FLEX

- Yacc entre 1975 et 1978
- Lex en 1975 (Mike Lesk et summer intern Eric Schmidt AT&T)
- 1987 Flex (*Fast Lexical Analyzer Generator.)*
- *Les expressions régulières de Flex sont basées sur POSIX (***Portable Operating System Interface) le X UniX**

Structure d'un fichier Flex

- Un programme Flex est constitué de trois sections séparées par des les lignes contenat %%:
  - Une section de déclarations en C entre %{ et %}
  - Une section constituée par une liste de règles chaque règle est formée d'un patron (ou modèle), le patron doit commencer au début de la ligne (chaque ligne qui commence par une espace est considérée par Flex comme une ligne de code C)

```
%{
int caracteres= 0;
int mots= 0;
int lignes= 0;
%}
%%
[a-zA-Z]+ { mots++; chars += strlen(yytext); }
\n {caracteres ++; lignes++; }
. {caracteres ++; }
%%
main(int argc, char **argv)
{
yylex(); // Appel de l'analyseur lexical
printf("%8d%8d%8d\n", lignes, mots, caracteres);
}
```

Un premier programme Flex

- Nous allons créer un programme Flex qui lit des chaînes à partir de l'entrée standard et réalise les actions suivantes :
  - Affiche le message « Commande STOP active », si la chaîne en entrée est « STOP »
  - Affiche le message « commande GO active si la chaîne en entrée est GO

Etape 1 : écrite du fichier de définition des règles « exemple1.l »

```
%%
STOP            printf("Commande STOP active ") ;
GO   printf("Commande GO active ") ;
```

- Etape 2 : création du fichier lex.yy.c
  - Flex exemple1.l
- Etape 3 : Compilation gcc lex.yy.c

- Flex est gourmand, Flex utilise en premier l'expression qui capture le maximum de caractères, dans le cas ou deux expressions capturent la même chaine alors la première qui apparait dans la liste des règles est utilisée, exemple:
- "+" { return ADD; }
- "=" { return ASSIGN; }
- "+=" { return ASSIGNADD; }

Expressions régulières

- **Les métacaractères :**
- Un métacaractère est un caractère spécial qui a une signification particulière dans une expression régulière.
- Les métacaractères suivants sont supportés par Flex : ^ . [ ] $ ( ) * + ? | { } \
  - [ ] : définit une classe de caractères.

Les symboles supportés dans une expression régulière

c	le caractère c
.	Un caractère quelconque à l'exception du caractère "nouvelle ligne".
^	Indique le début d'une ligne.
$	Indique la fin d'une ligne
[abg]	un caractère appartenant à l'ensemble {a,b,g}.
[f-m]	un caractère compris entre f et m
[^a-z\n]	un caractère n'appartenant pas à la classe [a-z] et différent du caractère nouvelle ligne
exp*	0 ou plusieurs occurrences de l'expression règulière exp
Exp?	0 ou 1 occurrence de l'expression rationnelle exp
exp+	1 ou plusieurs occurrences de l'expression rationnelle exp
exp{3,8}	Une concaténation de trois à huit occurrences de l'expression exp
exp{3,}	au minimum une concaténation de trois occurrences de l'expression exp
'exp{4}	exactement quatre occurrences de l'expression exp
\x	Une séquence d'échappement ANSI-C si x est un 'a', 'b', 'f', 'n', 'r', 't', ou 'v', sinon le caracètre \ est utilisé pour échapper un caractère spécial, par exemple pour avoir le caractère ^ dans une expression régulière il faut le faire précéder du caractère \^).
exp1/exp2	exp1 mais seulement si elle est suivie de **exp2**.

Options Flex

- Options en ligne de commande
  - –oNomFichier : l'analyse lexicale sera générée dans un fichier nommé NomFichier
  - -d : exécution en mode débogage.
  - –i : génère un analyseur lexical insensible à la casse.
  - -+ : l'analyseur est généré en c++.

- Options dans le fichier de description de l'analyseur.
  - caseless ou case-insensitive : équivalente à l'option –i
  - yylineno : Flex maintient le numéro de la ligne en cours dans la variable yylineno.
  - noyywrap: la fonction noyywrap() n'est pas appelée dans l'analyseur lexical généré.
  - nodefault: Flex ne doit pas généré la règle par défaut, mais elle doit être traitée dans l'analyseur Flex.

