Correction TD de Compilation n°1

Licence d'informatique

———

Introduction à l'analyse lexicale

Le but de ce TD est d'écrire des expressions régulières caractérisant certains lexèmes, puis d'écrire le programme `flex` correspondant.

▸ Exercice 1. *Expressions régulières*

Écrire des expressions régulières pour reconnaître :

1. *les identificateurs du C (commencent par une lettre ou un _, puis une suite de chiffres, lettres ou _, le tout avec au moins une lettre) ;*
2. *les chaînes de caractères du Pascal (`' ... '` avec `''` pour faire une apostrophe) ;*
3. *les chaînes de caractères du C ;*
4. *les commentaires du C++ (`// ...`) ;*
5. *les entiers (décimal, hexadécimal `0x...` ou `0X...`, et octal `0...`) ;*
6. *les réels avec exposant.*

✂ -

1. `'` (`''`+`NON-APOSTROPHE`)* `'`
2. `"` ((`\` `CARACTERE`)+`NON-GUILLEMET`)* `"`
3. `//` (`NON-RETOUR-CHARIOT`)*
4. — décimal : (ε+`-`+`+`)`0` + (ε+`-`+`+`)(`[1-9]` `[0-9]`*)
   — hexadécimal : (ε+`-`+`+`)`0`(`x`+`X`) `[0-9A-Fa-f]` `[0-9A-Fa-f]`*
   — octal : (ε+`-`+`+`)`0` `[0-7]` `[0-7]`*
5. (ε+`-`+`+`)(`[0-9]`$^+$(ε+`.`)+`[0-9]`*`.``[0-9]`$^+$)`e` ($\varepsilon$+`-`+`+`) `[0-9]`$^+$

- ✂

▸ **Exercice 2.** ***Analyse d'un programme*** `flex`

Que fait le programme `flex` *suivant ?*

```
%%

"/*" printf("<SE>");
"*/" printf("<ASE>");
"\\n" printf("<ASN>");
"("|"["|"{" printf("<PO>");
")"|"]"|"}" printf("<PF>");
"?"+ printf("<?;%zd>",yyleng);
. ;

%%
int yywrap() {
  return 1;
}

int main() {
  yylex();
  return 0;
}
```

✂ ----------

Ce programme :

- remplace les séquences `/*` et `*/` par `<SE>` et `<ASE>` ;
- remplace les séquences constituées des caractères `\` et `n` par `<ASN>` ;
- remplace toute occurrence d'un des caractères `(`, `[` ou `{` par `<PO>` ;
- remplace toute occurrence d'un des caractères `)`, `]` ou `}` par `<PF>` ;
- remplace toute séquence de points d'interrogation par une séquence `<?;N>`, où `N` représente le nombre de points d'interrogation de la séquence lue ;
- supprime tous les autres caractères, car ils sont lus sans que rien ne soit produit en sortie.

---------- ✂

▸ **Exercice 3.** ***Codage circulaire***

Ecrire un programme `flex` *qui remplace dans un texte chaque lettre par sa suivante en conservant la casse (*`a` *par* `b`*,* `B` *par* `C`*,* `z` *par* `a`*). Exemple pour "Je me sens VRAIMENT bien !" :*

```
Kf nf tfot WSBJNFOU cjfo!
```

```
%%
[a-z] printf("%c",'a'+(yytext[0]+1-'a')%26);
[A-Z] printf("%c",'A'+(yytext[0]+1-'A')%26);
.|\n ECHO;
%%
int main() {
yylex();
return 0;
}
```

▸ **Exercice 4.** ***Écriture d'un programme*** `flex`

1. *Transposer en* `flex` *les expressions régulières du premier exercice.*

   Penser à rappeler que les espaces ont une signification précise en `flex` et qu'ils ne peuvent pas être utilisés pour aérer la présentation du code. Présenter la possibilité d'utiliser des variables comme `LETTRE`, et dire à quoi correspondent les variables prédéfinies `yytext` et `yyleng`.

```
 // rappel 1: toujours au moins un espace avant du C
 //  (par exemple, des commentaires)
 // rappel 2: bien sauver un programme flex avec des
 // retours chariot style Unix
 // rappel 3: yytext est un pointeur sur le buffer d'entrée:
 //  il est susceptible de changer, donc il faut recopier son contenu si besoin
LETTRE [a-zA-Z]
CHIFFRE [0-9]

%%
( {LETTRE}({LETTRE}|\_|{CHIFFRE})* ) | ( \_(({LETTRE}|\_|{CHIFFRE})*L({LETTRE}|\_|{CHI
"'"([^']|"''")*"'" {printf("chaine pascal=%s\n",yytext);}
\"([^"]|\\.)*\" {printf("chaine C=%s\n",yytext);}
"//".* {printf("commentaire C=%s\n",yytext);}
0|([1-9]{CHIFFRE}*) {printf("entier decimal=%s\n",yytext);}
0x([a-fA-F]|{CHIFFRE})+ {printf("entier hexadecimal=%s\n",yytext);}
0[0-7]+ {printf("entier octal=%s\n",yytext);}
0|([1-9]{CHIFFRE}*)"."{CHIFFRE}* {printf("flottant=%s\n",yytext);}
"."{CHIFFRE}+ {printf("flottant=%s\n",yytext);}
```

```
{CHIFFRE}"."{CHIFFRE}*e[-+]?{CHIFFRE}+ {printf("flottant=%s\n",yytext);}
. ;

%%

int main() {
  yylex();
  return 0;
}
```

