Ⅰ. Purpose of the experiment

Lexical analysis is the process of converting a sequence of characters from source program into a sequence of tokens. A lexical analyzer is a pattern matcher for character strings:

A lexical analyzer is a “front-end” for the parser

A lexical analyzer identifies substrings of the source program that belong together.

A program which performs lexical analysis is termed as a lexical analyzer (lexer), tokenizer or scanner.

For this assignment, you will design a state diagram that describes the tokens and write a program that implements the state diagram which recognizes a specific form of C-based comments: those that begin with /* and end with */.

Ⅱ. Experimental content

1. Design/present a state diagram which recognizes this form of C-based comments: those that begin with /* and end with */
2. In the programming language of your choice, write and test code which implements the state diagram of this problem (recognizes the C style comments described above)
3. Discussion report (written formally in full sentences):
    a. Introduce the assignment/the problem to be solved
    b. Briefly discuss the steps you took and decisions you made to start and solve this problem
    c. Answer: did the state diagram help you design the code? Briefly explain your answer.
    d. Concludes the report: wrap up the report (summary of assignment and what you did to meet the requirements
4. Deliverables: You will submit a single Word or PDF document which includes:
    a. The written discussion elements found in #3 (in the above section)
    b. Your state diagram (all components of the diagram must be clearly readable)
    c. Video of your program .(the working URL to the video should be listed in the report document; at the end of the report):

i. compiling

ii. Showing the program run from start to finish with the output of the solution for three test cases

d. List (working URLs are sufficient) all resources used to complete this work (you are expected to explore at least two sources)

e. Text of your code copied at the end of the report

5. Expectations:

a. Compiling errors: Your solution must compile

b. Readability. Your code should meet basic readability principles:

i. Separate each component/part with white space.

ii. Align everything in a meaningful way.

c. Comments: you must include enough comments to ensure that the code is described in sufficient detail such that anyone else looking at the code can easily understand the design and the purpose of the code.

d. All code and work associated with this assignment is your own work/written by you.

III. Experimental steps

Step1:Study the lexical analyzer code and state diagrams provided in the book and collect relevant information online.

Step2:Use the state diagram in the book as a basis for your diagram. The function of the lexical analyzer is designed to recognize C annotations according to the question.

For example /*This is a function*/, where /* and */ are both comment symbols and the middle statement is a comment. The lexical analyzer will group and display them.

Step3:Based on the designed state diagram, program based on the code provided in the book:

i. According to the function, charClass will be divided into Symbol and EOF categories only

ii. Divide the comments into /... / and /*... */ two categories, the rest of the type characters are Uknown

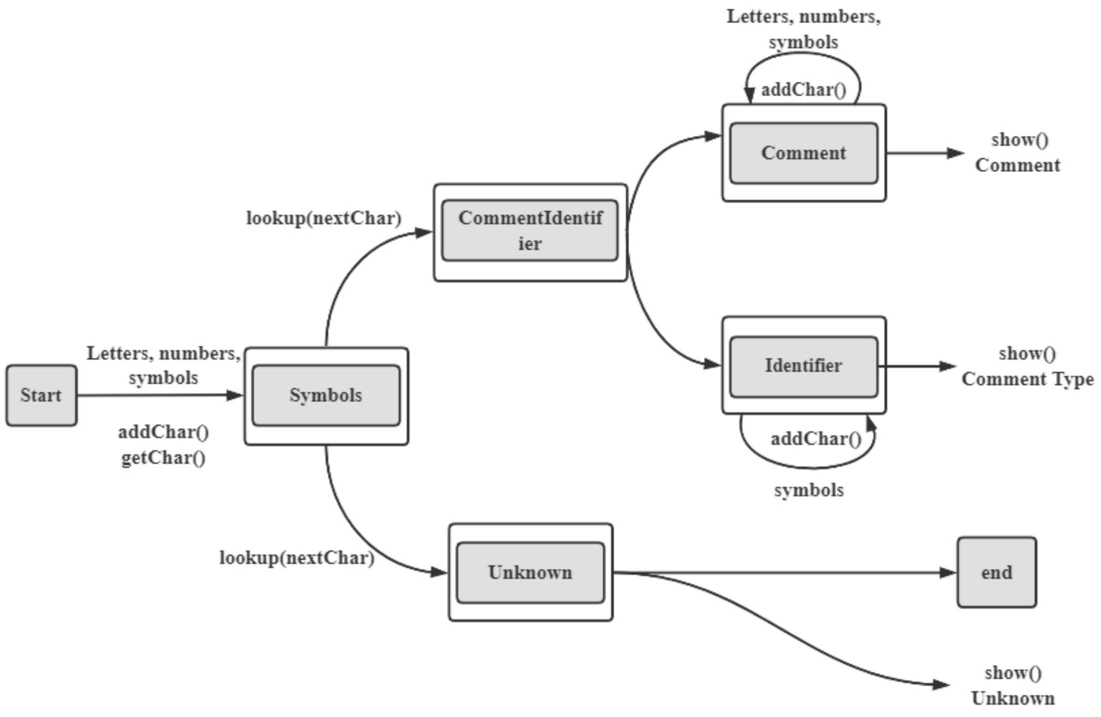
iii. When a comment symbol is detected, the content of the comment is divided into a separate category.This means that the annotation identifier and the annotation are two separate categories, which is the ultimate function of this lexical parser.

