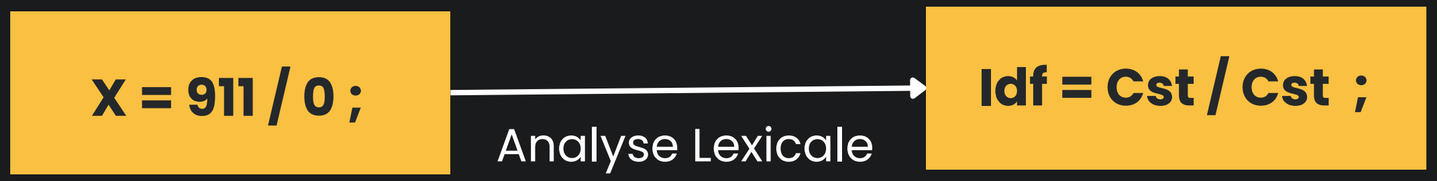
PARTAGE DE VARIABLE ENTRE FLEX ET BISON

Comment Bison peut-il traiter les valeurs des entités lexicales, sachant que l'analyseur lexical doit coder toutes les entités du programme source ?



Solution :

Flex peut transmettre la valeur d'une entité avant de la coder en utilisant la variable prédéfinie **yylval**.

-> La variable **yylval** est nécessaire pour détecter les erreurs sémantiques. L'analyseur lexical l'enverra donc en complément du code de l'entité.

L'utilisation de la yylval

1. Déclaration

Lexical.l

//Declaration de yylval dans flex

```
%{
extern YYSTYPE yylval;
%}
```

Syntax.y

//Declaration de type YYSTYPE dans bison Comme type composé

```
%union {
int   entier;
char* str;
float numvrg;
}
```

2. Utilisation

Lexical.l

```
//Envoie de valeur se fait dans flex
  %%
  {cst} { yylval.entier=atoi(yytext);
            return cst; }

  {idf} {yylval.str=strdup(yytext);
          return idf; }
  {reel} { yyval.numvrg=atof(yytext);
  return reel;}
   ....
  %%
```

Syntax.y

```
//Traitement de la de valeur dans bison
%token <str>idf   <entier>cst <numvrg> reel
%%
s : idf aff  idf  div cst   { if ($5==0)  {printf("
erreur : division par zéro");}
else printf("la divion est%s= %s/%d",
$1,$3,$5);
        YYACCEPT;
}
....
.%%
```

Remarques :

- En C, on ne peut pas comparer directement deux chaînes de caractères avec l'opérateur == car cela comparerait les adresses mémoire des chaînes et non leur contenu.
- La fonction **strdup** en C crée une copie d'une chaîne de caractères en allouant dynamiquement de la mémoire pour cette copie.
- Le symbole **$$** référence la valeur associée au **non terminal** de la **partie gauche** d’une règle de grammaire.
- Le symbole **$i** référence la valeur associée au **i ème symbole terminal ou non terminal** de la **partie droite** d’une règle de grammaire.

TABLE DE SYMBOLES

Création de la table de symboles

- Elle doit être programmée manuellement dans flex et bison.
- La table de symbole doit contenir des informations sur les entités.
- Chaque entité reconnu par le langage doit être insérée dans la table de symboles.
- Elle est très utile pour la détection des erreurs sémantiques.

Programme source

```
Program L3Compil
integer x;
Const reel y=5;
begin
x=y+1;
end
```

NomEntité	CodeEntité	Type	Const
L3Compil	Idf	/	Non
x	Idf	integer	Non
y	Idf	reel	Oui

Déclaration de la table de symbole

Lexical.l

```
%{
//structure de la table de symbole
typedef struct
{
char NomEntite[20];
char CodeEntite[20];
} TypeTS;
//Initialisation d'un tableau pour stocker les éléments de la table des symboles.
TypeTS ts[100];
// compteur global pour la table de symbole
int CpTS = 0;
```

Fonction de recherche dans la TS

- la fonction **recherche**: cherche est ce que l'entité existe ou non dans la table de symbole.
- Entrée: Entité
- Sortie:

Si l'entité existe déjà, il va renvoyer sa position.

