Flex

يستخدم البرنامج فليكس لانتاج محلل نصي بلغة السي يكون قادرا على تحليل النص باستخدام التعبيرات النمطية regular expressions وهو تطبيق سهل يمكن استخدامه في نظام التشغيل ويندوز ولينكس.

التركيب:

يتكون برنامج فليكس من ثلاثة أجزاء رئيسية هي:

جزء التعريفات

ويتم في هذا الجزء تعريف المتغيرات التي يتم استخدامها داخل البرنامج ويمكن استخدام التعريفات بلغة السي وتضمين المكتبات في هذا الجزء مثال:

```
%{
#include <stdio.h>
int chars = 0;
int words = 0;
int lines = 0;
%}
```

جزء القوانين

ويتم في هذا كتابة القوانين التي تستخدم في تحليل النص باستخدام التعابير النمطية. ويتكون سطر القوانين من جزئين هما:
التعبير النمطي و الفعل المطلوب عند المطابقة

regex-rule { // c action-code }

ويتم حصر القوانين في هذا الجزء باستخدام الرمز %%
مثال

```
%%

[a-zA-Z]+  { words++; chars += strlen(yytext); }
\n         { chars++; lines++; }
.          { chars++; }

%%
```

جزء دوال المستخدم

يتم في هذا الجزء كتابة دوال المستخدم مثل الدالة main() ويتم فيها كتابة الكود للتحقق من الشروط ويتم حذفها عند استخدام المحلل النحوي لاستخدام هذه القوانين. كما يمكن كتابة بعض الدوال الخاصة مثل yyerrorr() لطباعة الأخطاء وغيرها من الدوال المستخدمة من قبل فليكس.

متغيرات مهمة

المتغير yytext في برنامج فليكس يحتوي على البطاقة التي تم التعرف عليها
المتغير yyleng يحتوي على طول البطاقة التي تم التعرف عليها

مثال لبرنامج يقوم بإنشاء محلل نصي يقوم بحساب عدد الكلمات والحروف والاسطر المدخلة.

```
%{
int chars = 0;
int words = 0;
int lines = 0;
%}

%%

[a-zA-Z]+  { words++; chars += strlen(yytext); }
\n         { chars++; lines++; }
.          { chars++; }

%%

main(int argc, char **argv)
{
  yylex();
  printf("%8d%8d%8d\n", lines, words, chars);
}
```

تنفيذ برنامج flex

يتم عادة كتابة برامج flex باستخدام الامتداد l مثلا يتم تخزين البرنامج السابق تحت اسم wc.l

لتنفيذ هذا البرنامج يتم استخدام الأمر التالي

```
%%

C:\>flex wc.l
```

عند تنفيذ هذا الامر يقوم برنامج فليكس بإنتاج او انشاء برنامج بلغة السي يحمل الاسم lex.yy.c

ملاحظة: يتم انشاء اسم البرنامج تحت هذا الاسم تلقائيا. اذا رغبت في تغيير اسم البرنامج يمكن استخدام الخيار o- مثلا الامر flex -o wordcount.c wc.l سوف يقوم بتسمية ملف السي بـ wordcount.c بدلا من lex.yy.c

بعد الحصول على برنامج السي الذي سيقوم بتحليل النص بناء على القوانين التي تم تعريفها، يتم استخدام مترجم السي للحصول على نسخة تنفيذية من هذا المحلل.

```
C:\>gcc -o wordc.exe lex.yy.c -lfl
```

عند تنفيذ هذا الامر يتكون لدينا برنامج تنفيذي يمكن استخدامه لتحليل النص المدخل

```
C:\>wordc
I am learning flex
It made my life easy
^d
2  9  39
```

التحليل النحوي باستخدام bison

يتم استخدام برنامج bison للحصول على محلل نحوي syntax analysis مع برنامج flex

لشرح كيفية استخدام هذين التطبيقين معا للحصول سوف يتم كتابة برنامج الحاسبة الالية. سوف يتم كتابة محلل المفردات scanner باستخدام فليكس ثم نقوم بكتابة المحلل النحوي باستخدام bison وفي النهاية يتم ربطهما معا للحصول على المترجم النهائي.

أولا برنامج الحاسبة باستخدام فليكس

```
/* recognize tokens for the calculator and print them out */
%{
   enum yytokentype {
     NUMBER = 258,
     ADD = 259,
     SUB = 260,
     MUL = 261,
     DIV = 262,
     ABS = 263,
     EOL = 264
   };
```

```
   int yylval;
%}

%%
"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
"|"     { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n      { return EOL; }
[ \t]  { /* ignore whitespace */ }
.       { printf("Mystery character %c\n", *yytext); }
%%
main(int argc, char **argv)
{
  int tok;

  while(tok = yylex()) {
    printf("%d", tok);
    if(tok == NUMBER) printf(" = %d\n", yylval);
    else printf("\n");
  }
}
```

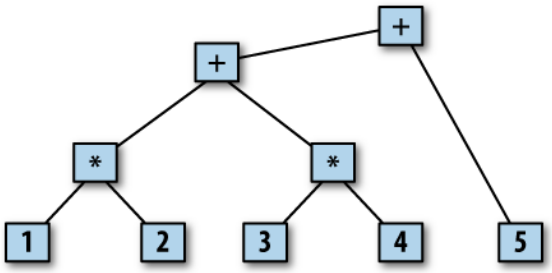
يقوم هذا البرنامج بإنشاء محلل نصي يقوم بقراءة ملف نصي ويقوم بتحليل النص وطباعة رقم العملية عندما تكون البطاقة احد العمليات المعرفة في البرنامج مثلا اذا كان الرمز "+" يقوم البرنامج بطباعة رقم العملية كما تم تعريفها في البرنامج وهو 259 اما اذا كان رقم فانه يطبع الرقم 258 ويقوم بطباعة الرقم المدخل بعد علامة "=".
ملاحظة: تم تعريف هذه البطاقات بهذه الأرقام لتوضيح كيفية تعريف البطاقات في برنامج فليكس.
بعد تخزين هذا البرنامج تحت اسم calc.l يتم تنفيذه كما يلي:

```
C:\>flex calc.l
C:\>gcc -o c.exe lex.yy.c -lfl
C:\>c
```

```
a / 34 + |45
Mystery character a
262
258 = 34
259
263
258 = 45
264
^D
```