iv. Output the results in two categories: lexemes and tokens

Step4:Testing with three test sets.

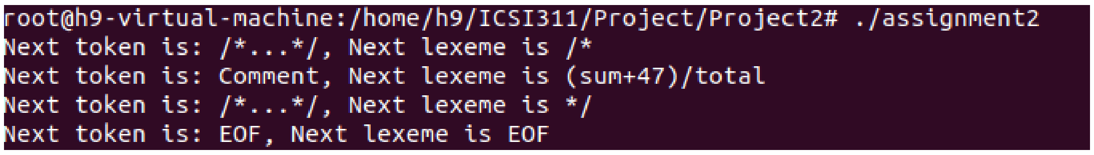
IV. Experimental results and analysis

State Gaph:



Graph 1 State Graph

Test set 1: /*(sum+47)/total*/



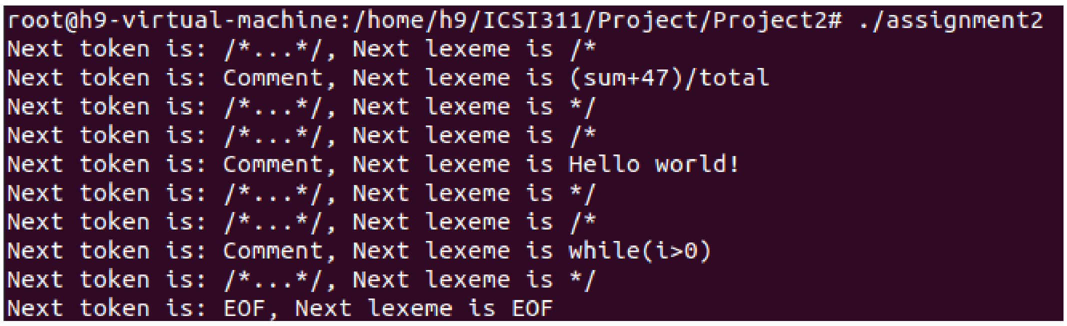
```
root@h9-virtual-machine:/home/h9/ICSI311/Project/Project2# ./assignment2
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is (sum+47)/total
Next token is: /*...*/, Next lexeme is */
Next token is: EOF, Next lexeme is EOF
```

Graph 2 Test1

Test set 2:/*(sum+47)/total*/

/*Hello world!*/

/*while(i>0)*/



```
root@h9-virtual-machine:/home/h9/ICSI311/Project/Project2# ./assignment2
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is (sum+47)/total
Next token is: /*...*/, Next lexeme is */
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is Hello world!
Next token is: /*...*/, Next lexeme is */
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is while(i>0)
Next token is: /*...*/, Next lexeme is */
Next token is: EOF, Next lexeme is EOF
```