Sinon -1, s'il elle n'existe pas dans la TS.

```
int recherche(char entite[])
{
 int i = 0;
 while (i < CpTS)
 {
  if (strcmp(entite, ts[i].NomEntite) == 0)
    return i;
  i++;
 }

 return -1;
}
```

Fonction d'insertion dans la TS

- fonction qui va insérer les entités de programme dans la table de symbole

```
void inserer(char entite[], char code[])
{
 if (recherche(entite) == -1) //elle n'existe pas
 {
  strcpy(ts[CpTS].NomEntite, entite);
  strcpy(ts[CpTS].CodeEntite, code);
  CpTS++;
 }
}
```

Fonction d'affichage de la TS

- Pour afficher le contenu de la table de symboles.

```
void afficher()
{
 printf("\n/***************Table des symboles ******************/\n");
 printf("_____________________________________\n");
 printf("\t| NomEntite | CodeEntite | \n");
 printf("_____________________________________\n");
 int i = 0;
 while (i < CpTS)
 {
  printf("\t|%10s |%12s |\n", ts[i].NomEntite, ts[i].CodeEntite);
  i++;
 }
}
```

Utilisation du fichier TS.h

- La déclaration de la table des symboles (TS) et les fonctions associées peuvent encombrer le fichier lexical.l.
- Pour plus de clarté, il est conseillé de les déplacer dans un fichier d'en-tête, par exemple **TS.h**, et de l'inclure dans lexical.l avec:

#include "TS.h"

TS.h

```
//Structure de la TS
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
typedef struct
{
 char NomEntite[20];
 char CodeEntite[20];
} TypeTS;

TypeTS ts[100];

// init compteur
int CpTS = 0;

void inserer(char entite[], char code[])
{
}

void recherche (char entite[])
{
}

void afficher()
{
}
```

Exemple:

Voici le programme suivant:

```
Program  L3ISILA
PDec
integer x,i,j;
Reel  y,z,a;
InBEGIN
 i=2;
 j=i/0;
 x=x+1;
InEnD
```

Lexical.l

```
%{
#include "syn.tab.h"
#include "TS.h"
extern int nb_ligne;
extern YYSTYPE yylval;
extern int col;
%}
lettre [a-zA-Z]
chiffre [0-9]
idf {lettre}({lettre}|{chiffre})*
cst [1-9][0-9]*|0
%%
Program {return mc_program;}
PDec {return mc_pdec;}
integer {yylval.str=strdup(yytext); return mc_integer;}
Reel  {yylval.str=strdup(yytext); return mc_real;}
InBEGIN {return mc_inbegin;}
InEnD {return mc_inend;}
[,;=/+] {return yytext[0];}
{cst} {yylval.entier=atoi(yytext); return cst;}
{idf} {yylval.str=strdup(yytext); inserer(yytext,"idf"); return idf;}
[ \t]
\n {nb_ligne++;col=1;}
. {printf ("erreur lexical sur l'entite %s a la ligne %d a la colonne %d",yytext,nb_ligne,col);
return err;}
%%
```

syn.y

```
%{
 int nb_ligne=1;
 int col=1;
 void yyerror(char *msg);
%}
%start S
%union
{
int entier;
char* str;
}
%token mc_inbegin mc_inend mc_program mc_pdec err
%token <str>idf <str>  mc_integer <str> mc_real
%token <entier> cst
%%
S: Header mc_pdec DECLARATION INSTRUCTION
{ printf ("programme syntaxiquement correcte . \n"); YYACCEPT;}
;
Header: mc_program idf
;
DECLARATION: type listeparams ';' DECLARATION
    | type listeparams ';'
;
listeparams: listeparams ',' idf
 | idf
 ;
type: mc_integer
  | mc_real
;
INSTRUCTION: mc_inbegin ListeInstr mc_inend
;
ListeInstr: ListeInstr Instruction
   | Instruction
;
Instruction : instaff
    | instdiv
    | instadd
;
instaff: idf '=' idf ';'
  | idf '=' cst ';'
;
instdiv: idf '=' idf '/' idf ';'
  | idf '=' idf '/' cst ';'
  ;
instadd: idf '=' idf '+' idf ';'
 | idf '=' idf '+' cst ';'
 ;
%%
void yyerror(char *msg) {
 printf("Erreur syntaxique %s, a la ligne %d \n", msg, nb_ligne);
}
main ()
{
yyparse();
afficher();
}
yywrap(){
}
```

Résultat de l'execution:

TSv1-ISIL A > test.txt

```
1   Program  L3ISILA
2   PDec
3   integer x,i,j;
4   Reel  y,z,a;
5   InBEGIN
6    i=2;
7    j=i/0;
8    x=x+1;
9   InEnD
```

```
programme syntaxiquement correcte .

/****************Table des symboles ******

-----------------------------------
         | NomEntite |  CodeEntite  |
-----------------------------------
         |   L3ISILA |          idf |
         |         x |          idf |
         |         i |          idf |
         |         j |          idf |
         |         y |          idf |
         |         z |          idf |
         |         a |          idf |
```