---------- ✂

✂ ----------

```
 // %x X : X est une start condition
/*
flex provides a mechanism for conditionally activating rules. Any rule whose pattern i

<STRING>[^"]*        { /* eat up the string body ... */
            ...
            }
will be active only when the scanner is in the "STRING" start condition, and

<INITIAL,STRING,QUOTE>\.        { /* handle an escape ... */
            ...
            }
will be active only when the current start condition is either "INITIAL", "STRING", or

Start conditions are declared in the definitions (first) section of the input using un
*/
%x CHAINEPASCAL
%x CHAINEC
%x COMMENTAIRE

%%
<CHAINEPASCAL>"'" {printf("'\n"); BEGIN INITIAL;}
<CHAINEPASCAL>"''"|. ECHO;
<CHAINEC>\" {printf("\"\n"); BEGIN INITIAL;}
<CHAINEC>\\?. ECHO;
<COMMENTAIRE>. ECHO;
<COMMENTAIRE>"*/" {printf("*/\n"); BEGIN INITIAL;}
' {printf("chaine Pascal='"); BEGIN CHAINEPASCAL;}
\" {printf("chaine C=\""); BEGIN CHAINEC;}
"/*" {printf("commentaire C=/*"); BEGIN COMMENTAIRE;}
```

```
. ;

%%

int main() {
  yylex();
  return 0;
}
```

▸ **Exercice 5.** *Ecrire une expression rationnelle lex qui défini les commentaires du C (`/* ... */`). Rappel : la norme précise qu'on ne peut inclure un commentaire C dans un autre. Améliorer la lisibilité de la reconnaissance des commentaires en utilisant des `start conditions`.*

```
%x COMMENT

/* Si plusieurs règles correspondent, celle qui consomme le plus de caractères est prise.
   Si plusieurs consomment le même nombre de caractères, la première est prise.
   Attention: flex essaie de trouver l'entrée la plus grande possible.
   Aussi, parfois il faut attendre la fin du flux: pour être sûr, fermer le flux (ctrl-d) !!
   Pour les commentaires utilisant les sart condition: ça fonctionne car flex essaie de
   trouver des correspondances les plus longues possibles.
   Ainsi, en cas de ba, on prefere la regle "ba" à ".", et on sort immediatement du commenta
 */
%%
"/""*"((((*"[^"/"])|[^"*"])*)|([^"*"]*("*"+)))"*""/" { printf("Commentaire C\n");  }
"/""*" { BEGIN COMMENT; }
<COMMENT> .
<COMMENT> "*""/" { printf("Commentaire C\n"); BEGIN INITIAL; }
%%
int main(void) {
  while (yylex()) {
    fflush(stdout);
  }
  return 0;
}
```

▸ **Exercice 6.** ***Repérage des noms de fonctions en C***

Ecrire un programme `flex` *qui prend un programme C et qui affiche tous les noms de fonctions utilisés dans ce programme. On fera attention à ne pas repérer abusivement les noms qui apparaissent dans les chaînes et les commentaires. Si l'on prend en entrée le programme suivant :*

```
/* la fonction plus(int,int) renvoie
   la somme de ses paramètres */
int plus
   (int a,int b) {
return a+b;
}
int main (void) {
printf ("plus(4,7)=%d\n",plus(4,7));
getchar(); // getchar() attend un retour chariot
return 0;
}
```

on devra obtenir la liste suivante : `plus main printf plus getchar`

✂ ----------

```
IDENT [a-zA-Z_][a-zA-Z_0-9]*
%x COMMENTAIRE
%x CHAINE
%%
"/*" BEGIN COMMENTAIRE;
<COMMENTAIRE>.|\n ;
<COMMENTAIRE>"*/" BEGIN INITIAL;
\" BEGIN CHAINE;
<CHAINE>\" BEGIN INITIAL;
<CHAINE>\\?. ;
"if"/[\ \r\n\t]*\( ; /* Pareil pour les autres mots clefs (for, while, sizeof,...)
{IDENT}/[\ \r\n\t]*\( ECHO;
.|\n ;
"//".*\n? ;
%%
int main() {
yylex();
return 0;
}
```

