Rapport : Projet de compilation 2A 2019

Semestre 2

Ilham ANOUAR - Jaber HMIZA - Souhail FAZOUANE - Walid REHIOUI

27 avril 2020

Table des matières

1 Introduction :

L'objectif de ce projet est de réaliser un compilateur pour le langage ALGOL60. Nous avons commencé par faire une analyse syntaxique qui se résume dans l'écriture de la grammaire et ainsi la génération de l'arbre syntaxique. Nous avons ensuite réalisé l'analyse sémantique et la génération du code assembleur en langage Java. La première étape a été déjà décrite précédemment lors du premier semestre. C'est pour cela, ce rapport présente la suite du projet. Nous allons décrire l'implémentation de la table des symboles, des contrôles sémantiques ainsi que de la génération du code assembleur.

2 Analyse sémantique

2.1 La table des symboles

La première étape dans l'analyse sémantique est de définir la table des symboles. Cela représente la partie principale, du fait qu'elle contient toutes les déclarations des variables et leur portée dans chaque bloc. On se base sur la table des symboles pour définir les déplacements des variables.
Lors de la construction de la table, on a commencé par définir les types de symboles qu'on va utiliser, à savoir : les variables, les labels, les fonctions/procédures, le label d'un switch et les arrays.
Ensuite, nous avons défini le contenu des informations enregistrées dans la table des symboles. On y trouve notamment le type du symbole, son nom , le déplacement, et si c'est own ou local. Les symboles sont stockés dans une Map<String, Symbol> qui représente la clé et son nom. Cette structure qui contient l'ensemble des symboles est donc la table des symboles.

On utilise une pile Stack<HashTable> comme une pile de région, et elle va servir pour créer ces tables (quand on rencontre un 'begin'), et insérer les symboles correctement dans la région qui contient leurs déclarations.

Le programme suivant va générer des tables des symboles (figure 2) qui lui sont associées.

```
1  begin
2      integer i;
3      integer a;
4      integer b;
5      integer temp;
6      integer procedure FIBO(N);
7          value N;
8          integer N;
9          begin
10             switch S := L1, L2, if N>0 then L1 else L2
11               a := 0;
12               b := 1;
13               for i:=1 step 1 until N do
14                   begin
15                       temp := a + b;
16                       a := b;
17                       b := temp;
18                       outinteger(1,temp);
19                       L1:
20                           begin
21                               boolean k;
22                               k:=true;
23                           end;
24                       L2:
```

```
25                          end
26              end;
27          FIBO(8)
28  end
```

```
PROG|1|0
--------------------------------------------------------------------------
FIBO            |PROC           |Integer          |8          |LOCAL
a               |VAR            |Integer          |2          |LOCAL
b               |VAR            |Integer          |4          |LOCAL
temp            |VAR            |Integer          |6          |LOCAL
i               |VAR            |Integer          |0          |LOCAL
outreal         |PROC           |UNDEFINED        |0          |OWN
PROCEDURE|2|1
--------------------------------------------------------------------------
S               |SWITCH         |None             |0          |LOCAL        |||||L1||L2||L1||
N               |PARAM          |Integer          |-2         |LOCAL
BLOCK|3|2
--------------------------------------------------------------------------
L1              |LABEL          |None             |0          |OWN
L2              |LABEL          |None             |2          |OWN
BLOCK|4|3
--------------------------------------------------------------------------
k               |VAR            |Boolean          |0          |LOCAL
```

FIGURE 1 – Table des symboles

2.2 Les erreurs sémantiques :

Pour les contrôles sémantiques, pendant le parcours de l'AST, on teste les instructions. Sur chacune d'elle, on effectue des tests afin d'extraire les éventuelles erreurs sémantiques, ainsi afficher le message d'erreur qui lui correspond.
Parmi les erreurs sémantiques qu'on a défini :

Déclaration/Affectation : Les erreurs sémantiques liées aux déclarations sont :

- Le message **variable/function/array is Undefined/Undeclared** s'affiche dans le cas où la variable, la fonction ou l'array ne sont pas bien déclarés ou le programme ne les reconnaît pas.
- La partie droite de l'affectation doit être compatible avec la partie gauche.(Même si la partie droite est une expression arithmétique et peut contenir des accès à un tableau ou un appel de fonctions)
- Dans le cas où on affecte à un boolean un number alors qu'on doit lui affecter un true ou un false : **Boolean must be true or false**

Accès à un tableau : (Array Access) :

- Le tableau doit être défini.
- Le nombre d'indices doit être égal à la dimension du tableau.