Graph 3 Test2

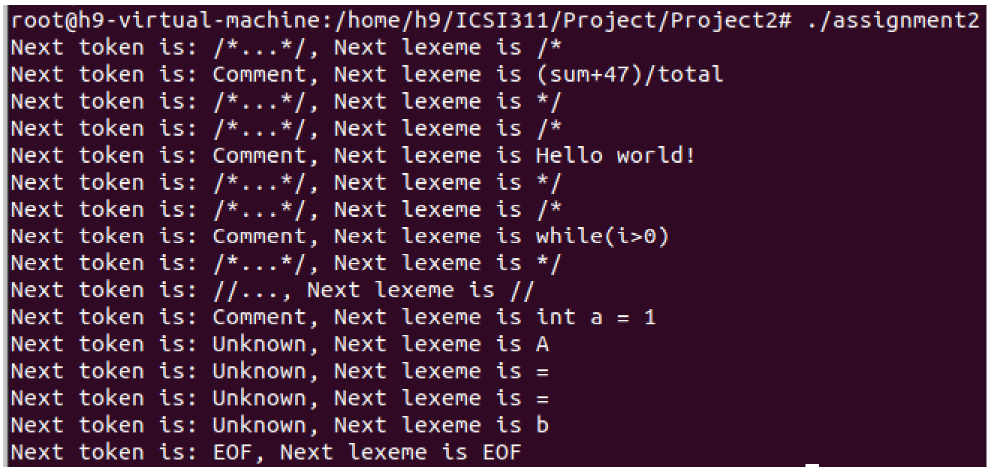
Test set 3:/*(sum+47)/total*/

/*Hello world!*/

/*while(i>0)*/

//int a = 1

A == B



```
root@h9-virtual-machine:/home/h9/ICSI311/Project/Project2# ./assignment2
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is (sum+47)/total
Next token is: /*...*/, Next lexeme is */
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is Hello world!
Next token is: /*...*/, Next lexeme is */
Next token is: /*...*/, Next lexeme is /*
Next token is: Comment, Next lexeme is while(i>0)
Next token is: /*...*/, Next lexeme is */
Next token is: //..., Next lexeme is //
Next token is: Comment, Next lexeme is int a = 1
Next token is: Unknown, Next lexeme is A
Next token is: Unknown, Next lexeme is =
Next token is: Unknown, Next lexeme is =
Next token is: Unknown, Next lexeme is b
Next token is: EOF, Next lexeme is EOF
```

Graph 4 Test3

Analysis:

The program basically completes the design of the state diagram, is able to recognize the C comments as required, and adds the "//..." comment recognition.

Ⅴ. Experimental experience and gains

1. Learn about state graph.
2. Further understanding and learning of lexical analyzer.
3. The code given in the book was studied in depth, and further optimized and modified.
4. I have benefited from searching the Internet for lexical analyzers written in various languages.

5. Some minor difficulties were encountered in the preparation. For example, how to distinguish between a comment with * in its termination symbol and a comment with * in it? I finally solved the problem by judging that the last two characters are equal to '*' and '/' at the same time.

