COMPILATION RÉVISION TP EMSI - 4ÈME IIR 2017/2018

Prof. M. D. RAHMANI

Rappel: traitement de caractères

Remarques:

- Je rappelle que l'examen porte sur tout le programme du semestre et naturellement le code en langages C, Flex et Bison.
- Attention à copier/coller des programmes joints, certains apostrophes ou d'autres caractères ne sortent pas de la même manière !
- Vous trouverez ci-joints des exemples de codes que nous avons faits dans les séances de TP et des exercices de l'examen de l'année dernière.
- Les exemples donnés ici ne couvrent pas tout le programme mais je les ai mis suite à la demande de certains étudiants.

Bon courage.

Rappel: traitement de caractères

Les macros d'analyse de caractères:

Ces macros sont incluses dans la bibliothèque `<ctype.h>`.

Elle acceptent comme argument un char ou un int et retournent un entier différent de 0 si l'argument est compris dans les limites indiquées ci-dessous:

macros	**condition**
`isalpha(c)`	A-Z, a-z
`isupper(c)`	A-Z
`islower(c)`	a-z
`isdigit(c)`	0-9
`isxdigit(c)`	0-9, A-F, a-f
`isspace(c)`	blanc
`isalnum(c)`	0-9, A-Z, a-z

Analyse lexicale : exo1.cpp

- Exercice 1: Ecrire un programme qui vérifie que les parenthèses d'une chaîne sont bien équilibrées.

Analyse lexicale : exo1.cpp


```
#include<stdio.h>
#include<conio.h>  // pour getch()
#include<string.h>
int main()
{
    int i,          // indice pour parcourir la chaine
        j=0;        //entier qui s'incrémente s'il rencontre '('
                // et se decremente si ')'
    char c,         // le caractère courant
      ch[30];   //la chaine lue au clavier
      printf("Donnez votre chaine a analyser:\t");
     gets(ch);
     i=0;
while (i<strlen(ch)) //boucle qui teste la fin de la chaine
    {
        c=ch[i];
        if (c=='(') ++j;
        else if (c==')') --j;
        i++;
    }
    if(!j) printf("les parentheses sont bien equilibrees");
    else if(j<0) printf("il manque %d  parenthese ouvrantes",-j);
          else printf("il manque %d parenthese fermantes",j);
    getch();

return 0;
}
```

Analyse lexicale

- Exercice 2:

- Ecrire un programme qui implante un automate qui indique si un nombre est constitué d'un nombre paire ou impaire de '**a**'.

Analyse lexicale : exo2.cpp


```
#include <stdio.h>
#include <string.h>  // pour strlen
#include <stdlib.h> // pour pause
#include <conio.h>  // pour getch()
int main() {
  char chaine[20];
  int i=0, etat=0, car, longueur;
  printf("donnez une chaine constituee d'une suite de 'a'\n");
  gets(chaine);
  longueur = strlen(chaine);
  while (i<=longueur) {
    switch(etat) {
      case 0: {car=chaine[i];
              if (i==longueur) {printf("\t Nombre paire de 'a'");
                                getch(); exit(0);}
              else { if (car=='a') {etat=1; i++;}
                     else {printf("!!! Erreur la chaine n'est pas une suite de 'a'");
                           getch(); exit(1);}}
           break;
      }
      case 1: {car= chaine[i];
              if (i==longueur) {printf("\t\t Nombre impaire de 'a'");
                           getch(); exit(0);}
              else { if (car=='a') {etat=0; i++;}
                     else {printf("!!! Erreur pas une suite de 'a'");
                           getch(); exit(1);}}
           break;
      }
    }
   }

    //system("pause");
    //getch();
return 0;
}
```

Analyse lexicale : exo3.cpp

- Exercice 3: On veut reconnaitre un horaire sous la forme : **12 : 15** pour *midi et 15 minutes*.
  1. Proposer un automate à états finis qui reconnait ces formes.
  2. Ecrire un programme qui implante cet automate.

