Projet de Traduction

Licence d'informatique

—2014-2015—

Le but du projet est d'écrire un compilateur en utilisant les outils `flex` et `bison`.

Le langage source sera un petit langage de programmation appelé TPC, qui *ressemble* à un sous-ensemble du langage C. Le langage cible est le langage d'une machine virtuelle dont la description a été vue précédemment. Vous vérifierez le résultat de la compilation d'un programme en faisant exécuter le code obtenu par la machine virtuelle qui vous est fournie.

Une version toujours à jour est disponible à l'adresse suivante :

`http://www-igm.univ-mlv.fr/~cmorvan/ens/fichiers/compilation/sujetProjet.pdf`

1 Définition informelle du langage source

Un programme TPC est une suite de fonctions. Chaque fonction est constituée de déclarations de constantes et variables (locales à la fonction), et d'une suite d'instructions. Les fonctions peuvent être récursives. Il peut y avoir des constantes et variables de portée globale. Elles sont alors déclarées avant les fonctions.

Tout programme doit comporter la fonction particulière `main` qui est toujours la dernière fonction du programme et celle par laquelle commence l'exécution. Les types de base du langage sont le type `entier` et le type `caractere`. Le mot clé `void` est utilisé pour indiquer qu'une fonction ne fournit pas de résultat ou n'a pas d'arguments. Les arguments d'une fonction sont transmis par valeur lorsqu'ils sont d'un type de base, et par référence lorsque ce sont des tableaux.

Le langage utilisera le mot-clé *print* pour afficher un entier ou un caractère ; le mot-clé *readch* permet d'obtenir un caractère lu au clavier (*read* permet d'obtenir un entier).

2 Définition des éléments lexicaux

Les identificateurs sont constitués d'une lettre, suivie éventuellement de lettres, chiffres, symbole souligné ("_"). Vous pouvez fixer une longueur maximale pour un identificateur. Il y a distinction entre majuscule et minuscule. Les mots clés comme *if*, *else*, *return*, etc., doivent être écrits en minuscule. Ils sont reconnus par l'analyseur lexical et ne peuvent pas être utilisés comme identificateurs.

Les entiers non signés sont des suites de chiffres.

Les caractères littéraux, dans le programme sont délimités par le symbole ', dans le style du C.

Les commentaires sont délimités par $/*$ et $*/$ et ne peuvent pas être imbriqués.

Les différents opérateurs et autres éléments lexicaux sont :

=	: opérateur d'affectation
+	: addition ou plus unaire
−	: soustraction ou moins unaire
*	: multiplication
/ et %	: division et reste de la division entière
!	: négation (booléenne)
==, ! =, <, >, <=, >=	: les opérateurs de comparaisons
&&, ||	: les opérateurs booléens
; et ,	: le point virgule et la virgule
(,), {, }, [et]	: les parenthèses, les accolades et les crochets

Chacun de ces éléments sera identifié par l'analyse lexicale qui devra produire une erreur pour tout élément ne faisant pas partie du lexique du langage.

3 Notations et sémantique du langage

Dans ce qui suit,

- **CARACTERE**, et **NUM** désignent respectivement un caractère littérale et un entier non signé ;
- **IDENT** désignent un identificateur ;
- **TYPE** désigne un nom de type qui peut être *entier* et *caractere* ;
- **COMP** désigne un quelconque des opérateurs de comparaisons ;
- **ADDSUB** désigne les opérateurs '+' ou '−' (binaire ou unaire) ;
- **DIVSTAR** désigne les opérateurs '∗', '/' ou '%' ;
- **BOPE** désigne les deux opérateurs booléens '&&', '||' ;
- **NEGATION** l'opérateur de négation booléenne, '!' ;
- Les mots clés sont notés par des unités lexicales qui leur sont identiques à la casse près.
- =, **;**, **,**, (,), {, }, [et] sont respectivement notés par **EGAL**, **PV**, **VRG**, **LPAR**, **RPAR**, **LACC**, **RACC**, **LSQB**, **RSQB**.

Tous les opérateurs binaires (sauf l'affectation) sont associatifs à gauche et l'opérateur unaire est associatif à droite. Ceux désignés par un même nom ont même niveau de priorité. L'ordre croissant des priorités est :

1. **BOPE**
2. **COMP**
3. **ADDSUB** (binaire)
4. **DIVSTAR**
5. **NEGATION**
6. **ADDSUB** (unaire)

Tout identificateur utilisé dans un programme doit être déclaré avant son utilisation (dans la partie de déclaration appropriée). La sémantique des instructions du langage est celle habituelle ou se déduit facilement de ce qui précède. L'instruction nulle est notée '**;**'.

Manipulation des tableaux Pour pouvoir manipuler les tableaux il y a deux choses indispensables :

- Pouvoir réserver la mémoire nécessaire.
- Être capable d'accéder convenablement au contenu d'une case.

Ainsi il est nécessaire d'attacher à chaque identifiant un attribut qui spécifie la taille du tableau, et d'effectuer la réservation correspondante en début de programme.