— Les indices doivent être des entiers (on teste aussi si c'est une expression arithmétique)

— Les indices doivent respecter les bornes.

Fonction/Procédures : Voici les erreurs sémantiques concernant les fonctions et les procédures :

— La procédure ne prend pas le bon nombre de paramètres en entrée : **Found a procedure but it needs to have more parameters**

— La procédure n'est pas définie : **Function is Undefined/Undeclared**

— La procédure ne doit rien retourner : **Procedure can't have a return**

— Le type des paramètes formels doit s'accorder avec les paramètres effectifs.

— La valeur retournée doit avoir le même type que celui du retour de la procédure (si défini)

For : Les erreurs sémantiques du for sont :

— L'itérateur n'existe pas ou n'est pas déclaré : **Variable is Undefined/Undeclared**.

— Le type de l'identifiant n'est pas un integer ou un real : **Integer or Real are expected in for expression** .

— Le for a beaucoup d'initialiseur ou s'il n'a pas un initialiseur : **Too many initializers in for.**

— Si le while est declaré dans le for, il testera si l'expression dans le while est correcte : **operation is not valid.**

— La vérification qu'il y a les mots clés : until, ou, step dans le for : **Semantic error**

IF : — Tester si l'expression arithmétique ou boolean après le if sont correctes : **Arithmetic/Boolean not correct.**

Switch : — Le numéro pris en index du switch ne correspond à aucun label : **Index of the label is not correct**

— L'index du switch doit être un entier : **Index of the label is not correct**

— Le switch demandé n'est pas défini ou n'est pas bien déclaré : **variable switchId is undefined**

— Lorsqu'il y a un if dans le switch, celui-ci doit retourner un label

GOTO : — Aprés le goto, le paramètre doit être un label ou un fonction.

3 Génération du code :

La dernière partie du projet consiste à générer du code assembleur correspondant au fichier test. La lecture de ce code se fait à partir de l'instruction main_, puis chaque instruction est lue. Pour les déclarations et les manipulations des variables, on alterne entre les registres et la pile.

Le code assembleur commence toujours de la même manière. On définit la pile, les registres ainsi que les valeurs de nul.

```
1 //Definitions des registres usuels
2 SP EQU R15       //SP => StackPointer
3 FR EQU R0        // FR => FunctionResult
4 WA EQU R11
5 WR EQU R14       //WR=> Work Register
6 BP EQU R13       // BP=> Base Pointer
```

```
SC EQU R12
EXIT_EXC EQU 64        // EXIT EXCEPTION
READ_EXC EQU 65        // READ EXCEPTION
WRITE_EXC EQU 66       // WRITE EXCEPTION
NUL EQU 0
NULL EQU 0
NIL EQU 0
STACK_ADRS EQU 0x1000
LOAD_ADRS EQU 0xF000


ORG LOAD_ADRS
START main_
```

On définit 2 fonctions print et printi qui seront toujours présents dans le code assembleur. Print va permettre d'afficher une chaine de caractère. Printi va afficher les nombres en utilisant la fonction ITOA qui convertit les nombres en chaine de caractères.

```
print_
LDQ 0, R1
STW BP, -(SP)
LDW BP, SP
SUB SP, R1, SP
LDW R0, (BP)4
TRP #WRITE_EXC
LDW SP, BP
LDW BP, (SP)+
RTS
```

```
printi_
STW BP, -(SP)
LDW BP, SP
ADI SP, SP, #-8
ADI SP, SP, #-2
LDW R0, (BP)4
STW R0, (BP)-10
LDW R0, #10
STW R0, -(SP)
ADI BP, R0, #-8
STW R0, -(SP)
LDW R0, (BP)-10
STW R0, -(SP)
JSR @itoa_
ADI SP, SP, #6
ADI BP, R0, #-8
STW R0, -(SP)
JSR @print_
ADI SP, SP, #2
LDW SP, BP
LDW BP, (SP)+
RTS
```

Ensuite on définit les procédures qui ont été déclarés et implémentés,pour qu'on puisse les utiliser plus tard

