Scanner

B11130038 Jia-Hong, Wang

Introduction

1. Include the library and define the token number, token name, line number, union type, and error function.

```
%{
	#include<stdio.h>
	enum token{
		VAR = 256, VAL, IDENTIFIER, INT, INTEGER,
		REAL,	FLOAT, CHAR, CHARACTOR, BOOL,
		TRUE, FALSE, STRING, CLASS, IF,
		ELSE, FOR, WHILE, EQ, NE,
		GE, LE, DO,
		SWITCH, CASE, FUN, RET, MAIN,PRINTLN
	};
	const char* tokenName[]={
		"VAR", "VAL",	"IDENTIFIER", "INT","INTEGER",
		"REAL", "FLOAT", "CHAR", "CHARACTOR", "BOOL",
		"TRUE", "FALSE", "STRING", "CLASS", "IF",
		"ELSE", "FOR", "WHILE", "EQ", "NE",
		"GE", "LE", "DO", "SWITCH", "CASE",
		"FUN", "RET","MAIN", "PRINTLN"
		};
	int lineno = 1;
	union type{
		int d;
		char c;
		char* s;
		float f;
	} yylval;
	char stringBuffer[1024];
	void yyerror(const char *s);
%}
```

→ Include library <stdio.h> because the program will use “printf”, “scanf” and other function

→ Define a token number from 256 to the end because the first 255 numbers are reserved for extended ASCII

→ Define token name to visualize the result

→ Define lineno to know which line the error occur

→ Define the type to choose which type the token is

→ Define the string buffer to get the string(maximum character is 1024)

→ Define yyerror to handle error message

2. Define all the state

```
%x CHARSTART
%x CHARESCAPE
%x MULTIPLECOMMENT
%x SINGLECOMMENT
```

```
%x STRINGSTATE
%x STRINGESCAPE
```

→ CHARSTART：charactor state

→ CHARESCAPE：charactor state with escape token

→ MULTIPLECOMMENT：multiple lines comment state

→ SINGLECOMMENT：single line comment state

→ STRINGSTATE：string state

→ STRINGESCAPE：string state with escape token

3. Scan the normal token we want

```
"main"                          { return MAIN;}
"var"                           { return VAR; }
"val"                           { return VAL; }
"bool"                          { return BOOL; }
"char"                          { return CHAR; }
"int"                           { return INT; }
"real"                          { return REAL; }
"true"                          { return TRUE; }
"false"                         { return FALSE; }
"class"                         { return CLASS; }
"if"                            { return IF; }
"else"                          { return ELSE; }
"for"                           { return FOR; }
"while"                         { return WHILE; }
"do"                            { return DO; }
"switch"                        { return SWITCH; }
"case"                          { return CASE; }
"fun"                           { return FUN; }
"ret"                           { return RET; }
```

→ scan all normal token and return

4. Scan identifier, number, and other single tokens.

```
[a-zA-Z_][a-zA-Z0-9_]*    { yylval.s=yytext;return IDENTIFIER; }
[0-9]+                          { yylval.d = atoi(yytext); return INTEGER; }
[0-9]+[a-zA-Z_\'\"]        {yyerror("invalid integer definition");yyterminate();}
[0-9]+\.[0-9]+                 { yylval.f = atof(yytext); return FLOAT; }
[(),\[\]{};,:\+\-\*\/<>=]     { return yytext[0]; }
```

→ identifier start with a-z, A-Z, or underscore, followed by a-z, A-Z, underscore, or 0-9

→ integer is a series of numbers

→ if numbers followed by other character, the token is invalid, so output error and terminate the program

→ real number is a series of number followed by a dot and a series of number

→ other single tokens we want to scan

5. Scan judge symbol, line, tab and single backslah

```
"=="                          { return EQ; }
"!="                          { return NE; }
">="                          { return GE; }
"<="                          { return LE; }
[\n\r]+                       { lineno++;}
[\t ]                         { ; }
```

```
"\\"                                {yyerror("invalid escape charactor");yyterminate();}
```

→ scan equal(==), not equal(!=), greater equal(>=), less equal(<=) to get the token

→ \n\r is a new line token, but Linux only has \n, MacOS only has \r, and Window has both, so we may count more than 1 new line token

→ ignore tab and blank

→ if the program scan only one backslash(\), the token is invalid.