Variables définies par Flex

- char * yytext: contient l'élément lexical (lexème ou token) reconnu par Flex .
- int yyleng : nombre de caractères du lexème courant.
- File* yyin : fichier en entrée.
- File * yyout : fichier en sortie.

```
main(argc, argv)
int argc;
char **argv;
{
if(argc > 1) {
if(!(yyin = fopen(argv[1], "r"))) {
perror(argv[1]);
return (1);
}
}
yylex();
printf("%8d%8d%8d\n", lines, words, chars);
}
```

Fonctions Flex

yyrestart()

- Permet de changer de fichier d'entrée au milieu d'un scan, il ne réinitialise pas l'état.

```
%option noyywrap
%{
/* Exemple5: Lecture de plusieurs
fichiers*/
int chars = 0;
int words = 0;
int lines = 0;
int totchars = 0;
int totwords = 0;
int totlines = 0;
%}
%%
[a-zA-Z]+ { words++; chars +=
strlen(yytext); }
\n { chars++; lines++; }
. { chars++; }
%%
main(argc, argv)
int argc;
char **argv;
{
int i;
if(argc < 2) { /* just read stdin */
yylex();
printf("%8d%8d%8d\n", lines, words,
chars);
return 0;
}
for(i = 1; i < argc; i++) {
FILE *f = fopen(argv[i], "r");
if(!f) {
perror(argv[i]);
return (1);
}
yyrestart(f);
yylex();
fclose(f);
printf("%8d%8d%8d %s\n", lines, words,
chars, argv[i]);
totchars += chars; chars = 0;
totwords += words; words = 0;
totlines += lines; lines = 0;
}
if(argc > 1)
printf("%8d%8d%8d total\n", totlines,
totwords, totchars);
return 0;
}
```

sélection

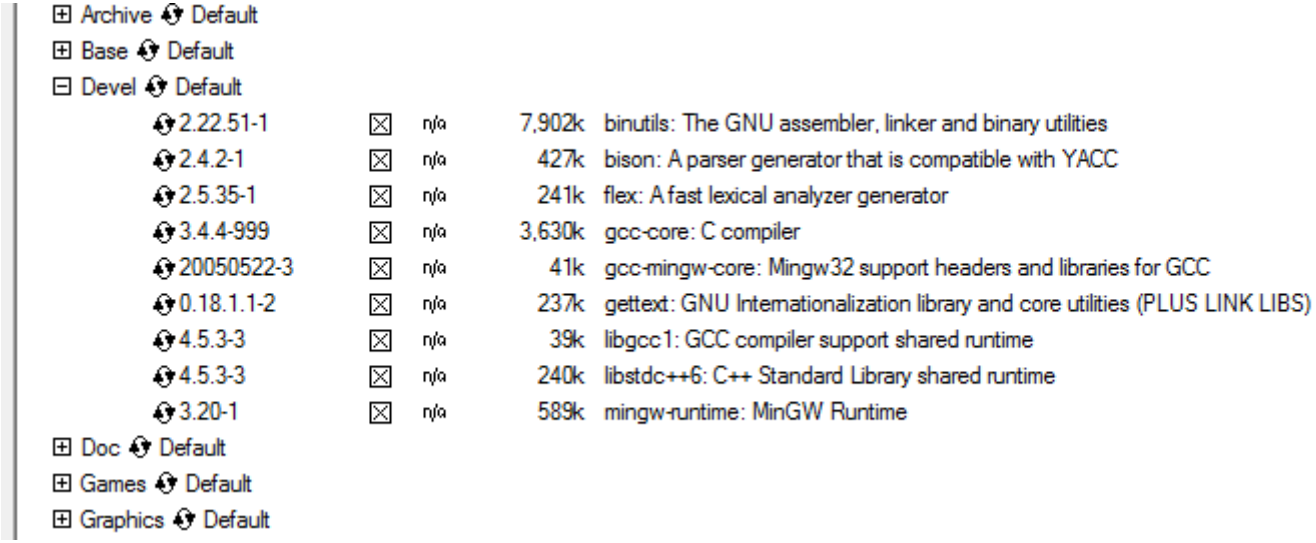
Archive Default
Base Default
Devel Default
2.22.51-1 n/a 7,902k binutils: The GNU assembler, linker and binary utilities
2.4.2-1 n/a 427k bison: A parser generator that is compatible with YACC
2.5.35-1 n/a 241k flex: A fast lexical analyzer generator
3.4.4-999 n/a 3,630k gcc-core: C compiler
20050522-3 n/a 41k gcc-mingw-core: Mingw32 support headers and libraries for GCC
0.18.1.1-2 n/a 237k gettext: GNU Internationalization library and core utilities (PLUS LINK LIBS)
4.5.3-3 n/a 39k libgcc1: GCC compiler support shared runtime
4.5.3-3 n/a 240k libstdc++6: C++ Standard Library shared runtime
3.20-1 n/a 589k mingw-runtime: MinGW Runtime
Doc Default
Games Default
Graphics Default

Table des symboles

- L'analyseur lexical les caractères en entrée et produit en résultat une suite d'unités lexicales qui seront utilisées par l'analyseur syntaxique.
- Une règle est formée par un modèle (une expression régulière) et une action.
- Un lexème est une chaine du fichier source qui concorde avec le modèle d'une règle.
- X=5.5 X est un lexème de l'unité lexicale « Identificateur »