

Utilisation de Lex & YACC

■ ■ ■ Utilisation de Lex & YACC

1 – a. Vous écrirez le fichier au format lex

```
%{
#include "contact_syntaxique.h"
#include <malloc.h>
#include <string.h>
}%

NUMERO [0-9]{10}
ADRESSE [a-z0-9]+@[a-z0-9\.\.]+\.[a-z]{2,}
%start CARACTERES
%%
[Pp]ersonne return PERSONNE;
[Cc]ontact return CONTACT;
"}" return ACCOLADE_FERMANTE;
"{" return ACCOLADE_OUVRANTE;
numero_tel return NUMTEL;
email return EMAIL;
<INITIAL>" BEGIN CARACTERES;
<CARACTERES>{NUMERO} {
    yylval.chaine = (char *) malloc(sizeof(char)*yyleng+1);
    strncpy(yylval.chaine, yytext, yylen+1);
    return CHAINE;
}
<CARACTERES>{ADRESSE} {
    yylval.chaine = (char *) malloc(sizeof(char)*yyleng+1);
    strncpy(yylval.chaine, yytext, yylen+1);
    return CHAINE;
}
<CARACTERES>" BEGIN INITIAL;
. /* vide */
\n /* vide */
%%
```

b. Complétez le fichier d'analyse syntaxique

```
%{
#include "contact_syntaxique.h"
#include <stdio.h>
}%

%union {
    char *chaine;
}

%token <chaine> PERSONNE CONTACT ACCOLADE_FERMANTE ACCOLADE_OUVRANTE NUMTEL EMAIL CHAINE
%type <chaine> Contact
%start Input
%%
Input : PERSONNE ACCOLADE_OUVRANTE Contact ACCOLADE_FERMANTE
;
Contact : /* vide */
| Contact CONTACT ACCOLADE_OUVRANTE NUMTEL CHAINE ACCOLADE_FERMANTE {
    printf("Numero de tel : %s\n", $5);
    free($5);}
| Contact CONTACT ACCOLADE_OUVRANTE EMAIL CHAINE ACCOLADE_FERMANTE {
    printf("Adresse electronique : %s\n", $5);
    free($5);}
;

%%
int yyerror(char *s)
{
    printf("%s", s);
}
```

```
int main()
{
    yyparse();
}
```

2 – On veut vérifier un fichier qui contient des définitions de figures en 2D :

```
struct Point{
    int x;
    int y; };
```

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "global.h"
    #include "geometrie_syntaxique.h"
    struct Point mon_point;
}%
ENTIER [1-9][0-9]*
ESPACE [ \t]*
%start PREM_ENTIER SEC_ENTIER
%%
Point{ESPACE}"("{ESPACE} BEGIN PREM_ENTIER;
<PREM_ENTIER>{ENTIER} mon_point.x = atoi(yytext);
<PREM_ENTIER>{ESPACE}", "{ESPACE} BEGIN SEC_ENTIER;
<SEC_ENTIER>{ENTIER} { mon_point.y = atoi(yytext);
    yylval.pointeur_point = (struct Point *) malloc (sizeof(struct Point));
    yylval.pointeur_point->x = mon_point.x;
    yylval.pointeur_point->y = mon_point.y;
    return POINT; }
<SEC_ENTIER>{ESPACE})" BEGIN INITIAL;
Figure\{ return FIGURE;
Segment\{ return SEGMENT;
}" return ACCOLADE_FERMANTE;
{ESPACE} /* ignorer */
\n /* ignorer */
%%
```

```
%{
    #include "geometrie_syntaxique.h"
    #include "global.h"
    #include <stdio.h>
}%
%union{
    struct Point *pointeur_point;
}
%token <pointeur_point> FIGURE SEGMENT POINT ACCOLADE_FERMANTE
%start Input

%%
Input : FIGURE ListeSegments ACCOLADE_FERMANTE
;
Un_Segment : SEGMENT POINT POINT ACCOLADE_FERMANTE
{
    printf("Segment : P(%d,%d) P(%d, %d)\n",
        $2->x, $2->y, $3->x, $3->y);
    free($2);
    free($3);
}
;

ListeSegments : Un_Segment
| Un_Segment ListeSegments
;

/*ListeSegmentsContrainte : Un_Segment Un_Segment Un_Segment
| Un_Segment Un_Segment Un_Segment Un_Segment ;
*/
%%
int yyerror(char *s)
{
    printf("%s", s);
}

int main()
{
    yyparse();
}
```

3 – Voir Projet.

4 – On veut vérifier qu'un programme écrit en Pascal est valide :

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "pascal_syntaxique.h"
}%
ESPACE [ \t]*
%%
INTEGER return TYPE;
VAR return VAR;
BEGIN return DEBUT;
END\. return ENDP;
PROGRAM return PROGRAM;
"==" return AFFECTATION;
(Read|ReadLn) return FONCTION;
(Write|WriteLn) return FONCTION;
\'.*\' return CHAINE;
, return VIRGULE;
"+" return OP;
\( return PO;
\) return PF;
":" return DPTS;
[A-Za-z][A-Za-z0-9]+ return ID;
"," return PV;
\{.*\} /* ignorer */
{ESPACE} /* ignorer */
\n /* ignorer */
%%
```

```
%{
    #include "pascal_syntaxique.h"
    #include <stdio.h>
}%
%token TYPE VAR DEBUT ENDP PROGRAM AFFECTATION FONCTION CHAINE VIRGULE OP PO PF DPTS
PV ID
%start Input
%%
Input : PROGRAM ID PV Declarations DEBUT ListeInstructions ENDP
      ;
Declarations : VAR ListeDeclarations
            ;
ListeDeclarations : ListeIdentifiants DPTS TYPE PV
                  | ListeIdentifiants DPTS TYPE PV ListeDeclarations
                  ;
ListeIdentifiants : ID
                  | ID VIRGULE ListeIdentifiants
                  ;
ListeInstructions : Instruction
                  | Instruction ListeInstructions
                  ;
Instruction : error PV
            | ID AFFECTATION Expression PV { printf("Instruction Affectation recon
nue\n"); }
            | FONCTION PO ListeArguments PF PV { printf("Instruction Fonction recon
nue\n"); }
            ;
ListeArguments : Argument
               | Argument VIRGULE ListeArguments
               ;
Argument : ID
         | CHAINE
         ;
Expression : ID
           | Expression OP Expression
           ;
%%
int yyerror(char *s) { printf("%s\n",s); }
int main() { yyparse(); }
```