بعد الحصول على المحلل النصي الذي يمكنه التعرف على رموز العمليات الحسابية والأرقام الصحيحة يمكننا الان التطرق الى كيفية استخدام bison في عملية التحليل النحوي او ما يسمى باللغة الإنجليزية parsing.

كما تم التطرق في مواضيع سابقة يتم تحليل العلاقة بين البطاقات المدخلة باستخدام شجرة التحليل النحوي او (الاعراب)
مثلا يمكن رسم شجرة التحليل النحوي للتعبير الرياضي 5 + 4 * 3 + 2 * 1 كما يلي:



لان عملية الضرب لها أولوية في حساب التعابير الرياضية على عملية الجمع والطرح يتم حسابها أولا وفق شجرة الاعراب.
يقوم bison باستخدام هذه الشجرة في حساب التعابير الرياضية. لانجاز هذا يتم استخدام القوانين النحوية للحصول على شجرة الاعراب ويتم استخدام تمثيل قوانين الاعراب باستخدام Backus-Naur Form (BNF)
الشكل التالي يعطي مثالا لتمثيل التعبير الرياضي باستخدام هذه الصيغة

```
<exp> ::= <factor>
      | <exp> + <factor>
<factor> ::= NUMBER
      | <factor> * NUMBER
```

يتكون برنامج bison من ثلاثة أجزاء بنفس التركيب السابق لبرنامج flex وهي جزء التعريفات والقوانين ودوال المستخدم.
ويتم تخزين برامج bison باستخدام الامتداد y.
المثال الثالي يوضح المحلل النحوي لبرنامج الحاسبة السابق.

```
%{
```

```
#include <stdio.h>
%}
/* declare tokens */
%token NUMBER
%token ADD SUB MUL DIV ABS
%token EOL

%%
calclist: /* nothing */
 | calclist exp EOL { printf("= %d\n", $2); }
 ;
exp: factor
 | exp ADD factor { $$ = $1 + $3; }
 | exp SUB factor { $$ = $1 - $3; }
 ;
factor: term
 | factor MUL term { $$ = $1 * $3; }
 | factor DIV term { $$ = $1 / $3; }
 ;
term: NUMBER
 | ABS term   { $$ = $2 >= 0? $2 : - $2; }
;
%%
main(int argc, char **argv)
{
  yyparse();
}
yyerror(char *s)
{
  fprintf(stderr, "error: %s\n", s);
}
```

يوضح البرنامج كيف تم تعريف البطاقات tokens في الجزء الأول من البرنامج (ملاحظة هي نفس البطاقات التي تم تعريفها في برنامج calc.l). في الجزء الثاني تم تعريف القوانين النحوية التي تستخدم لبناء شجرة الاعراب. و في الجزء الأخير تم استدعاء الدالة yyparse والتي تقوم بعملية التحليل النحوي.
بعد حفظ هذا الملف تحت اسم calparser.y يتم تنفيذ هذا البرنامج نقوم بالخطوات التالية:

أولا استخدام برنامج bison للحصول على المكتبة التي يتم استخدامها في برنامج flex كما يلي:

```
C:\>bison -d calparser.y
```

بعد تنفيذ هذا الامر بنجاح يقوم bison بإنشاء ملفين هما calparser.tab.c و calparser.tab.h
قبل ترجمة برنامج السي الذي قام بإنشائه المحلل النحوي يجب ان يتم تعديل ملف المحلل النصي calc.l وذلك بإلغاء تعريفات البطاقات وتضمين الملف calparser.tab.h في جزء التعريفات. كذلك يجب الغاء دوال المستخدم وذلك لان برنامج التحليل النحوي هو الذي يقوم بتنفيذ دوال المستخدم ()yyparse بدلا من الدوال الموجودة في برنامج التحليل النصي.
برنامج calc.l بعد التعديل:

```
/* recognize tokens for the calculator and print them out */
%{
 #include “calparser.tab.h”
   int yylval;
%}

%%
"+"    { return ADD; }
"-"    { return SUB; }
"*"    { return MUL; }
"/"    { return DIV; }
"|"    { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n     { return EOL; }
[ \t]  { /* ignore whitespace */ }
.      { printf("Mystery character %c\n", *yytext); }
%%
```

بعد تعديل ملف المحلل النصي يتم استخدام flex للحصول على ملف السي

```
C:\>flex -o callexical.c calc.l
```

بعد تنفيذ هذا الامر نتحصل على البرنامج **callexical.c**

الخطوة الأخيرة هي ترجمة البرنامجين معا باستخدام مترجم السي

```
C:\>gcc -o calculator calparser.tab.c callexical.c -lfl
```

ينتج عن هذا البرنامج ملف تنفيذي اسمه calculator.exe

عند تنفيذ هذا البرنامج وكتابة أي تعبير رياضي يتم طباعة ناتج هذا التعبير كما يلي

```
C:\>calculator
2 + 3 * 4
= 14
2 * 3 + 4
= 10
20 / 4 - 2
= 3
20 - 4 / 2
= 18
```