Analyse lexicale : exo3.cpp


```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
  char chaine[20];
  int i=0, etat=0;
  printf("donnez un horaire a verfifer: \n\t");
  gets(chaine);
  while(1) {
       switch(etat) {
            case 0: {
                        if(chaine[i]=='0' || chaine[i]=='1') {etat=1; i++;}
                        else if (chaine[i]=='2') { etat=2; i++;}
                             else { printf("\n\t erreur !!!\n"); exit(1);}
                        break;
                       }
             case 1: {
                        if(chaine[i]>='0' && chaine[i]<='9') { etat=3; i++; }
                        else { printf("\n\t erreur !!!\n"); exit(1); }
                        break;
                      }
             case 2: {
                        if(chaine[i]>='0' && chaine[i]<='3') { etat=3; i++;}
                        else { printf("\n\t erreur !!!\n"); exit(1);}
                       break;
                        }
            case 3: { if (chaine[i]==':') { etat=4; i++; }
                          else { printf("\n\t erreur !!!\n"); exit(1); }
                        break;
                       }
            case 4: { if(chaine[i]>='0' && chaine[i]<='5') {etat=5; i++;}
                          else { printf("\n\t erreur !!!\n");     exit(1); }
                       break;
                       }
            case 5: { if(chaine[i]>='0' && chaine[i]<='9') {etat=6}
                          else { printf("\n\t erreur !!!\n"); exit(1);
                       }
                        break;
                       }
             case 6: { printf("\n\t horaire valide\n"); exit(0);
                        /* A  la fin obligatoirement exit(0) pas un break */}
       }
    }
  return 0;
}
```

Analyse lexicale : exo4.cpp

- Exercice 4: Ecrire un programme en langage C qui vérifie si une chaine de caractères entrée au clavier est un commentaire en C et affiche le corps du commentaire.

Analyse lexicale : exo4.cpp

Solution : **exo4.cpp**

```
#include <stdio.h>
#include <string.h>  // pour strlen
int main() {
  char chaine[20], com[20];
  int j =0;
  int i=0, etat=0, car, longueur;
  printf("donnez un commentaire en C \n");
  gets(chaine);
  longueur = strlen(chaine);
  while(1) {
     switch(etat) {
        case 0: { car=chaine[i];
                  if (i==longueur) {printf("\t Commentaire non
                                          valide !!!"); getch(); return 1;}
                  else if (car=='/') { etat=1; i++;}
                    else { printf("!!! Erreur \n\t "); return 1;}
                   break;
                 }
case 1: { car= chaine[i];
            if (i==longueur) {printf("\t Commentaire non valide !!!"); getch(); return 1;}
               else if (car=='*') {etat=2; i++;}
                     else { printf("!!! Erreur "); return 1;}
          break;
        }
case 2: { car= chaine[i];
            if (i==longueur) {printf("\t Commentaire non valide !!!"); getch(); return 1;}
                 else if (car == '*') {etat=3; i++;}
                       else { com[j] = car; j++; i++; etat=2; }
          break;
       }
case 3: { car= chaine[i];
            if (i==longueur) {printf("\t Commentaire non valide !!!"); getch(); return 1;}
              else if (  car == '*') {etat=3; com[j] = car; j++; i++;}
                   else if (car == '/') {etat=4; i++;}
                        else {etat=2; com[j]='*'; j++; com[j] = car; j++; i++;}
         break; }
case 4: { printf("Le corps du commentaire est: ");
          for(int n=0; n<j; n++) printf("%c", com[n]); printf("\n \t");
              return 0;}
    } }
return 0;
}
```

