27/04/2024

UNAY Arhan
GULTEKIN Kivanc
GEMICIOGLU Utku

COMPTE RENDU POUR LE PROJET D'INF404

TP5: Interpréteur de séquences d'affectations

L'objectif de ce TP était de réaliser un interpréteur pour des programmes constitués de séquences d'affectations. Nous avons commencé par définir le lexique du langage, en choisissant la syntaxe pour les identificateurs, les symboles d'affectation et de séparation. Ensuite, nous avons défini la syntaxe des affectations et avons commencé à implémenter l'analyse lexicale et syntaxique pour reconnaître les séquences d'affectations. Nous avons également commencé à travailler sur la gestion de la table des symboles pour stocker les identificateurs et leurs valeurs.

TP6: Ajout d'instructions d'entrée/sortie et construction d'un arbre abstrait

Ce TP visait à étendre l'interpréteur pour prendre en charge les instructions d'entrée/sortie au clavier et à l'écran, ainsi que la construction d'un arbre abstrait du programme à interpréter. Nous avons ajouté la reconnaissance des nouvelles instructions de lecture et d'écriture dans l'analyse lexicale, et généralisé la syntaxe des instructions pour prendre en compte ces ajouts. Ensuite, nous avons travaillé sur la construction de l'arbre abstrait en définissant les nouveaux nœuds de l'AST et en étendant les fonctions de construction et de parcours de l'AST.

TP7: Ajout d'une instruction conditionnelle

Dans ce TP, nous avons étendu l'interpréteur pour prendre en charge les instructions conditionnelles. Nous avons ajouté la reconnaissance des instructions conditionnelles dans l'analyse lexicale et syntaxique, en prenant en compte les expressions booléennes et les opérateurs de comparaison. Ensuite, nous avons commencé à travailler sur l'interprétation du programme, en modifiant l'analyse syntaxique pour construire un arbre abstrait complet et en écrivant une fonction de parcours pour évaluer les valeurs des variables à chaque instruction.

TP8: Ajout d'une instruction itérative

Dans ce dernier TP, nous avons complété l'interpréteur en prenant en charge les instructions itératives. Nous avons étendu l'analyse lexicale et syntaxique pour reconnaître les nouvelles instructions itératives, en prenant en compte les expressions booléennes et les opérateurs de comparaison. Ensuite, nous avons continué à travailler sur l'interprétation du programme en définissant un arbre abstrait complet pour représenter le programme et en écrivant une fonction de parcours pour évaluer les valeurs des variables à chaque instruction.

Notre Calculette et Notre Interpréteur : Nous avons 2 programmes distincts : 'calculette.c' et 'interpreteur.c'.

Calculette :

Notre calculette accepte les entiers ainsi que les nombres flottants et les opérateurs tels que '+', '-', '*', '/'. Nous pouvons utiliser les parenthèses pour donner la priorité aux calculs mais avec la condition de fermer toutes les parenthèses qui sont ouverts au début.

Exemple correct : **(25+25)** , **((42+5)*2)**

Exemple faux : **(25+25** , **31+31)** , **(25+25))** , **a2 ,** **-- +4 -3**

Interpréteur :

- On peut faire des affectations avec l'interpréteur.
- On peut utiliser les comparateur booléens tell que « et, ou, non » dans les expressions conditionnelles
- Les identificateurs doivent commencer par les lettres mais ensuite on peut également utiliser des chiffres.
- Il faut toujours mettre des espaces après le symbole ' = '.
- Les instructions doivent être terminés par ' ; '.

Avertissement : Les nombres décimaux ne sont pas soutenu dans notre interpréteur ! On peut les écrire dans l'intérieur de la fonction écrire comme il appelle la fonction rec_eag (utilisé dans calculette) mais on affiche en arrondissant la valeur en entier. Donc on n'utilise pas les nombres décimaux dans l'interpréteur.

Exemple : x = 1; et x3 = 1; sont corrects mais x=1 et 3 = 1 ne sont pas corrects.

- Si nous avons déjà déclaré une valeur pour un identificateur, on peut utiliser cet identificateur comme une valeur d'un autre identificateur.

Exemple correct :	**Exemple faux** :
x = 10;	x = 10;
z = x;	x = z;

- Nous avons les fonctions 'lire()' (défini en interpreter_lire) pour lire la valeur des données en argument et 'ecrire()' (défini en interpreter_ecrire) pour calculer et afficher la valeur des données en argument en utilisant la fonction 'evaluation'.

Exemple :	**Résultat :**
x = 5 ;	Table des Symboles :
lire (y) ;	x : 5
y = x+1 ;	y : 6

ecrire (y*2) ; formule formée: ((x5)((lire(y)((y+(x, 1))(ecrire(*(y, 2))))))

- Il y a des boucles ‘si’ et ‘tantque’. Les deux peuvent terminés par l’instruction ‘fsi’.

Exemples de programme

./interpreteur entree18.txt

Erreur : identificateur non déclaré

./interpreteur entree20.txt

Table des Symboles :
x : 0
r : 6
a : 5
Formule formée: (aff(x, 3)(aff(r, 2)(aff(a, 5) si <(x, a) alors (aff(x, -(x, 3))aff(r, +(r, 4))) sinon aff(x, 2) fsi)))

./interpreteur entree17.txt

Table des Symboles :
x : 3
y : 4
z : 7
Formule formée: (aff(x, 3)(aff(y, 4)(aff(z, +(x, y)))))

./test_lexeme entree2.txt

(ligne 1, colonne 1) : [nature = IDENTIFICATEUR, chaine = a2,]
Erreur_Lexicale

./calculette entree12.txt

Division par zéro!

./calculette entree11.txt

Il n'y a pas d'erreur de syntaxe.
Formule formée: +(42, *(12, 3))
Voici le résultat de l'évaluation : 78