Compiler Simulation for Lexical Analyzation of C++

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Perumalla Dharan AP21110010201

NVSS Pavan Sastry AP21110010209

V.Phalgun AP21110010223

G.Dinesh AP21110010240

Under the Guidance of

Dr.Arnab Mitra

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November, 2023]

Table of Contents

Certificate

Date: 16-Nov-23

This is to certify that the work present in this Project entitled "**Compiler Simulation for Lexical Analyzation of C++** " has been carried out by **Perumalla Dharan AP21110010201 , NVSS Pavan Sastry AP21110010209 , V.Phalgun AP21110010223 , G.Dinesh AP21110010240** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. / Dr. Arnab Mitra

Designation,

Affiliation.

Co-supervisor

(Signature)

Prof. / Dr. [Name]

Designation,

Affiliation.

Acknowledgements

I would like to express my sincere gratitude to the Dean, the Vice Chancellor, and the Head of the Organization for giving me the chance to work on this project. Their constant support and encouragement have inspired me to strive for excellence.

I am grateful to Dr. Arnab Mitra, my Faculty Mentor at SRM University, Amaravati, for his ongoing support, critical insight, and intellectual mentorship. I cannot thank him enough for being a consistent source of inspiration and help throughout this academic adventure. His excellent aid has played a big influence in enhancing my research process and has been crucial in the effective completion of this task.

I also want to express my gratitude to my family, friends, and anybody else that guided and assisted me in my studies. Their encouragement has inspired me to overcome challenges and achieve my research goals.

Lastly, I would like to express my sincere thanks to the whole SRM University academic community for creating a welcoming environment that aided in researching this project. I am appreciative of the chance to collaborate with SRM University, Amaravati's esteemed and encouraging academic community.

Thank you all for your major help and advice in creating this research project.

Introduction

Background

Compilers play a fundamental role in converting high-level programming languages into machine code. Our project focuses on building a C++ to Assembly language compiler, particularly concentrating on the initial phase of Lexical Analysis.

Objective

The goal of our project is to develop the Lexical Analysis phase of the C++ compiler. This phase involves the recognition and tokenization of different elements within C++ code, serving as the primary step in the compilation process.

Scope

The scope of this project is limited to Lexical Analysis. We will tokenize C++ code, identify keywords, operators, literals, and other components to generate a stream of tokens that can be used as input for subsequent phases of the compiler.

LEXICAL ANALYSIS PHASE

Overview

Lexical analysis is the initial phase of the compiler, responsible for reading the source code and converting it into a stream of tokens. These tokens represent the fundamental building blocks of the programming language, such as identifiers, keywords, operators, and literals.

Tokenization

Tokenization is the process of breaking the input source code into smaller units, known as tokens. These tokens are the basic elements of the programming language and will be used in the subsequent compilation phases.

Regular Expressions

Regular expressions are patterns that define how different tokens in C++ code are recognized. We have defined regular expressions for keywords, operators, literals, and other elements.

Implementation

Our group has implemented the Lexical Analysis phase in C++. Below is a simplified code snippet for tokenizing C++ source code