Expressions régulières du langage FLEX

Expression	Signification	Exemple
c	tout caractère 'c' qui n'est pas un opérateur	a
\c	caractère littéral 'c'	*
"s"	chaîne littérale s	"**"
.	tout caractère sauf fin de ligne	a.b
^r	r en début de ligne	^abc
r$	r en fin de ligne	abc$
[s]	tout caractère appartenant à s	[abc]
[^s]	tout caractère n'appartenant pas à s	[^abc]
[a-z]	tout caractère (lettre minuscule) entre 'a' et 'z'	d
[^a-z]	tout caractère qui n'est pas une lettre minuscule	5
r*	zéro ou plusieurs r	a*
r+	un ou plusieurs r	a+
r?	zéro ou un r	a?
r{m,n}	entre m et n occurrences de r	a{1,5}
r{3,}	trois r ou plus	b{3,}
r{2}	exactement deux r	c{2}
rs	r puis s	ab
r|s	r ou s	a|b
(r)	r	(a|b)
r/s	r quand suivi de s	abc / 123

Fonctions prédéfinies en FLEX:

- **char yytext[]**: tableau de caractères qui contient la chaîne reconnue.
- **`int yyleng`**: longueur de la chaîne reconnue.
- **`int yylex()`**: fonction qui lance l'analyseur (et appelle **`yywrap()`**).
- **`int yywrap()`**: fonction toujours appelée en fin du flot d'entrée. Elle ne fait rien par défaut, mais l'utilisateur peut la redéfinir dans la section des fonctions supplémentaires. **`yywrap()`** retourne 0 si l'analyse doit se poursuivre (sur un autre fichier d'entrée) et 1 sinon.
- **`int yylineno`**: numéro de la ligne courante.
- **`int main()`**: la fonction principale du langage C, elle doit appeler la fonction **`yylex()`**.

Exercice 1

Ecrire un programme en langage ***FLEX*** qui vérifie si une expression est correctement parenthésée.

Exercice 1: Ecrire un programme en langage ***FLEX*** qui vérifie si une expression est correctement parenthésée.


```
/* Exo1.l Vérification de l'équilibre des parenthèses */
%{
#include <stdio.h>
int n =0;
%}
%%
"(" {++n;}
")" {--n;}
fin {affichage(); exit(0);}
. {}
%%
int main() {
    printf("Entrez votre texte finissant avec 'fin' :\n\t");
    yylex();
    return 0;
}
```

```
affichage() {
    if (n == 0) printf("\t parentheses
    equilibrees\n");
    if (n > 0) printf("\t parentheses non
    equilibrees, %d '(' de plus\n",n);
    if (n < 0) printf("\t parentheses non
    equilibrees, %d ')' de plus\n",-n);
}

int yywrap() {
    return 1;
}
```

Exercice 1: Ecrire un programme en langage ***FLEX*** qui vérifie si une expression est correctement parenthésée.


```
//Version v2 avec les notations du c++: Exo4v2.l
//Vérification de l'équilibre des parenthèses

/* TP4_Exo4v2.l Verification de l'equilibre des parentheses */
/***  Compilation:  flex TP4_Exo1v2.l
****                gcc lex.yy.c -o TP4_Exo1
***/

%{
#include <iostream>

int n(0);
using namespace std;
void affichage();
%}
%%
"(" {++n;}
")" {--n;}
\n {affichage(); exit(0);}
. {}
%%
int main() {
     cout<<"***>"Entrez votre texte finissant avec 'fin' : ";
     yylex();
     return 0;
}
```

```
void affichage() {
     if (n == 0) cout<<"==> parentheses equilibrees\n"<<endl;

     if (n > 0) cout<<"==> parentheses non equilibrees,
     "<<n<<" '(' de plus\n"<<endl;

     if (n < 0) cout<<"==> parentheses non equilibrees,
     "<<-n<< "')' de plus\n"<<endl;
}

int yywrap() {
     return 1;
}
```

Exercice 2

Ecrire un programme en langage ***FLEX*** qui traduit les abréviations contenues dans un texte donnée.