6. Scan the charactor

```
\'                              {BEGIN(CHARSTART);}
<CHARSTART>[\'\"\n]         {yyerror("invalid character");yyterminate();}
<CHARSTART><<EOF>>           {yyerror("missing terminating ' character");yyterminate();}
<CHARSTART>\\\'              {yyerror("missing terminating ' character");yyterminate();}
<CHARSTART>\\\'\'           {yylval.c='\'';BEGIN(INITIAL);return CHARACTER;}
<CHARSTART>\\                 {BEGIN(CHARESCAPE);}
<CHARSTART>.\'                {yylval.c=*yytext;BEGIN(INITIAL);return CHARACTER;}
<CHARESCAPE>(\\|\'|\"|\?)\' { yylval.c=yytext[0]; BEGIN(INITIAL);return CHARACTER;}
<CHARESCAPE>t\'               {yylval.c=9;BEGIN(INITIAL);return CHARACTER;}
<CHARESCAPE>n\'                {yylval.c=10;BEGIN(INITIAL);return CHARACTER;}
<CHARESCAPE><<EOF>>         {yyerror("invalid escape character");yyterminate();}
<CHARESCAPE>.                  {yyerror("invalid escape character");yyterminate();}
```

→ if the program scan single quotation mark, start character state

→ in character state

  → if the program scan ‘, “, \n, EOF or \’, the token is invalid.

  → if the program scan \, go to character escape state

  → if the program scan \’’, the value is ‘ and go to initial state.

  → if the program scan other single character with single quotation mark, set the value to the character.

→ in character escape state

  → if the program scan \, ‘, “, ? followed by ‘, set the value to the first character

  → if the program scan t, n followed by ‘, set the value to 9, 10 due to ASCII.

  → if the program scan EOF or other character, the escape character is invalid.

7. Scan the string

```
\"                               { BEGIN(STRINGSTATE); stringBuffer[0] = '\0'; }
<STRINGSTATE>\"               { yylval.s = strdup(stringBuffer); BEGIN(INITIAL); return STRING; }
<STRINGSTATE>\\               { BEGIN(STRINGESCAPE); }
<STRINGSTATE>[^\\\n\"]+    { strcat(stringBuffer, yytext); }
<STRINGSTATE>\n                { yyerror("missing terminating \" character"); yyterminate(); }
<STRINGESCAPE>n                { strcat(stringBuffer, "\n"); BEGIN(STRINGSTATE); }
<STRINGESCAPE>t                { strcat(stringBuffer, "\t"); BEGIN(STRINGSTATE); }
<STRINGESCAPE>\"               { strcat(stringBuffer, "\""); BEGIN(STRINGSTATE); }
<STRINGESCAPE>\\               { strcat(stringBuffer, "\\"); BEGIN(STRINGSTATE); }
<STRINGESCAPE>\'               { strcat(stringBuffer, "\'"); BEGIN(STRINGSTATE); }
<STRINGESCAPE>\?               { strcat(stringBuffer, "\?"); BEGIN(STRINGSTATE); }
<STRINGESCAPE>.                { yyerror("invalid escape character"); yyterminate(); }
<STRINGESCAPE><<EOF>>       { yyerror("EOF in string constant"); yyterminate(); } }
```

→ if the program scan “, start string state

→ string state

  → if the program scan “, set the value to the string buffer address, back to initial state and return the token

→ if the program scan \, go to string escape state

→ if the program scan \n, the string is invalid

→ if the program other character, put the character in the string buffer

→ string escape state

→ if the program scan \, ‘, “, ?,n or t, put them to the string buffer and back to string state

→ if the program scan EOF or other character, the string is invalid.