Code

LexicalAnalyser.h

```
#program once
#include <string>
#include <map>

using namespace std;

// Reserved Words
#define MAIN 0    // main
#define INT 1     // int
#define FLOAT 2    // float
#define RETURN 3  // return
#define CHAR 4  // char
#define STRING 5  // string
#define IF 6     // if
#define ELSE 7    // else
#define FOR 8   // for
#define WHILE 9    // while
#define KEY_DESC "Reserved Word"
// Identifier
#define IDENTIFIER 10
#define IDENTIFIER_DESC "Identifier"

// Operators
#define ADD 11         // +
#define SUB 12         // -
#define MUL 13         // *
#define DIV 14        // /
#define DIVMOD 14        // %
#define EQUAL 15       // ==
#define LESS_THAN 16   // <
#define LESS_EQUAL 17  // <=
#define GRT_THAN 18  // >
#define GRT_EQUAL 19   // >=
#define NOT_EQUAL 20   // !=
#define ASSIGN 21     // =
#define OPE_DESC "Operator"
```

```
// Delimiters
#define LEFT_BRACKET 22  // (
#define RIGHT_BRACKET 23 // )
#define LEFT_BOUNDER 24 // {
#define RIGHT_BOUNDER 25 // }
#define SEMICOLON 26     // ;
#define DOLLAR 27        // $
#define CLE_OPE_DESC "Delimiter"

// Constants
#define CONST 28 // Unsigned integer constants
#define CONSTANT_DESC "Constant"

// Error Types
#define INT_ERROR "Not an integer constant"
#define INT_ERROR_NUM 1
#define EXCLAMATION_ERROR "Invalid '!' symbol"
#define EXCLAMATION_ERROR_NUM 2
#define SYMBOL_ERROR "Invalid symbol"
#define SYMBOL_ERROR_NUM 3
#define LEFT_BRACKET_ERROR "Unmatched '('"
#define LEFT_BRACKET_ERROR_NUM 4
#define RIGHT_BRACKET_ERROR "Unmatched ')'"
#define RIGHT_BRACKET_ERROR_NUM 5
#define LEFT_BOUNDER_ERROR "Unmatched '{'"
#define LEFT_BOUNDER_ERROR_NUM 6
#define RIGHT_BOUNDER_ERROR "Unmatched '}'"
#define RIGHT_BOUNDER_ERROR_NUM 7
#define END_ERROR "Does not end with $"
#define END_ERROR_NUM 8

#define _NULL "null"

map<string, int> keyMap;
map<string, int> operMap;
map<string, int> limitMap;

// Reserved Word | Identifier | Operator | Constant
struct NormalNode
{
    string content;   // Content
    string describe;  // Describe whether it is a reserved word or identifier
```

```
    int type;         // Type code
    string iden_type; // Identifier type
    int line;         // Line number
    NormalNode *next; // Next node
} * normalHead;       // Head node

// Error Node
struct ErrorNode
{
    string content;  // Error content
    string describe; // Error description
    int type;
    int line;        // Line number
    ErrorNode *next; // Next node
} * errorHead;       // Head node

void initKeyMap();
// Initialize the reserved word dictionary
void initOperMap();
// Initialize the operator dictionary
void initLimitMap();
// Initialize the delimiter dictionary
void initNode();
// Initialize nodes
void createNewNode(string content, string describe, int type, int addr, int
line);    // Insert a node
void createNewError(string content, string describe, int type, int line);
// Insert an error node
void scanner();
// Word scanning
void printNodeLink();
// Print node information
void outputNodeLink();
// Export node information
void printErrorLink();
// Print error node information
void clear();
// Recycle node chain and error chain
```

