Théorie des langages et outils pour la compilation

ESGI 3A AL

Projet mini langage

Informations pratiques :

- Le projet est à réaliser en monôme et à rendre le dimanche 27 février
- Les soutenances auront lieu le mardi 1 et mercredi 2 mars

REMARQUE : Les discussions entre groupes de projet sont encouragées mais vous devez impérativement écrire vous-même le code. Le partage de code entrainera un 0 pour les deux parties.

Sujet :

L'objectif du projet est de concevoir un interpréteur pour un mini langage. Il est obligatoire de baser l'interprétation sur un arbre de syntaxe abstrait construit au cours de l'analyse syntaxique du « programme – input » donné en entrée.

Spécifications de la version minimale (8/20) :

1. Votre interpréteur devra gérer les noms de variables à plusieurs caractères.
2. Gérer les instructions suivantes :
    a. affectation
    b. affichage d'expressions numériques (pouvant contenir des variables numériques)
    c. instructions conditionnelles : implémenter le si-alors-sinon/si-alors
    d. structures itératives : implémenter le while et le *for*
    e. Affichage de l'arbre de syntaxe (sur la console ou avec graphViz)

Améliorations majeures (entre autre):

1. Gérer les fonctions avec/ sans paramètre avec/sans valeur de retour
2. Gérer le scope des variables
3. Gérer les fonctions récursives terminales
4. Les tableaux
5. la POO
6. Gérer le passage des paramètres par référence (cf id() en python)

Améliorations mineures (entre autre):

1. Gestion des erreurs (variable non initialisée, …)
2. Gérer la déclaration explicite des variables
3. Gestion du type chaine de caractères (et extension d'autant de l'instruction d'affichage)
4. Gestion des variables globales
5. affectations multiples à la python : a, b = 2, 3
6. comparaison multiples à la python : 1<2<3
7. print multiples : print(x+2, « toto ») ;
8. incrémentation et affectation élargie : x++, x+=1
9. possibilités de mettre des commentaires dans le code (et génération automatique d'une docString)

Rendu :

- votre **code** (1 ou plusieurs fichiers)
- un fichier **readme** qui détaille
    - les fonctionnalités implémentées
    - les différents inputs associés aux fonctionnalités ci dessus

Exemple de fichier readme :

1. Cas du rendu 1 seul fichier : non zippé

Vous devez avoir quelquechose qui ressemble à ça (à la syntaxe près) :

```
# affectation, print
s1='x=4;x=x+3;print(x);'

# affectation élargie, affectation
s2='x=9; x+=4; x++; print(x);'

# while, for
s3='''x=4;while(x<30){x=x+3;print(x);} ; for(i=0 ;i<4 ;i=i+1 ;){print(i*i) ;} ;'''

#fonctions void avec paramètres
s4='fonctionVoid toto(a, b){print(a+b) ;} toto(3, 5) ;'

#fonctions value avec paramètres et return explicite
s5='fonctionValue toto(a, b){c=a+b ;return c ;} toto(3, 5) ;'

#fonctions value avec paramètres et return implicite
s5='fonctionValue toto(a, b){c=a+b ; toto=c ;} toto(3, 5) ;'


#fonctions value avec paramètres et return coupe circuit
s6='fonctionValue toto(a, b){c=a+b ;return c ; print(666) ;} x=toto(3, 5) ; print(x) ;'

#fonctions value avec paramètres, return coupe circuit et scope des variables
s7='fonctionValue toto(a, b){if(a==0) return b ; c=toto(a-1, b-1) ;return c ; print(666) ;} x=toto(3, 5) ;
print(x) ;'
```

2. Cas du rendu en plusieurs fichiers : zippé

```
# affectation, print → fichier1.txt
#'x=4;x=x+3;print(x);'

# affectation élargie, affectation → fichier2.txt
#'x=9; x+=4; x++; print(x);'

# while, for → fichier3.txt
```

#’’’x=4;while(x<30){x=x+3;print(x);} ; #for(i=0 ;i<4 ;i=i+1 ;){print(i*i) ;} ;’’’

#fonctions void avec paramètres → fichier4.txt
#'fonctionVoid toto(a, b){print(a+b) ;} toto(3, 5) ;’

#fonctions value avec paramètres → fichier5.txt
#'fonctionValue toto(a, b){c=a+b ;return c ;} toto(3, 5) ;’

#fonctions value avec paramètres et return coupe circuit → fichier6.txt
#'fonctionValue toto(a, b){c=a+b ;return c ; print(666) ;} x=toto(3, #5) ; print(x) ;’

#fonctions value avec paramètres, return coupe circuit et scope des variables → fichier7.txt
#'fonctionValue toto(a, b){if(a==0) return b ; c=toto(a-1, b-#1) ;return c ; print(666) ;} x=toto(3, 5) ; print(x) ;’

Remarques importantes :

- **une exécution non verbeuse** : Le run doit aboutir à des affichages sur console (préfixées par « calc > »), et éventuellement, les opérations d’écriture des valeurs des variables. En aucun cas, on ne doit voir les traces des fonctions d’évaluations.

- **l’arbre de l’input doit être visible sur la console comme un tuple. En revanche, le lancement de graphviz doit être commenté**

```
def p_start(p):
    '''start : body'''
    p[0] = p[1]
    #printTreeGraph(p[0])
    print(p[0]))
    evalInst(p[0])
```

- A EVITER ABSOLUMENT : le nom du fichier en argument

```
if len(sys.argv) >= 2:
    path = sys.argv[1]
    f = open(path, 'r')
    s = f.read()

    yacc.parse(s)
```