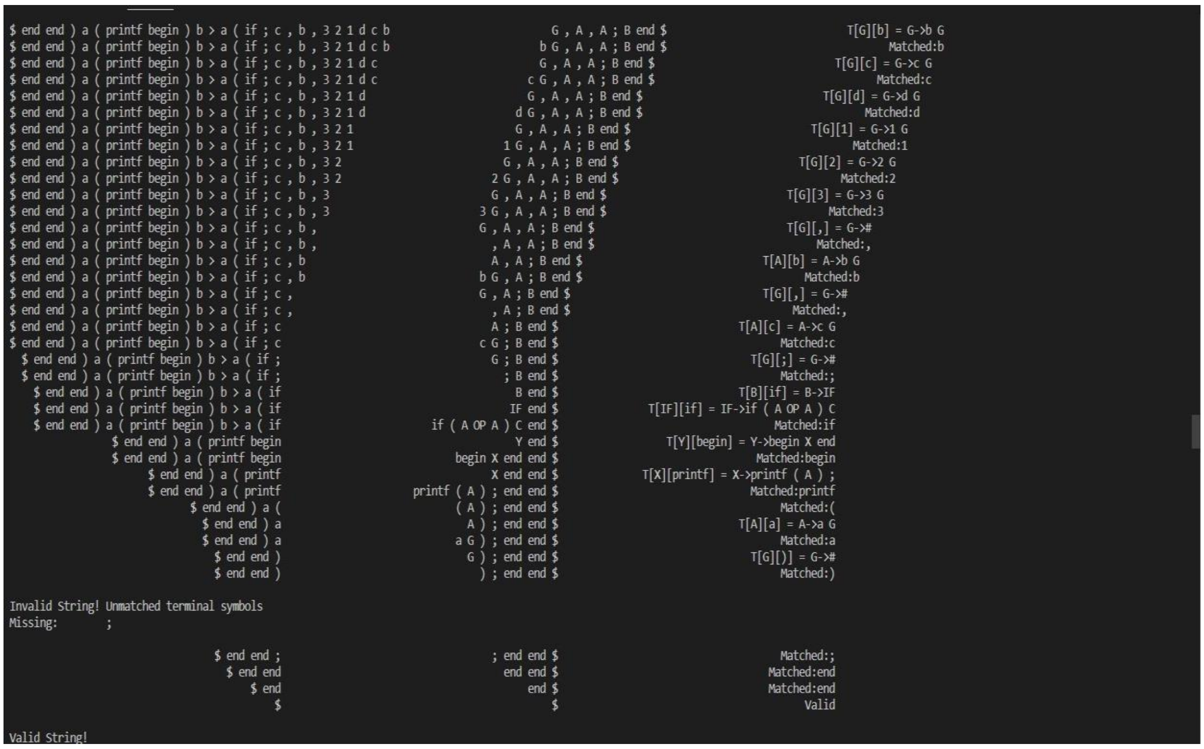
```
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                            G , A , A ; B end $          T[G][c] = G->c G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                          c G , A , A ; B end $              Matched:c
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                              G , A , A ; B end $          T[G][d] = G->d G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                            d G , A , A ; B end $              Matched:d
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                                G , A , A ; B end $          T[G][1] = G->1 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                              1 G , A , A ; B end $              Matched:1
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                  G , A , A ; B end $          T[G][2] = G->2 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                2 G , A , A ; B end $              Matched:2
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                    G , A , A ; B end $          T[G][3] = G->3 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                  3 G , A , A ; B end $              Matched:3
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                      G , A , A ; B end $            T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                        , A , A ; B end $              Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c , b                                            A , A ; B end $          T[A][b] = A->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b                                          b G , A ; B end $              Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c ,                                              G , A ; B end $            T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c ,                                                , A ; B end $              Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c                                                    A ; B end $          T[A][c] = A->c G
$ end end ) a ( printf begin ) b > a ( if ; c                                                  c G ; B end $              Matched:c
  $ end end ) a ( printf begin ) b > a ( if ;                                                    G ; B end $            T[G][;] = G->#
  $ end end ) a ( printf begin ) b > a ( if ;                                                      ; B end $              Matched:;
    $ end end ) a ( printf begin ) b > a ( if                                                        B end $           T[B][if] = B->IF
    $ end end ) a ( printf begin ) b > a ( if                                                       IF end $  T[IF][if] = IF->if ( A OP A ) C