On considérera les abréviations suivantes :

- cad : abréviation de c'est à dire,
- ssi : abréviation de si et seulement si,
- afd : automate à états finis déterministe

Exercice 2 solution


```
/* Exo2.l Les abbreviations */
%{
#include <stdio.h>
%}
%%
"cad" {printf("c'est a dire");}
"ssi" {printf("si et seulement si");}
"afd" {printf("automate a etats finis deterministe");}
fin {exit(0);}
%%
int main() {
  printf("Entrez votre texte avec les abreviations :\n\t");
  yylex();
  return 0;
}

int yywrap() {
     return 1;
}
```

Exercice 2 solution


```
/* Exo2v2.l Les abbreviations */
%{
#include <iostream>
using namespace std;
%}
%%
"cad" {cout<<"c'est a dire";}
"ssi" {cout<<"si et seulement si";}
"afd" cout<<"Entrez votre texte avec les abreviations :\n\t";
fin {exit(0);}
%%
```

```
int main() {
     cout<<"Entrez votre texte avec les abrreviations :\n\t";
     yylex();
     return 0;
}

int yywrap() {
     return 1;
}
```

Exercice 3: 1- Ecrire un programme en langage ***FLEX*** qui compte le nombre de mots d'un texte saisi au clavier.


```
/* Exo3_1.l */
/****Compilation: 1-  flex TP4_Exo3_1.l  *** 2- gcc lex.yy.c   *** 3-  a < Exo3_1.l  ****/
%{
#include <stdio.h>
#define OK (1)
unsigned int le_compte = 0;
%}
%%
[a-zA-Z]+("-"[A-Z]+)* {le_compte++; return OK;}
. |\n {}
%%
int main() {
    while (yylex())
        printf("%4d. %s\n",le_compte,yytext);
    printf("--Le texte contient %d mots--\n",le_compte);
}
int yywrap() {
    return OK;
}
```

Exercice 3: 2- Ecrire un programme en langage ***FLEX*** qui compte le nombre de mots d'un texte à partir d'un fichier.


```
/* Exo3_2.l */
%{
#include <stdio.h>
#define OK (1)  // pour avoir tous les mots
unsigned int le_compte = 0;
%}
%%
[a-zA-Z]+("-"[A-Z]+)* {le_compte++;}
.   |
\n {}
%%
```

```
int main() {
    char fichier[20];
    printf("Donnez le nom de fichier a analyser : ");
    scanf("%s", fichier);
    yyin = fopen(fichier,"r"); // initialisation de yyin
                               // par fichier au lieu de stdin
    yylex();
    //   printf("%4d. %s\n",le_compte,yytext);
    printf("--Le texte contient %d mots--\n",le_compte);
    fclose(yyin);
}
int yywrap() {
    return OK;
}
```

Exercice 4_1: 1- Ecrire un programme en langage ***FLEX*** qui compte le nombre de caractères et de lignes d'un texte source.


```
/* calcul du nombre de lignes et de caractères */
/* compilation : 1>flex -c nombre_lc.c nombre_lc.l
********** 2>gcc -o nombre_lc nombre_lc.c ********** 3>a < Exo4_1.l
*/
%{
#include <stdio.h>
int nom_lignes =0, nom_car = 0;
%}
%%
\n {++nom_lignes; ++nom_car;}
. {++nom_car;}
%%
int main() {
        yylex();
        printf("nombre de lignes = %d, nombre de caracteres = %d\n", nom_lignes, nom_car);
        return 0;
}
int yywrap() {
        return 1;
}
```

Exercice 4_2: Ecrire un programme en langage ***FLEX*** qui compte le nombre de caractères, de mots et de lignes d'un texte source.


```
/* Exo4_2.l  nombre de caractères, de mots et de lignes */
%{
#include <stdio.h>
int nom_lignes =1, nom_mots=0, nom_cars = 0;
%}
%%
[^\t\n ]+ {nom_mots++; nom_cars += yyleng;}
.       {nom_cars++;}
\n       {nom_cars++; nom_lignes++;}
%%
```

```
int main() {
        yylex();
        printf("nombre de caracteres = %d,
                nombre de    mots = %d,
                nombre de lignes = %d\n",
                nom_cars, nom_mots, nom_lignes);
        return 0;
}

int yywrap() {
        return 1;
}
```

Exercice 5

Ecrire un programme en langage ***FLEX*** qui remplace toute suite de blancs ou de tabulations par un seul blanc, supprime les espaces du début et de fin de ligne, ainsi que les lignes blanches.

