College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

August 26, 2020

CS431 - Compiler Design Lab · 2020 ·

Exp 1

1 Lexical analyzer

1.1 Aim

Design and Implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new line.

1.2 Theory

The very first phase of a compiler deals with lexical analysis. A lexical analyser, also known as scanner, converts the high level input program into a sequence of tokens. A lexical token is a sequence of characters which is treated as a unit in the grammar of the programming languages.
The common type of tokens include:
Keyword: A keyword is a word reserved by a programming language having a special meaning.
Identifier: It is a user-defined name used to uniquely identify a program element. It can be a class, method, variable, namespace etc.
Operator: It is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.
Separator: Separators are used to separate one programming element from the other.
Literals: A literal is a notation for representing a fixed value and do not change during the course of execution of the program.

1.3 Algorithm

Algorithm 1: Lexical Analyser Algorithm

```
START
Get the input file and read from the file word by word .
Split the word into meaningful tokens with the help of delimiters
Read each token one by one
      If token is a keyword
         print < token , keyword >
      If token is an operator
         print < token , operator >
      If token is a seperator / delimiter
         print < token , delimiter >
      If token is a literal
         print < token , literal >
      If token is an identifier
         print < token , identifier >
STOP
```

1.4 Code

```
#include <bits/stdc++.h>
using namespace std;
vector<string> split_vect(vector<string> s)
{
    vector<string> words;
    for (auto x : s)
    {
        int n = x.size();
        int j = 0;
        for (int i = 0; i < n; ++i)
        {
            if (x[i] == ' ' || x[i] == '\t')
            {
                if (i != j)
```

```
                {
                    words.push_back(x.substr(j, i - j));
                    j = i + 1;
                }
                else
                {
                    j++;
                }
            }
            if (x[i] == '{' || x[i] == '}' || x[i] == '(' || x[i] == ')' || x[i] == ',' || x[i] == ';')
            {
                if (i != j)
                {
                    words.push_back(x.substr(j, i - j));
                    j = i;
                }
                string samp = "";
                samp += x[i];

                if (i + 1 != n)
                {
                    words.push_back(samp);
                    j = i + 1;
                }
            }
        }
        words.push_back(x.substr(j, n - j));
    }
    return words;
}

bool is_key(string s)
{
    if (s == "if" || s == "else" || s == "int" || s == "for" || s == "bool" || s == "string" || s == "float" || s == "return" || s == "printf")
        return true;
    else
        return false;
}

bool is_id(string s)
{
    int size = s.size();
    if (!isalpha(s[0]))
    {
        return false;
    }
    else
    {
        for (int i = 1; i < size; ++i)
        {
            if (!isalnum(s[i]))
            {
                return false;
            }
        }
    }
    return true;
}

bool is_bop(string s)
{
    if (s == "+" || s == "-" || s == "*" || s == "/" || s == "&&" || s == "||" || s == "=")
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool is_uop(string s)
{
    if (s == "++" || s == "--" || s == "!")
```

```
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool is_par(string s)
{
    if (s == "{" || s == "}" || s == "(" || s == ")")
    {
        return true;
    }
    return false;
}

bool is_relop(string s)
{
    if (s == " < " || s == ">" || s == ">=" || s == "<=" || s == "==")
    {
        return true;
    }
    return false;
}

bool is_num(string s)
{
    int n = s.size();
    if (n == 0)
        return false;
    if (s[0] == '"' && s[n - 1] == '"' && n - 1 != 0)
        return true;
    for (int i = 0; i < n; ++i)
    {
        if (!isdigit(s[i]))
        {
            if (s[i] != '.')
                return false;
            else
            {
                for (int j = i + 1; j < n; ++j)
                {
                    if (!isdigit(s[j]))
                    {
                        return false;
                    }
                }
                return true;
            }
        }
    }
    return true;
}

bool is_sup(string s)
{
    if (s == "," || s == ";")
        return true;
    return false;
}
int main()
{
    vector<string> lines;
    string s;
    ifstream file("input.c");
    cout << "Reading from input.c" << endl;
    while (getline(file, s))
    {
        cout << s << endl;
        lines.push_back(s);
    }
    vector<string> words;
    words = split_vect(lines);
    vector<string> tokens;
```

```
    for (auto x : words)
    {
        if (is_key(x))
        {
            tokens.push_back("< " + x + " , " + "keyword" + " >");
        }
        else if (is_id(x))
        {
            tokens.push_back("< " + x + " , " + "identifier" + " >");
        }
        else if (is_par(x))
        {
            tokens.push_back("< " + x + " , " + "paranthesis" + " >");
        }
        else if (is_bop(x))
        {
            tokens.push_back("< " + x + " , " + "operator_b" + " >");
        }
        else if (is_uop(x))
        {
            tokens.push_back("< " + x + " , " + "operator_u" + " >");
        }
        else if (is_relop(x))
        {
            tokens.push_back("< " + x + " , " + "relop" + " >");
        }
        else if (is_sup(x))
        {
            tokens.push_back("< " + x + " , " + "seperator" + " >");
        }
        else if (is_num(x))
        {
            tokens.push_back("< " + x + " , " + "literal" + " >");
        }
        else
        {
            tokens.push_back("< " + x + " , " + "no_idea" + " >");
            cout << "un identified tocken " << x << " program forced to quit" << endl;
            return 0;
        }
    }
    for (auto x : tokens)
    {
        cout << x << endl;
        ;
    }
    cout << endl;
    return 0;
}
```