    $ end end ) a ( printf begin ) b > a ( if                                       if ( A OP A ) C end $             Matched:if
       $ end end ) a ( printf begin ) b > a (                                          ( A OP A ) C end $              Matched:(
         $ end end ) a ( printf begin ) b > a                                            A OP A ) C end $          T[A][a] = A->a G
         $ end end ) a ( printf begin ) b > a                                          a G OP A ) C end $              Matched:a
           $ end end ) a ( printf begin ) b >                                            G OP A ) C end $            T[G][>] = G->#
           $ end end ) a ( printf begin ) b >                                              OP A ) C end $           T[OP][>] = OP->>
           $ end end ) a ( printf begin ) b >                                               > A ) C end $              Matched:>
             $ end end ) a ( printf begin ) b                                                 A ) C end $          T[A][b] = A->b G
             $ end end ) a ( printf begin ) b                                               b G ) C end $              Matched:b
               $ end end ) a ( printf begin )                                                 G ) C end $            T[G][)] = G->#
               $ end end ) a ( printf begin )                                                   ) C end $              Matched:)
                 $ end end ) a ( printf begin                                                     C end $        T[C][begin] = C->Y
                 $ end end ) a ( printf begin                                                     Y end $   T[Y][begin] = Y->begin X end
                 $ end end ) a ( printf begin                                           begin X end end $              Matched:begin
                       $ end end ) a ( printf                                                 X end end $  T[X][printf] = X->printf ( A ) ;
                       $ end end ) a ( printf                                     printf ( A ) ; end end $            Matched:printf
                              $ end end ) a (                                            ( A ) ; end end $              Matched:(
                                $ end end ) a                                              A ) ; end end $          T[A][a] = A->a G
                                $ end end ) a                                            a G ) ; end end $              Matched:a
                                  $ end end )                                              G ) ; end end $            T[G][)] = G->#
                                  $ end end )                                                ) ; end end $              Matched:)

Invalid String! Unmatched terminal symbols
```