---------- ✂

Extrait de la page man de la commande `flex`

Les motifs en entrée sont écrits en utilisant un ensemble étendu d'expressions rationnelles. Ce sont :

`x`	correspond au caractère 'x'
`.`	n'importe quel caractère (octet) sauf le retour à la ligne
`[xyz]`	une « classe de caractères » ; dans ce cas, le motif convient pour un 'x', un 'y', ou un 'z'
`[abj-oZ]`	une « classe de caractères » contenant un intervalle ; convient pour un 'a', un 'b', n'importe quelle lettre allant de 'j' à 'o', ou un 'Z'
`[^A-Z]`	une « classe de caractères niée », c.-à-d. tout caractère sauf ceux dans la classe. Dans cet exemple, tout caractère SAUF une lettre majuscule.
`[^A-Z\n]`	tout caractère SAUF une lettre majuscule ou un retour à la ligne
`r*`	zéro ou plusieurs r, où r est une expression rationnelle quelconque
`r+`	un ou plusieurs r
`r?`	zéro ou un r (c.-à-d. un r optionnel)
`r{2,5}`	entre deux et cinq r
`r{2,}`	deux r ou plus
`r{4}`	exactement 4 r
`{nom}`	le développement de la définition de « nom » (voir au dessus)
`"[xyz]\"foo"`	la chaîne de caractères littérale : [xyz]"foo
`\X`	si X est 'a', 'b', 'f', 'n', 'r', 't' ou 'v', alors la représentation C ANSI de \x. Sinon, un 'X' littéral (utilisé pour protéger des opérateurs comme '*')
`\0`	un caractère NUL (de code ASCII 0)
`\123`	le caractère de valeur octale 123
`\x2a`	le caractère de valeur hexadécimale 2a
`(r)`	reconnaît un r ; les parenthèses sont utilisées pour surcharger la priorité (voir plus bas)
`rs`	l'expression rationnelle r suivie de l'expression

rationnelle s ; appelé « concaténation »

r|s un r ou un s

r/s un r mais seulement s'il est suivi par un s.
Le texte reconnu par s est inclus quand on détermine si cette règle est la « correspondance la plus longue », mais est ensuite renvoyé sur l'entrée avant que l'action ne soit exécutée. Ainsi, l'action ne voit que le texte auquel correspond r. Ce type de motif est appelé « contexte de queue ». (trailing context) (Il y a certaines combinaisons de r/s que flex ne peut détecter correctement ; voyez les notes consacrées aux contextes de queue dangereux dans la section Défectuosités/Bogues.)

^r un r, mais uniquement au début d'une ligne (c.-à-d. au début de l'analyse, ou juste après qu'un saut de ligne ait été détecté).

r$ un r, mais seulement à la fin d'une ligne (c.-à-d. juste avant un saut de ligne). Équivalent à « r/\n ».

Notez que la notion qu'a flex d'un « saut de ligne » est exactement celle dont le compilateur C utilisé pour compiler flex interprète '\n' ; en particulier, sur certains systèmes DOS, vous devez soit filtrer vous-même les \r de l'entrée vous-même, soit utiliser explicitement r/\r\n pour « r$ ».

<s>r un r, mais seulement dans la condition de démarrage s (voyez en dessous pour une discussion sur les conditions de démarrage)

<s1,s2,s3>r
idem, mais dans une des conditions de démarrage s1, s2 ou s3

<*>r un r dans n'importe quelle condition de démarrage, même une exclusive.

<<EOF>> un end-of-file (fin de fichier)

<s1,s2><<EOF>>

un end-of-file quand on se trouve dans la condition de démarrage s1 ou s2

Notez qu'à l'intérieur d'une classe de caractères, tous les opérateurs d'expressions rationnelles perdent leur signification spéciale sauf l'échappement ('\') et les opérateurs de classes de caractères, « - », «] » et, au début de la classe, « ^ ».

Les expressions rationnelles listées plus haut sont groupées en fonction de leur priorité, allant de la plus haute au sommet à la plus basse en bas.

En plus des caractères et des intervalles de caractères, les classes de caractères peuvent également contenir des expressions de classes de caractères. Ce sont des expressions enfermées dans des délimiteurs [: et :] (qui doivent elles-mêmes apparaître entre le '[' et le ']' de la classe de caractères ; d'autres éléments peuvent également être présents dans la classe de caractères). Les expressions valides sont :

```
[:alnum:] [:alpha:] [:blank:]
[:cntrl:] [:digit:] [:graph:]
[:lower:] [:print:] [:punct:]
[:space:] [:upper:] [:xdigit:]
```

Ces expressions désignent toutes un groupe de caractères équivalent à la fonction C standard correspondante isXXX.