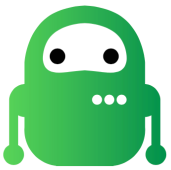
UE Compilation - Projet



1. Définition du langage

Le langage **YAL** est un langage de programmation typé rudimentaire incluant les variables entières, les tableaux, les structures de contrôle élémentaires et les fonctions.

Grammaire

L'axiome de la grammaire est **PROGRAMME**. Les non terminaux sont écrits en majuscules ; les terminaux qui ne sont pas des symboles sont écrits en minuscules et soulignés. La notation $\{\alpha\}*$ signifie que la séquence α peut être répétée un nombre quelconque de fois, éventuellement nul ; la notation $\{\alpha\}+$ signifie que la séquence α peut être répétée un nombre quelconque de fois, jamais nul ; la notation $\{\alpha\}$ signifie que la séquence α est optionnelle.

En **YAL**, les commentaires commencent par la séquence de caractères // et se terminent à la fin de la ligne.

Les terminaux génériques sont constitués de la façon suivante :

- **csteEntiere** est une suite non vide de chiffres décimaux ;
- **idf** est une suite non vide de lettres et de chiffres commençant par une lettre, sans limitation de taille ; la différence entre majuscules et minuscules est significative.

Les mots clés sont écrits en minuscules (exactement comme dans la grammaire) et sont réservés. Par exemple, un programme **YAL** peut contenir un identificateur SI qui ne sera pas confondu avec le mot clé réservé **si**.

PROGRAMME	→	**programme** **idf** **debut** { DECLARATION }* { INSTRUCTION }+ **fin**
DECLARATION	→	DECL_VAR | DECL_FONCTION
DECL_VAR	→	**entier** **idf** **;** | **entier** **[** **EXP** **]** **idf** **;**
DECL_FONCTION	→	**fonction** **idf** PARAMETRES **debut** { DECL_VAR }* { INSTRUCTION }+ **fin**
PARAMETRES	→	**(** **)** | **(** { **entier** **idf** { **;** **entier** **idf** }* } **)**
INSTRUCTION	→	AFFECT | BOUCLE | CONDITION | LIRE | ECRIRE | RETOURNE
AFFECT	→	**idf** **=** EXP **;** | **idf** **[** EXP **]** **=** EXP **;**
BOUCLE	→	**tantque** EXP **repeter** { INSTRUCTION }+ **fintantque**
CONDITION	→	**si** EXP **alors** { INSTRUCTION }* { **sinon** { INSTRUCTION }+ } **finsi**
LIRE	→	**lire** **idf** **;**
RETOURNE	→	**retourne** EXP **;**
ECRIRE	→	**ecrire** EXP **;**
EXP	→	**idf** **.** **longueur** | **idf** **(** PAR_EFF **)**
		| **csteEntiere** | **idf** | **idf** **[** EXP **]**
		| **(** EXP **)** | **non** EXP | **-** EXP | EXP OPER EXP
OPER	→	**+** | **-** | ***** | **>** | **<** | **/** | **==** | **!=** | **et** | **ou**
PAR_EFF	→	{ EXP { **,** EXP }* }

Sémantique

1. Un programme est constitué de déclarations de variables, de fonctions et d'instructions. À l'exécution du programme, chacune de ces instructions est exécutée dans l'ordre d'écriture.

2. La portée d'une déclaration de variable ou de fonction est constituée de l'intégralité de la région qui la contient, moins les régions imbriquées où le même identificateur est déclaré dans le même espace des noms. Une région est délimitée par les mots clés **debut** et **fin**.

3. Les variables sont initialisées à 0.

4. Dans une déclaration de variable (alternative 3), la notation **entier [EXP]** désigne un type tableau à une dimension dont les éléments sont de type **entier** et la taille fixée par la valeur de l'expression. Si la déclaration est dans le programme principal, l'expression doit être constante ; si la déclaration se trouve dans une fonction, l'expression est quelconque.

5. Les doubles déclarations de variables sont interdites. La surcharge des fonctions est autorisée, sous réserve de définir des profils différents, c'est-à-dire avec des nombres de paramètres différents. Une variable et une fonction peuvent porter le même nom ; la différence est faite syntaxiquement.

6. Dans une affectation, le type de l'expression doit être identique ou concorder avec celui de la notation d'accès en partie gauche. Deux types tableau concordent s'ils ont le même nombre d'éléments.

7. Dans un accès à un élément de tableau (notation **idf [EXP]**), l'expression entre crochets est de type entier ; elle désigne le rang de l'élément. Le rang du premier élément est 0.

8. Le résultat des fonctions est **entier.** Dans le corps d'une fonction, l'instruction **retourne** est obligatoire. Son exécution provoque l'arrêt de la fonction en retournant comme résultat la valeur de l'expression.

