

Utilisation de Lex & YACC

■ ■ ■ Utilisation de Lex & YACC

1 – On veut vérifier un fichier qui contient un carnet d'adresse contenant des informations concernant plusieurs personnes :

- chaque personne est défini par une liste de contacts ;
- chaque contact est défini par :
 - ◊ un type: numero_tel, email ;
 - ◊ une chaîne de caractère, qui peut contenir un ou plusieurs numéros de téléphone, une ou plusieurs adresses mail *etc.* ;
- pour définir une personne, on donne la liste des contacts qui la définissent :

```
1 Personne{
2   Contact{
3       numero_tel "0567596867"
4   }
5   Contact{
6       email "toto@laposte.net"
7   }
8   Contact{
9       numero_tel "0661496867"
10  }
11 }
```

a. Vous écrirez le fichier au format `lex`, nécessaire à l'analyse lexical et qui renvoie les **tokens** nécessaires à l'analyse syntaxique.

La liaison avec la partie « analyse syntaxique », est décrite dans le début du fichier au format `yacc`, qui est le suivant :

```
1 %{
2   #include "contact_syntaxique.h"
3   #include <stdio.h>
4   %}
5
6   %union {
7       char *chaine;
8   }
9
10  %token <chaine> PERSONNE CONTACT ACCOLADE_FERMANTE ACCOLADE_OUVRANTE NUMTEL
11  EMAIL CHAINE
12  %type <chaine> Contact
13  %start Input
14  %%
```

b. Complétez le fichier d'analyse syntaxique pour vérifier qu'un fichier contact est correctement écrit.
Vous afficherez la définition de chaque contact lors de l'analyse.

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

- ◇ chaque figure est composée d'une liste de segments ;
- ◇ chaque segment est défini par deux points ;
- ◇ chaque point est défini par ses coordonnées (x, y) ;
- ◇ pour définir une figure, on donne la liste des 3 ou 4 segments qui la définissent :

```
Figure{
    Segment{
        Point(1, 30)
        Point(20, 30)
    }
    Segment{
        Point(20, 30)
        Point(2, 10)
    }
    Segment{
        Point(2, 10)
        Point(1, 30)
    }
}
```

Vous disposez du fichier, `global.h`, suivant :

```
typedef struct Point Point;

struct Point{
    int x;
    int y;
}

#define YYSTYPE Point
extern YYSTYPE yylval;
```

Vous disposez déjà du fichier au format `lex`, qui vous permet de faire l'analyse lexical et qui vous renvoie les tokens nécessaires à l'analyse syntaxique.

Le début du fichier d'analyse syntaxique, au format `yacc`, est le suivant :

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

%token FIGURE SEGMENT POINT ACCOLADE_FERMANTE
%start Input

%%
```

Question : Complétez le fichier d'analyse syntaxique pour vérifier qu'un fichier est correctement écrit. Vous afficherez la définition de chaque segment lors de l'analyse.

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

```
1 PROGRAM Addition;
2 VAR
3   Somme      : INTEGER;
4   Nombre1,
5   Nombre2    : INTEGER;
6
7 BEGIN
8
9   Write ('Premier nombre ? : ');
10  ReadLn (Nombre1); { Lecture 1er nombre }
11
12  Write ('Deuxième nombre ? : ');
13  ReadLn (Nombre2); { Lecture 2ème nombre }
14
15  Somme := Nombre1 + Nombre2;
16  WriteLn ('La somme vaut: ', Somme);
17
18 END.
```

- 3 – a. Écrivez un analyseur syntaxique permettant de vérifier si un programme écrit en Pascal est valide ou non ;
- b. Améliorez votre analyseur pour indiquer en cas d'erreur :
- ◊ le numéro de la ligne où l'erreur s'est produite ;
 - ◊ le début de la ligne.
- c. Améliorez votre analyseur pour pouvoir continuer l'analyse après une erreur.