1.5 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Reading from input.c
int main()
{
    int a, b, c;
    c = a + b;
    string s;
    if (a > b)
        printf(b);
    printf("how-are-you");
    return 0;
}
< int , keyword >
< main , identifier >
< ( , paranthesis >
< ) , paranthesis >
< { , paranthesis >
< int , keyword >
< a , identifier >
< , , seperator >
< b , identifier >
< , , seperator >
< c , identifier >
< ; , seperator >
< c , identifier >
< = , operator_b >
< a , identifier >
< + , operator_b >
< b , identifier >
< ; , seperator >
< string , keyword >
< s , identifier >
< ; , seperator >
< if , keyword >
< ( , paranthesis >
< a , identifier >
< > , relop >
< b , identifier >
< ) , paranthesis >
< printf , keyword >
< ( , paranthesis >
< b , identifier >
< ) , paranthesis >
< ; , seperator >
< printf , keyword >
< ( , paranthesis >
< "how-are-you" , literal >
< ) , paranthesis >
< ; , seperator >
< return , keyword >
< 0 , literal >
< ; , seperator >
< } , paranthesis >
```

```
Reading from input.c
int main()
{
    int a, b, c;
    c = a + b;
    string s;
    if (a > b)
        printf(b);
    printf("how-are-you");
    return 0;
}
< int , keyword >
< main , identifier >
< ( , paranthesis >
< ) , paranthesis >
< { , paranthesis >
< int , keyword >
< a , identifier >
< , , seperator >
< b , identifier >
< , , seperator >
< c , identifier >
< ; , seperator >
< c , identifier >
< = , operator_b >
< a , identifier >
< + , operator_b >
< b , identifier >
< ; , seperator >
< string , keyword >
< s , identifier >
< ; , seperator >
< if , keyword >
< ( , paranthesis >
< a , identifier >
< > , relop >
< b , identifier >
< ) , paranthesis >
< printf , keyword >
< ( , paranthesis >
< b , identifier >
< ) , paranthesis >
< ; , seperator >
< printf , keyword >
< ( , paranthesis >
< "how-are-you" , literal >
< ) , paranthesis >
< ; , seperator >
< return , keyword >
< 0 , literal >
< ; , seperator >
< } , paranthesis >
```

1.6 Result

Implemented the program to develop a lexical analyzer for C language in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained. The input file is read word by word. The words are further divided using the help of delimiters to form meaningful tokens. After proper pre-processing of read words, we get all the required tokens. All these tokens are tested for keywords, operators(relative operator,binary,unary), literals, delimiters(paranthesis, seperator) and identifiers. They are checked and output is displayed.