3.1 Niveau 1

3.1.1 Les opérations arithmétiques :

```
begin
    integer n;
    n:= ((2+4)-(7*5)+(4-7))/2;
    outinteger(1,n);
end
```

```
main_
LDW SP,#STACK_ADRS  // initialising Stack Pointer
LDQ NIL, BP         // BP loaded with 0
STW BP, -(SP)       // Enqueue BP value
LDW BP, SP          // BP points at the same adresse as SP
LDQ NIL, R7
STW R1, -(SP)
ADQ -2,SP
// n:= ((2+4)-(7*5)+(4-7))/2;
LDW R1, #2
STW R1,-(SP)
LDW R1, #4
LDW R2,(SP)+
ADD R2,R1,R1    // Addition
STW R1,-(SP)
LDW R1, #7
STW R1,-(SP)
LDW R1, #5
LDW R2,(SP)+
MUL R2,R1,R1    // Mutiplication
LDW R2,(SP)+
SUB R2,R1,R1    // Soustraction
STW R1,-(SP)
LDW R1, #4
STW R1,-(SP)
LDW R1, #7
LDW R2,(SP)+
SUB R2,R1,R1    // Soustraction
LDW R2,(SP)+
ADD R2,R1,R1    // Addition
STW R1,-(SP)
LDW R1, #2
LDW R2,(SP)+
DIV R2,R1,R1    // Division
STW R1, (BP)-4
// outinteger(1,n);
LDW R1,(BP)-4 // Charging variable to R1
STW R1, -(SP)
JSR @printi_
ADI SP, SP, #2
TRP #EXIT_EXC
JEA @main_
```

3.1.2 If, Label, FOR

```
    begin
    integer L;
    integer i;
    integer k;
      TESTLABEL :
        begin
            if (i < 3) then L:=11 else L:=12;
```

```
            k:=4;
            outinteger(1,L)
        end;
        IsHere :
        begin
        integer i;
        for i := 1 step 1 until 5 do
        begin
        outinteger(1,L)
        end
        end
        ;
      L:=12;
      i:=1;
      outinteger(1,L);
      goto TESTLABEL;
      goto isIsHere
    end
```

Voici le code assembleur correspondant à la partie du if :

```
condition_if963601816
LDW R2,BP
ADQ -2,R2        //R2 holds a Static Chain
LDW R2,(R2)      //LEAP to change environment
LDW R1,(R2)-6    // Found variable : i... loading it
LDW R2, #3
CMP R2,R1
BLE 4
LDQ  1,R2
BMP 2
LDQ 0,R2
STW  R2 , -(SP)
LDW R2, (SP)+
TST R2
JEQ #end_condition_if963601816-$-2
// then L:=11
LDW R1, #11
LDW R2,BP
ADQ -2,R2        //R2 holds a Static Chain
LDW R2,(R2)      //LEAP to change environment
STW R1,(R2)-4    // Found variable... Storing it
JMP  #end_cond_if_if963601816-$-2
end_condition_if963601816
else_begin_if963601816
//else L:=12;
LDW R1, #12
LDW R2,BP
ADQ -2,R2        //R2 holds a Static Chain
LDW R2,(R2)      //LEAP to change environment
STW R1,(R2)-4    // Found variable... Storing it
end_cond_if_if963601816
```

Voici le code assembleur correspondant au for

```
// for i := 1 step 1 until 5 do
LDW R1, #1
STW R1, (BP)-4
begin_cond_For103887628
LDW R1,(BP)-4            // CHARGING VARIABLE TO R1
LDW R2, #5
CMP R1, R2
```

```
JGT #end_cond__For103887628-$-2
// outinteger(1,L) Test2
STW BP,-(SP)              //save BP content in the stack LINKING
LDW R8,BP                 // saves BP for SC
LDW BP, SP                // update of current BP
STW R8,-(SP)              // STATIC CHAIN
//NEW BLOCK 2,4
// outinteger(1,L)
LDW R2,BP
ADQ -2,R2                 //R2 holds a Static Chain
LDW R2,(R2)               //LEAP to change environment
ADQ -2,R2                 //R2 holds a Static Chain
LDW R2,(R2)               //LEAP to change environment
LDW R1,(R2)-4             // Found variable : L... loading it
STW R1, -(SP)
JSR @printi_
ADI SP, SP, #2
LDW SP, BP                //UNLINKING
LDW BP, (SP)
ADQ 2, SP
// outinteger(1,L)
LDW R1,(BP)-4             // CHARGING VARIABLE TO R1
STW R1, (BP)-4
//
// i
LDQ 1, R3
ADD R3,R1,R1
STW R1, (BP)-4
JMP #begin_cond_For103887628-$-2
end_cond__For103887628
```

