# Projet de Compilation

## Licence d'informatique

*—2017-2018—*

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

Le langage source est un petit langage de programmation appelé TPC, qui ressemble à un sous-ensemble du langage C. Le langage cible est l'assembleur `nasm` 64 bits. Vous vérifierez le résultat de la compilation d'un programme en exécutant le code obtenu.

Les dates limite de rendu sont :

- le dimanche 25 février 2018 à 23h55 au plus tard pour la partie analyse syntaxique [(1)] (1/4 de la note de projet) ;
- le mrecredi 6 juin 2018 à 23h55 au plus tard pour le compilateur complet.

## 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` (entier signé) 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 `readc` permet d'obtenir un caractère lu au clavier ; `reade` 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 éléments lexicaux pour les constantes numériques 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.

(1). La partie analyse syntaxique consiste uniquement à détecter les erreurs lexicales et syntaxiques et à afficher les messages d'erreur correspondants. Vous n'avez pas à vous occuper de calculs d'attributs. L'analyseur syntaxique créé avec le Makefile doit être nommé `tcompil`. La commande `./tcompil <` *test*`.tpc` doit tester l'exécutable. Votre anayseur syntaxique doit renvoyer 0 si et seulement si *test*`.tpc` est syntaxiquement correct. Dans les deux cas, votre analyseur doit se terminer sans plantage. S'il n'y a aucune erreur dans *test*`.tpc`, il doit se terminer sans afficher aucun message.

# 3 Notations et sémantique du langage

Dans ce qui suit,

- **CARACTERE** et **NUM** désignent respectivement un caractère littéral et une constante numérique ;
- **IDENT** désigne un identificateur ;
- **TYPE** désigne un nom de type qui peut être `entier` et `caractere` ;
- **EQ** désigne les opérateurs d'égalité ('`==`') et d'inégalité ('`!=`') ;
- **ORDER** désigne les opérateurs de comparaison '`<`', '`<=`', '`>`' et '`>=`' ;
- **ADDSUB** désigne les opérateurs '`+`' et '`-`' (binaire ou unaire) ;
- **DIVSTAR** désigne les opérateurs '`*`', '`/`' et '`%`' ;
- **OR** et **AND** désignent les deux opérateurs booléens '`||`' et '`&&`'.
- Les mots clés sont notés par des unités lexicales qui leur sont identiques à la casse près.

L'instruction nulle est notée '**;**'.

# 4 Grammaire du langage TPC

Prog : DeclConsts DeclVars DeclFoncts ;
DeclConsts : DeclConsts **CONST** ListConst **';'**
| ;
ListConst : ListConst **','** **IDENT** **'='** Litteral
| **IDENT** **'='** Litteral ;
Litteral : NombreSigne
| **CARACTERE** ;
NombreSigne : **NUM**
| **ADDSUB** **NUM** ;
DeclVars : DeclVars **TYPE** Declarateurs **';'**
| ;
Declarateurs : Declarateurs **','** Declarateur
| Declarateur ;
Declarateur : **IDENT**
| **IDENT** **'['** **NUM** **']'** ;
DeclFoncts : DeclFoncts DeclFonct
| DeclFonct ;
DeclFonct : EnTeteFonct Corps ;
EnTeteFonct : **TYPE** **IDENT** **'('** Parametres **')'**
| **VOID** **IDENT** **'('** Parametres **')'** ;
Parametres : **VOID**
| ListTypVar ;
ListTypVar : ListTypVar **','** **TYPE** **IDENT**
| **TYPE** **IDENT** ;
Corps : '{' DeclConsts DeclVars SuiteInstr '}' ;
SuiteInstr : SuiteInstr Instr
| ;
Instr : Exp **';'**
| **';'**
| **RETURN** Exp **';'**
| **RETURN** **';'**
| **READE** **'('** **IDENT** **')'** **';'**
| **READC** **'('** **IDENT** **')'** **';'**
| **PRINT** **'('** Exp **')'** **';'**
| **IF** **'('** Exp **')'** Instr
| **IF** **'('** Exp **')'** Instr **ELSE** Instr
| **WHILE** **'('** Exp **')'** Instr
| '{' SuiteInstr '}' ;
Exp : LValue **'='** Exp
| EB ;

```
EB          : EB OR TB
            | TB   ;
TB          : TB AND FB
            | FB   ;
FB          : FB EQ M
            | M   ;
M           : M ORDER E
            | E   ;
E           : E ADDSUB T
            | T    ;
T           : T DIVSTAR F
            | F    ;
F           : ADDSUB F
            | '!' F
            | '(' Exp ')'
            | LValue
            | NUM
            | CARACTERE
            | IDENT '(' Arguments ')'   ;
LValue      : IDENT
            | IDENT '[' Exp ']'   ;
Arguments   : ListExp
            |  ;
ListExp     : ListExp ',' Exp
            | Exp   ;
```

# 5 Sémantique

La sémantique de la plupart des expressions et instructions du langage est la sémantique habituelle.

Tout identificateur utilisé dans un programme doit être déclaré avant son utilisation et dans la partie de déclaration appropriée.

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.

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 que les expressions sont correctes.

De même, la grammaire n'impose pas que le programme comporte la fonction `main`, mais l'analyse sémantique devra vérifier sa présence.

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

# 6 Langage cible

Le langage cible est un sous-ensemble de l'assembleur `nasm` 64 bits. Les commandes autorisées sont :

`mov`, `call`, `ret`, `syscall`,

`add`, `sub`, `idiv`, `imul`,
`and`, `or`, `xor`,
`cmp`, `je`, `jg`, `jne`, `jng`, `jmp`,
`push`, `pop`,
`resb`, `resd`, `resq`, `db`, `dd`, `dq`.

On codera les entiers et les caractères sur 4 octets.

Les registres autorisés sont :

`rax`, `rbx`,
`rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`,
`rsp`, `rbp`

et leurs sous-registres `eax`, `ax`, `ah`, `al`, `ebx`, `bx`, `bh`, `bl`, `ecx`, `cx`, `ch`, `cl`, `edx`, `dx`, `dh`, `dl`, `edi`, `di`, `esi` et `si`.

# 7 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 des 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.

Le répertoire que vous déposerez doit contenir un Makefile nommé `Makefile` ou `makefile` et être organisé correctement : un répertoire pour les sources, un autre pour la documentation, un autre pour les exemples... Le compilateur créé avec le Makefile doit être nommé `tcompil`. La commande suivante doit exécuter votre compilateur :

**./tcompil <** *prog*.**tpc [-o prog.asm]**

Si l'option "**-o**" est présente, le résultat de la compilation sera placé dans le fichier indiqué, sinon le résultat sera affiché à l'écran.

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.

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.