Documentation

du projet de fin de semestre

Compilation 2021

DU Wenke

MURATI Vidan

# Difficultés

Une des grandes difficultés de ce projet a été de comprendre comment implémenter les différentes structures de données nécessaires à la compilation d’un programme.

Savoir par exemple par quoi se traduis la construction d’un arbre abstrait dans notre code, ou encore l’implémentation d’une table des symboles mais surtout quels mécanismes interviennent lors de la traduction en code intermédiaire.

Sur l’arbre abstrait,

Il a tout d’abord fallu comprendre que sa construction se faisait directement dans le programme bison, et ensuite décider des nœuds importants pour la traduction en code intermédiaire.

Sur la table des symboles,

La plus grosse difficulté a été de gérer le stockage des variables locales, la définition de structure, de fonction et la déclaration de symbole.

Sur le code intermédiaire,

La traduction en code NASM à été notre plus grande difficulté.

Une compréhension trop partielle de la pile ne nous à malheureusement pas permis de terminer le projet.

# Choix

Nous avons fait l’impasse sur une certaine partie de la traduction de notre code en nasm,

faute de temps et de compréhension. Mais nous avons malgré tout eu le temps de construire l’arbre abstrait, de produire la table de symbole, de procéder analyse sémantique sur expression et instruction.

## Arbre abstrait

L’arbre abstrait se construit au fur et à mesure avec l’analyse syntaxique, nous avons essayé au maximal de réduire le nombre de type de nœud, en manipulant la structure de nœud, par exemple, pour le nœud LValue, on a le variable simple ou le membre d’une structure, au lieu de définir deux types de nœud, on les identifier par l’existence du premier fils.

## Table de symbole

La table de symbole se compose de deux parties, une pour gérer les symboles déclarés, l’autre pour gérer les types définis par programmeur.

### Structure de la table de symbole

Pour gérer le symbole déclaré, on utilise la pile de scope. On empile un scope au début de compilation pour variable globale, puis quand on entre dans une fonction, on en empilera d’une et la dépilera quand on sort de la fonction.

Le scope contient des symboles, le nom du scope ainsi, un pointeur vers le scope père. Les symboles sont celles déclarés dans l’endroit courant. Le nom du scope normalement est le nom de la fonction où on a empilé ce scope. Le pointeur sert à aller dans le scope père quand on ne trouve pas le symbole dans le scope courant.

le symbole se compose de l’identifiant, son descripteur de type et l’adresse. l’identifiant est le nom de variable ou de fonction. Le descripteur de type est similaire que descripteur de fichier, c’est une représentation des types définis dans programme (voir dessous). L’adresse sert à traduction au code assembleur.

### Management de type défini

Durant la compilation, la table de symbole doit enregistrer les nouveaux types définis par programmeur, ici on propose un mécanisme qui s’appelle type descripteur pour les gérer.

Pour un type de symbole, on enregistre son type (type simple, struct ou fonction), son nom (void, int, char, nom de struct, nom de fonction). Si le type est struct, on enregistre ses membres par une liste, les membre de compose du type et son identifiant, c’est un sous-ensemble de symbole, on réutilise le symbole pour les gérer. La fonction est similaire que structure, sauf que la valeur de retour n’a pas l’identifiant, la définition des paramètres est pareil que celle de membre de structure, donc on mettre une convention que le premier élément de la liste stocke les information de la valeur de retour.

Pour stocker les types définis, on met une liste de type dans la table de symbole, chaque fois il y a une nouvelle définition de type à enregistrer, on le met dans cette liste, l’indice de ce type dans la liste est le type descripteur.

Concernant la définition de structure et de fonction, on vérifie chaque membre (paramètre) si le type du membre sont déjà défini. Puisque on insère la définition de structure avant analyser ses membres, c’est possible de définir une structure contenante elle-même comme un membre.

On insère les types void, int, char comme le type prédéfinit dans la liste. Comme ça on peut les manipuler comme les autres.

### Insertion de symbole

Quand on rencontre une déclaration de variable, d’abord, on vérifie si son type est déjà défini puis l’insère dans le scope courant avec l’identifiant et son type descripteur.

## Analyse sémantique

Analyse sémantique s’effectue sur instruction et expression. La contrôle d’expression est avant celle sur instruction.

### Contrôle de type sur expression

Une expression peut être une variable, une opération. Une variable doit être utilisé après sa déclaration. Pour une opération, on vérifie si les types des opérandes compatible, grâce à type descripteur, la vérification est seulement la comparaison d’entier. Si les type sont différents mais compatible, un avertissement se produira, sinon, on aura une erreur.

### Contrôle de type sur instruction

Après vérifier la validation d’expression, on fera sur instruction, la procédure est similaire, on vérifie si les variables utilisées sont bien définies ou les opérandes sont compatible.

## Traduction en assembleur

Dans le nasm, on réserve la place pour les variables globales dans une section « .bss », les variables locales elles seront stockées sur la pile. On empile les variables nouvellement déclarées et on les dépilera à la sortie de la fonction dans laquelle elles ont été déclarées.

Les fonctions de calcul se font en dépilant les 2 dernières valeurs de la pile, en effectuant l’opération, puis en empilant le résultat.

Pour le print, n’ayant pas réussi à compiler notre programme avec un point d’entrée main au lieu du point d’entrée \_start, nous n’avons pas pu utiliser la fonction printf de la libc.

Nous avons donc pris le parti de réécrire un print nous même défini dans le fichier functions.asm.

# Conclusion

Ce projet nous a permis la compréhension de nombreux mécanismes sur lesquels reposent la plupart de nos programmes.

Que ce soit la décomposition d’instructions en un arbre abstrait, ou encore la gestion de la mémoire lors de la traduction de ces instructions en langage machine, ce projet nous a vraiment permis d’élargir notre compréhension du domaine informatique. En enlevant le voile devant les mécanismes moins évidents ( Gestion de la pile / Registres ).

Même si nous n’avons malheureusement pas eu le temps de finaliser le projet, nous avons quand même pu observer les différentes difficultés qu’implique la mise en place d’un langage qui simplifie l’écriture de programmes informatique.