Exercice 5 solution


```
/* Exo5.l */
/* Suppression des lignes blanches */
/* Suppression des blancs au début et au fin des lignes */
/* remplacement de plusieurs espaces par un seul */
%{
#include <stdio.h>
%}
%%
^[ \t]*\n {/* supprimer les lignes blanches */}
^[ \t]+    {/* supprimer les débuts blancs */}
[ \t]+$    {/* supprimer les blancs en fin de ligne */}
[ \t]+      {printf(" ");}
%%
```

```
int main() {
        yylex();
        return 0;
}
int yywrap() {
        return 1;
}
```

Exercice 6

Ecrire un programme en langage ***FLEX*** qui remplace toute suite de blancs ou de tabulations par un seul blanc et supprime les lignes vides.

Exercice 6 solution


```
/* Exo6.l */
/* Suppression des lignes vides et remplacement de plusieurs espaces par un seul */
%{
#include <stdio.h>
%}
%%
^\n {/* supprimer les lignes vides */}
[ \t]+      {printf(" ");}
%%
int main() {
        yylex();
        return 0;
}
int yywrap() {
        return 1;
}
```

TP: Calculatrice scientifique(1) (+ et -)


```
/* test avec Flex a.l */
%{
#include <math.h>
#include "b.tab.h"
%}
nombre [0-9]+
%%
{nombre} {yylval=atoi(yytext); return NOMBRE;}
"\n"     {return FIN;}
"+"      {return PLUS;}
"-"      {return MOINS;}
.        { }
%%
int yywrap(void) {
  return 1;
}
```

```
/* test avec bison b.y */
%{
#include <stdio.h>
int yyerror(char *s);
int yylex();
%}
%token FIN NOMBRE
%left PLUS MOINS
%start R
%%
R : E FIN {printf(" Resultat = %d",$1); return 0;}
  ;
E : E PLUS T  {$$=$1+$3;}
  | E MOINS T {$$=$1-$3;}
  | T         {$$=$1;}
  ;
T : NOMBRE
  ;
%%
int yyerror(char *s) {
  printf("%s",s);
  return 1;
}
int main() {
  printf(" Donnez une expression arithmétique :  "); yyparse(); return 0;
}
```

TP: Calculatrice scientifique (2)

Les étapes de compilation :

1 étape: > ***bison -d b.y***

en sortie on a, ***b.tab.c*** et ***b.tab.h***

l'analyseur syntaxique: ***b.tab.c***

et l'interface avec flex: ***b.tab.h***

2ème étape: > ***flex a.l***

en sortie on a l'analyseur lexical: ***lex.yy.c***

3ème étape: > Production du code exécutable

gcc –o ab lex.yy.c b.tab.c

en sortie on a : ***ab.exe***

Test du + et - entre entiers

Voir le contenu du fichier ***b.tab.h***

```
#define FIN=258
# define NOMBRE=259
#define MOINS=260
#define PLUS=261
#define MOINS=262
typedef int YYSTYPE;
extern YYSTYPE yylval;
```

TP: Calculatrice scientifique (3)

Les étapes suivantes:

1- L'ajout de la multiplication pour vérifier la priorité (***a1.l*** et ***b1.y***). Exemple: 3 + 5 * 2 = ?

2- L'ajout des parenthèses pour forcer une priorité (***a2.l*** et ***b2.y***). Exemple: (3+5) * 2 = ?

3- L'ajout de la division et les réels (***a3.l*** et ***b3.y***). l'ajout du fichier global.h, atof:et le format d'affichage.

