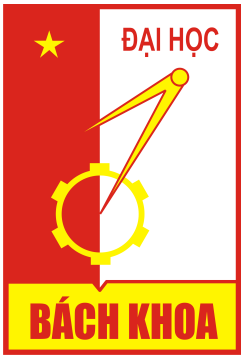
TRƯỜNG ĐAI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Môn học

Lí thuyết ngôn ngữ và phương pháp dịch

Tên đề tài

Tìm hiểu về FLEX

Giảng viên hướng dẫn: **TS. Phạm Đăng Hải**

Sinh viên thực hiện: **Phạm Minh Tâm**
Lâm Xuân Thư

Lớp: **KSTN CNTT K60**

Hà Nội, Ngày 24 tháng 10 năm 2018

Mục lục

1 Giới thiệu flex **3**
1.1 Tổng quan về flex 3
1.2 Sử dụng flex xây dựng bộ phân tích cho ngôn ngữ PL/0 . . 4

2 Cấu trúc file đầu vào flex **6**
2.1 Defintions 6
2.2 Rules 7
2.3 User Code 8
2.4 Comment 8

3 Biểu thức chính quy **9**

4 Ngôn ngữ PL0 **11**

5 Demo chương trình phân tích cú pháp ngôn ngữ PL/0 **13**

Chương 1

Giới thiệu flex

1.1 Tổng quan về flex

flex là một công cụ dùng để sinh các *scanner*. Một *scanner* là một chương trình nhận dạng các mẫu từ vựng (lexical patterns) trong một văn bản. Chương trình *flex* đọc các file đầu vào được cho trước, hoặc đầu vào mặc định nếu không có tên file nào được cung cấp, như là mô tả của một scanner sẽ được sinh. Mô tả này có dạng các cặp biểu thức chính quy và code C, gọi là các luật. *flex* sinh ra một file C ở output, mặc định là *lex.yy.c*, dùng để định nghĩa hàm *yylex*(). File này có thể được compile và link với flex runtime library để tạo ra một file thực thi. Khi ta chạy file thực thi, nó phân tích đầu vào dựa trên sự xuất hiện của các biểu thức chính quy. Bất cứ khi nào nó tìm thấy một biểu thức chính quy thoả mãn, nó sẽ thực thi phần code C tương ứng với biểu thức đó.

Dưới đây là một ví dụ đầu vào của flex. Scanner được sinh ra ở ví dụ này có chức năng đếm số lượng kí tự và số lượng dòng trong input của nó. Output của nó đơn giản chỉ in ra số lượng kí tự và dòng đếm được. Dòng đầu tiên khai báo hai biến toàn cục và chúng có thể truy cập được trong cả hàm yylex() và main() được khai báo ở sau '%%" thứ hai. Có hai luật, một luật nhận dạng kí tự xuống dòng ('\n') và tăng cả số dòng cà số kí tự, và một luật nhận dạng kí tự bất kì khác kí tự xuống dòng (biểu thức '.').

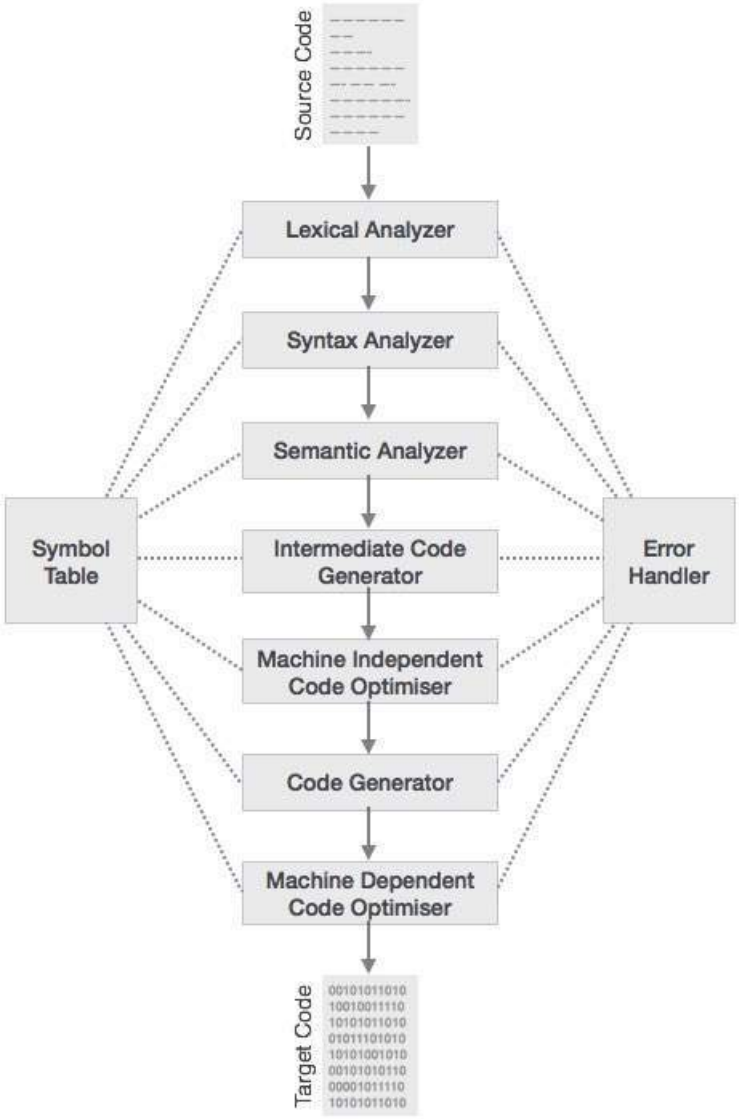
```
            int num_lines = 0, num_chars = 0;
%%
\n        ++num_lines; ++num_chars;
.         ++num_chars;
%%
int main()
{
yylex();
printf( "# of lines = %d, # of chars = %d", num_lines, num_chars );
}
```