9. Dans une itération conditionnelle (mot clé **tantque**), l'expression est de type booléen ; elle est obligatoirement évaluée avant toute entrée dans la boucle.

10. Dans une instruction conditionnelle (mot clé **si**), l'expression est de type booléen.

11. La fonction prédéfinie **lire** lit un entier sur l'entrée standard.

12. La fonction prédéfinie **ecrire** écrit, soit un entier, soit un booléen, suivi d'un retour à la ligne. Les booléens sont écrits sous la forme **vrai** ou **faux**.

13. Les opérandes des opérateurs +, - , *, / et - unaire sont entiers, le résultat est entier. Les opérandes des opérateurs **et**, **ou** et **non** sont booléens. Les opérateurs relationnels < et > sont à opérandes entiers et résultat booléen, les opérateurs relationnels == et != sont à opérandes de même type et résultat booléen.

14. Les opérateurs binaires sont tous associatifs gauche-droite. Les priorités des opérateurs sont définies dans l'ordre strictement décroissant suivant : **( )** , **non** , * et / , + et - , < et > , == et != , **et** , **ou**.

15. Dans un appel de fonction, le compilateur choisit le profil de la fonction à partir du nombre de paramètres effectifs. Le passage des paramètres se fait par valeur.

16. La notation **idf.longueur** désigne le nombre d'éléments du tableau de nom **idf**. C'est une expression de type **entier**.

2. Le compilateur YAL

Le compilateur YAL doit être développé en langage Java en utilisant les générateurs d'analyseurs **JFlex** et **JavaCup**. Il génère du code **MIPS**.

De sorte que les tests soient facilement automatisables, il est impératif de respecter les contraintes ci-dessous :

- le compilateur est livré dans l'archive **yal.jar**. Son exécution nécessite exactement un argument : le nom du fichier contenant le texte du programme à compiler, suffixé **.yal**.. Elle provoque la compilation du texte et, si celle-ci se déroule sans erreur, crée un fichier de même préfixe, suffixé **.mips**, contenant le texte cible correspondant.
- l'exécution n'est pas conversationnelle ; en dehors du fichier **.mips** créé, elle doit laisser intact l'environnement de celui qui l'appelle.
- selon le cas, l'exécution produit sur la sortie standard l'un des résultats suivants (et rien d'autre …) :

  ERREUR LEXICALE : no ligne d'erreur : message d'erreur explicite
  ERREUR SYNTAXIQUE : no ligne d'erreur : message d'erreur explicite
  ERREUR SEMANTIQUE : no ligne d'erreur : message d'erreur explicite
  COMPILATION OK

- une erreur lexicale ou syntaxique stoppe la compilation du texte source ; toutes les erreurs sémantiques détectées doivent être signalées.

  Si une erreur se produit lors de l'exécution du code généré, l'exécution s'arrête immédiatement avec l'affichage du message ERREUR EXECUTION sur la sortie standard.

3. Déroulement du projet

Vous travaillerez en groupe de 3. Inscrivez rapidement vos noms dans la feuille prévue à cet effet sur Arche.

La réalisation de ce compilateur doit se faire par noyaux successifs du langage. Vous trouverez ci-dessous la composition des différents noyaux à compiler. Lorsque le compilateur d'un noyau est terminé et complètement testé, vous pouvez commencer le développement du noyau suivant. Et inversement, il est inutile de commencer un nouveau noyau tant que le précédent n'est pas totalement testé. Les dates de dépôt sont impératives.

L'évaluation de ce projet se fera de façon automatique, version par version et s'arrêtera dès que l'une des versions montrera des signes de déficience.

Grammaire YAL0 - que des affichages

Le compilateur ne traite que des programmes avec des instructions d'écriture ; on se contente de l'extrait de grammaire ci-dessous.

PROGRAMME → **programme** **idf** **debut** { INSTRUCTION } + **fin**

INSTRUCTION → ECRIRE

ECRIRE → **ecrire** EXP **;**

EXP → **csteEntiere**

Noyaux du langage et dates des dépôts

Noyau	Constructions reconnues par le compilateur	Date limite de dépôt sur Arche
YAL0	Commentaires Instruction d'écriture	**Mercredi 16 janvier - à la fin de la séance - 16h**
YAL1	Déclaration de variables entières Affectation Expression réduite à une constante entière Lecture	**Dimanche 27 janvier - 20h** **ou** **Mardi 29 janvier - 20h**
YAL2	Expressions quelconques Instruction conditionnelle Instruction itérative	**Mercredi 13 février - 20h**
YAL3	Fonction sans paramètre ni variable locale	**Mercredi 6 mars - 20h**
YAL4	Fonction avec paramètres et/ou variables locales entières	
YAL5	Tableaux	