8. Scan the single line comment and ignore it

```
"//"                              { BEGIN SINGLECOMMENT; }
<SINGLECOMMENT>[^\n]*             { ; }
<SINGLECOMMENT>\n                 { lineno++;BEGIN 0; }
```

→ if scan //, start single comment state

→ in single comment state, the program will ignore characters until new line

9. Scan the multiple line comment and ignore it and scan other characters

```
"/*"                              { BEGIN(MULTIPLECOMMENT); }
<MULTIPLECOMMENT>"*/"             { BEGIN(INITIAL); }
<MULTIPLECOMMENT>.                { ; }
<MULTIPLECOMMENT>\n               { lineno++; }
<MULTIPLECOMMENT><<EOF>>          { yyerror("Unclosed comment at end of file."); yyterminate(); }

.                                 {yyerror("scanner error");yyterminate();}
%%
```

→ if scan /*, start multiple comment state

→ multiple comment state

→ if the program scan */ go to initial state

→ if the program scan \n, line number add 1

→ if the program scan other characters, ignore it

→ if the program scan EOF, the comment is invalid

→ if none of the tokens is scanned, the token is invalid

10. Handle end function and error message

```
int yywrap(void) {
	return 1;
}

void yyerror(const char *s) {
	printf("scanner error. line %d: %s at yytext:(%s)\n", lineno, s, yytext);
}
```

→ after end of the lex, the program will not scan other things

→ yyerror is used to handle error

11. main function

```
int main(void) {

	int mode;
	while(1){
		printf("input 1 to input mode and input 2 to file mode:");
		scanf("%d",&mode);
		if(mode==1||mode==2) break;
		else printf("invalid input to choose the mode\n");
	}
	while (mode == 2) {
		char sFile[256];
		printf("Input the path of the file: ");
		scanf("%255s", sFile);
		FILE *fp = fopen(sFile, "r");
		if (fp == NULL) {
			printf("Cannot open %s\n", sFile);
		}
		else {
			yyin = fp;
			break;
		}
	}
	int token;
	while(token = yylex())
	{
		if(token>255){
			printf("<%d,%s", token, tokenName[token-256]);
			switch(token){
			case IDENTIFIER:
				printf(",%s>\n",yylval.s);
				break;
			case INTEGER:
				printf(",%d>\n",yylval.d);
				break;
			case FLOAT:
				printf(",%f>\n",yylval.f);
				break;
			case CHARACTOR:
				printf(",%c>\n",yylval.c);
				break;
			case STRING:
				printf(",%s>\n",yylval.s);
				break;
			default:
				printf(">\n");
				break;
			}
		}
		else if(token<=255){
			printf("<%d,%c>\n", token,(char)token);
		}
	}
}
```
→ the program should start to input 1 or 2 to choose input mode or read mode. If inputting 1, the scanner will scan line by line. If inputting 2 and input file address, the program will scan the file

How to use

lex

Input 1 or 2 to choose input mode or file mode
Input mode: input and scan line by line
File mode: input the file address and can the file

Token Format

- <tokenNumber, token> If the token number is less than 256
- < tokenNumber, tokenName> If the token number is higher or equal to 256 without value
- < tokenNumber, tokenName, value> If the token number is higher or equal to 256 within value

Demo program

1. sample1.txt
   A. input

```
// qv Sample Program No. 1
fun main () {                          // Function definition
        var i: int = 10;               // Integers; always signed
        var j: real = 3.14159; // Real numbers; always signed
        var k: char = 'c';          // Character; in ASCII encoding
        var l: int[5];                  // 1D array (/vector) with 5 integers
        var m: int[3][4];             // 2D array with 3 rows, each with 4 integers
        var n: char[10] = "Hello, world!"; // 1D arrays with characters are strings
        println(i);                       // Function call; print i and a new line character
        i = 20;                             // Assign a new value 20 for i
        println(i);
        l = {1, 2, 3, 4, 5};      // Assign a vector with 5 integers 1, 2, 3, 4, 5 in order
        println(l);
        k = '\\';                              // Assign a char with new value '\\' (backslash)
        println(k);
        println(n);
        n = "Another string";      /*Test C-style comments*/ n = "Third string";
        println(n);
        ret;                                      // Return nothing to terminate the function body
}
```