1.2 Sử dụng flex xây dựng bộ phân tích cho ngôn ngữ PL/0

Quá trình biên dịch một chương trình là một chuỗi tuần tự các pha. Mỗi pha lấy đầu vào từ pha trước, và đưa ra output cho pha tiếp theo trong compiler.

Trong đó pha đầu tiên (Lexical Analyzer) hoạt động như một text scanner. Pha này scan suorce code như một dòng liên tiếp các kí tự. Nó duyệt từng ký tự của văn bản nguồn, loại bỏ các ký tự không cần thiết như dấu cách, chú thích,.. Sau đó xây dựng từ vựng từ những ký tự đọc được, nhận dạng từ tố và gửi tới pha tiếp.

Trong bài này chúng em sẽ trình bày các đặc trưng của flex và sau đó sử dụng flex để tạo ra một scanner thực hiện chức năng phân tích từ vựng cho ngôn ngữ PL/0.



Hình 1.1: Compiler phases

Chương 2

Cấu trúc file đầu vào flex

File đầu vào flex có ba phần defintions, rules và user code, ngăn cách với nhau bởi một dòng chứa duy nhất '%%'.
defintions
%%
rules
%%
user code

Cụ thể mỗi phần như sau:

2.1 Defintions

Phần definition chứa các khai báo định nghĩa tên, khai báo các điều kiện bắt đầu.

Định nghĩa tên có dạng: *name definition*.
Trong đó, 'name' là một từ bắt đầu bởi một chữ cái hoặc một dấu gạch dưới ('_'), sau đó là không có hoặc nhiều chữ cái, chữ số, gạch dưới hoặc dấu '-' (dấu gạch ngang).
Phần 'definition' được xác định bắt đầu từ kí tự khác dấu cách trắng đầu tiên ở sau tên, và kéo dài đến hết dòng.

Ví dụ:
DIGIT [0-9]
*ID [a-z][a-z0-9]**

Trong phần 'definition' có thể sử dụng đến các định nghĩa đã khai báo trước đó. Ví dụ
*DIGIT+"."DIGIT**
tương đương với
*([0-9])+"."([0-9])**

Một comment không thụt đầu dòng (dòng bắt đầu bởi '/*') được copy nguyên vẹn ra output cho đến khi gặp '*/'.
Đoạn văn bản thụt đầu dòng hoặc văn bản nằm trong cặp '%{' và '%}' cũng được copy nguyên vẹn ra output.
Một khối %top tương tự như một khối '%{' và '%}', ngoại trừ việc code trong %top sẽ được đặt lại lên trên cùng của file sẽ được sinh. Nhiều khối %top có thể được sử dụng, và thứ tự của chúng được giữ nguyên khi thực hiện.

2.2 Rules

Phần rules chứa tập các luật có dạng:
pattern action
Trong đó, pattern bắt buộc không được thụt đầu dòng, và action phải bắt đầu trên cùng dòng đó. Pattern là biểu thức chính quy và action là các câu lệnh C. Nếu pattern được match, thì phần code trong action được thực thi. Chi tiết về pattern sẽ được trình bày kĩ hơn ở phần sau của bài.

Ngoài ra, bất kì đoạn văn bản thụt đầu dòng hoặc nằm trong khối '%{' và '%}' mà xuất hiện trước luật đầu tiên có thể được dùng để khai báo biến địa phương cho hàm scanning. Các trường hợp văn bản thụt đầu dòng hoặc nằm trong khối '%{' và '%}' khác được copy ra output, nhưng nó có thể gây ra các lỗi compile-time.