Le code assembleur correspondant au goto est :

```
JSR @TESTLABEL_
```

Le label est défini de la manière suivante :

```
JSR @TESTLABEL_end
TESTLABEL_
// On trouvera ici les instructions concernant la procedure du label
RTS
TESTLABEL_end
```

3.2 Niveau 2

```

begin
comment integer procedure without arguments;
  integer n;

  integer procedure test;
  begin
    test := 12345
  end;

  n := test;
  outinteger (1,n)
end
```

Voici le code assembleur correspondant au programme ci-dessus de niveau 2 :

```
1
2
3 sqr_ STW BP,-(SP) //save BP content in the stack LINKING
4 LDW BP, SP // update of current BP
5 STW SC,-(SP) //STATIC CHAIN
6 //function bloc
7 // sqr := m * m
8 LDW FR,(BP)4// CHARGING PARAM TO FR
9 STW FR,-(SP)
10 LDW FR,(BP)4// CHARGING PARAM TO FR
11 LDW R2,(SP)+
12 MUL R2,FR,FR
13 LDW SP, BP //UNLINKING
14 LDW BP, (SP)
15 ADQ 2, SP
16 RTS // end of sqr
17 //FUNC DEC END
18 main_ LDW SP,#STACK_ADRS //initialising Stack Pointer
19 LDQ NIL, BP //BP loaded with 0
20 STW BP, -(SP) // Enqueue BP value
21 LDW BP, SP //BP points at the same adresse as SP
22 LDQ NIL, R7
23 STW R1, -(SP)
24 ADQ -4,SP
25 // n := 123;
26 LDW R1, #123
27 STW R1, (BP)-4
28 // nn := sqr(n);
29 // nn := sqr(n);
30 STW R1 ,-(SP) //saving registers
31 STW R2 ,-(SP) //saving registers
32 STW R3 ,-(SP) //saving registers
33 STW R4 ,-(SP) //saving registers
34 STW R5 ,-(SP) //saving registers
35 STW R6 ,-(SP) //saving registers
36 STW R7 ,-(SP) //saving registers
37 STW R8 ,-(SP) //saving registers
38 STW R9 ,-(SP) //saving registers
39 STW R10 ,-(SP) //saving registers
40 STW R11 ,-(SP) //saving registers
41 LDW R5,(BP)-4 // CHARGING VARIABLE TO R5
42 STW R5,-(SP) // pushing parameter
43 //Computing Static chaining
44 LDW SC,BP //puting BP in SC
45 JSR @sqr_ // function call
46 ADQ 2,SP //popping parameters
47 LDW R11 ,(SP)+ //reloading registers
48 LDW R10 ,(SP)+ //reloading registers
49 LDW R9 ,(SP)+ //reloading registers
50 LDW R8 ,(SP)+ //reloading registers
51 LDW R7 ,(SP)+ //reloading registers
52 LDW R6 ,(SP)+ //reloading registers
53 LDW R5 ,(SP)+ //reloading registers
54 LDW R4 ,(SP)+ //reloading registers
55 LDW R3 ,(SP)+ //reloading registers
56 LDW R2 ,(SP)+ //reloading registers
57 LDW R1 ,(SP)+ //reloading registers
58 //End of FunctionCall
59 LDW R1, R0 // getting the result of sqr's call
60 STW R1, (BP)-6
```

```
// outinteger (1,nn)
LDW R1,(BP)-6 // CHARGING VARIABLE TO R1
STW R1, -(SP)
JSR @printi_
ADI SP, SP, #2
```

3.3 Niveau 3

```

begin
comment classic recursive procedure;
  integer nn, nf;

  integer procedure factorial(n); value n; integer n;
  begin
    if n = 1 then factorial := 1
    else factorial := n * factorial(n-1)
  end;

  nn := 5;
  nf := factorial(nn);
  outinteger ( 1 , nf)

end
```

