TP01 - ANALYSE LEXICALE

1. Objectif

L'objectif de ce TP est de programmer un analyseur lexical pour le langage L. L'analyseur lexical sera d'abord décrit à l'aide d'expressions régulières dans un fichier `.flex`, puis compilé à l'aide du logiciel `flex`. Le code C généré contiendra une fonction appelée `yylex` qui, à chaque appel, renvoie comme résultat le code de l'unité lexicale dont c'est le tour. De plus, cette fonction met automatiquement dans la variable `yytext` la valeur de l'unité lexicale reconnue. Ainsi, lorsque `yylex` reconnaît la chaîne de caractères `123`, elle renverra le type `NOMBRE` et mettra dans la variable `yytext` la chaîne de caractères "`123`".

2. Le langage L

Le langage L est un langage de programmation minimaliste, inspiré du langage C.

Types: Le langage L connaît deux types de variables :
- Un type simple : le type `entier`.
- Un type dérivé : les tableaux d'entiers, déclarés avec des crochets `[ ... ]`.

Opérateurs: Le langage L connaît les opérateurs suivants :
- arithmétiques : `+`, `-`, `*`, `/`
- comparaison : `<`, `=`
- logiques : `&` (et), `|` (ou), `!` (non)

Instructions: Le langage L connaît les instructions suivantes :
- Instruction vide `;`
- Bloc d'instructions, délimité par des accolades `{ ... }`
- Affectation `$a = $b + 1;`
- Instruction `si` *expression* `alors { ... }` et `si` *expression* `alors { ... } sinon { ... }`
- Instruction `tantque` *expression* `faire { ... }`
- Instruction `retour` *expression* `;`
- Instruction d'appel à fonction `fonction(` *liste d'expressions* `);`

Fonctions: un programme L est une suite de fonctions, dont `main`
- Ce sont des fonctions à résultat entier.
- Le passage des arguments se fait par valeur.
- Les fonctions possèdent des variables locales.
- Une fonction ne peut pas être déclarée à l'intérieur d'une autre.
- On peut ignorer le résultat rendu par une fonction.

Fonctions prédéfinies: Les entrées-sorties de valeurs entières se font à l'aide de deux fonctions prédéfinies, `$a = lire();` et `ecrire($a);`.

Voici un exemple d'un programme en L :

```
f(entier $a, entier $b)  # déclaration d'une fonction à deux arguments
entier $c, entier $k;    # déclaration de deux variables locales
{                        # début d'un bloc d'instruction
  $k = $a + $b;          # affectation et expression arithmétique
  retour $k;             # valeur de retour de la fonction
}                        # fin du bloc d'instruction
main()                   # point d'entrée dans le programme
entier $d;
{
  $d = f($d, 2);         # affectation et appel de fonction
  ecrire($d + 1);        # appel de la fonction prédéfinie ecrire
}
```

3. Unités lexicales

On distingue 5 types d'unités lexicales :
- les symboles simples (`+`, `;`, `(`, `{`, ...),
- les mots-clefs, dont :
  - les instructions de contrôle (`si`, `alors`, `retour`, ...),
  - les types (`entier`),
  - les fonctions spéciales `lire` et `ecrire`,
- les noms de variables,
- les noms de fonctions et
- les nombres, uniquement entiers.

Les unités lexicales de type symboles simples et mots-clefs constituent des **classes fermées**, on peut en faire l'inventaire exhaustif.
Les unités lexicales de type noms de variables et de fonction ainsi que les nombres constituent des **classes ouvertes**, on ne peut pas en faire l'inventaire. Cependant, on peut les décrire à l'aide de contraintes de forme (par exemple : un nombre entier est une suite de chiffres).
Les unités lexicales des deux différentes classes distinguées ci-dessus ne sont pas traités de la même manière. Dans le cas d'une classe fermée, c'est l'unité lexicale elle même qui est retournée par `yylex` (en fait un code correspondant à l'unité lexicale, comme expliqué ci-dessous). Dans le cas d'une classe ouverte, c'est le (code du) *type d'unité lexicale* qui est renvoyé ; la valeur de l'unité lexicale est copiée dans `yytext`.
Chaque unité lexicale d'une classe fermée ou type d'unité lexicale d'une classe ouverte est représentée par une constante symbolique, définie comme ceci :

```
#define TANTQUE 20
```

Les déclarations des constantes symboliques sont regroupées dans le fichier `symboles.h` qui vous est fourni et qui sera partagé avec l'analyseur syntaxique. Il convient d'utiliser les constantes données dans ce fichier pour pouvoir tester facilement le programme. Pour le moment, nous ne regarderons que la section de symboles terminaux dans ce fichier. Les constantes correspondant aux symboles non terminaux seront utilisées plus tard, pendant l'analyse syntaxique. Pour utiliser `symboles.h` avec flex, vous pouvez inclure `symboles.h` dans votre fichier `.flex`.
Le dernier code renvoyé par l'analyseur lexical doit être le symbole (`FIN`). C'est-à-dire que, quand l'analyseur lexical arrive à la fin du fichier (EOF), il doit encore renvoyer l'unité lexicale "`FIN`" pour signaler qu'il est arrivé à la fin du fichier. Ceci sera pratique lors de l'analyse syntaxique.

