Projet de Traduction

Licence d'informatique

—2015-2016—

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 de la machine virtuelle de l'UPEM. Vous vérifierez le résultat de la compilation d'un programme en faisant exécuter le code obtenu par la machine virtuelle.

Cette description du projet est disponible à l'adresse suivante :

`http://igm.univ-mlv.fr/~laporte/compil/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` 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.

Le langage TPC utilise `print` pour afficher un entier ou un caractère. Le mot-clé `readch` permet d'obtenir un caractère lu au clavier ; `read` permet de lire 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 minuscules. 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 comparaison
`&&`, `||`	: 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 littéral et un entier non signé ;
– **IDENT** désigne un identificateur ;
– **TYPE** désigne un nom de type qui peut être `entier` et `caractere` ;
– **COMP** désigne un quelconque des opérateurs de comparaison ;
– **ADDSUB** désigne les opérateurs '`+`' et '`-`' (binaire ou unaire) ;
– **DIVSTAR** désigne les opérateurs '`*`', '`/`' et '`%`' ;
– **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 **EGAL**, **PV**, **VRG**, **LPAR**, **RPAR**, **LCUR**, **RCUR**, **LSQB** et **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. L'opérateur ternaire dans le style Python : exp `if (`exp`) else` exp
2. **BOPE**
3. **COMP**
4. **ADDSUB** binaire
5. **DIVSTAR**
6. **NEGATION**
7. **ADDSUB** unaire

Tout identificateur utilisé dans un programme doit être déclaré avant son utilisation et dans la partie de déclaration appropriée. La sémantique des instructions du langage est la sémantique 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 : DeclConsts DeclVars DeclFoncts ;
DeclConsts : DeclConsts **CONST** ListConst **PV**
| ;
ListConst : ListConst **VRG IDENT EGAL** Litteral
| **IDENT EGAL** Litteral ;
Litteral : NombreSigne
| **CARACTERE** ;
NombreSigne : **NUM**
| **ADDSUB NUM** ;
DeclVars : DeclVars **TYPE** Declarateurs **PV**
| ;
Declarateurs : Declarateurs **VRG** Declarateur
| Declarateur ;
Declarateur : **IDENT**
| **IDENT LSQB NUM RSQB** ;
DeclFoncts : DeclFoncts DeclFonct
| DeclFonct ;
DeclFonct : EnTeteFonct Corps ;
EnTeteFonct : **TYPE IDENT LPAR** Parametres **RPAR**
| **VOID IDENT LPAR** Parametres **RPAR** ;
Parametres : **VOID**

```
                | ListTypVar   ;
ListTypVar      : ListTypVar VRG TYPE IDENT
                | TYPE IDENT   ;
Corps           : LCUR DeclConsts DeclVars SuiteInstr RCUR   ;
SuiteInstr      : SuiteInstr Instr
                |  ;
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
                | Bloc   ;
Bloc            : LCUR SuiteInstr RCUR   ;
Arguments       : ListExp
                |  ;
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
                | Exp IF LPAR Exp RPAR ELSE Exp
                | IDENT LPAR Arguments RPAR   ;
LValue          : IDENT
                | IDENT LSQB Exp RSQB   ;
```

5 Complément sur les expressions

Il n'y a pas de type booléen. Toute valeur entière peut être interprétée comme booléenne, avec la convention que l'entier 0 représente "faux" et tout autre entier "vrai". L'opérateur de négation produit comme résultat l'entier 1 quand on l'applique à l'argument 0.

La grammaire est particulièrement souple pour les expressions, mais certaines expressions syntaxiquement valides pour cette grammaire n'ont pas de sens pour le langage. Par exemple, les caractères n'ont pas de valeur booléenne, on ne peut pas leur appliquer d'opérateur logique. La vérification de type devra valider la correction. De même, la grammaire n'impose pas que le programme comporte la fonction `main`, c'est donc l'analyse sémantique qui devra vérifier sa présence.

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 vos modifications ne peuvent affecter le langage engendré que si cela enrichit le langage TPC. Vous décrirez dans votre documentation vos choix et les difficultés que vous avez rencontrées.

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

Quand vous aurez un compilateur fonctionnel pour TPC, vous êtes encouragé à apporter une amélioration intéressante au langage : permettre de passer des tableaux en *paramètres de fonction*. Dans ce cas, passez-les par référence (au contraire des types de base, qui sont passés par valeur) : ceci induira une complexité mais évitera d'avoir à copier le tableau dans la pile au moment de l'appel.

La commande pour exécuter 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, seule l'extension change), sinon le résultat sera affiché à l'écran.

Déposez votre projet sur la plateforme elearning dans la zone prévue à cet effet, sous la forme d'une archive tar compressée de nom "ProjetTraductionL3_NOM1_NOM2.tar.gz", qui, au désarchivage, crée 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... La date de rendu est le dimanche 5 juin 2016 à 23h55 au plus tard.