Inclure le fichier ***global.h*** dans les 2 programmes:

```
/* le fichier global.h */
#define YYSTYPE double
extern YYSTYPE yylval;
```

Voir le contenu du fichier ***b.tab.h***

```
#define FIN=258
# define NOMBRE=259
#define MOINS=260
#define PLUS=261
#define MOINS=262
typedef int YYSTYPE;
extern YYSTYPE yylval;
```

TP: Calculatrice scientifique (4)


```
/* Ajout de la multiplication b1.y */
%{
#include <stdio.h>
int yyerror(char *s);
int yylex();
%}
%token FIN NOMBRE
%left PLUS MOINS
%left FOIS
%start R
%%
R : E FIN {printf(" Resultat = %d",$1);}
  ;
E : E PLUS T    {$$=$1+$3;}
  | E MOINS T {$$=$1-$3;}
  | T               {$$=$1;}
  ;
T : T FOIS F {$$=$1*$3;}
  | F            {$$=$1;}
  ;
F : NOMBRE
  ;
%%
int yyerror(char *s) {
  printf("%s",s);
  return 1;
}
int main() {
  printf(" Donnez une expression : ");
  yyparse();
  return 0;
}
```

A compiler avec: >bison -d b1.y
en sortie on a : b1.tab.c et b1.tab.h

TP: Calculatrice scientifique (5)


```
/* Ajout de la multiplication a1.l */
%{
#include <math.h>
#include "b1.tab.h"
%}
nombre [0-9]+
%%
{nombre} {yylval=atoi(yytext); return NOMBRE;}
"\n"     {return FIN;}
"+"      {return PLUS;}
"-"      {return MOINS;}
"*"      {return FOIS;}
 .       { }
```

```
%%
int yywrap(void) {
  return 1;
}
```

A compiler avec >flex a1.l
en sortie on a : lex.yy.c

La dernière étape de compilation:

> gcc -o ab1 lex.yy.c b1.tab.c

Lancer l'exécutable: ab1.exe

TP: Calculatrice scientifique (6)


```
/* Ajout des parenthèses b2.y */
%{
#include <stdio.h>
int yyerror(char *s);
int yylex();
%}
%token FIN NOMBRE PO PF
%left PLUS MOINS
%left FOIS
%start R
% %
R : E FIN {printf(" Resultat = %d",$1);}
  ;
E : E PLUS T    {$$=$1+$3;}
  | E MOINS T {$$=$1-$3;}
  | T           {$$=$1;}
  ;
T : T FOIS F    {$$=$1*$3;}
  | F           {$$=$1;}
  ;
F : NOMBRE
  | PO E PF       {$$=$2;}
  ;
% %
int yyerror(char *s) {
  printf("%s",s);
  return 1;
}
int main() {
  printf(" Donnez une expression : ");
  yyparse();
  return 0;
}
```

A compiler avec: >bison -d b2.y
en sortie on a : b2.tab.c et b2.tab.h

TP: Calculatrice scientifique (7)


```
/* Ajout des parenthèses a2.l */
%{
#include <math.h>
#include "b2.tab.h"
%}
nombre [0-9]+
%%
{nombre} {yylval=atoi(yytext); return NOMBRE;}
"\n"     {return FIN;}
"+"      {return PLUS;}
"-"      {return MOINS;}
"*"      {return FOIS;}
"("      {return PO;}
")"      {return PF;}
 .       { }
%%
int yywrap(void) {
  return 1;
}
```

A compiler avec >flex a2.l
en sortie on a : lex.yy.c

La dernière étape de compilation:

```
> gcc -o ab2 lex.yy.c b2.tab.c
```

TP: Calculatrice scientifique (8)


```
/* Ajout de la division et les réels: b3.y */
%{
#include <stdio.h>
#include "global.h"
int yyerror(char *s);
int yylex();
%}
%token FIN NOMBRE PO PF
%left PLUS MOINS
%left FOIS DIV
%start R
%%
R : E FIN {printf(" Resultat = %f",$1); return 0;}
  ;
E : E PLUS T    {$$=$1+$3;}
  | E MOINS T {$$=$1-$3;}
  | T           {$$=$1;}
  ;
T : T FOIS F    {$$=$1*$3;}
  | T DIV F     {if ($3==0) {printf("division par zero interdite !"); return 1;}
                 else  $$=$1/$3;}
  | F           {$$=$1;}
  ;
F : NOMBRE
  | PO E PF       {$$=$2;}
  ;
%%
int yyerror(char *s) {
  printf("%s",s);
  return 1;
}
int main() {
  printf("donner une expression : ");
  yyparse();
  return 0;
}
```

```
/* inclusion d'un fichier d'interface global.h */
#define YYSTYPE double
extern YYSTYPE yylval;
```

