編譯程式

Programming Assignment 1

Lexical analyzer

for miniC language

系級： 資工三
學號： 410121021
姓名： 林育慈

2015. 4. 7

Problem Description

1. Use lex or flex to implement a lexical analyzer for the miniC language.

   ➢ The lexical rules in details.

   - **Integer**
     Sequence of digits denoting an integer number in the range -32768...32767. This should be stored in a two's-complement representation.
     (Hint: The value range will be checked in th ephase of semantic analusis.)

   - **Identifiers**
     Sequence of letters, digits and underscores that may only be initiated with inderscores or letters, no longer than 16 characters.
     (Hint: The length of squence will be checked in the phase of semantic analysis.)

   - **Strings**
     A string negins with a " and ends with a ". No new-line or " is allowed to appear in a string.
     (Hint: Strings are only for the use in printf.)

   - **Reserved words**
     `break continue else if int return while printf`
     (Hint: Exah reserved word is a token type.)

   - **Special characters**
     `+ - * / % ! ? : = , < > ( ) { } || && == "`
     (Hint: Each special character or sequence of special characters is a token type.)

   - **Comments**
     A comment begins with `//` and goes to the end of the line.

   ➢ You are requested to separate the C code and the Lex specification into distinct files.

Program listing

Makefile // 用來呼叫flex並編譯產生執行檔

```
main:  c_lex.o main.o
	gcc -o scanner.exe c_lex.o main.o

c_lex.o: c_lex.c
	gcc -c c_lex.c
c_lex.c: c_lex.l
	flex -oc_lex.c c_lex.l
main.o: main.c
	gcc -c main.c

# To clean the generated files
clean:
	rm *.o c_lex.c scanner.exe
```

c_lex.h // lexical的定義

```
#define BREAK 0
#define CONTINUE 1
#define ELSE 2
#define IF 3
#define INT 4
#define RETURN 5
#define WHILE 6
#define PRINTF 7
#define ADD 10
#define MINUS 11
#define TIMES 12
#define DIV 13
#define MOD 14
#define NOT 15
#define QUES 16
#define COLON 17
#define ASSIGN 28
#define COMMA 19
#define LT 20
#define GT 21
#define LP 22
#define RP 23
#define LSP 24
#define RSP 25
#define OR 26
#define AND 27
#define EQ 18
#define QUOTE 29
#define SEMI 30
#define ID 31
#define NUM 32
#define STRING 9
#define COMMENT 8

extern int yylex();
extern FILE *yyin;
extern char c_name[16];
extern int c_val;
```

c_lex.l // lexical定義、規則，讓flex用來產生c_lex.c

```
%{
#include "c_lex.h"
%}

ID [A-Za-z_][A-Za-z0-9_]*
NUM [0-9]+
STRING \"(\\.|[^"]|[^\n])*\"
COMMENT "//"[^"\n"]*

%%
break       {return BREAK; }
continue    {return CONTINUE; }
else        {return ELSE; }
if          {return IF; }
int         {return INT; }
return      {return RETURN; }
while       {return WHILE; }
printf      {return PRINTF; }
{STRING}    {sscanf(yytext, "%s", c_name); return STRING; }
"+"         {return ADD; }
"-"         {return MINUS; }
"*"         {return TIMES; }
"/"         {return DIV; }
"%"         {return MOD; }
"!"         {return NOT; }
"?"         {return QUES; }
":"         {return COLON; }
"="         {return ASSIGN; }
","         {return COMMA; }
"<"         {return LT; }
">"         {return GT; }
"("         {return LP; }
")"         {return RP; }
"{"         {return LSP; }
"}"         {return RSP; }
"||"        {return OR; }
"&&"        {return AND; }
"=="        {return EQ; }
"\""        {return QUOTE; }
";"         {return SEMI; }
{ID}        {sscanf(yytext, "%s", c_name); return ID; }
{NUM}       {sscanf(yytext, "%d", &c_val); return NUM; }
{COMMENT}   {return COMMENT; }
[ \t\n]     {}
.           {}
%%

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

main.c // 結果的輸出