LexicalAnalyser.cpp

```cpp
#include "LexAnalyse.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstring>
#include <cmath>
#include <string>
#include<ctype.h>

using namespace std;

// Initialize the reserved word dictionary
void initKeyMap()
{
    keyMap.clear();
    keyMap["main"] = MAIN;
    keyMap["int"] = INT;
    keyMap["float"] = FLOAT;
    keyMap["char"] = CHAR;
    keyMap["return"] = RETURN;
    keyMap["if"] = IF;
    keyMap["else"] = ELSE;
    keyMap["for"] = FOR;
    keyMap["while"] = WHILE;
    keyMap["string"] = STRING;
}

// Initialize the operator dictionary
void initOperMap()
{
    operMap.clear();
    operMap["+"] = ADD;
    operMap["-"] = SUB;
    operMap["*"] = MUL;
    operMap["/"] = DIV;
    operMap["%"] = DIVMOD;
    operMap[">"] = GRT_THAN;
    operMap[">="] = GRT_EQUAL;
    operMap["<"] = LESS_THAN;
    operMap["<="] = LESS_EQUAL;
```

```
    operMap["!="] = NOT_EQUAL;
    operMap["=="] = EQUAL;
    operMap["="] = ASSIGN;
}

// Initialize the delimiter dictionary
void initLimitMap()
{
    limitMap["{"] = LEFT_BOUNDER;
    limitMap["}"] = RIGHT_BOUNDER;
    limitMap["("] = LEFT_BRACKET;
    limitMap[")"] = RIGHT_BRACKET;
    limitMap[";"] = SEMICOLON;
}

// Initialize the nodes
void initNode()
{
    normalHead = new NormalNode();
    normalHead->content = "";
    normalHead->describe = "";
    normalHead->type = -1;
    normalHead->iden_type = "";
    normalHead->line = -1;
    normalHead->next = NULL;

    errorHead = new ErrorNode();
    errorHead->content = "";
    errorHead->describe = "";
    errorHead->line = -1;
    errorHead->next = NULL;

    cout << "Initialization of word nodes and error nodes completed" << endl;
}

// Insert a node
void createNewNode(string content, string describe, int type, int line)
{
    NormalNode *p = normalHead;
    NormalNode *temp = new NormalNode();

    while (p->next)
```

```
    {
        p = p->next;
    }

    temp->content = content;
    temp->describe = describe;
    temp->type = type;
    temp->iden_type = "";
    temp->line = line;
    temp->next = NULL;

    p->next = temp;
}

// Insert an error node
void createNewError(string content, string describe, int type, int line)
{
    ErrorNode *p = errorHead;
    ErrorNode *temp = new ErrorNode();

    temp->content = content;
    temp->describe = describe;
    temp->type = type;
    temp->line = line;
    temp->next = NULL;
    while (p->next)
    {
        p = p->next;
    }
    p->next = temp;
}

// Print node information
void printNodeLink()
{
    cout << "*****************************Analysis
Table*****************************" << endl
         << endl;
    cout << setw(15) << "Content"
         << setw(15) << "Description"
         << "\t"
         << setw(3) << "Type Code"
```

```
         << "\t"
         << "Line Number" << endl;
    NormalNode *p = normalHead;
    p = p->next;
    while (p)
    {
        cout << setw(15) << p->content
             << setw(15) << p->describe << "\t"
             << setw(3) << p->type << "\t"
             << setw(8) << p->iden_type << "\t"
             << p->line << endl;

        p = p->next;
    }
    cout << endl;
}

// Export node information
void outputNodeLink()
{
    ofstream fout("words.txt");
    if (!fout)
    {
        cout << "Failed to open words.txt!" << endl;
        return;
    }
    fout << "****************************Analysis
Table*****************************" << endl
         << endl;
    fout << "Content"
         << "\t"
         << setw(10) << "Description"
         << "\t"
         << setw(3) << "Type Code"
         << "\t"
         << "Line Number" << endl;
    NormalNode *p = normalHead;
    p = p->next;
    while (p)
    {
        fout << p->content << "\t"
             << setw(10) << p->describe << "\t"
```

```
            << setw(3) << p->type << "\t"
            << p->line << endl;

        p = p->next;
    }
    fout << endl;

    cout << "Update of words.txt completed!" << endl;
    fout.close();
}

// Print error node information
void printErrorLink()
{
    cout << "*****************************Error
Table*****************************" << endl
         << endl;
    cout << setw(15) << "Content" << setw(15) << "Description"
         << "\t"
         << "Type"
         << "\t"
         << "Line Number" << endl;
    ErrorNode *p = errorHead;
    p = p->next;
    while (p)
    {
        cout << setw(15) << p->content << setw(15) << p->describe << "\t" <<
p->type << "\t" << p->line << endl;
        p = p->next;
    }
    cout << endl
         << endl;
}

// Word scanning
void scanner()
{
    string filename;