2.3 User Code

Phần user code được copy nguyên vẹn ra file *lex.yy.c*. Nó được dùng để gọi hoặc được gọi bởi scanner.
Phần này là tuỳ chọn, có thể không có.
Ví dụ:

```
int main( int argc, char **argv ) {
        ++argv, –argc; /* skip over program name */
        if ( argc > 0 )
            yyin = fopen( argv[0], "r" );
        else
            yyin = stdin;
        yylex();
}
```

2.4 Comment

Flex hỗ trợ comment giống như trong ngôn ngữ C, tức lf mọi thứ nằm trong '/*' và '*/' được coi như là comment. Bất cứ khi nào flex gặp một comment, nó sẽ copy nguyên vẹn comment đó ra file được sinh.
Comment có thể xuất hiện ở bất cứ đâu, trừ:

- Comment không được đặt ở đầu dòng trong phần Rules.
- Comment không được đặt ở một dòng '%option' trong phần Definition.

Chương 3

Biểu thức chính quy

Biểu thức chính quy là một chuỗi miêu tả một bộ các chuỗi khác, theo những quy tắc cú pháp nhất định. Một biểu thức chính quy định nghĩa một khuôn mẫu tìm kiếm chuỗi. Flex sử dụng biểu thức chính quy để làm khuôn mẫu định nghĩa xâu đầu vào. Bất cứ khi nào xâu đầu vào thỏa mãn một khuôn mẫu bất kỳ, chương trình sẽ thực hiện các hành động tương ứng đã được khai báo sẵn.

Khi chương trình flex thực thi, nó sẽ quét các xâu đầu vào và tìm kiếm khuân mẫu phù hợp nhất với xâu đó. Nếu tìm được nhiều hơn một khuân mẫu phù hợp với xâu đầu vào, chương trình sẽ chọn thực thi hành động của luật xuất hiện sớm hơn.

Khi một xâu đã được xác định phù hợp với một khuân mẫu nào đó, xâu đó sẽ được lưu trong biến yytext và độ dài của nó được lưu trong biến yyleng.
Không có luật nào được tìm thấy, chương trình sẽ tự động quét các từ tiếp theo.

Một số biểu thức chính quy:

Biểu thức chính quy	Mô tả
‘.’	Khớp với bất cứ ký tự nào
[abc]	Khớp với ký tự a hoặc b hoặc c
[abj-oZ]	Ký tự a hoặc b hoặc 1 ký tự từ j đến o hoặc Z
[Â-Z]	Không phải là một ký tự trong khoảng từ A đến Z
r*	Không hoặc nhiều ký tự r cạnh nhau
r+	Một hoặc nhiều ký tự r cạnh nhau
r?	Không hoặc một ký tự r
r2,5	2 đến 5 ký tự r cạnh nhau
r2,	Nhiều hơn 2 ký tự r
‘x’	Ký tự x
‘r$’	Ký tự r ở cuối dòng

Chương 4

Ngôn ngữ PL0

Là ngôn ngữ lập trình đơn giản, phục vụ trong giảng dạy, có cấu trúc tựa Pascal, chứa đặc trưng của một ngôn ngữ lập trình bậc cao.

Từ vựng của PL/0:

- Chữ cái: a-z,A-Z
- Chữ số 0-9
- Dấu đơn + - * /
- Dấu kép := >= <= <>
- Từ khóa begin, end, if, then, while, do, call, odd, to const, var, procedure, program, else, for
- Số nguyên có tối đa 9 chữ số
- Định danh độ dài tối đa 10 ký tự

Từ tố của PL/0:

- Số nguyên NUMBER
- Định danh IDENT (Nếu từ vựng trùng với khóa, từ vựng sẽ mang ý nghĩa từ tố trùng với tên từ khóa)

- Toán tử

  + (PLUS) - (MINUS)

  * (TIMES) / (SLASH) %(PERCENT)

  = (EQU) <> (NEQ)

  < (LSS) <= (LEQ)

  > (GRT) >= (GEQ)

  ((LPARENT)) (RPARENT)

  [(LBRACK)] (RBRACK)

  . (PERIOD)

  , (COMMA)

  ; (SEMICOLON)

  := (ASSIGN)

- Các trường hợp khác: từ tố NONE

Chương 5

Demo chương trình phân tích cú pháp ngôn ngữ PL/0

Chương trình phân tích cú pháp ngôn ngữ PL/0:

```
%{

#include<string.h>
%}
DIGIT    [0-9]
ID       [a-zA-Z][A-Za-z0-9]*


%%



{DIGIT}+    {
        int value = 0;
        int i= 0;
        int L = strlen(yytext);
```

```
        if(L>9){
                    printf("\nSo qua lon %s",yytext);
        }
        else{
                value = value*10 + yytext[i]-48;
                i=i+1;
                while(i<L){
                        value = value*10 + yytext[i]-48;
                        i = i + 1;
                }


                if(i==L) {
                         printf("\nNUMBER: %s", yytext);
                }
        }

        }

BEGIN|CALL|CONST|DO|ELSE|END|FOR|IF|ODD|PROCEDURE|PROGRAM|THEN|TO|VAR|WHILE     {
        printf("\n%s", yytext );
        }


{ID}        {
                char s[11];
                int i=0;
                if(strlen(yytext) >=10){
                        while(i<10){
                                s[i] = yytext[i];
                                //printf("\nIDENT: %c", yytext[i] );
                                i+=1;
                        }
                        s[i]='\0';
                        printf("\nIDENT: %s", s );
                }else
                        printf("\nIDENT: %s", yytext );
```