4. Identificateurs et nombres

Un identificateur est une suite de caractères qui n'est pas un nombre ni un mot dans le tableau des mots-clefs. Il s'agit d'une suite de lettres non accentuées majuscules ou minuscules, et les symboles dollar ($) ou souligné (_). Un identificateur de variable commence toujours par un symbole dollar tandis qu'un identifiant de fonction commence toujours par un caractère différent d'un chiffre ou d'un dollar (c'est-à-dire une lettre majuscule, minuscule ou souligné). De plus, un identificateur de fonction ne peut pas être identique à un mot-clef du langage L. Par exemple, on ne peut pas déclarer une fonction qui s'appelle `tantque` ou `ecrire`. Le langage L est sensible à la casse.

5. Lecture du texte source

L'analyseur lexical accède au texte source caractère par caractère. Avec flex, la lecture du fichier contenant le programme source se fera à travers un flux appelé `yyin`. Le programme de test fourni initialise la variable `yyin` avant de faire appel à la fonction `yylex` de l'analyseur lexical généré par flex.

6. Commentaires et blancs

Les unités lexicales peuvent être séparées les unes des autres dans le programme source par un nombre quelconque de blancs, caractères de tabulation (`\t`) et retours chariot (`\n`). Tous ces caractères doivent être ignorés. C'est l'analyseur lexical qui doit s'occuper de supprimer les espaces.

Cependant, l'analyseur doit s'en servir pour séparer les unités lexicales. Par exemple, il est obligatoire de mettre un (ou plusieurs) espaces dans la déclaration d'une variable, entre le mot-clef et le nom, par exemple, `entier $max`. Cependant, dans une expression comme `$max = 1;` les espaces sont optionnels car le caractère `=` ne peut pas faire partie de l'identificateur.
Il est possible de rajouter des commentaires d'une ligne. Ces commentaires commencent par un caractère dièse (`#`) et terminent à la fin de la ligne. Le caractère dièse et tout ce qui suit jusqu'à la fin de la ligne (`\n`) doit être ignoré par le compilateur. Il ne doit rester aucune trace des commentaires au niveau de l'analyseur syntaxique. Notamment, les commentaires ne sont mentionnés nulle part dans la grammaire.

7. Programmation avec flex

Dans ce TP, vous n'allez pas écrire du code C directement. Vous allez décrire les unités lexicales de L dans un fichier `.flex` nommé `analyseur_lexical.flex`, qui sera ensuite compilé à l'aide de la commande ci-dessous :

```
flex -o analyseur_lexical.c analyseur_lexical.flex
```

Ceci va générer un fichier source appelé `analyseur_lexical.c` que vous allez ensuite compiler séparément vers un fichier objet :

```
gcc -c analyseur_lexical.c
```

Puis finalement, lors du linking, vous allez utiliser ce fichier pour compiler votre programme de test nomme `test_yylex` :

```
gcc -o test_yylex test_yylex.c analyseur_lexical.o util.o
```

8. Exemple

Voici un exemple possible de résultat de l'analyse syntaxique d'une instruction *si ... alors* :

source	`si`	`$n`	`<`	`1`	`alors`	`$n`	`=`	`$n`	`+`	`1`
retour	`SI`	`ID_VAR`	`INFERIEUR`	`NOMBRE`	`ALORS`	`ID_VAR`	`EGAL`	`ID_VAR`	`PLUS`	`NOMBRE`

9. Programme de test

La fonction `yylex` sera appelée par l'analyseur syntaxique qui ne sera programmé que lors des prochains TP. Il est fortement conseillé de valider l'analyseur lexical à partir d'un programme de test. Nous fournissons le programme `test_yylex.c` qui ouvre un fichier passé en paramètre, l'affecte à la variable `yyin`, puis exécute une boucle de lecture de ce fichier en appelant la fonction `yylex` générée par flex, puis en affichant le nom et la valeur de l'unité lexicale retournée. Ce programme de test utilise une bibliothèque de fonctions utiles appelée `util.c` qui est aussi fournie, et qui pourra vous aider à développer votre compilateur.