JUNIA ISEN

CIR 3

TP / Théorie des langages

TP N° 5

Prérequis (les mêmes que pour les TP 3 et 4)

Installer cygwin (https://cygwin.com/install.html) en incluant les packages suivants :

- make (The GNU version of 'make' utility)
- gcc-g++ (GNU Compiler Collection C++)
- flex (A fast lexical analyzer generator)
- bison (GNU yacc-compatible parser generator)

Alternative : Ce TP peut aussi être l'occasion de tester le nouveau WSL (Windows Subsystem for Linux) et qui facilite l'installation et l'utilisation des outils cités précédemment sur Windows.
https://docs.microsoft.com/en-us/windows/wsl/

Vous pouvez toujours tester vos expressions régulières sur les sites :

- https://regexr.com/
- https://regex101.com/
- https://www.regexpal.com

Documentation :

Récupérer les manuels de « flex » et « bison » depuis Teams

Exercice 1 :

A partir du tutoriel vu en cours (Calculette), récupérer le projet et l'enrichir pour :

- Analyser un fichier source (la calculette actuelle utilise le flux clavier)
- Introduire plus de fonctions mathématiques (sin, tan, …)

Exercice 2 : Table de symbole

Dans la suite de l'exercice précédent, le but de cet exercice est d'introduire les variables :

Alpha = 5
Pi = 3.14
Cos (2*Pi*Alpha)

Pour cela il faut :

- Déclarer un structure pour la table de symbole (Une map <string,double>) pour stocker le nom de la variable et sa valeur.
- Déclarer la règle de production pour l'affectation (ex. VAR '=' expression) qui permet de donner une valeur à la variable
- Déclarer la ou les règles de productions qui permettent d'utiliser une variable dans une expression.
- Sortir un warning lorsqu'une variable est utilisée sans être initialisée. Une valeur par défaut de 0 lui sera affectée.

Pour que les **$$, $1 , $2, …** puissent stocker à la fois la valeur d'un double ou le nom d'un variable, nous ferons appel au type **%union** expliqué page 59 du manuel bison.

Cet opérateur transforme les **$n** en structures qui peuvent être accessible avec un « . » :

Exemple : **$1.valeur** ou **$1.nom**

Exercice 3 : Grammaire d'un langage de programmation

Développer une grammaire pour vérifier syntaxiquement :

- Une boucle de type for.
- Une condition (if then else) avec des expressions booléennes (and, or, égalité, !=, >=, <, …)

Il n'est pas nécessaire d'exécuter le code, le but étant juste de détecter les erreurs syntaxiques.