VI. Core Code

```c
/***********************************************************************************
* @File name: assignment2.c
* @Author: H9
* @Version: 1.1
* @Date: 2022-10-22
* @Description: Implementing a lexical parser for identifying C annotations.
* It can only identify two types of C comments.
***********************************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <string.h>

/* Global declarations */
/* Variables */
int charClass;
char lexeme[100];
char nextChar;
int lexLen;
int token;
char nextToken[50];
FILE* fptr;

/* Function declarations */
void addChar();
void getChar();
void getNonBlank();
char* lex();
void show();

/* Character classes */
#define Symbol 0

/* Token codes */
#define CommentIdentifier1 "/*...*/"
#define CommentIdentifier2 "//..."
#define Comment "Comment"
```

```
#define Unknown_OP "Unknown"

/* main driver */
int main(int argc, char *argv[]) {
    /* Open the input data file and process its contents */
    if ((fptr = fopen("data1.txt", "r")) == NULL)
        printf("ERROR - cannot open front.in \n");
    else {
        getChar();
        do {
            lex();
        } while (strcmp(nextToken,"EOF"));
    }
}

/* lookup - a function to lookup operators and parentheses
            and return the token */
char* lookup(char ch) {
    char second;
    switch (ch)
    {
         case '/':
             addChar();
             //Comment Type1
             second = getc(fptr);
             if(second  == '/')
             {
                 //Analysis comment identifier
                 nextChar = second;
                 addChar();
                 strcpy(nextToken, CommentIdentifier2);
                 show();

                 //Analysis commnet
                 lexLen = 0;
                 while((nextChar = getc(fptr)) != EOF && nextChar != '\n')
                 {
                  addChar();
                 }
                 strcpy(nextToken, Comment);
                 show();

             }
             //Comment Type2
            else if(second == '*')
            {
               //Analysis comment identifier
               nextChar = second;
               addChar();
               strcpy(nextToken, CommentIdentifier1);
                         show();

               //Analysis comment
               char last;
               lexLen = 0;
                   while((nextChar = getc(fptr)) != EOF && !(last == '*' && nextChar ==
'/'))
               {
                last = nextChar;
                addChar();
               }
               for(int i = 0;i<sizeof(lexeme);i++)
               {
                    if(lexeme[i] =='\0')
                 {
                    lexeme[i-1]='\0';
                 }
               }
              strcpy(nextToken, Comment);
              show();


              //Analysis comment identifier
              lexLen = 0;
              nextChar = '*';
              addChar();
              nextChar = '/';
              addChar();
              strcpy(nextToken, CommentIdentifier1);
              show();
               }
            //Not comment
            else
            {
            nextChar = second;
                      addChar();
            strcpy(nextToken, Unknown_OP);
            show();
```

```
             }
             break;

        //other situations
            default:
           addChar();
           strcpy(nextToken, Unknown_OP);
           show();
           break;
    }
    return nextToken;
}

/* addChar - a function to add nextChar to lexeme */
void addChar() {
    if (lexLen < sizeof(lexeme)-1) {
        lexeme[lexLen++] = nextChar;
        lexeme[lexLen] = 0;
    } else
        printf("Error - lexeme is too long \n");
}

/* getChar - a function to get the next character of
             input and determine its character class */
void getChar() {
    if ((nextChar = getc(fptr)) != EOF) {
            charClass = Symbol;
    } else
        charClass = EOF;
}

/* getNonBlank - a function to call getChar until it
                 returns a non-whitespace character */
void getNonBlank() {
    while (isspace(nextChar))
        getChar();
}

/* lex - a simple lexical analyzer for arithmetic
         expressions */
char* lex() {
    lexLen = 0;
    getNonBlank();
    switch (charClass) {
        /* Parse identifiers */

        /* Comments and others */
      case Symbol:
        lookup(nextChar);
    getChar();
        break;

        /* EOF */
      case EOF:
        strcpy(nextToken, "EOF");
        lexeme[0] = 'E';
        lexeme[1] = 'O';
        lexeme[2] = 'F';
        lexeme[3] = 0;
    show();
        break;
    } /* End of switch */
    return nextToken;
}

//Shows nextToken and lexeme
void show()
{
    printf("Next token is: %s, Next lexeme is %s\n",
           nextToken, lexeme);
}
```

VII. Reference

1. [12th] Robert W. Sebesta - Concepts of Programming Languages - libgen.lc

2. **https://stackoverflow.com/questions/73751818/lexical-analyzer-to-c/73753954#73753954**

3. **https://github.com/linyacool/lexical_syntax_analysis/blob/master/%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90.cpp**

4. **https://www.thecrazyprogrammer.com/2017/02/lexical-analyzer-in-c.html**

5. **http://techteach.no/publications/state_diagrams/index.htm**