Output for invalid string with error recovery:

Code -> int main ( ) begin int abcd1213 , b , c ; if (a > b) begin printf (a) end end



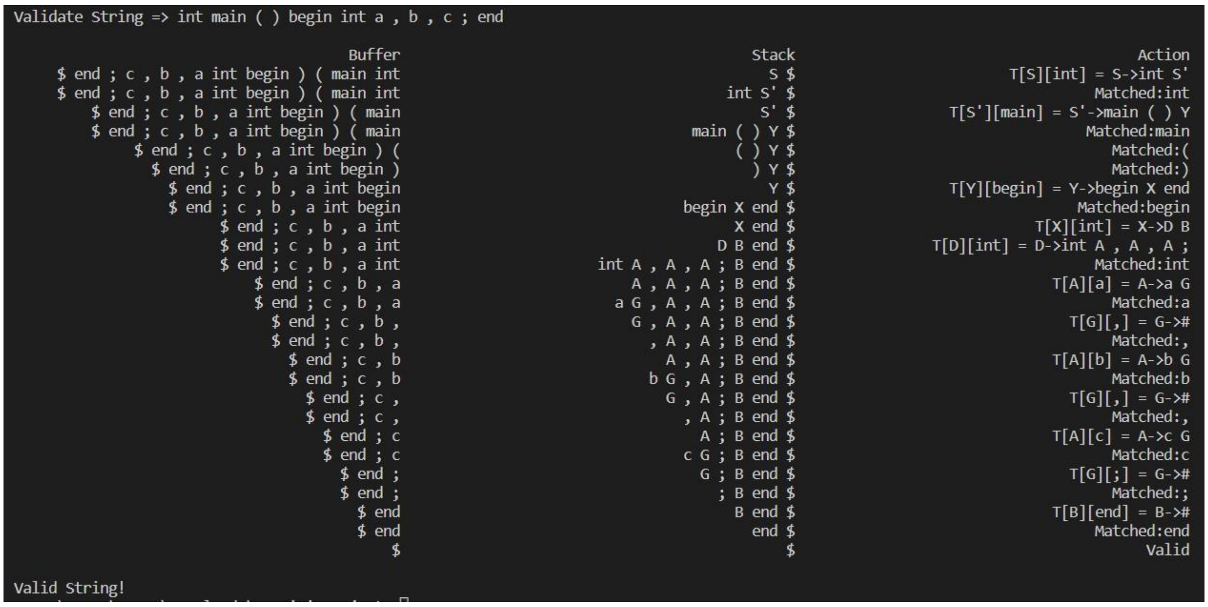
```
                        Buffer                                        Stack                          Action
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main int                    S $                    T[S][int] = S->int S'
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main int                int S' $                             Matched:int
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main                    S' $             T[S'][main] = S'->main ( ) Y
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) ( main              main ( ) Y $                          Matched:main
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin ) (                     ( ) Y $                             Matched:(
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin )                         ) Y $                             Matched:)
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin                           Y $            T[Y][begin] = Y->begin X end
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int begin                  begin X end $                        Matched:begin
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                              X end $              T[X][int] = X->D B
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                            D B end $      T[D][int] = D->int A , A , A ;
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a int                int A , A , A ; B end $                 Matched:int
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a                        A , A , A ; B end $            T[A][a] = A->a G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b a                      a G , A , A ; B end $                 Matched:a
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                          G , A , A ; B end $           T[G][b] = G->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                        b G , A , A ; B end $                Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                          G , A , A ; B end $          T[G][c] = G->c G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                        c G , A , A ; B end $               Matched:c
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                            G , A , A ; B end $         T[G][d] = G->d G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                          d G , A , A ; B end $              Matched:d
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                              G , A , A ; B end $        T[G][1] = G->1 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                            1 G , A , A ; B end $             Matched:1
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                G , A , A ; B end $       T[G][2] = G->2 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                              2 G , A , A ; B end $            Matched:2
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                  G , A , A ; B end $      T[G][3] = G->3 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                3 G , A , A ; B end $           Matched:3
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                    G , A , A ; B end $      T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                      , A , A ; B end $          Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c , b                                          A , A ; B end $    T[A][b] = A->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b                                        b G , A ; B end $        Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c ,                                          G , A ; B end $    T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c ,                                            , A ; B end $        Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c                                                A ; B end $  T[A][c] = A->c G
$ end end ) a ( printf begin ) b > a ( if ; c                                              c G ; B end $      Matched:c
  $ end end ) a ( printf begin ) b > a ( if ;                                                G ; B end $   T[G][;] = G->#
  $ end end ) a ( printf begin ) b > a ( if ;                                                  ; B end $      Matched:;
    $ end end ) a ( printf begin ) b > a ( if                                                  B end $   T[B][if] = B->IF
    $ end end ) a ( printf begin ) b > a ( if                                                 IF end $  T[IF][if] = IF->if ( A OP A ) C
    $ end end ) a ( printf begin ) b > a ( if                                 if ( A OP A ) C end $      Matched:if
                $ end end ) a ( printf begin                                                   Y end $  T[Y][begin] = Y->begin X end
                $ end end ) a ( printf begin                                         begin X end end $     Matched:begin