    string word;
    int i;
    int line = 1; // Line number
```

```
    fstream fin("test.txt", ios::in);
    if (!fin)
    {
        cout << "Failed to open the file!" << endl;
        return;
    }
    else
    {
        cout << "File opened successfully!" << endl;
    }

    char ch;
    char x ;
    fin.get(ch);
    while (!fin.eof() && ch != '$')
    {
        word.clear();

        if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))
        {
            while ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch
>= '0' && ch <= '9'))
            {

                word += tolower(ch);
                fin.get(ch);
            }

            // If it is a reserved word

            map<string, int>::iterator it = keyMap.find(word);
            if (it != keyMap.end())
                createNewNode(word, KEY_DESC, it->second, line);
            // If it is an identifier
            else
                createNewNode(word, IDENTIFIER_DESC, IDENTIFIER, line);

        }
        // Starts with a number
        else if (ch >= '0' && ch <= '9')
        {
            while (ch >= '0' && ch <= '9')
```

```
            {
                word += ch;
                fin.get(ch);
            }
            createNewNode(word, CONSTANT_DESC, CONST, line);        }
        else if (ch == '+')
        {
            createNewNode("+", OPE_DESC, ADD, line);
            fin.get(ch);
        }
        else if (ch == '-')
        {
            createNewNode("-", OPE_DESC, SUB, line);
            fin.get(ch);
        }
        else if (ch == '*')
        {
            createNewNode("*", OPE_DESC, MUL, line);
            fin.get(ch);
        }
        else if (ch == '/')
        {
            createNewNode("/", OPE_DESC, DIV, line);
            fin.get(ch);
        }
        else if (ch == '%')
        {
            createNewNode("%", OPE_DESC, DIVMOD, line);
            fin.get(ch);
        }
        else if (ch == '<')
        {
            fin.get(ch);
            if (ch == '=')
                createNewNode("<=", OPE_DESC, LESS_EQUAL, line);
            else
                createNewNode("<", OPE_DESC, LESS_THAN, line);
        }
        else if (ch == '>')
        {
            fin.get(ch);
            if (ch == '=')
```

```
                createNewNode(">=", OPE_DESC, GRT_EQUAL, line);
            else
                createNewNode(">", OPE_DESC, GRT_THAN, line);

        }
        else if (ch == '!')
        {
            fin.get(ch);

            if (ch == '=')
                createNewNode("!=", OPE_DESC, NOT_EQUAL, line);
            else
            {
                createNewError("!", EXCLAMATION_ERROR, EXCLAMATION_ERROR_NUM,
line);
            }
        }
        else if (ch == '=')
        {
            fin.get(ch);

            if (ch == '=')
                createNewNode("==", OPE_DESC, EQUAL, line);
            else
                createNewNode("=", OPE_DESC, ASSIGN, line);

        }
        else if (ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n')
        {
            if (ch == '\n')
                line++;
            fin.get(ch);
        }
        else if (ch == '(')
        {
            createNewNode("(", CLE_OPE_DESC, LEFT_BRACKET, line);
            fin.get(ch);
        }
        else if (ch == ')')
        {
            createNewNode(")", CLE_OPE_DESC, RIGHT_BRACKET, line);
            fin.get(ch);
```

```
        }
        else if (ch == '{')
        {
            createNewNode("{", CLE_OPE_DESC, LEFT_BOUNDER, line);
            fin.get(ch);
        }
        else if (ch == '}')
        {
            createNewNode("}", CLE_OPE_DESC, RIGHT_BOUNDER, line);
            fin.get(ch);
        }
        else if (ch == ';')
        {
            createNewNode(";", CLE_OPE_DESC, SEMICOLON, line);
            fin.get(ch);
        }
        else if (ch == '$')
        {
            createNewNode(";", CLE_OPE_DESC, DOLLAR, line);
            fin.get(ch);
        }
        else
        {
            word+=ch;
            createNewError(word, SYMBOL_ERROR, SYMBOL_ERROR_NUM, line);
            fin.get(ch);
        }


    }
    if (ch == '$')
    {
        word.clear();
        word += ch;
        createNewNode(word, CLE_OPE_DESC, DOLLAR, line);
    }

    fin.close();
}

// Recycle node chain and error chain
void clear()
```

```
{
    while (normalHead)
    {
        NormalNode *next = normalHead->next;
        delete normalHead;
        normalHead = next;
    }
    while (errorHead)
    {
        ErrorNode *next = errorHead->next;
        delete errorHead;
        errorHead = next;
    }
}
```