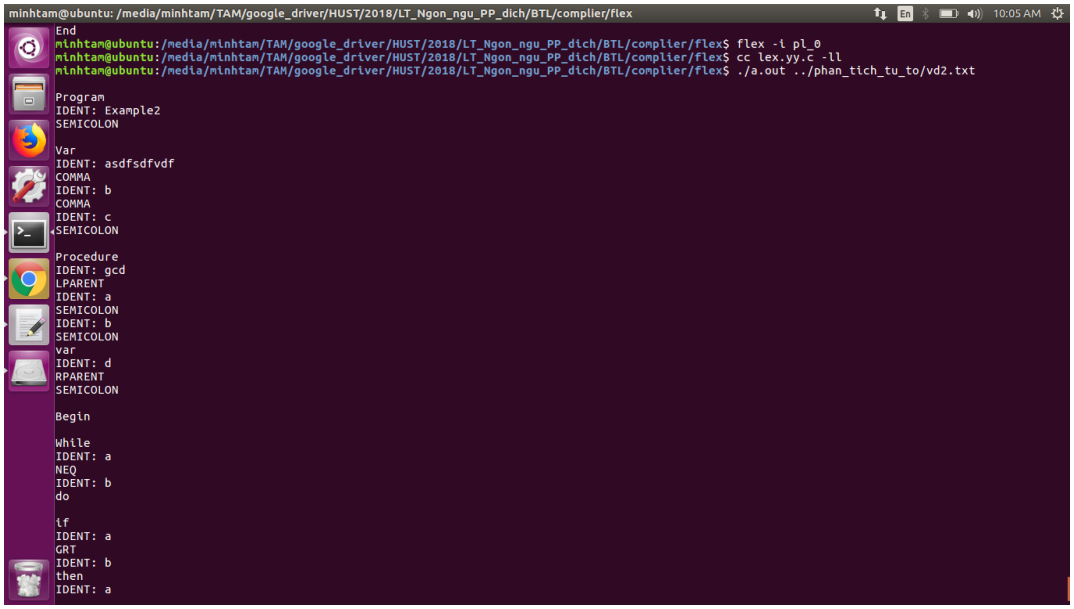
```
        }
"+"       printf( "\nPLUS");
"-"       printf( "\nMINUS");
"*"       printf( "\nTIMES");
"/"       printf( "\nSLASH");
"<>"      printf( "\nNEQ");
"<="      printf( "\nLEQ");
"<"       printf( "\nLSS");
">="      printf( "\nGEQ");
">"       printf( "\nGRT");
":="      printf( "\nASSIGN");
"%"       printf( "\nPERCENT");
"="       printf( "\nEQU");
"("       printf( "\nLPARENT");
")"       printf( "\nRPARENT");
"["       printf( "\nLBRACK");
"]"       printf( "\nRBRACK");
"."       printf( "\nPERIOD");
","       printf( "\nCOMMA");
";"       printf( "\nSEMICOLON");
'.'       ;
%%

int main( int argc, char **argv )
{
++argv, --argc;  /* skip over program name */
if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
else
        yyin = stdin;

yylex();
}
```

```
Program Example2;
Var asdfsdfvdfvfdfdbdfvcvdsxcdsdsdxdscvf,b,c;
Procedure gcd(a; b; var d);
        Begin
          While a <> b do
                if a > b then a := a - b
                else b := b - a;
           d := b;
        End;
Begin
  Call Readln(a);       Call Readln(b);
        Call gcd(a,b,c);
        Call Writeln(c);
End
```

Hình 5.1: Ví dụ file đầu vào



Hình 5.2: Kết quả thu được khi chạy ví dụ

Nếu ta chọn một file đầu vào để phân tích cú pháp như hình 5.1 thì ta sẽ thu được kết quả như hình 5.2 trên đây.

Tài liệu tham khảo

[1] Jonathan Engelsma. *Part 01: Tutorial on lex/yacc. https://www.youtube.com/watch?v=54bo1qaHAfk&t=746s.*

[2] *Lexical Analysis With Flex, for Flex 2.6.2. http://westes.github.io/flex/manual/.*

[3] Phạm Đăng Hải. *Ngôn ngữ và phương pháp dịch.*