   B. output

```
<281,FUN>
<283,MAIN>
<40,(>
<41,)>
<123,{>
<256,VAR>
<258,IDENTIFIER,i>
<58,:>
<259,INT>
<61,=>
<260,INTEGER,10>
<59,;>
<256,VAR>
<258,IDENTIFIER,j>
<58,:>
```

```
<261,REAL>
<61,=>
<262,FLOAT,3.141590>
<59,;>
<256,VAR>
<258,IDENTIFIER,k>
<58,:>
<263,CHAR>
<61,=>
<264,CHARACTOR,c>
<59,;>
<256,VAR>
<258,IDENTIFIER,l>
<58,:>
<259,INT>
```

```
<91,[>
<260,INTEGER,5>
<93,]>
<59,;>
<256,VAR>
<258,IDENTIFIER,m>
<58,:>
<259,INT>
<91,[>
<260,INTEGER,3>
<93,]>
<91,[>
<260,INTEGER,4>
<93,]>
<59,;>
```

```
<256,VAR>
<258,IDENTIFIER,n>
<58,:>
<263,CHAR>
<91,[>
<260,INTEGER,10>
<93,]>
<61,=>
<268,STRING,Hello,
world!>
<59,;>
<284,PRINTLN>
<40,(>
<258,IDENTIFIER,i>
<41,)>
<59,;>
<258,IDENTIFIER,i>
<61,=>
<260,INTEGER,20>
<59,;>
<284,PRINTLN>
<40,(>
<258,IDENTIFIER,i>
<41,)>
<59,;>
```

```
<258,IDENTIFIER,l>
<61,=>
<123,{>
<260,INTEGER,1>
<44,,>
<260,INTEGER,2>
<44,,>
<260,INTEGER,3>
<44,,>
<260,INTEGER,4>
<44,,>
<260,INTEGER,5>
<125,}>
<59,;>
<284,PRINTLN>
<40,(>
<258,IDENTIFIER,l>
<41,)>
<59,;>
<258,IDENTIFIER,k>
<61,=>
<264,CHARACTOR,\>
<59,;>
<284,PRINTLN>
<40,(>
```

```
<258,IDENTIFIER,k>
<41,)>
<59,;>
<284,PRINTLN>
<40,(>
<258,IDENTIFIER,n>
<41,)>
<59,;>
<258,IDENTIFIER,n>
<61,=>
<268,STRING,Another
string>
<59,;>
<258,IDENTIFIER,n>
<61,=>
<268,STRING,Third string>
<59,;>
<284,PRINTLN>
<40,(>
<258,IDENTIFIER,n>
<41,)>
<59,;>
<282,RET>
<59,;>
<125,}>
```

2. test1.qv

A. input

```
// qv Sample Test No. 1: bubble sort
fun main () {                       // Function definition
    var list: char[5];
    var i : int;
    var j : int;
    var tmp : char;
    var length : int = 5;
    list = {'1','3','e','\\','2'};
    for(i=0;i<length;i=i+1){
        for(j=0;j<length-i;j=j+1){
            if(list[j+1]>list[j]){
                tmp = list[j+1];
          list[j+1] = list[j];
          list[j] = tmp;
            }
        }
    }
    for(i=0;i<length;i=i+1){
        println(list[i]);
    }
}
```

B. output

```
<281,FUN>
<283,MAIN>
<40,(>
<41,)>
<123,{>
<256,VAR>
<258,IDENTIFIER,list>
```

```
<58,:>
<263,CHAR>
<91,[>
<260,INTEGER,5>
<93,]>
<59,;>
<256,VAR>
```

```
<258,IDENTIFIER,i>
<58,:>
<259,INT>
<59,;>
<256,VAR>
<258,IDENTIFIER,j>
<58,:>
```

```
<259,INT>
<59,;>
<256,VAR>
<258,IDENTIFIER,tmp>
<58,:>
<263,CHAR>
<59,;>
<256,VAR>
<258,IDENTIFIER,length>
<58,:>
<259,INT>
<61,=>
<260,INTEGER,5>
<59,;>
<258,IDENTIFIER,list>
<61,=>
<123,{>
<264,CHARACTER,1>
<44,,>
<264,CHARACTER,3>
<44,,>
<264,CHARACTER,e>
<44,,>
<264,CHARACTER,\>
<44,,>
<264,CHARACTER,2>
<125,}>
<59,;>
<272,FOR>
<40,(>
<258,IDENTIFIER,i>
<61,=>
<260,INTEGER,0>
<59,;>
<258,IDENTIFIER,i>
<60,<>
<258,IDENTIFIER,length>
<59,;>
<258,IDENTIFIER,i>
<61,=>
<258,IDENTIFIER,i>
<43,+>
<260,INTEGER,1>
<41,)>
<123,{>
<272,FOR>
```

```
<40,(>
<258,IDENTIFIER,j>
<61,=>
<260,INTEGER,0>
<59,;>
<258,IDENTIFIER,j>
<60,<>
<258,IDENTIFIER,length>
<45,->
<258,IDENTIFIER,i>
<59,;>
<258,IDENTIFIER,j>
<61,=>
<258,IDENTIFIER,j>
<43,+>
<260,INTEGER,1>
<41,)>
<123,{>
<270,IF>
<40,(>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<43,+>
<260,INTEGER,1>
<93,]>
<62,>>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<93,]>
<41,)>
<123,{>
<258,IDENTIFIER,tmp>
<61,=>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<43,+>
<260,INTEGER,1>
<93,]>
<59,;>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<43,+>
```

```
<260,INTEGER,1>
<93,]>
<61,=>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<93,]>
<59,;>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,j>
<93,]>
<61,=>
<258,IDENTIFIER,tmp>
<59,;>
<125,}>
<125,}>
<125,}>
<272,FOR>
<40,(>
<258,IDENTIFIER,i>
<61,=>
<260,INTEGER,0>
<59,;>
<258,IDENTIFIER,i>
<60,<>
<258,IDENTIFIER,length>
<59,;>
<258,IDENTIFIER,i>
<61,=>
<258,IDENTIFIER,i>
<43,+>
<260,INTEGER,1>
<41,)>
<123,{>
<258,IDENTIFIER,println>
<40,(>
<258,IDENTIFIER,list>
<91,[>
<258,IDENTIFIER,i>
<93,]>
<41,)>
<59,;>
<125,}>
<125,}>
```