main.cpp

```
#include "LexAnalyse.cpp"
#include <cstring>

int main()
{
    // Lexical Analysis Section
    initKeyMap();
    initOperMap();
    initLimitMap();
    initNode();

    cout << "Lexical Analysis Results: " << endl;
    scanner();

    printNodeLink();
    outputNodeLink();
    printErrorLink(); // error table not working , fix if possible


    clear();
    return 0;
}
```

test.txt:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

bool is_valid(const char *input_string)
{
   int length = strlen(input_string);
   if (length < 2)
   {
      return false;
   }
   return input_string[length - 2] == input_string[length - 1];
}

int main()
{
   const char *test_strings[] = {"aabb", "bbaa", "aab", "bb", "aa", "abab", "aaab",
"ababa"};

   int num_test_strings = sizeof(test_strings) / sizeof(test_strings[0]);

   for (int i = 0; i < num_test_strings; i++)
   {
      bool result = is_valid(test_strings[i]);
      printf("'%s' is %s\n", test_strings[i], result ? "accepted" : "not accepted");
   }

   return 0;
}
```

Results

Output:

```
D:\cd>cd "d:\cd\project\" && g++ main.cpp -o main && "d:\cd\project\"main
Initialization of word nodes and error nodes completed
Lexical Analysis Results:
File opened successfully!
****************************Analysis Table*****************************

        Content    Description  Type Code       Line Number
        include     Identifier   10                    1
              <       Operator   16                    1
          stdio     Identifier   10                    1
              h     Identifier   10                    1
              >       Operator   18                    1
        include     Identifier   10                    2
              <       Operator   16                    2
        stdbool     Identifier   10                    2
              h     Identifier   10                    2
              >       Operator   18                    2
        include     Identifier   10                    3
              <       Operator   16                    3
         string  Reserved Word    5                    3
              h     Identifier   10                    3
              >       Operator   18                    3
           bool     Identifier   10                    5
             is     Identifier   10                    5
          valid     Identifier   10                    5
              (      Delimiter   22                    5
          const     Identifier   10                    5
           char  Reserved Word    4                    5
              *       Operator   13                    5
          input     Identifier   10                    5
         string  Reserved Word    5                    5
              )      Delimiter   23                    5
              {      Delimiter   24                    6
            int  Reserved Word    1                    7
         length     Identifier   10                    7
```

```
           (      Delimiter   22                  8
      length     Identifier   10                  8
           <       Operator   16                  8
           2       Constant   28                  8
           )      Delimiter   23                  8
           {      Delimiter   24                  9
      return  Reserved Word    3                 10
       false     Identifier   10                 10
           ;      Delimiter   26                 10
           }      Delimiter   25                 11
      return  Reserved Word    3                 12
       input     Identifier   10                 12
      string  Reserved Word    5                 12
      length     Identifier   10                 12
           -       Operator   12                 12
           2       Constant   28                 12
          ==       Operator   15                 12
           =       Operator   21                 12
       input     Identifier   10                 12
      string  Reserved Word    5                 12
      length     Identifier   10                 12
           -       Operator   12                 12
           1       Constant   28                 12
           ;      Delimiter   26                 12
           }      Delimiter   25                 13
         int  Reserved Word    1                 15
        main  Reserved Word    0                 15
           (      Delimiter   22                 15
           )      Delimiter   23                 15
           {      Delimiter   24                 16
       const     Identifier   10                 17
        char  Reserved Word    4                 17
           *       Operator   13                 17
        test     Identifier   10                 17
     strings     Identifier   10                 17
           =       Operator   21                 17
```