                      $ end end ) a ( printf                                               X end end $  T[X][printf] = X->printf ( A ) ;
                      $ end end ) a ( printf                                 printf ( A ) ; end end $      Matched:printf
                             $ end end ) a (                                        ( A ) ; end end $      Matched:(
                               $ end end ) a                                          A ) ; end end $   T[A][a] = A->a G
```



```
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                          G , A , A ; B end $           T[G][b] = G->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c b                        b G , A , A ; B end $                Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                          G , A , A ; B end $          T[G][c] = G->c G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d c                        c G , A , A ; B end $               Matched:c
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                            G , A , A ; B end $         T[G][d] = G->d G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1 d                          d G , A , A ; B end $              Matched:d
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                              G , A , A ; B end $        T[G][1] = G->1 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2 1                            1 G , A , A ; B end $             Matched:1
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                                G , A , A ; B end $       T[G][2] = G->2 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3 2                              2 G , A , A ; B end $            Matched:2
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                  G , A , A ; B end $      T[G][3] = G->3 G
$ end end ) a ( printf begin ) b > a ( if ; c , b , 3                                3 G , A , A ; B end $           Matched:3
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                    G , A , A ; B end $      T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c , b ,                                      , A , A ; B end $          Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c , b                                          A , A ; B end $    T[A][b] = A->b G
$ end end ) a ( printf begin ) b > a ( if ; c , b                                        b G , A ; B end $        Matched:b
$ end end ) a ( printf begin ) b > a ( if ; c ,                                          G , A ; B end $    T[G][,] = G->#
$ end end ) a ( printf begin ) b > a ( if ; c ,                                            , A ; B end $        Matched:,
$ end end ) a ( printf begin ) b > a ( if ; c                                                A ; B end $  T[A][c] = A->c G
$ end end ) a ( printf begin ) b > a ( if ; c                                              c G ; B end $      Matched:c
  $ end end ) a ( printf begin ) b > a ( if ;                                                G ; B end $   T[G][;] = G->#
  $ end end ) a ( printf begin ) b > a ( if ;                                                  ; B end $      Matched:;
    $ end end ) a ( printf begin ) b > a ( if                                                  B end $   T[B][if] = B->IF
    $ end end ) a ( printf begin ) b > a ( if                                                 IF end $  T[IF][if] = IF->if ( A OP A ) C
    $ end end ) a ( printf begin ) b > a ( if                                 if ( A OP A ) C end $      Matched:if
                $ end end ) a ( printf begin                                                   Y end $  T[Y][begin] = Y->begin X end
                $ end end ) a ( printf begin                                         begin X end end $     Matched:begin
                      $ end end ) a ( printf                                               X end end $  T[X][printf] = X->printf ( A ) ;
                      $ end end ) a ( printf                                 printf ( A ) ; end end $      Matched:printf
                             $ end end ) a (                                        ( A ) ; end end $      Matched:(
                               $ end end ) a                                          A ) ; end end $   T[A][a] = A->a G
                               $ end end ) a                                        a G ) ; end end $      Matched:a
                                 $ end end )                                          G ) ; end end $   T[G][)] = G->#
                                 $ end end )                                            ) ; end end $      Matched:)

Invalid String! Unmatched terminal symbols
Missing:       ;

                                $ end end ;                                               ; end end $      Matched:;
                                  $ end end                                                 end end $      Matched:end
                                      $ end                                                     end $      Matched:end
                                          $                                                         $          Valid

Valid String!
```