A compiler avec: >bison -d b3.y

en sortie on a : b3.tab.c et b3.tab.h

TP: Calculatrice scientifique (9)


```
/* Ajout de la division et des réels: a3.l */
%{
#include <math.h>
#include "global.h"
#include "b3.tab.h"
%}
nombre [0-9]+(\.[0-9]+)?((e|E)(\+|\-)?[0-9]+)?
%%
{nombre} {yylval=atof(yytext); return NOMBRE;}
"\n"     {return FIN;}
"+"      {return PLUS;}
"-"      {return MOINS;}
"*"      {return FOIS;}
"/"      {return DIV;}
"("      {return PO;}
")"      {return PF;}
 .       { }
%%
int yywrap(void) {
  return 1;
}
```

A compiler avec >flex a3.l
en sortie on a : lex.yy.c

Les autres étapes de compilation:

```
> gcc -c lex.yy.c -o a3.o
> gcc -c b3.tab.c -o b3.o
> gcc -o ab3 a3.o b3.o
```

TP: Calculatrice scientifique (10)


```
/* Ajout du plus et du moins unaires: b4.y */
%{
#include <stdio.h>
#include "global.h"
int yyerror(char *s);
int yylex();
%}
%token FIN NOMBRE PO PF
%left PLUS MOINS
%left FOIS DIV
%nonassoc POS NEG
%start R
%%
R : E FIN {printf(" Resultat = %f",$1); return 0;}
   ;
E : E PLUS T    {$$=$1+$3;}
   | E MOINS T {$$=$1-$3;}
   | T          {$$=$1;}
  ;
T : T FOIS F    {$$=$1*$3;}
   | T DIV F    {if ($3==0) {printf("division par zero interdite !");
                  return 1;} else  $$=$1/$3;}
   | F          {$$=$1;}
  ;
F : NOMBRE      {$$=$1;}
   | PO E PF    {$$=$2;}
   | MOINS F  %prec NEG {$$= -$2;}
   | PLUS F   %prec POS {$$= +$2;}
%%
int yyerror(char *s) {
  printf("%s",s);
  return 1;
}

int main() {
  printf("donner une expression : ");
  yyparse();
  return 0;
}

/* inclusion d'un fichier d'interface global.h */
#define YYSTYPE double
extern YYSTYPE yylval;
```

A compiler avec: >bison -d b4.y
en sortie on a : b4.tab.c et b4.tab.h

TP: Calculatrice scientifique (11)


```
/* Ajout du plus et du moins unaires: a4.l */
%{
#include <math.h>
#include <stdlib.h>
#include "global.h"
#include "b4.tab.h"
%}
nombre [0-9]+(\.[0-9]+)?((e|E)(\+|\-)?[0-9]+)?
%%
{nombre} {yylval=atof(yytext); return NOMBRE;}
"\n"     {return FIN;}
"+"      {return PLUS;}
"-"      {return MOINS;}
"*"      {return FOIS;}
"/"      {return DIV;}
"("      {return PO;}
")"      {return PF;}
 .       { }
%%
int yywrap(void) {
  return 1;
}
```

A compiler avec >flex a4.l
en sortie on a : lex.yy.c

Les autres étapes de compilation:

```
> gcc -c lex.yy.c -o a4.o
> gcc -c b4.tab.c -o b4.o
> gcc -o ab4 a4.o b4.o
```

