Projet de Traduction

Licence d'informatique

—2013-2014—

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 en faisant exécuter le code obtenu par la machine virtuelle qui vous est fournie.

1 Définition informelle du langage source

Un programme TPC est une suite de fonctions. Chaque fonction est constituée 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 seuls types admis dans le langage sont le type `entier`, et le type `pointeur` (sur `entier` ou sur `pointeur`). Le mot clé `void` est utilisé pour indiquer qu'une fonction ne retourne pas de valeur ou n'a pas d'arguments. Les arguments d'une fonction, lorsqu'ils sont présents, sont transmis uniquement "par valeur" (naturellement une fonction peut avoir des pointeurs comme paramètres).

Le programme utilisera deux mot-clés *malloc* et *free* (utilisés comme des fonctions) pour obtenir des zone de mémoire dont la gestion sera faite par du code produit par le compilateur. Si la mémoire disponible n'est pas suffisante, le programme s'arrête avec un message d'erreur.

2 Définition des éléments lexicaux

Les identificateurs sont constitués d'une lettre, suivie éventuellement de lettres ou de chiffres ou du 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 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

==, ! =, <, >, <=, >= : les opérateurs de comparaisons
; et , : le point virgule et la virgule
(,), {, }, [et] : les parenthèses, les accolades et les crochets

Chacun de ces éléments sera identifié par le lexeur 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,
- **IDENT**, et **NUM** désignent respectivement un identificateur et un entier non signé ;
- **COMP** désigne un quelconque des opérateurs de comparaisons ;
- **ADDSUB** désigne les opérateurs '+' ou '−' (binaire ou unaire) ;
- **STAR** désigne l'opérateurs '∗' (binaire ou unaire) ;
- **DIV** désigne les opérateurs '/' ou '%' ;
- **ADR** désigne l'opérateur &
- Les mots clés sont notés par des tokens 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 les opérateurs unaires sont associatifs à droite. Ceux désignés par un même nom ont même niveau de priorité. L'ordre croissant des priorités est :

1. **COMP**
2. **ADDSUB** (binaire)
3. **DIV, STAR**
4. **ADDSUB** (unaire)
5. **STAR, ADR** (unaires)

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 pointeurs Les pointeurs sont manipulés de façon similaire à celle du C.

L'instruction *malloc* s'utilise avec un paramètre représentant la taille demandée. En cas d'échec le programme s'achève brutalement. En cas de succès (assez de mémoire disponible), la valeur de retour est un pointeur sur le début de l'espace mémoire concerné.

L'instruction *free* est utilisée avec comme paramètre un pointeur. Ce dernier devra impérativement correspondre à une valeur obtenue par *malloc*. Dans le cas contraire le programme généré par votre compilateur est autorisé à planter !

Un pointeur qui ne contient pas l'adresse d'une variable mais celle du début d'une zone mémoire allouée peut être manipulé comme un tableau en utilisant les crochets **LSQB** et **RSQB** encadrant l'indice de l'élément de la zone mémoire concerné. Ainsi $t[i]$ désigne le i-ème élément de la zone mémoire allouée à l'adresse pointée par t.

Indication : en interne, il vous est conseiller de consituer un *tas* de taille fixe au début de votre pile mémoire. Vous pouvez également placer à deux adresses convenue l'espace disponible dans le tas et la première adresse libre. À chaque allocation réussie, placer à la première adresse libre la taille de l'allocation, et renvoyer cette adresse plus 1 comme valeur pour *malloc* (ainsi chaque allocation de taille n consomme $n + 1$).

4 Grammaire du langage TPC

Prog	:	DeclConst DeclVar DeclFonct DeclMain
DeclConst	:	DeclConst **CONST** ListConst **PV**
	|	ε
ListConst	:	ListConst **VRG IDENT EGAL** NombreSigne
	|	**IDENT EGAL** NombreSigne
NombreSigne	:	**NUM**
	|	**ADDSUB NUM**
DeclVar	:	DeclVar **VAR** ListVar **PV**
	|	ε
ListVar	:	ListVar **VRG** Variable
	|	Variable
Variable	:	**STAR** Variable
	|	**IDENT**
DeclMain	:	EnTeteMain Corps
EnTeteMain	:	**MAIN LPAR RPAR**
DeclFonct	:	DeclFonct DeclUneFonct
	|	ε
DeclUneFonct	:	EnTeteFonct Corps
EnTeteFonct	:	Type **IDENT LPAR** Parametres **RPAR**
Type	:	**ENTIER**
	|	**VOID**
Parametres	:	**VOID**
	|	ListVar
Corps	:	**LACC** DeclConst DeclVar SuiteInstr **RACC**
SuiteInstr	:	SuiteInstr Instr
	|	ϵ
InstrComp	:	**LACC** SuiteInstr **RACC**
Instr	:	**IDENT EGAL** Exp **PV**
	|	**STAR IDENT EGAL** Exp **PV**
	|	**IDENT LSQB** Exp **RSQB EGAL** Exp **PV**
	|	**IDENT EGAL MALLOC LPAR NUM RPAR PV**
	|	**FREE LPAR** Exp **RPAR 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** LValue **RPAR PV**
| **PRINT LPAR** Exp **RPAR PV**
| **PV**
| InstrComp

Arguments : ListExp
| ε

LValue : **IDENT**
| **STAR IDENT**
| **IDENT LSQB** Exp **RSQB**

ListExp : ListExp **VRG** Exp
| Exp

Exp : Exp **ADDSUB** Exp
| Exp **STAR** Exp
| Exp **DIV** Exp
| Exp **COMP** Exp
| **ADDSUB** Exp
| **LPAR** Exp **RPAR**
| LValue
| **ADR** LValue
| **NUM**
| **IDENT LPAR** Arguments **RPAR**

5 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 gère pas les pointeurs. Ensuite introduisez les pointeurs sans mettre en place *malloc* et *free*. Enfin, mettez en place la gestion du tas.

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'entré, 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, ainsi qu'un rapport expliquant le fonctionnement de votre projet et les différents choix que vous avez fait et être organisé correctement (un répertoire pour les sources, un autre pour la documentation, un autre pour les exemples, etc). Votre code devra être largement commenté pour faciliter sa compréhension. La date de rendu est le dimanche 25 mai 2014 à minuit au plus tard.