Voici le code assembleur correspondant au programme ci-dessus de niveau 3 :

```

factorial_ STW BP,-(SP) //save BP content in the stack LINKING
LDW BP, SP // update of current BP
STW SC,-(SP) //STATIC CHAIN
//function bloc
condition_if1078694789
LDW R1,(BP)4// CHARGING PARAM TO R1
LDW R2, #1
CMP R2,R1
BNE 4
LDQ  1,R2
BMP 2
LDQ 0,R2
STW  R2 , -(SP)
LDW R2, (SP)+
TST R2
JEQ #end_condition_if1078694789-$-2
// if n = 1 then factorial := 1
LDW FR, #1
JMP  #end_cond_if_if1078694789-$-2
end_condition_if1078694789
else_begin_if1078694789
// else factorial := n * factorial(n-1)
LDW FR,(BP)4// CHARGING PARAM TO FR
STW FR,-(SP)
// else factorial := n * factorial(n-1)
STW R1 ,-(SP) //saving registers
STW R2 ,-(SP) //saving registers
STW R3 ,-(SP) //saving registers
STW R4 ,-(SP) //saving registers
STW R5 ,-(SP) //saving registers
STW R6 ,-(SP) //saving registers
```

```
33 STW R7 ,-(SP) //saving registers
34 STW R8 ,-(SP) //saving registers
35 STW R9 ,-(SP) //saving registers
36 STW R10 ,-(SP) //saving registers
37 STW R11 ,-(SP) //saving registers
38 LDW R5,(BP)4// CHARGING PARAM TO R5
39 STW R5,-(SP)
40 LDW R5, #1
41 LDW R2,(SP)+
42 SUB R2,R5,R5
43 STW R5,-(SP) // pushing parameter
44 //Computing Static chaining
45 LDW SC,BP //puting BP in SC
46 ADQ -2,SC //SC holds a Static Chain
47 LDW SC,(SC) //SC LEAP
48 JSR @factorial_ // function call
49 ADQ 2,SP //popping parameters
50 LDW R11 ,(SP)+ //reloading registers
51 LDW R10 ,(SP)+ //reloading registers
52 LDW R9 ,(SP)+ //reloading registers
53 LDW R8 ,(SP)+ //reloading registers
54 LDW R7 ,(SP)+ //reloading registers
55 LDW R6 ,(SP)+ //reloading registers
56 LDW R5 ,(SP)+ //reloading registers
57 LDW R4 ,(SP)+ //reloading registers
58 LDW R3 ,(SP)+ //reloading registers
59 LDW R2 ,(SP)+ //reloading registers
60 LDW R1 ,(SP)+ //reloading registers
61 //End of FunctionCall
62 LDW FR, R0 // getting the result of factorial's call
63 LDW R2,(SP)+
64 MUL R2,FR,FR
65 end_cond_if_if1078694789
66 LDW SP, BP //UNLINKING
67 LDW BP, (SP)
68 ADQ 2, SP
69 RTS // end of factorial
70 //FUNC DEC END
71 main_ LDW SP,#STACK_ADRS //initialising Stack Pointer
72 LDQ NIL, BP //BP loaded with 0
73 STW BP, -(SP) // Enqueue BP value
74 LDW BP, SP //BP points at the same adresse as SP
75 LDQ NIL, R7
76 STW R1, -(SP)
77 ADQ -4,SP
78 // nn := 5;
79 LDW R1, #5
80 STW R1, (BP)-4
81 // nf := factorial(nn);
82 // nf := factorial(nn);
83 STW R1 ,-(SP) //saving registers
84 STW R2 ,-(SP) //saving registers
85 STW R3 ,-(SP) //saving registers
86 STW R4 ,-(SP) //saving registers
87 STW R5 ,-(SP) //saving registers
88 STW R6 ,-(SP) //saving registers
89 STW R7 ,-(SP) //saving registers
90 STW R8 ,-(SP) //saving registers
91 STW R9 ,-(SP) //saving registers
92 STW R10 ,-(SP) //saving registers
```

```
93 STW R11 ,-(SP) //saving registers
94 LDW R5,(BP)-4 // CHARGING VARIABLE TO R5
95 STW R5,-(SP) // pushing parameter
96 //Computing Static chaining
97 LDW SC,BP //puting BP in SC
98 JSR @factorial_ // function call
99 ADQ 2,SP //popping parameters
100 LDW R11 ,(SP)+ //reloading registers
101 LDW R10 ,(SP)+ //reloading registers
102 LDW R9 ,(SP)+ //reloading registers
103 LDW R8 ,(SP)+ //reloading registers
104 LDW R7 ,(SP)+ //reloading registers
105 LDW R6 ,(SP)+ //reloading registers
106 LDW R5 ,(SP)+ //reloading registers
107 LDW R4 ,(SP)+ //reloading registers
108 LDW R3 ,(SP)+ //reloading registers
109 LDW R2 ,(SP)+ //reloading registers
110 LDW R1 ,(SP)+ //reloading registers
111 //End of FunctionCall
112 LDW R1, R0 // getting the result of factorial's call
113 STW R1, (BP)-6
114 // outinteger ( 1 , nf)
115 LDW R1,(BP)-6 // CHARGING VARIABLE TO R1
116 STW R1, -(SP)
117 JSR @printi_
118 ADI SP, SP, #2
```

