System Programming Assignment 3

Lexical Analysis

Assignment document

Name : GENG XU

Student ID: 12132031d

Submission contains:

This document

A lex.c source code

A compiled a.out file

A readme file for running the c code

Test sample code(input_file and input_file2)

1.regular expression:

1.1 KEYWORD -> var|begin|end.

1.2 COMMA -> ,

1.3 SEMICOLON -> ;

1.4 ASSIGN -> =

1.5 PERIOD -> .

1.6 PLUS -> +

1.7 MINUS -> -

1.8 MUL -> *

1.9 DIV -> /

1.10 LBRACE -> (

1.11 RBRACE ->)

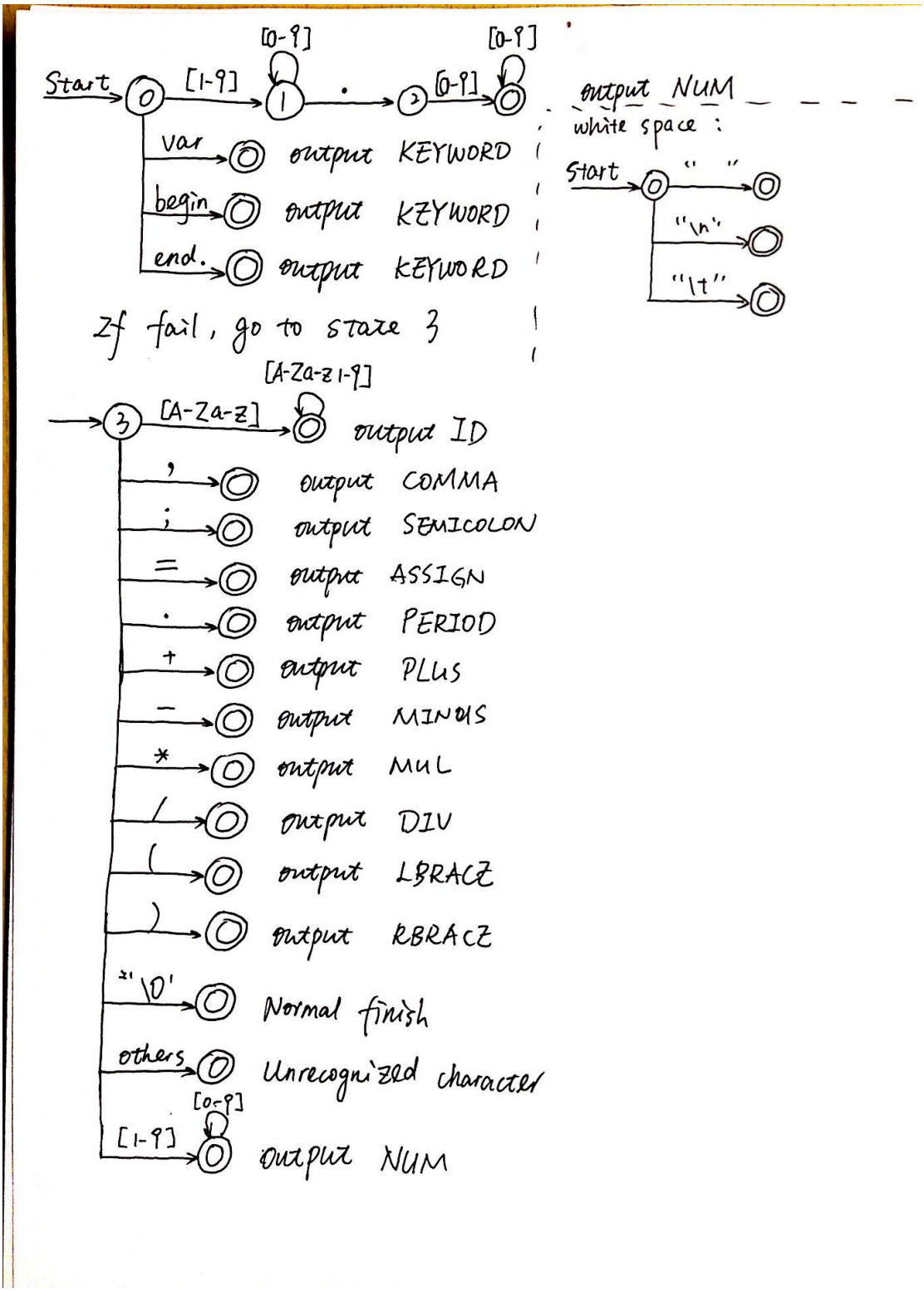
1.12 NUMBER -> digit*.?digit*

1.13 ID letter(letter|digit)*

--> with letter -> A|B|C|...|Z|a|b|c|...|z

digit -> 0|1|2|...|8|9

2.Finite Automata



3.program description

3.1 main function

Function:

In main function, first of all, check the argument

```
if( argc != 2){
        printf("Usage: scan input_file \n");
        return -1;
}
```

Then ,open the file indicated by the argument

```
if( (fd = open(argv[1], O_RDONLY)) < 0 ){...}
```

finally, call the scan() function to begin lexical analysis

```
if (scan() > 0)
```

3.2 scan() function

Essential variables:

input_buf: pointer pointing to the string read out from the file

start_pt: pointer pointing to the start of current token

forward: pointer pointing to the current character of current token

function:

first scan the content of the target file by

```
rbytes = read( fd, input_buf, MAXBUFSIZE );
```

and append an end character to it

```
input_buf[rbytes-1]='\0';
```

then start to get tokens by get_next_token() function

```
while( ( ret=get_next_token() ) >= 0 ){
        print_token(ret);
        start_pt = start_pt + forward +1;
}
```

After calling get_next_token():

Normal case:

(1) The return value, which is the content of the token,indicated by a tokenType enum, will be assigned to var ret.

(2) Print_token() function will accept ret and print the table on the terminal

(3) The start_pt will be re-located to start of next token.

Terminate case:

If the get_next_token() return -1, the lexical analysis terminate successfully.

If the get_next_token() return -2, the lexical analysis encounter undefined character

Finally, close file

close(fd);

3.3 get_next_token() function

Essential variables:

c: current character

forward: offset of c relative to the start of current token

current_state: state in finite automata

some basic statements:

1. Finish current token and return

```
// use forward to backtrack until the start of the token
    for(i=0; i<=forward; i++)
        token_val[i]=start_pt[i];
    token_val[i]='\0';  // end current token with '\0'
    return tokenType;  // return the tokenType enum of current token
```

2. Deal with a looping acceptance state:

For example, to deal with numbers -> [0-9]*

```
    if(c>47&&c<58){//is 0-9
        while(1){
            forward++;
            c = start_pt[forward]; // go to see next character
            if(!(c>47&&c<58))
                break;   // if not fulfill, then break the while(1) loop
        }
            forward--; // move backward to remove overhead
            for(i=0; i<=forward; i++) // same as 1
                token_val[i]=start_pt[i];
```

```
                token_val[i]='\0';
                return NUM;
        }else{…}
```

function:

This function is an infinite loop to read characters in a token until the token ends

First of all, get a character to analyze

```
c = start_pt[forward];
```

Then, according to the current_state, make judgement according to the finite automata

At state 0 (case 0):

Firstly, need to deal with white space ‘ ’ | ’\t’ | ’\n’, accepted

```
if(c == ' '||c=='\t'||c=='\n'){//deal with white space
        for(i=0; i<=forward; i++)
                token_val[i]=start_pt[i];
        token_val[i]='\0';
        return delim;
}
```

Secondly, need to deal with NUM (actually this is only for decimal numbers), next state = 1

```
else if(c>48&&c<58){//is 1-9
        forward++;
        current_state = 1;
}
```

