Rapport TP n°2 :

Compilateur VSL + - ANTLR/JAVA

Auteurs : DIALLO, Djiby – **KHEIRI**, Sarah

Professeur : MASSON, Véronique

Master 1, Génie logiciel, Groupe **1.1**

Introduction

Ce TP consistait en la réalisation d'un compilateur du langage VSL+ à l'aide de ANTLR, et produire un code 3 adresses vus en cours et TD de compilation.
Nous devrions pour cela écrire un arpenteur d'arbre qui effectue la vérification de type et la génération de code 3 adresses pour chaque constructions du langage VSL+, qui constitue la face avant de notre compilateur.
Puis finalement, transformer le code 3 adresses en un fichier assembleur MIPS afin de produire des exécutables.

Méthodologie de travail

Avant de commencer le projet nous avons tout d'abord lus, compris et analysé les différentes classes fournies avec le projet, puis compilé et testé la version qui nous a été donné pour mieux comprendre notre arpenteur d'arbre.

La réalisation du TP c'est fait dans l'esprit du développement agile, c'est-à-dire de façon itérative. En effet, on a commencé par les expressions simples, la déclaration des variables, les instructions, et finalement les fonctions et les tableaux.

Dans chacune de ces parties on a effectivement réaliser à chaque fois des tests pour être sûre que tout marche et ne pas accumuler les bugs, car certaines règles sont reliées avec d'autres.

Modifications apportées:

- VSLTreeParser.g: la vérifications des types s'est fait à ce niveau, et la modification de l'axiome pour pouvoir faire des tests.
- Code3aGenerator.java: Ajout des méthodes de du code génération du code 3 adresses pour les différentes instructions de VSL+.
- DoubleExp.java: Cette classe a été spécialement ajoutée pour résoudre le problème de l'affectation concernant les éléments d'un tableau (voir commentaire dans le fichier .java).

Bilan

Le Travail demandé a été traité complètement dans l'ordre suivant:

- Les expressions simples
- L'instruction d'affectation
- La déclaration des variables
- Les expressions avec variables
- La gestion des blocs
- Les instructions de contrôle if , while et la séquence
- La définition et l'appel de fonctions (avec les prototypes)
- Les fonctions de la bibliothèque : PRINT et READ
- La gestion des tableaux (déclaration, expression, affectation et lecture)

→ On a pas pu traiter tous les cas d'erreurs que peut générer un programme VSL+, comme le contrôle du nombre de paramètres d'une fonction déjà prototypée.

Exemple de compilation d'un programme VSL+

- **Programme VSL+**

```
PROTO INT fact(k)
FUNC VOID main()
{
```

```
INT n, i, t[11]
PRINT "Input n between 0 and 11:\n"
READ n
i := 0
WHILE n-i
DO
{
t[i] := fact(i)
i := i+1
}
DONE
i := 0
WHILE n-i
DO
{
PRINT "f(", i, ") = ", t[i], "\n"
i := i+1
}
DONE
}
FUNC INT fact(n)
{
IF n THEN RETURN n*fact(n-1)
ELSE RETURN 1
FI
}
```

- **Production du code 3 adresses**

```
label main
 beginfunc
 var n
 var i
 var t
 arg L10
```

```
call L4
n =  call L8
i = 0
label L11
var T_0
T_0 = n - i
ifz T_0 goto L12
var T_1
arg i
T_1 =  call fact
t [ i ] = T_1
var T_2
T_2 = i + 1
i = T_2
goto L11
label L12
i = 0
label L16
var T_3
T_3 = n - i
ifz T_3 goto L17
arg L13
call L4
arg i
call L2
arg L14
call L4
var T_4
T_4 = t [ i ]
arg T_4
call L2
arg L15
call L4
var T_5
```

```
 T_5 = i + 1
 i = T_5
 goto L16
 label L17
 endfunc
 label fact
 beginfunc
 var n
 ifz n goto L18
 var T_7
 var T_6
 T_6 = n - 1
 arg T_6
 T_7 =  call fact
 var T_8
 T_8 = n * T_7
 return T_8
 goto L19
 label L18
 return 1
 label L19
 endfunc
L10:Input n between 0 and 11:\n
L13:f(
L14:) =
L15:\n
```