```
#include <stdio.h>
#include "c_lex.h"

char c_name[16];
int c_val;

void print_lex(int t){
	switch(t){
		case BREAK:		printf("BREAK\n");			break;
		case CONTINUE:	printf("CONTINUE\n");		break;
		case ELSE:		printf("ELSE\n");			break;
		case IF:		printf("IF\n");				break;
		case INT:		printf("INT\n");			break;
		case RETURN:	printf("RETURN\n");			break;
		case WHILE:		printf("WHILE\n");			break;
		case PRINTF:	printf("PRINTF\n");			break;
		case ADD:		printf("ADD\n");			break;
		case MINUS:		printf("MINUS\n");			break;
		case TIMES:		printf("TIMES\n");			break;
		case DIV:		printf("DIV\n");			break;
		case MOD:		printf("MOD\n");			break;
		case NOT:		printf("NOT\n");			break;
		case QUES:		printf("QUES\n");			break;
		case COLON:		printf("COLON\n");			break;
		case ASSIGN:	printf("ASSIGN\n");			break;
		case COMMA:		printf("COMMA\n");			break;
		case LT:		printf("LT\n");				break;
		case GT:		printf("GT\n");				break;
		case LP:		printf("LP\n");				break;
		case RP:		printf("RP\n");				break;
		case LSP:		printf("LSP\n");			break;
		case RSP:		printf("RSP\n");			break;
		case OR:		printf("OR\n");				break;
		case AND:		printf("AND\n");			break;
		case EQ:		printf("EQ\n");				break;
		case QUOTE:		printf("QUOTE\n");			break;
		case SEMI:		printf("SEMI\n");			break;
		case ID:		printf("ID: %s\n", c_name);	break;
		case NUM:		printf("NUM: %d\n", c_val);	break;
		case COMMENT:	printf("COMMENT");			break;
		case STRING:	printf("STRING: %s\n", c_name);	break;
		default:		printf("******** error!!\n");
	}
}

int main(int argc, char *argv[]){
	int t;
	yyin = fopen(argv[1], "r");
	t = yylex();
	while(t){
		print_lex(t);
		t = yylex();
	}
	fclose(yyin);
	return 0;
}
```

Test run results

test.c // 測試程式

```
int ComputeFac(int num){
	int num_aux;
	if (num < 1)
		num_aux = 1;
	else
		num_aux = num * ComputeFac(num - 1);
	return num_aux;
}

int main(){
	printf("%d\n", ComputeFac(10));
	// test comment
}
```

result.txt // 根據測試程式分析後的結果

```
1    INT
2    ID: ComputeFac
3    LP
4    INT
5    ID: num
6    RP
7    LSP
8    INT
9    ID: num_aux
10   SEMI
11   IF
12   LP
13   ID: num
14   LT
15   NUM: 1
16   RP
17   ID: num_aux
18   ASSIGN
19   NUM: 1
20   SEMI
21   ELSE
22   ID: num_aux
23   ASSIGN
24   ID: num
25   TIMES
26   ID: ComputeFac
27   LP
28   ID: num
29   MINUS
30   NUM: 1
31   RP
32   SEMI
33   RETURN
34   ID: num_aux
35   SEMI
36   RSP
37   INT
38   ID: main
39   LP
40   RP
41   LSP
42   PRINTF
43   LP
44   STRING: "%d\n"
45   COMMA
46   ID: ComputeFac
47   LP
48   NUM: 10
49   RP
50   RP
51   SEMI
52   COMMENT
53   RSP
```

Discussion

起初，根據Example的檔案去進行修改時，產生出來的檔案與應有的結果差異頗大，故開始思考原因。

首先第一個問題是，未將底線（_）視作ID可接受的字元，因而產生錯誤；第二個問題是，字串的判定，會一直將字串的內容分別去匹配；第三個問題則是註解的判定，註解應忽視該行 // 後所有字元，然而實際上還是會去做匹配。

第一個問題直接在定義的部份，將ID的regular expression做修改，使得ID能接受的變為 [A-Za-z_][A-Za-z0-9_] 便解決了問題；第二個問題卡了相當久的時間，後來想通是先抓到雙引號為token，再判斷後面字元不為換行或雙引號並重複，最後再以一個雙引號做結；註解的判斷亦類似於字串，先抓到//為token，判斷後面字元直到換行為止。

最後順利完成了這次的作業，也更了解了編譯器掃描時的運行原理了。