```
strings     Identifier   10                17
      =       Operator   21                17
      {      Delimiter   24                17
   aabb     Identifier   10                17
   bbaa     Identifier   10                17
    aab     Identifier   10                17
     bb     Identifier   10                17
     aa     Identifier   10                17
   abab     Identifier   10                17
   aaab     Identifier   10                17
  ababa     Identifier   10                17
      }      Delimiter   25                17
      ;      Delimiter   26                17
    int  Reserved Word    1                19
    num     Identifier   10                19
   test     Identifier   10                19
strings     Identifier   10                19
      =       Operator   21                19
 sizeof     Identifier   10                19
      (      Delimiter   22                19
   test     Identifier   10                19
strings     Identifier   10                19
      )      Delimiter   23                19
      /       Operator   14                19
 sizeof     Identifier   10                19
      (      Delimiter   22                19
   test     Identifier   10                19
strings     Identifier   10                19
      0       Constant   28                19
      )      Delimiter   23                19
      ;      Delimiter   26                19
    for  Reserved Word    8                21
      (      Delimiter   22                21
    int  Reserved Word    1                21
      i     Identifier   10                21
      =       Operator   21                21
```

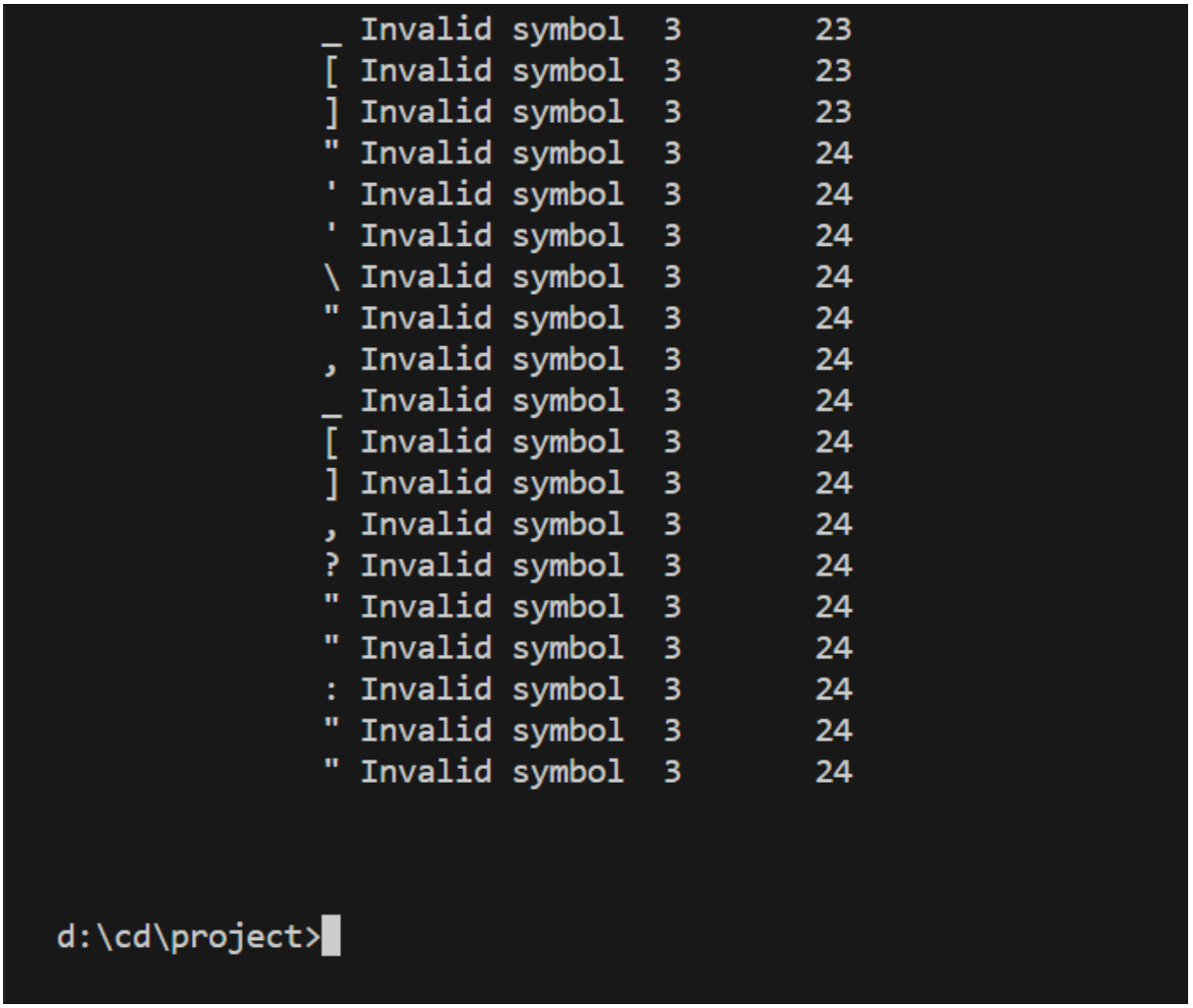
```
         =       Operator   21                21
         0       Constant   28                21
         ;      Delimiter   26                21
         i     Identifier   10                21
         <       Operator   16                21
       num     Identifier   10                21
      test     Identifier   10                21
   strings     Identifier   10                21
         ;      Delimiter   26                21
         i     Identifier   10                21
         +       Operator   11                21
         +       Operator   11                21
         )      Delimiter   23                21
         {      Delimiter   24                22
      bool     Identifier   10                23
    result     Identifier   10                23
         =       Operator   21                23
        is     Identifier   10                23
     valid     Identifier   10                23
         (      Delimiter   22                23
      test     Identifier   10                23
   strings     Identifier   10                23
         i     Identifier   10                23
         )      Delimiter   23                23
         ;      Delimiter   26                23
    printf     Identifier   10                24
         (      Delimiter   22                24
         %       Operator   14                24
         s     Identifier   10                24
        is     Identifier   10                24
         %       Operator   14                24
         s     Identifier   10                24
         n     Identifier   10                24
      test     Identifier   10                24
   strings     Identifier   10                24
         i     Identifier   10                24
```

