Flex/Bison

Table des matières

1 Introduction

2 Flex

Suivre le manuel info, qui par ailleurs contient des exemple de paternes (commentaires, adresses IP...).

```
Flex generates as output a C source file, lex.yy.c, which defines a routine yylex().
This file is compiled and linked with the -lfl library to produce an executable. When the executable
Whenever it finds one, it executes the corresponding C code.
```

Voici le programme le plus simple : ex01

Les fonctions `{main()` et `yywrap` sont embarquées dans la librairie. Les implémenter permet de se passer de la librairie (cf ex02).

En gros il y a 4 points d'entrée :

- `yylex()` pour (re-)appeler le scanner
- `yywrap()` pour enchaîner éventuellement sur un autre buffer d'entrée une fois le premier lu.
- `yyin` pour lire le texte en entrée
- `yyout` pour écrire les sorties

Flex ne retourne jamais de code d'erreur, seulement les éventuels tokens trouvés via `return`, puis continue sa lecture au prochain appel.

2.1 Utilisation

cf ex03_flex et ex03_main :

- On extrait un header pour isoler le scanner.
- Le scanner est en principe utilisé pour retrouner des tokens (via `return`) ; le parseur appelant la fonction `yylex()` pour les obtenir.
- Une UNION est mise à disposition pour retrouner des valeures à l'appelant.

2.2 Lire la ligne de commande

cf ex04_flex, pas besoin de redéfinir `YY_INPUT(buf,result,max_size)` :

```
void* buf = 0;

getCommandLine(argc, argv, input);
buf = yy_scan_string(input);

do {
  rc = yylex();
} while (rc);

yy_delete_buffer(buf);
```

2.3 Optimisations

cf ex05_flex et ex05_main :

```
'-f, --full, '%option full''
     specifies "fast scanner".  No table compression is done and 'stdio'
     is bypassed.  The result is large but fast.  This option is
     equivalent to '--Cfr'
```

2.4 Scanner ré-entrant

cf ex06_main : on doit utiliser des fonctions membre pour accéder aux variables.

2.5 Union pour passer les valeures

Il s'agit d'une extension prévue pour bison (`%option bison-bridge`) qui (bison) définit le type `YYSTYPE` donnant lieu à la variable interne `yylval`.

cf ex07_main ;

```
  YYSTYPE bisonUnion;

  do {
    rc = ex_yylex (&bisonUnion);
...
```

Flex l'instancie comme ça :

```
[:digit:]* {
 yylval->num = atoi(yytest)
 return TOKEN1;
}
```

2.6 Erreur de syntaxe

Structure `YYLTYPE` et variable `yylloc` définie et utilisée (je ne sais pas comment) par bison. Sinon, à part un lexeme inconnu, on ne peut pas vraiment renvoyer d'erreur.

3 Bison

Suivre le manuel info (du paquet bison-doc). Attention, il semble y avoir des regressions entre les versions 2.5 et 3.0.

Bison retourne 0 lorsque l'ensemble du texte en entrée a été traité, ou 1 s'il est tombé en erreur (erreur de syntaxe, YYABORT...).

3.1 Exemple sans flex

cf ex10_bison ;

```
int yyparse (void)
{
...
      YYDPRINTF ((stderr, "Reading a token: "));
      yychar = yylex ();
```

exemple :

```
$ ./ex10
     4 9 +
     => 13
```

3.1.1 Exemple sans flex avec gestion des erreurs

cf ex11_bison ;

```
               fprintf (stderr, "%d.%d-%d.%d: division by zero",
                        @3.first_line, @3.first_column,
                        @3.last_line, @3.last_column);
```

```
   This code shows how to reach locations inside of semantic actions, by
using the pseudo-variables '@N' for rule components, and the
pseudo-variable '@$' for groupings.
```

exemple :

```
$ ./ex11
100/0000
1.5-1.9: division by zero1
```

3.1.2 Exemple sans flex avec une union

cf ex12_bison ;

```
     %define api.value.type union /* Generate YYSTYPE from these types:  */
     %token <double>  NUM         /* Simple double precision number.  */
     %token <symrec*> VAR FNCT    /* Symbol table pointer: variable and function.  */
     %type  <double>  exp
```

```
   The special 'union' value assigned to the '%define' variable
```

```
'api.value.type' specifies that the symbols are defined with their data
types.  Bison will generate an appropriate definition of 'YYSTYPE' to
store these values.

typedef union YYSTYPE YYSTYPE;
union YYSTYPE
{
  double NUM;
  double exp;
  symrec* VAR;
  symrec* FNCT;
};
```

exemple :

```
$ ./ex12
     pi = 3.141592653589
     => 3.1415926536
     sin(pi)
     => 0.0000000000
     alpha = beta1 = 2.3
     => 2.3000000000
     alpha
     => 2.3000000000
     ln(alpha)
     => 0.8329091229
     exp(ln(beta1))
     => 2.3000000000
```

3.2 Push parser

cf ex13_bison. On passe les tokens un par un à l'aide de la fonction `yypush_parseb`.

cf ex14_bison et ex14_main C'est pratique pour impléter un parseur SAX XML, qui au passage valide la grammaire.

```
xmllint --schema ex14.xsd ex14.xml
```

3.3 Exemple avec flex

cf ex20_bison et ex20_flex.

3.3.1 Parseur ré-entrant

cf ex21_bison et ex20_flex. L'union de bison est passée en paramètre ; seul le parseur est réentrant (pas le scanner).

```
// ex21_flex.l
%option   bison-bridge

// ex21_bison.y
%define   api.pure full

rq:  extern int yylex (YYSTYPE * yylval_param );
```

Note : yylval et YYSTYPE étant à présent locales, ne sont plus renommées (préfix `yy`).

3.3.2 Parseur et scanner ré-entrant

cf ex22_bison et ex22_flex.

```
// ex22_flex.l
%option   bison-bridge
%option   reentrant

// ex22_bison.y
%define   api.pure full
%lex-param {yyscan_t yyscanner}

rq: #define YY_DECL int yylex (YYSTYPE * yylval_param, yyscan_t yyscanner)
```

Note : C'est à nous d'initialiser le scanner avant de la passer au parseur. Ici on ne peut plus éviter de générer un header pour le scanner.

3.3.3 Gestion des erreurs

cf ex23_bison et ex23_flex.

```
// ex23_flex.l
%option   bison-bridge
%option   bison-locations

// ex24_bison.y
%define   api.pure full
%locations
```

A priori le lexer ne renseigne pas ces champs mais se serait à nous de le faire à la main dans les règles du parseur...