- **Production du code assembleur MIPS**

```
.rdata
.align 2
$LC0:
    .ascii      "%d\000"
    .align 2
L10:
    .ascii      "Input n between 0 and 11:\012\000"
```

```
        .align 2
L13:
        .ascii      "f(\000"
        .align 2
L14:
        .ascii      ") = \000"
        .align 2
L15:
        .ascii      "\012\000"
.text
# label main
# beginfunc
            .align   2
            .globl main
            .ent main
main:
             addiu  $sp,$sp,-132
             sw     $31,16($29)
             sw     $16,24($29)
             sw     $17,28($29)
             sw     $18,32($29)
             sw     $19,36($29)
             sw     $20,40($29)
             sw     $21,44($29)
             sw     $22,48($29)
             sw     $23,52($29)
# var n
# var i
# var t
# arg L10
        lui   $8,%hi(L10)
        addiu   $8,$8,%lo(L10)
             sw   $8,0($29)
            move   $4,$8
```

```
# call L4
        jal   n_printf
        nop
# n =  call L8
    addiu   $5,$29,56
    lui   $4,%hi($LC0)
    addiu   $4,$4,%lo($LC0)
        jal   n_read_int
        nop
    sw   $2,56($29)
# i = 0
    li   $8, 0
# label L11
    sw  $2,56($29)
    sw  $8,60($29)
L11:
# var T_0
# T_0 = n - i
    lw  $9,56($29)
    lw  $8,60($29)
       sub $8, $9, $8
# ifz T_0 goto L12
    sw  $8,108($29)
       beq  $8,0,L12
# var T_1
# arg i
    lw  $10,60($29)
        sw   $10,0($29)
       move   $4,$10
# T_1 =  call fact
        jal   fact
        nop
# t [ i ] = T_1
    lw  $8,60($29)
```

```
    li   $9,4
    mult   $8,$9
    mflo   $8
    add   $9,$29,64
    add   $8,$8,$9
    move  $9,$2
    sw  $9,0($8)
# var T_2
# T_2 = i + 1
    lw  $10,60($29)
    li   $8, 1
        add $8, $10, $8
# i = T_2
    sw  $8,116($29)
# goto L11
    sw  $2,112($29)
    sw  $8,60($29)
    sw  $9,112($29)
        j  L11
# label L12
L12:
# i = 0
    li   $8, 0
# label L16
    sw  $8,60($29)
L16:
# var T_3
# T_3 = n - i
    lw  $9,56($29)
    lw  $8,60($29)
        sub $8, $9, $8
# ifz T_3 goto L17
    sw  $8,120($29)
        beq  $8,0,L17
```

```
# arg L13
      lui   $10,%hi(L13)
      addiu   $10,$10,%lo(L13)
          sw   $10,0($29)
         move   $4,$10
# call L4
          jal   n_printf
          nop
# arg i
      lw  $8,60($29)
          sw   $8,0($29)
         move   $4,$8
# call L2
      move $5,$4
      lui   $4,%hi($LC0)
      addiu   $4,$4,%lo($LC0)
          jal   n_printf
          nop
# arg L14
      lui   $8,%hi(L14)
      addiu   $8,$8,%lo(L14)
          sw   $8,0($29)
         move   $4,$8
# call L4
          jal   n_printf
          nop
# var T_4
# T_4 = t [ i ]
      lw  $8,60($29)
      li   $9,4
      mult   $8,$9
      mflo   $8
      add   $9,$29,64
      add   $8,$8,$9
```

```
      lw  $8,0($8)
# arg T_4
      sw  $8,124($29)
          sw   $8,0($29)
         move   $4,$8
# call L2
      move $5,$4
      lui   $4,%hi($LC0)
      addiu   $4,$4,%lo($LC0)
          jal   n_printf
          nop
# arg L15
      lui   $8,%hi(L15)
      addiu   $8,$8,%lo(L15)
          sw   $8,0($29)
         move   $4,$8
# call L4
          jal   n_printf
          nop
# var T_5
# T_5 = i + 1
      lw  $9,60($29)
      li   $8, 1
         add $8, $9, $8
# i = T_5
      sw  $8,128($29)
# goto L16
      sw  $8,60($29)
         j  L16
# label L17
L17:
# endfunc
          lw      $16,20($29)
          lw      $17,24($29)
```

```
        lw      $18,28($29)
        lw      $19,32($29)
        lw      $20,36($29)
        lw      $21,40($29)
        lw      $22,44($29)
        lw      $23,48($29)
        lw      $31,16($29)
        addiu  $sp,$sp,132
        jr  $31
        nop
        .end main
# label fact
# beginfunc
        .align   2
        .globl fact
        .ent fact
fact:
        addiu  $sp,$sp,-36
        sw      $31,16($29)
# var n
# ifz n goto L18
      lw  $8,36($29)
         beq  $8,0,L18
# var T_7
# var T_6
# T_6 = n - 1
      li   $9, 1
         sub $9, $8, $9
# arg T_6
      sw  $9,28($29)
         sw   $9,0($29)
         move   $4,$9
# T_7 =  call fact
         jal   fact
```

```
            nop
# var T_8
# T_8 = n * T_7
        lw  $9,36($29)
        move  $8,$2
            mult $9, $8
            mflo $8
# return T_8
        sw  $2,24($29)
        move  $2,$8
            lw      $31,16($29)
            addiu  $sp,$sp,36
            jr  $31
            nop
# goto L19
        sw  $2,32($29)
            j  L19
# label L18
L18:
# return 1
        li   $2, 1
            lw      $31,16($29)
            addiu  $sp,$sp,36
            jr  $31
            nop
# label L19
L19:
# endfunc
            lw      $31,16($29)
            addiu  $sp,$sp,36
            jr  $31
            nop
           .end fact
```

- **Résultat de l'exécution sur NACHOS**

Input n between 0 and 11:
10
f(0) = 1
f(1) = 1
f(2) = 2
f(3) = 6
f(4) = 24
f(5) = 120
f(6) = 720
f(7) = 5040
f(8) = 40320
f(9) = 362880

Conclusion

L'objectif de ce TP était de mettre en œuvre le fonctionnement d'un compilateur de la production de code compilable à l'exécution.
Ce travail a été fait sur plusieurs parties en commençant par les instructions simples (les expressions, ..) et en finissant avec les introductions complexes (les tableaux et fonctions).
Durant ce TP on a rencontré des difficultés sur les parties fonctions et prototypes, ainsi tableaux. Ce qui nous pris beaucoup de temps pour les traiter et au final rendre un code fonctionnel.