Lex Tutorial

最後更新於2022/3/17

Lex的工作

- Lex會把input當作 a sequence of characters
  - 一個以上連續的character會形成一個token
- Lex的目的是檢查token是否合法
  - 例如不合法的變數名稱(identifier)
- Lex必須事先定義規則
  - Regular expression
    - 可以被辨識的token

Lex的input

- 辨識下列兩句話的token
- 要求的token類別：String, Symbol

This is a Lex Example.

I love compiler, I love coding.

Lex格式

- 分成三部分，每個部分以%%區隔開來

Definition

%%

Lex Rules

%%

User code

example.lex完整內容

```
%{
#include<stdio.h>
unsigned charCount = 0,lineCount = 0;
void print_char();
%}
String      [a-zA-Z]+
space [ ]
eol \n
Symbol   .

%%
{String}    {charCount += yyleng;
        printf("String : \'%s\'\n", yytext);}
{space} {}
{eol}   {lineCount++;}
{Symbol} {print_char();}
%%
```

```
int main()
{
            yylex();
            printf("\n\nThe number of characters: %d\n",charCount);
            printf("The number of lines: %d\n", lineCount);
            return 0;
}

void print_char()
{
            printf("[In print_char]Symbol:\'%s\'\n",yytext);charCount++;
}
```

Definition

```
%{
#include<stdio.h>
unsigned charCount = 0,lineCount = 0;
void print_char();
%}
String      [a-zA-Z]+
Space       [ ]
eol         \n
Symbol      .
```

Rules

- 定義token及對應的action

- {String}　　{charCount += yyleng; printf("String : \'%s\'\n", yytext);}
  {space}　　{ }
  {eol}　　{lineCount++;}
  {Symbol} {print_char();}

- Scanner所匹配規則的優先順序
  - scanner會scan出長度最長的token去進行匹配
  - 如果匹配長度一樣，則看被定義的先後順序(由上到下)
  - 把space跟character的順序對調的話，則space沒辦法被匹配到

Our code

- int main()
  {
  yylex();
  printf("\n\nThe number of characters: %d\n",charCount);
  printf("The number of lines: %d\n", lineCount);
  return 0;
  }

- void print_char()
  {
  printf("[In print_char]Symbol:\'%s\'\n",yytext);charCount++;
  }

Output

```
wangyc@wangyc-ubuntu:~/Desktop/lex/testfile_lab1_2022
/example$ ./a.out < test.in
String : 'This'
String : 'is'
String : 'a'
String : 'Lex'
String : 'Example'
[In print_char]Symbol:'.'
'In print_char]Symbol:'
String : 'I'
String : 'love'
String : 'compiler'
[In print_char]Symbol:','
String : 'I'
String : 'love'
String : 'coding'
[In print_char]Symbol:'.'


The number of characters: 45
The number of lines: 1
```

Lex file中的特殊字元

- 這些字元在regular expression中有特殊意義，如果要當成一般字元，請在前面加上\這一個跳脫字元(Escape character)
  - ? * + | ( ) ^ $. [ ] { } " \
- Digit [0-9]
- Letter [a-zA-Z]
- Operator [\+\-*]

如何使用Lex file

- 我們的目的要將demo.l編譯成可以執行的scanner
- 首先必須安裝flex這個程式來編譯我們的lex file，以ubuntu為例
  - sudo apt-get install flex
- 透過flex將demo.l編譯成C source file，這個C source file就是我們的scanner
  - flex demo.l
- C source file預設檔名為lex.yy.c，最後我們可以利用gcc將其編譯成可執行檔
  - gcc lex.yy.c **-lfl**
- 執行檔為a.out，假設我們要scan的檔案為test.in
  - ./a.out < test.in
- 也可以直接執行a.out，<Ctrl-D>可以送出EOF

example2

- 以pascal為例(test.pas)

```
1  program test;
2  begin
3     writeln (10);
4  end.
```

Definition

(example2.l) **這份檔案只有部分正確**

```
%{
#include<stdio.h>
unsigned charCount = 1,lineCount = 1;
%}
reserved program|begin|end|writeln
space [ ]
eol \n
symbol [.;\(\)]
/* You should write your own regular expression. */
```

Rules

(example2.l) **這份檔案只有部分正確**

```
%%
{reserved} {
    printf("Line: %d, 1st char: %d, \"%s\" is a \"reserved word\".\n", lineCount, charCount, yytext);
    charCount += yyleng;
}
{space} {
    charCount++;
}
{eol} {
    lineCount++;
    charCount = 1;
}
{symbol} {
/* You shold write your own code */
}

%%
```

Output

```
wangyc@wangyc-ubuntu:~/Desktop/lex/testfile_lab1_2022/
example2$ ./a.out < test.pas
Line: 1, 1st char: 1, "program" is a "reserved word".
Line: 2, 1st char: 1, "begin" is a "reserved word".
Line: 3, 1st char: 3, "writeln" is a "reserved word".
Line: 4, 1st char: 1, "end" is a "reserved word".
```

對 Regular Expression不熟的同學

- 網路上對正規表示式的資源非常豐富

- Online regular expression tester
  - https://regexr.com/

作業繳交注意事項

- **due: (4/10) 23:59**
- 程式Demo環境是Ubuntu 20.04，因此請保證你們的程式碼能夠在Ubuntu上面編譯執行
- 請參考課程網頁中的測試檔案來驗證你的程式
- 助教會自行設計額外的測試檔案，因此請保證你所寫的Regular Expression可以匹配到大部分的case
  - 例如一些複雜的變數名稱、浮點數必須要可以是負數...
- 請準時繳交作業，作業遲交一天打七折
- 請把作業包成一個壓縮包，上傳至網大，檔名命為「學號_hw1」，學號輸錯，此項作業分數-10，沒輸學號分數-50，請同學注意
- 作業繳交之後，在繳交截止隔周會安排時間，到EC5023找助教Demo。