3.4 Niveau 4

```
1
2
3 begin
4 comment array allocation of maximal size;
5   integer n;
6   n := 100;
7   begin
8     integer array nArr[1:n];
9     nArr[n-1] := 2334
10  end
11 end
```

Voici le code assembleur correspondant au programme ci-dessus de niveau 4 pour les tableaux :

```
1
2
3
4 main_ LDW SP,#STACK_ADRS //initialising Stack Pointer
5 LDQ NIL, BP //BP loaded with 0
6 STW BP, -(SP) // Enqueue BP value
7 LDW BP, SP //BP points at the same adresse as SP
8 LDQ NIL, R7
9 STW R1, -(SP)
10 ADQ -2,SP
11 // n := 100;
12 LDW R1, #100
13 STW R1, (BP)-4
14 STW BP,-(SP) //save BP content in the stack LINKING
15 LDW R8,BP // saves BP for SC
```

```
16 LDW BP, SP // update of current BP
17 STW R8,-(SP)// STATIC CHAIN
18 //NEW BLOCK 1,2
19 ADQ -0,SP
20 // nArr[n-1] := 2334
21 LDW R1, #2334
22 LDW R3,#0
23 LDW R2,BP
24 ADQ -2,R2 //R2 holds a Static Chain
25 LDW R2,(R2) //LEAP to change environment
26 LDW R6,(R2)-4// Found variable : n... loading it
27 STW R6,-(SP)
28 LDW R6, #1
29 LDW R2,(SP)+
30 SUB R2,R6,R6
31 LDW R7, #2
32
33 MUL R6, R7, R6 // multiplication avec l'enjamb s
34
35 ADD R3, R6, R3 // sommer les termes
36
37 LDW R5,BP
38 LDW R4,#4
39 SUB R5,R4,R5//@impl
40 LDW R4,#2
41 ADD R5,R4,R5 //(@Impl - v)
42 LDW R4,R3
43 SUB R5,R4,R5
44 STW R1,(R5)//Storing in array
45 LDW SP, BP //UNLINKING
46 LDW BP, (SP)
47 ADQ 2, SP
48 TRP #EXIT_EXC
49 JEA @main_
```

4 Schémas de traduction

4.1 Schéma de traduction du begin :

Pour la génération du begin, on commence par :

- Créer l'environnement en chargeant le chaînage dynamique et statique (le premier begin a des valeurs : NIL/NULL =0)
- On crée une étiquette main_, qui sera notre adresse du début du programme.
- Effectuer les générations correspondantes en code assembleur.
- A la fin du block, on dépile l'environnement.
- A la fin du code, on ajoute TRP #EXIT_EXC et JEA @main_.

4.2 Schéma de traduction des opérations

Pour la génération de code d'une opération arithmétique, on le fait d'une façon récursive. D'abord, on génère l'expression de la partie gauche de l'arbre , on la charge dans un registre et on la stocke dans la pile(STW). Ensuite, on génère l'expression de la partie droite et on la stocke dans un registre. Enfin, on dépile la pile pour récupérer le résultat de la partie gauche et on effectue l'opération qu'on veut.

4.2.1 Accès à une variable locale/non locale

Accéder à une variable locale est facile, car il faut juste récupérer le déplacement (Offset) de cette variable à travers la table de symbole courante, la variable se trouvera ainsi dans l'adresse (BP)+déplacement.

Pour accéder à une variable non locale , il faut sauter des chaînages statiques (Nx-Ny ou Nx est le numéro d'imbrication du bloc actuel et Ny est le numéro d'imbrication de la table où la variable est déclarée) ,et ensuite faire la même opération précédente.