```
             i     Identifier   10                    24
        result     Identifier   10                    24
      accepted     Identifier   10                    24
           not     Identifier   10                    24
      accepted     Identifier   10                    24
             )      Delimiter   23                    24
             ;      Delimiter   26                    24
             }      Delimiter   25                    25
        return  Reserved Word    3                    27
             0       Constant   28                    27
             ;      Delimiter   26                    27
             }      Delimiter   25                    28

Update of words.txt completed!
*****************************Error Table*******************************

       Content    Description  Type    Line Number
             # Invalid symbol  3       1
             . Invalid symbol  3       1
             # Invalid symbol  3       2
             . Invalid symbol  3       2
             # Invalid symbol  3       3
             . Invalid symbol  3       3
             _ Invalid symbol  3       5
             _ Invalid symbol  3       5
             _ Invalid symbol  3       7
             _ Invalid symbol  3       12
             [ Invalid symbol  3       12
             ] Invalid symbol  3       12
             _ Invalid symbol  3       12
             [ Invalid symbol  3       12
             ] Invalid symbol  3       12
             _ Invalid symbol  3       17
             [ Invalid symbol  3       17
             ] Invalid symbol  3       17
             " Invalid symbol  3       17
```

```
_ Invalid symbol  3       17
[ Invalid symbol  3       17
] Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
, Invalid symbol  3       17
" Invalid symbol  3       17
" Invalid symbol  3       17
_ Invalid symbol  3       19
_ Invalid symbol  3       19
_ Invalid symbol  3       19
_ Invalid symbol  3       19
[ Invalid symbol  3       19
] Invalid symbol  3       19
_ Invalid symbol  3       21
_ Invalid symbol  3       21
_ Invalid symbol  3       23
  Invalid symbol  3       23
```



words.txt:

**********Analysis Table**********

Content	Description	Type Code	Line Number
include	Identifier	10	1
<	Operator	16	1
stdio	Identifier	10	1
h	Identifier	10	1
>	Operator	18	1
include	Identifier	10	2
<	Operator	16	2
stdbool	Identifier	10	2
h	Identifier	10	2
>	Operator	18	2
include	Identifier	10	3

<	Operator	16		3	
string	Reserved Word	5			3
h	Identifier	10		3	
>	Operator	18		3	
bool	Identifier	10		5	
is	Identifier	10		5	
valid	Identifier	10		5	
(	Delimiter	22		5	
const	Identifier	10		5	
char	Reserved Word	4			5
*	Operator	13		5	
input	Identifier	10		5	
string	Reserved Word	5			5
)	Delimiter	23		5	
{	Delimiter	24		6	
int	Reserved Word	1			7
length	Identifier	10		7	
=	Operator	21		7	
strlen	Identifier	10		7	
(	Delimiter	22		7	
input	Identifier	10		7	
string	Reserved Word	5			7
)	Delimiter	23		7	
;	Delimiter	26		7	
if	Reserved Word	6			8
(	Delimiter	22		8	
length	Identifier	10		8	
<	Operator	16		8	
2	Constant	28		8	
)	Delimiter	23		8	
{	Delimiter	24		9	
return	Reserved Word	3			10
false	Identifier	10		10	
;	Delimiter	26		10	