3. Error1.qv

A. input

```
// qv Sample Test No. 1: bubble sort with escape micro syntax error
fun main () {                        // Function definition
        var list: char[5];
        var i : int;
        var j : int;
        var tmp : char;
        var length : int = 5;
        list = {'1','3','e','\','2'};
        for(i=0;i<length;i=i+1){
```

```
        for(j=0;j<length-i;j=j+1){
            if(list[j+1]>list[j]){
                tmp = list[j+1];
                    list[j+1] = list[j];
                    list[j] = tmp;
            }
        }
    }
    for(i=0;i<length;i=i+1){
        println(list[i]);
    }
}
```

B. output

```
<281,FUN>
<283,MAIN>
<40,(>
<41,)>
<123,{>
<256,VAR>
<258,IDENTIFIER,list>
<58,:>
<263,CHAR>
<91,[>
<260,INTEGER,5>
<93,]>
<59,;>
<256,VAR>
<258,IDENTIFIER,i>
<58,:>
<259,INT>
<59,;>
<256,VAR>
<258,IDENTIFIER,j>
<58,:>
<259,INT>
<59,;>
```

```
<256,VAR>
<258,IDENTIFIER,tmp>
<58,:>
<263,CHAR>
<59,;>
<256,VAR>
<258,IDENTIFIER,length>
<58,:>
<259,INT>
<61,=>
<260,INTEGER,5>
<59,;>
<258,IDENTIFIER,list>
<61,=>
<123,{>
<264,CHARACTER,1>
<44,,>
<264,CHARACTER,3>
<44,,>
<264,CHARACTER,e>
<44,,>
scanner error. line 8: missing terminating ' character at yytext:(\')
```

4. error2.qv

A. input

```
/* qv Sample Test No. 3: invalid comment
fun main () {

}
```

B. output

```
scanner error. line 4: Unclosed comment at end of file. at yytext:()
```

5. error3.qv

A. input

```
// qv Sample Test No. 4: invalid symbol
fun main () {                    // Function definition
    var list: char# ='1';

}
```

B. output

```
<281,FUN>
<283,MAIN>
```

```
<40,(>
<41,)>
<123,{>
<256,VAR>
<258,IDENTIFIER,list>
<58,:>
<263,CHAR>
scanner error. line 3: scanner error at yytext:(#)
```

Reference

1. Lex & Yacc 學習筆記 :: 2023 iThome 鐵人賽：
   https://ithelp.ithome.com.tw/users/20157613/ironman/6494
2. 简易编译器实现（一）使用 Flex 创建词法分析器 | 胡刘郏的技术博客 (huliujia.com)
   https://www.huliujia.com/blog/7bdf23e1aadefa13286b73c3aa4063a5836b1a37/