4 Grammaire du langage TPC

Prog	: DeclConst DeclVarPuisFonct DeclMain
DeclConst	: DeclConst **CONST** ListConst **PV**
	| ε
ListConst	: ListConst **VRG IDENT EGAL** Litteral
	| **IDENT EGAL** Litteral
Litteral	: NombreSigne
	| **CARACTERE**
NombreSigne	: **NUM**
	| **ADDSUB NUM**
DeclVarPuisFonct	: **TYPE** ListVar **PV** DeclVarPuisFonct
	| DeclFonct
	| ε
ListVar	: ListVar **VRG** Ident
	| Ident
Ident	: **IDENT** Tab
Tab	: Tab **LSQB ENTIER RSQB**
	| ε
DeclMain	: EnTeteMain Corps
EnTeteMain	: **MAIN LPAR RPAR**

DeclFonct : DeclFonct DeclUneFonct
| DeclUneFonct
DeclUneFonct : EnTeteFonct Corps
EnTeteFonct : **TYPE IDENT LPAR** Parametres **RPAR**
| **VOID IDENT LPAR** Parametres **RPAR**
Parametres : **VOID**
| ListTypVar
ListTypVar : ListTypVar **VRG TYPE IDENT**
| **TYPE IDENT**
Corps : **LACC** DeclConst DeclVar SuiteInstr **RACC**
DeclVar : DeclVar **TYPE** ListVar **PV**
| ε
SuiteInstr : SuiteInstr Instr
| ϵ
InstrComp : **LACC** SuiteInstr **RACC**
Instr : LValue **EGAL** Exp **PV**
| **IF LPAR** Exp **RPAR** Instr
| **IF LPAR** Exp **RPAR** Instr **ELSE** Instr
| **WHILE LPAR** Exp **RPAR** Instr
| **RETURN** Exp **PV**
| **RETURN PV**
| **IDENT LPAR** Arguments **RPAR PV**
| **READ LPAR IDENT RPAR PV**
| **READCH LPAR IDENT RPAR PV**
| **PRINT LPAR** Exp **RPAR PV**
| **PV**
| InstrComp
Arguments : ListExp
| ε
LValue : **IDENT** TabExp
TabExp : TabExp **LSQB** Exp **RSQB**
| ε
ListExp : ListExp **VRG** Exp
| Exp
Exp : Exp **ADDSUB** Exp
| Exp **DIVSTAR** Exp
| Exp **COMP** Exp
| **ADDSUB** Exp
| Exp **BOPE** Exp
| **NEGATION** Exp
| **LPAR** Exp **RPAR**
| LValue
| **NUM**
| **CARACTERE**
| **IDENT LPAR** Arguments **RPAR**

5 Complément sur les expressions

On note tout d'abord l'absence de type booléen. Toute valeur entière pourra être acceptée comme valeur booléenne, avec la convention que l'entier 0 représente faux et tout autre entier vrai. Par convention, l'opérateur de négation produira l'entier 1 pour l'argument 0.

D'autre part, la grammaires est particulièrement souple pour les expressions. Pour autant de nombreuses expressions valides pour cette grammaire n'ont pas de sens pour le langage. La vérification de type devra valider la correction. Par exemple, les caractères n'ont ni valeur booléenne, ni valeur entière, on ne peut pas leur ajouter d'entier ou les multiplier.

6 Travail demandé

Écrire un compilateur de ce langage en utilisant `flex` pour l'analyse lexicale et `bison` pour l'analyse syntaxique et la traduction. Vous pouvez modifier la grammaire pour lever les conflits d'analyse ou faciliter la traduction, mais ces modifications ne doivent pas affecter le langage engendré. Par contre, il n'est absolument pas interdit d'enrichir le langage TPC !

Il est conseillé de d'abord écrire un compilateur qui ne traite pas les tableaux. Dans un second temps, il est conseillé d'introduire les tableaux à une dimension, puis enfin les tableaux multidimensionnels.

Une amélioration intéressante (encouragée) du langage consiste à permettre de passer des tableaux en *paramètres de fonction*. Dans ce cas, on encourage fortement à opter pour un passage par adresse. Ceci induira une complexité mais évite d'avoir à réaliser la copie du tableau dans la pile au moment de l'appel. Vous ne traiterez ce problème que si vous avez un compilateur pour TPC fonctionnel, et vous décrirez dans votre documentation les choix que vous avez effectués et les difficultés que vous avez rencontré.

L'exécution de votre compilateur sera :

tcompil prog.tpc [-o]

Si l'option "**-o**" est présente, le résultat de la compilation sera placé dans le fichier **prog.vm** (même nom que le fichier d'entrée, seul l'extension change), sinon le résultat sera affiché à l'écran.

Votre projet devra être rendu par courriel à votre chargé de travaux dirigés (sujet du courriel : Projet Traduction L3), sous la forme d'une archive tar compressée de nom "ProjetTraductionL3_NOM1_NOM2.tar.gz", qui, au désarchivage, créé un répertoire "ProjetTraductionL3_NOM1_NOM2" contenant le projet. Il devra contenir un makefile, et être organisé correctement (un répertoire pour les sources, un autre pour la documentation, un autre pour les exemples, etc). La date de rendu est le dimanche 31 mai 2015 à minuit au plus tard.