}	Delimiter	25		11	
return	Reserved Word	3			12
input	Identifier	10		12	
string	Reserved Word	5			12
length	Identifier	10		12	
-	Operator	12		12	
2	Constant	28		12	

==	Operator	15		12	
=	Operator	21		12	
input	Identifier	10		12	
string	Reserved Word		5		12
length	Identifier	10		12	
-	Operator	12		12	
1	Constant	28		12	
;	Delimiter	26		12	
}	Delimiter	25		13	
int	Reserved Word		1		15
main	Reserved Word		0		15
(	Delimiter	22		15	
)	Delimiter	23		15	
{	Delimiter	24		16	
const	Identifier	10		17	
char	Reserved Word		4		17
*	Operator	13		17	
test	Identifier	10		17	
strings	Identifier	10		17	
=	Operator	21		17	
{	Delimiter	24		17	
aabb	Identifier	10		17	
bbaa	Identifier	10		17	
aab	Identifier	10		17	
bb	Identifier	10		17	
aa	Identifier	10		17	
abab	Identifier	10		17	
aaab	Identifier	10		17	
ababa	Identifier	10		17	
}	Delimiter	25		17	
;	Delimiter	26		17	
int	Reserved Word		1		19
num	Identifier	10		19	
test	Identifier	10		19	
strings	Identifier	10		19	
=	Operator	21		19	
sizeof	Identifier	10		19	
(	Delimiter	22		19	
test	Identifier	10		19	
strings	Identifier	10		19	
)	Delimiter	23		19	

/	Operator	14		19	
sizeof	Identifier	10		19	
(	Delimiter	22		19	
test	Identifier	10		19	
strings	Identifier	10		19	
0	Constant	28		19	
)	Delimiter	23		19	
;	Delimiter	26		19	
for	Reserved Word		8		21
(	Delimiter	22		21	
int	Reserved Word		1		21
i	Identifier	10		21	
=	Operator	21		21	
0	Constant	28		21	
;	Delimiter	26		21	
i	Identifier	10		21	
<	Operator	16		21	
num	Identifier	10		21	
test	Identifier	10		21	
strings	Identifier	10		21	
;	Delimiter	26		21	
i	Identifier	10		21	
+	Operator	11		21	
+	Operator	11		21	
)	Delimiter	23		21	
{	Delimiter	24		22	
bool	Identifier	10		23	
result	Identifier	10		23	
=	Operator	21		23	
is	Identifier	10		23	
valid	Identifier	10		23	
(	Delimiter	22		23	
test	Identifier	10		23	
strings	Identifier	10		23	
i	Identifier	10		23	
)	Delimiter	23		23	
;	Delimiter	26		23	
printf	Identifier	10		24	
(	Delimiter	22		24	
%	Operator	14		24	
s	Identifier	10		24	

is	Identifier	10		24	
%	Operator	14		24	
s	Identifier	10		24	
n	Identifier	10		24	
test	Identifier	10		24	
strings	Identifier	10		24	
i	Identifier	10		24	
result	Identifier	10		24	
accepted	Identifier	10		24	
not	Identifier	10		24	
accepted	Identifier	10		24	
)	Delimiter	23		24	
;	Delimiter	26		24	
}	Delimiter	25		25	
return	Reserved Word		3		27
0	Constant	28		27	
;	Delimiter	26		27	
}	Delimiter	25		28	

Conclusion

The primary goal of the project is to construct the Lexical Analysis stage, which is the first part of a C++ compiler. This crucial step entails locating and tokenizing various components included in C++ source code, such as operators, literals, and keywords. The main objective is to provide a stream of tokens that serve as the foundation for further compiler stages. Even though the project's focus is limited to lexical analysis, its successful completion creates the framework for the addition of more C++ to Assembly language compiler stages. The project advances the development of the compiler by highlighting the importance of this step and laying the groundwork for thorough language translation. This allows high-level C++ code to be translated into machine-readable instructions.