Finally, deal with KEYWORD (var|begin|end.), accepted

```
else{
        char c_kw[5];
        c_kw[0] = start_pt[forward];
        c_kw[1] = start_pt[forward+1];
        c_kw[2] = start_pt[forward+2];
        c_kw[3] = start_pt[forward+3];
        c_kw[4] = start_pt[forward+4]; //get 5 chars because the longest is “begin”
```

```
        if(c_kw[0]=='v'&&c_kw[1]=='a'&&c_kw[2]=='r'){// if var
                forward = 2;
        }elseif(c_kw[0]=='b'&&c_kw[1]=='e'&&c_kw[2]=='g'&&c_kw[3]=='i'&&c_kw
[4]=='n'){// if begin
                forward = 4;
        }else if(c_kw[0]=='e'&&c_kw[1]=='n'&&c_kw[2]=='d'){// if end.
                forward = 2;
        }else{//if fail, call fail function to go to state 3
                current_state=fail(current_state);
                break;
        }
        for(i=0; i<=forward; i++)
                token_val[i]=start_pt[i];
        token_val[i]='\0';
        return KEYWORD;
}
```

At state 1:

If the character is a number(actually, for decimal numbers), still in state 1

```
        if(c>47&&c<58){//is 0-9
                forward++;
                current_state = 1;
        }
```

If the character is a decimal point(for decimal numbers), go to state 2

```
        else if(c=='.'){
                forward++;
                current_state = 2;
        }
```

If neither, call fail function to go to state 3

```
        else{
                current_state=fail(current_state);
```

```
}
```

At state 2

Accept state with a loop until the character is no longer a digit(0-9) (for decimal number)

```
if(c>47&&c<58){//is 0-9
        while(1){
                forward++;
                c = start_pt[forward];
                if(!(c>47&&c<58))
                        break;
                }
                forward--;
                for(i=0; i<=forward; i++)
                        token_val[i]=start_pt[i];
                token_val[i]='\0';
                return NUM;
        }
        else{// if fail, go to state 3
                current_state=fail(current_state);
        }
```

At state 3:

If the character is one of : “,;=.+-*/()” , accept

COMMA as an example: very similar for others

```
        case ',':
                for(i=0; i<=forward; i++)
                        token_val[i]=start_pt[i];
                token_val[i]='\0';
                return COMMA;
```

If it is the end character ‘\0’, return -1 to indicate finish

```
        case '\0':
```

```
            return -1;
```

For the default of switch statement:

Firstly check for integer number

```
    if(c>48&&c<58){//is number 1-9
            while(1){// accept state with loop
                    forward++;
                    c = start_pt[forward];
                    if(!(c>47&&c<58))//not 0-9
                            break;
            }
            forward--;
            for(i=0; i<=forward; i++)
                    token_val[i]=start_pt[i];
            token_val[i]='\0';
            return NUM;
    }
```

Secondly, check for ID

```
else if((c>64&&c<91)||(c>96&&c<123)){//is letter
while(1){ //accept state with loop
    forward++;
    c = start_pt[forward];
    if(!((c>64&&c<91)||(c>96&&c<123)||(c>47&&c<58)))//not 0-9 nor letter
            break;
}
    forward--;
    for(i=0; i<=forward; i++)
            token_val[i]=start_pt[i];
    token_val[i]='\0';
    return ID;
}
```

Finally, fail in state 3 means unrecognized characters, return -2

```
else{
    return -2;
}
```

3.4 fail()

For finite automata, if it fails in one state, the program should know what state is the next state. In my design, fail in state 0-2 will lead the program to state 3

```
if( cstate < 3 )
    next_state = 3;
```

if fail in state 3, it indicates unrecognized token, the handler is in the get_next_token() switch case 3 return -2

3.5 print_token()

Print the token in the terminal by accepting the TokenType enum and make judgement by switch statement

```
switch(token){
    case NUM:
        printf("NUM\t\t\t| %s\n",token_val);
        break;
    …
}
```

3. Source code

Attached as lex.c

4. Input_file sample

Attached as input_file and input_file2

Input_file is the one given with the assignment description

Input_file2 is revised a little bit by me to test IDs with mixed digits and letters.