Output for valid string without if statement:

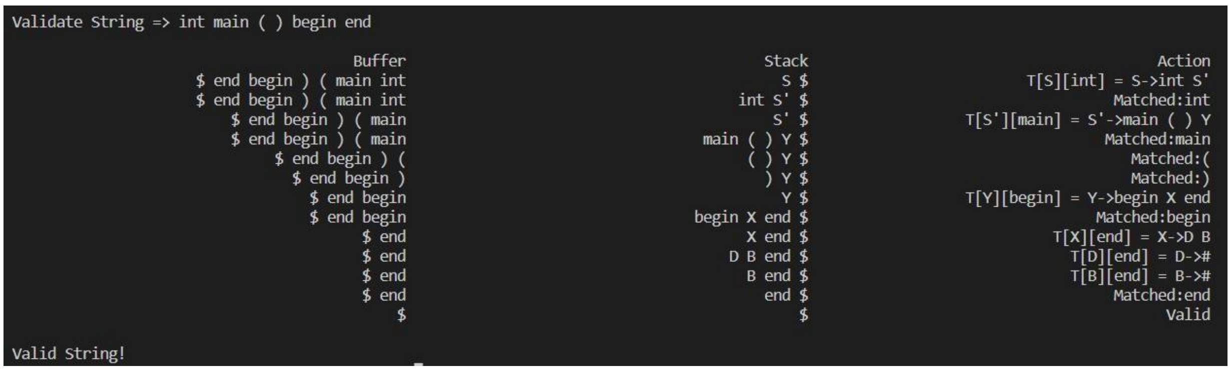
Code -> int main ( ) begin int a , b , c ; end



```
Validate String => int main ( ) begin int a , b , c ; end

                                      Buffer                         Stack                            Action
    $ end ; c , b , a int begin ) ( main int                           S $             T[S][int] = S->int S'
    $ end ; c , b , a int begin ) ( main int                      int S' $                       Matched:int
        $ end ; c , b , a int begin ) ( main                          S' $      T[S'][main] = S'->main ( ) Y
        $ end ; c , b , a int begin ) ( main                  main ( ) Y $                      Matched:main
             $ end ; c , b , a int begin ) (                       ( ) Y $                         Matched:(
               $ end ; c , b , a int begin )                         ) Y $                         Matched:)
                 $ end ; c , b , a int begin                           Y $      T[Y][begin] = Y->begin X end
                 $ end ; c , b , a int begin                 begin X end $                     Matched:begin
                       $ end ; c , b , a int                       X end $                T[X][int] = X->D B
                       $ end ; c , b , a int                     D B end $    T[D][int] = D->int A , A , A ;
                       $ end ; c , b , a int       int A , A , A ; B end $                       Matched:int
                           $ end ; c , b , a           A , A , A ; B end $                  T[A][a] = A->a G
                           $ end ; c , b , a         a G , A , A ; B end $                         Matched:a
                             $ end ; c , b ,           G , A , A ; B end $                    T[G][,] = G->#
                             $ end ; c , b ,             , A , A ; B end $                         Matched:,
                               $ end ; c , b               A , A ; B end $                  T[A][b] = A->b G
                               $ end ; c , b             b G , A ; B end $                         Matched:b
                                 $ end ; c ,               G , A ; B end $                    T[G][,] = G->#
                                 $ end ; c ,                 , A ; B end $                         Matched:,
                                   $ end ; c                   A ; B end $                  T[A][c] = A->c G
                                   $ end ; c                 c G ; B end $                         Matched:c
                                     $ end ;                   G ; B end $                    T[G][;] = G->#
                                     $ end ;                     ; B end $                         Matched:;
                                       $ end                       B end $                  T[B][end] = B->#
                                       $ end                         end $                       Matched:end
                                           $                             $                             Valid

Valid String!
```

Output for valid string without statements:

Code -> int main ( ) begin end



```
Validate String => int main ( ) begin end

                      Buffer                         Stack                            Action
    $ end begin ) ( main int                           S $             T[S][int] = S->int S'
    $ end begin ) ( main int                      int S' $                       Matched:int
        $ end begin ) ( main                          S' $      T[S'][main] = S'->main ( ) Y
        $ end begin ) ( main                  main ( ) Y $                      Matched:main
             $ end begin ) (                       ( ) Y $                         Matched:(
               $ end begin )                         ) Y $                         Matched:)
                 $ end begin                           Y $      T[Y][begin] = Y->begin X end
                 $ end begin                 begin X end $                     Matched:begin
                       $ end                       X end $                T[X][end] = X->D B
                       $ end                     D B end $                  T[D][end] = D->#
                       $ end                       B end $                  T[B][end] = B->#
                       $ end                         end $                       Matched:end
                           $                             $                             Valid

Valid String!
```

5. CONCLUSION

The project is mainly aimed at implementing Lexical and Syntax
Analyzer 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.

In lexical analyses, when we give a program statement as input, the input is parsed. If it is parsed successfully, the success message is displayed in the output. Otherwise, the error message is displayed.

The compiler technology is applied in various computer fields such as HLL implementation, program translation, and computer architecture.
In the future, we may experience complex compiler technologies that will be integrated with various computer applications.

6. REFERENCES

[1] – GeeksforGeeks
https://www.geeksforgeeks.org/introduction-compiler-design/

[2] – tutorials point
https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm

[3] – WIKIPEDIA
https://en.wikipedia.org/wiki/LR_parser

[4] – javaTpoint
https://www.javatpoint.com/clr-l-parsing