



YACC Tutorial

YACC的工作

- YACC會把input當作 a sequence of tokens
 - 一個以上連續的token可以被表示成一個grammar(語法)
- YACC的目的是檢查語法是否合法
- Lex只是YACC的一個routine
 - 負責回傳token給YACC

YACC如何表示語法

- 假設現在要做一個簡單的計算機的parser
- 設計語法，其中NUMBER為lex抓到的token:
 - $\text{expression} \rightarrow \text{NUMBER}$
 - $\text{expression} \rightarrow \text{expression} + \text{NUMBER}$
 - $\text{expression} \rightarrow \text{expression} - \text{NUMBER}$
- 同一個LHS可以合併在一起，再用| 隔開每個RHS
- 以上語法，在YACC會被表示成
 - $\text{expression} : \text{NUMBER}$
 $\quad \quad \quad | \text{expression} + \text{NUMBER}$
 $\quad \quad \quad | \text{expression} - \text{NUMBER}$

YACC格式

- 總共分成三個部分

- definition

%%

grammars

%%

user code

- 每個部分以%%區隔開來

YACC範例-以計算機為例

- 此範例在課程網頁中的YACC program example
- 功能為輸入一個數學式子，回傳答案

Definition

```
% {  
#include <stdio.h>  
void yyerror(char *); //syntax error時自動呼叫yyerror()  
//利用%union去定義token type及yylval的值  
% }  
  
%union{  
double dval;  
};  
  
%token <dval> NUMBER  
%left '-' '+' //優先序的定義  
%left '*' '/' //比+-還高一級  
%nonassoc UMINUS //最高優先  
%type <dval> expression
```

Rules

lines : lines expression '\n' {printf("answer=%lf\n", \$2);}

| {/*empty string*/}

expression: expression '+' expression {\$\$ = \$1+\$3}

| expression '-' expression {\$\$ = \$1-\$3;}

| expression '*' expression {\$\$ = \$1*\$3;}

| expression '/' expression {

if(\$3 == 0)

yyerror("divide by zero");

else

\$\$ = \$1/\$3;

}

| '-' expression %prec UMINUS {\$\$ = -\$2;}

| '(' expression ')' {\$\$ = \$2;}

| NUMBER {\$\$ = \$1;}

Our code

- 這部分完全是以C語言來撰寫
- //some global variable, for example, symbol table

```
int main(){
```

```
    yyparse(); //YACC透過yyparse()呼叫yylex()，並且開始做  
    parsing
```

```
    return 0;
```

```
}
```

```
void yyerror(char *str){ fprintf(stderr,"%s\n",no,str);}
```

/*當發生syntax error的時候，YACC預設會呼叫yyerror()，而且傳入的參數為"syntax error"，而預設只會把傳入的參數給印出來(也就是印出"syntax error")，但我們可以定義自己的yyerror()。*/

Lex 內容

```
% {  #include <stdio.h>
      #include "y.tab.h"
% }
number [0-9]*[\.]?((([0-9]*[Ee][\+|-]?[0-9]+)|([0-9]*))?)
error   [^\n\t()+-\\*/]+
%%
{number} {yylval.dval = atof(yytext);return(NUMBER); }
"+" {return('+');}
"-" {return('-');}
"*" {return('*');}
"/" {return('/');}
"(" {return('(');}" )" {return(')');}
[ ] {/*do nothing*/}
"\n" {line_no++;return("\n");}
{error} {exit(1);}
%%
```

編譯流程

- 請安裝bison來編譯我們的yacc file，以calc為例
 - `sudo apt-get install bison`
- 透過bison將calc.y編譯成y.tab.c及產生y.tab.h
 - `bison -y -d calc.y`
- 透過flex編譯calc.lex產生lex.yy.c
 - `flex calc.lex`
- 透過gcc產生可執行檔
 - `gcc lex.yy.c y.tab.c -ly -lfl`
- 執行方式
 - `./a.out < testfile`

Yacc作業繳交注意事項

- **due: 6/7, 23:59**
- YACC的設計要比Lex要複雜很多，因此請馬上開始撰寫。
- 程式Demo環境是Ubuntu 14.04，因此請保證你們的程式碼能夠在Ubuntu上面編譯執行
- 作業說明有提供input file，可自行驗證
- 請把作業Email給我，linjt@db.cse.nsysu.edu.tw
- 請準時繳交作業，作業遲交一天打九折
- 助教不會幫忙debug你們的程式