TP 2 – Introduction à Bison

Introduction

GNU Bison permet de générer des analyseurs syntaxiques LALR(1). Il est alors possible de définir la spécification d'un langage non*contextuelle en bison pour réaliser un compilateur pour celui-ci. Flex est souvent utilisé pour définir les différents lexèmes qui pourront être utilisés par Bison. Un lexème correspond à un terme identifiable reconnue par Flex à l'aide d'une expression régulière, par exemple un entier, un réel, une constante, etc... Avec Bison comme pour Flex nous allons définir des règles que nous allons interpréter, les règles ne correspondrons cette fois-ci pas à un terme mais à une suite de termes. (La documentation du logiciel `bison` est disponible en utilisant les commandes `man bison` et `info bison`. Bien sûr, vous avez aussi la documentation donnée en cours, en TP et sur internet à cette URL `https://www.gnu.org/software/bison/manual/html_node/`.)

Calculatrice V1

Nous souhaitons dans un premier temps réaliser une calculatrice basique permettant l'utilisation uniquement d'entiers et des opérateurs + et ×. En ajoutant la gestion de priorité d'opération pour les calculs avec ou sans parenthèses.

Écrire un analyseur syntaxique `bison` qui permet de réaliser une calculatrice basique. Dans la première version, l'utilisateur doit pouvoir taper :

- Une seule expression par ligne (e.g. $4 + 5$)
- Des expressions avec des entiers uniquement (pas de réels)
- Les opérateurs + et ×
- Le signe - pour les nombres négatifs (utilisation du moins unaire)
- Des parenthèses pour donner la priorité au calcul (e.g. le programme doit afficher 8 lorsqu'on tape "$(2 + 2) \times 2$")
- Le mot clé "fin" est utilisé pour quitter le programme
- Attention à la gestion des erreurs

Calculatrice V1.2

Vous pouvez maintenant ajouter à votre calculatrice :

- D'autres opérateurs : soustraction, division, racine carrée, puissance, ...
- La gestion des nombres réels.

Structure d'un fichier Bison (.yy)

Les fichiers Bison sont divisés en trois sections, comme les fichiers Lex, séparées par des lignes commençant par deux signes %. Les sections sont les suivantes :

— La section définition qui permet d'ajouter des déclarations en C/C++ et des déclaration en bison.

— La déclaration de code C/C++ nécessaire est encadrée par :

```
%code require{
    //Declaration des classes ou du code necessaire au fonctionnement du parser
}
```

— L'inclusion de code C/C++ est encadrée par :

```
%code{
    //Ajout des include
    //Definition possible de fonctions C/C++
}
```

— La section définition qui permet de définir les différents symboles terminaux et non-terminaux qui seront utilisés dans la grammaire, les différentes options liées aux symboles (précédence par exemple). Les symboles peuvent être liés aux tokens définis dans le fichier Lex ou simplement typés dans le fichier yy. Exemple de définition :

```
%token              NL //retour    la ligne
%token              END // Fin de programme
%token <int>        NUMBER // Nombre entier
%type <int>         expression // Representation d'une expression arithmetique
//Symbole ne provenant pas du fichier Lex
```

— La section règles qui permet de définir les règles bison décrivant notre grammaire. On utilise ici les symboles terminaux et non-terminaux définis dans la section précédente et on exécute un code C/C++ associé. La structure de cette section est la suivante :

```
programme: //Symbole representant le resultat de la regle
    NUMBER NL {
        std::cout << "nombre : " << $1 << std::endl;
    } programme
    | END NL {
        YYACCEPT;
    }
```

— La section code C/C++, contient des instructions et fonctions en C/C++ qui seront copiées dans le fichier source généré. Ces instructions peuvent contenir du code qui sera appelé par les règles de la section précédente. Lorsque les programmes sont d'une taille importante il est préférable d'écrire le code C/C++ dans d'autres fichiers séparés et de les linker lors de la compilation. C'est le cas dans notre exemple avec les fichier scanner.hh, driver.hh et driver.cc qui contiennent les définitions des classes Scanner et Driver.

Il est nécessaire pour chaque règle du fichier Lex de définir un token en retour de la fonction afin que celui-ci soit accessible dans le fichier bison.

L'accès à la valeur des symboles utilisés dans les règles se fait par le signe $ suivi du numéro du symbole dans la règle. $$ représente le résultat de la règle.

La syntaxe pour la grammaire bison est accessible à l'URL suivante :
`http://dinosaur.compilertools.net/bison/bison_6.html#SEC41`