Lancer l'exécutable: ab4.exe

Fonctions arithmétiques en C

Le fichier en-tête ***<math.h>*** *déclare des fonctions mathématiques. Tous les paramètres et résultats sont du type* ***double****; les angles sont indiqués en radians.*

double sin(double X) sinus de X

double cos(double X) cosinus de X

double tan(double X) tangente de X

double asin(double X) arcsin(X) dans le domaine [-π/2, π/2], x[-1, 1]

double acos(double X) arccos(X) dans le domaine [0, π], x[-1, 1]

double atan(double X) arctan(X) dans le domaine [-π/2, π/2]

double exp(double X): fonction exponentielle : e^X

double log(double X): logarithme naturel : ln(X), X>0

double log10(double X):
logarithme à base 10 : $\log_{10}(X)$, X>0

double pow(double X, double Y): X exposant Y : X^Y

double sqrt(double X): racine carrée de X : $\sqrt{X}$, X≥0

double fabs(double X): valeur absolue de X : |X|

double floor(double X): arrondir en moins: int(X)

double ceil(double X): arrondir en plus

Fonctions arithmétiques en C

Le fichier en-tête ***<math.h>*** *déclare des fonctions mathématiques. Tous les paramètres et résultats sont du type* ***double****; les angles sont indiqués en radians.*

Exemple1 : *valeur absolue*

```
#include <stdio.h>
#include <math.h>
main() {
    double nbre = -1234.0;
    printf("Valeur absolue de %lf =  %lf\n", nbre, fabs(nbre));
    return 0;
}
/*-- résultat de l'exécution ---------------------------------
Valeur absolue de -1234.000000 =  1234.000000
------------------------------------------------------------*/
```

Exemple 2 : arrondis

```
#include <stdio.h>
#include <math.h>

main() {
   double nbre = 1234.56;

   printf("ceil de  %lf = %lf\n", nbre, ceil(nbre) );
   printf("floor de %lf = %lf\n", nbre, floor(nbre) );
   return 0;
}
/*-- résultat de l'exécution ----------------------------
ceil de  1234.560000 = 1235.000000
floor de 1234.560000 = 1234.000000
------------------------------------------------------------*/
```

Fonctions arithmétiques en C


```
printf("%f", 100.123);        ==>    100.123000

printf("%12f",                ==>    __100.123000
100.123);

printf("%.2f",                ==>    100.12
100.123);

printf("%5.0f",               ==>    __100
100.123);

printf("%10.3f",              ==>    ___100.123
100.123);

printf("%.4f",                ==>
1.23456);                            1.2346
```

```
float N = 12.1234;

double M = 12.123456789;

long double P = 15.5;

printf("%f", N);              ==> 12.123400

printf("%f", M);              ==> 12.123457

printf("%e", N);              ==> 1.212340e+01

printf("%e", M);              ==> 1.212346e+01

printf("%Le", P);             ==> 1.550000e+01
```

La partie qui vous concerne

L'examen de l'année dernière:

Exercice 1

- Ecrire des programmes en langage **Flex** qui permettent de :

1- Supprimer les ***lignes vides*** d'un fichier.

2- Supprimer les ***lignes "blanches"*** c'est à dire ne contenant que des espaces et tabulations.

3- Supprimer les ***"blancs" inutiles en fin de ligne***.

Exercice 4

- Donner une expression régulière qui valide une forme simplifiée des adresses électroniques.
- Définition simplifiée :

- une adresse électronique est constituée d'un champ ou plusieurs suivies d'un @ suivi d'un champ ou plusieurs.

- un champ est constitué d'un caractère ou plusieurs (lettre, chiffre, -, _).

Exercice 5

Soit la grammaire simplifiée des expressions logiques,

P ⟶ P ∧ Q
P ⟶ P v Q
P ⟶ Q
Q ⟶ vrai
Q ⟶ faux

Analyser la phrase suivante : "vrai ∧ faux v faux" par la méthode ascendante de décalage/réduction en précisant à chaque étape, les types de conflits et en tenant compte des conditions suivantes :

- favoriser la réduction par rapport au décalage,

- favoriser la réduction du préfixe le plus long.

Exercice 6

- Ecrire une application avec les langages **flex** et **bison** qui calcule la puissance d'un nombre réel et affiche le résultat.
- Exemple : Donnée : **3 puissance 2** Résultat : **9**