Institut Supérieur d'Informatique et de Mathématiques de Monastir

Année universitaire 2024/2025
Section : L2 INFO
Enseignante : Aida Lahouij
Matière : Techniques de Compilation

TP1 : Initiation à Flex et Introduction à l'analyse lexicale

I. Introduction

Lex et YACC sont une paire d'outils qui peuvent être utilisés ensemble pour générer un compilateur ou son front-end. Parmi les versions les plus populaires figurent les outils Flex et Bison du système GNU de l'Open Software Foundation. Ces outils utilisent le langage C pour spécifier le *code d'action* (désigne le code écrit par un programmeur à exécuter à des points spécifiques d'une exécution lexer et/ou parser). On peut trouver de l'aide sur son utilisation dans les pages de manuel ou bien sur Internet, notamment sur le site http: www.gnu.org/software/flex/manual

II. Objectif

L'objectif de ce TP est de s'initier à l'outil Flex afin de programmer par la suite un analyseur lexical et un analyseur syntaxique qui seront produits automatiquement à partir d'un fichier de spécification. Ce fichier décrit les unités lexicales du langage, ainsi que ses règles de grammaire. Tout au long de ce TP, on utilisera :

- **Flex** comme générateur d'analyseur lexical.
- **Bison** comme générateur d'analyseur syntaxique.
- **Dev C++** comme environnement de développement intégré (IDE).

III. INSTALLATION DES OUTILS

1. Utilisez les liens suivants pour télécharger respectivement les binaires de Flex, Bison et Dev C++.

- https://sourceforge.net/projects/gnuwin32/files/flex/2.5.4a-1/flex-2.5.4a-1.exe/download?use_mirror=netix&download=
- https://sourceforge.net/projects/gnuwin32/files/bison/2.4.1/bison-2.4.1-setup.exe/download?use_mirror=netix
- https://sourceforge.net/projects/orwelldevcpp/

2. Ajouter les chemins d'installation de Flex et Bison à votre variable d'environnement **PATH**.

3. Exécuter les commandes suivantes :

> flex - - version **(pour vérifier la disponibilité de Flex sur votre machine)**

> bison - - version **(pour vérifier la disponibilité de Bison sur votre machine)**

> gcc - - version **(pour vérifier la disponibilité de Dev C++ sur votre machine)**

IV. Utilisation de Flex

Flex est normalement utilisé pour diviser un flux de caractères en jetons. Il prend comme entrée une spécification qui associe des expressions régulières avec des actions. À partir de cette spécification, Flex construit une fonction implémentant un automate fini déterministe (AFD) qui reconnaît les expressions régulières dans le temps linéaire. Au moment de l'exécution, lorsqu'une expression régulière est appariée, son action associée est exécutée. L'outil **Flex** permet de générer le code source en C de l'analyseur lexical à partir du fichier de spécification qui lui est fourni en paramètre. Le fichier généré s'appelle par défaut ***« lex.yy.c »***
La commande pour générer le code C de l'analyseur est : ***flex*** *nom_fichier.l*

Il faut compiler ensuite le code source pour obtenir un analyseur lexical exécutable. Pour ce faire, on utilise la commande : **gcc lex.yy.c -o** nomfichier.exe

Si le compilateur gcc produit un message d'erreur relatif à l'absence de la fonction yywrap(), ajouter une option %option noyywrap en en-tête du fichier .l, ou compiler le fichier lex.yy.c avec l'option de liaison -lfl.

Le format d'un fichier de spécifications est le suivant :

%{
déclarations de variable et inclusion de bibliothèques en C
%}
définitions :
<identificateur><expression régulière>
...
%%
règles :
<expression régulière> {<commandes en C>}
%%
<programme principal et fonctions en C>

V. TESTER FLEX

Etape1 : Créer un dossier nommé « **tp_compilation** » sous C:\

Etape2 : Ecrire le code et l'enregistrer sous l'extension « **TP1.l** » dans le dossier «**tp_compilation** ».

```
%{
int nbVoyelles, nbConsonnes, nbPonct ;
%}
%option noyywrap
consonne [b-df-hj-np-xz]
ponctuation [, ;:?!\,]
%%
[aeiouy] nbVoyelles++ ;
{consonne} nbConsonnes++ ;
{ponctuation} nbPonct++ ;
.|\n //ne rien faire
%%
main( ) {
nbVoyelles = nbConsonnes = nbPonct = 0 ;
yylex() ;
printf("Il y a %d voyelles, %d consonnes, %d signes de ponctuations.\n ",
nbVoyelles, nbConsonnes, nbPonct) ;
}
```

Etape 3 : Ecrire le texte suivant dans un fichier texte « test.txt » et l'enregistrer dans le même dossier « tp_compilation ».

```
Hello Dear …,
Welcome to the flex & bison course!!
This text will be used to test your lexical parser
```

Etape 4 : Ouvrir l'invite de commandes
Etape 5 : Exécuter la commande suivante

```
> cd tp_compilation
```

Etape 6 : Générer le fichier « **lex.yy** » avec la commande suivante :

```
> flex TP1.l
```

*Le fichier généré est d'extension « **.c** », c'est un code source écrit en langage C.*

Etape 7 : Générer le fichier « **executable.exe** » avec la commande suivante :

```
> gcc lex.yy.c -o TP1Exe.exe
```

Etape 8 : Utiliser la commande suivante pour exécuter l'exécutable (généré à l'étape 7) en lui donnant comme input le fichier texte de test « **test.txt** » (créé à l'étape 3).

```
> TP1Exe < test.txt
```

Exercice d'application 1 :

1. Écrire une spécification Flex « TP2.l » qui permet de reconnaître dans un fichier texte « test2.txt »:
   - Tous les mots qui commencent par une majuscule.
   - Tous les mots qui se terminent par "*er*".
   - Tous les mots qui commencent par "*j*" et se terminent par *"r"*.
   - Pour tout autre caractère ou retour à la ligne ne rien faire.

On désire obtenir un affichage similaire à l'exemple suivant :

- Contenu du fichier « test2.txt » :

```
En nuit, manger Et marcher.
En jour, manger et aller dormir.
```

- Le résultat de l'affichage doit être comme suit :

```
Mot qui commence par une majuscule: En
Mot qui se termine par er: manger
Mot qui commence par une majuscule: Et
Mot qui se termine par er: marcher
Mot qui commence par une majuscule: En
Mot qui commence par j et se termine par r: jour
Mot qui se termine par er: manger
Mot qui se termine par er: aller
```

2. Quelles sont les commandes nécessaires pour compiler et exécuter cet analyseur ?

Exercice d'application 2 :

1. Écrire une spécification Flex « TP3.l » qui permet de reconnaître dans un fichier texte « test3.txt »:
   - Toutes les chaînes de "0" et de "1" qui contiennent le facteur "011".
   - Toutes les chaînes de "0" et de "1" qui se terminent par "00".
   - Toutes les chaînes de "0" et de "1" qui commencent par "1" et se terminent par "10".
   - Pour tout autre caractère ou retour à la ligne ne rien faire.

On désire obtenir un affichage similaire à l'exemple suivant :

- Contenu du fichier « test3.txt » :

```
1100111111
aaaaa
11111100
bbbbb
10101010
```

- Le résultat de l'affichage doit être comme suit :

```
chaine qui contient le facteur 011: 1100111111
chaine qui se termine par 00: 11111100
chaine qui commence par 1 et se termine par 10: 10101010
```

2. Quelles sont les commandes nécessaires pour compiler et exécuter cet analyseur ?