4.2.2 Opérations arithmétiques

- On effectue l'opération entre le fils gauche et le fils droit.
- On ajoute l'opération correspondante à l'opération binaire dans le cas si :
  - Addition : ADD
  - Soustraction : SUB
  - Multiplication : MUL
  - Division : DIV
  - Négation : Dans ce cas on a qu'un seul fils R1 : NEG

4.2.3 Opérations binaires

- On compare le fils gauche et le fils droit : CMP
- On ajoute l'opération correspondante à l'opération binaire dans le cas si :
  - Strictement supérieur : BGT 4
  - Supérieur ou égal : BGE 4
  - Strictement inférieur : BLE 4
  - Inférieur ou égal : BLW 4
  - Égal :BNE 4
  - Différent :BEQ 4
- Si le résultat de l'opération est correcte, il continue avec l'instruction suivante. Sinon il saute deux lignes.
- Dans le cas si on fait un AND ou OR entre deux opération binaire, il commence récursivement pas la première opération et stocke le résultat dans le pile pour le charge dans le registre et il effectue soit :
- AND : avec AND et stock le résultat.
- OR : avec OR et stock le résultat
- Et pour les boolean True ou False , on fait l'opération avec TST R2.

4.3 Schéma de traduction du noeud ASSIGNMENT

Dans la partie assignment, il y a deux phases : la génération de l'expression de la partie droite, et le stockage du résultat dans l'emplacement exact dans la pile.

- On calcule l'expression droite et on la met dans un registre.
- En montant un certain nombre de chaînage statique, on peut accéder à l'adresse ou on va stocker cette valeur

4.4 Schéma de traduction du noeud FUNCTION_CALL

— **L'appelant :**

1. Sauvegarder l'état des registres.
2. Calculer le chaînage statique
3. Empiler les paramètres.
4. Faire le JSR.
5. Dépiler les paramètres.
6. Restaurer l'état des registres.

— **L'appelé :**

1. Créer le chaînage dynamique de l'environnement et mettre à jour BP.
2. Mettre en place le chaînage statique.
3. Créer les variables du bloc.
4. Exécuter les instructions.
5. Tout dépiler et restaurer la valeur du BP.
6. Positionner SP à l'adresse de retour.
7. Faire le RTS.

4.5 Schéma d'exécution d'un ARRAY_CALL

— On calcule d'abord l'adresse de l'@ d'implémentation

— On calcule les enjambés(ei)

— On soustrait(additionne en pile) de @impl :la somme de BIi*ei (avec BI les bornes inférieures de ses Bornes)

— On calcule : les indices i et ensuite le terme : ei*i pour les sommer après (les soustraire en pile) avec l'ancienne valeur. On obtient ainsi l'adresse où se trouve la valeur voulue.

4.6 Schéma de traduction du noeud FOR

— Pour la génération du for, on crée 3 étiquettes :

  — "begin_cond_"+hashCode() : Début de la condition for

  — "end_cond_for"+hashCode() : Fin de la condition for

  — "end_cond__+hashCode() : Fin du bloc for

  Ces étiquettes permettent d'accéder aux différentes parties du for.

— On affecte la valeur d'itération à la valeur d'initialisation selon les trois constructions : une construction WHILE, une construction avec un STEP, ou le cas d'une itération unique.

— après on génère le bloc DO.

4.7 Schéma de traduction du noeud IF

— Pour la génération du if, on crée 3 étiquettes :

  — "condition_if" + hashCode() : Début du bloc if

  — "end_condition_if" + hashCode() : Fin du bloc then

  — "end_condition_if_if"+hashCode() : Fin du bloc if

Ces étiquettes permettent d'accéder aux différents parties du if.

— On effectue l'opération binaire correspondante.

— Si celui-ci est vraie alors on met 1 dans R2, sinon 0 dans R2.

— On met la valeur de R2 dans la pile.

— Si R2 vaut 0 alors on va dans le else indiqué par la fin du bloc then

— Sinon on continue et à la fin du bloc then on ira directement à la fin du bloc if

4.8 Schéma de traduction du Label

— On crée une étiquette permettant d'indiquer où est créé le label : "nom_label_"

— On effectue les instructions qui suivent

4.9 Schéma de traduction du GOTO

— On récupère le nom de la fonction/label pris par le goto

— On effectue un JMP vers la fonction/label : JMP #nom_fonction-$-2

4.10 Schéma de traduction du Switch

Les étapes suivantes présentent la génération du switch :

— Lors de sa déclaration, on crée une étiquette qui porte le nom du switch.

— Puis, on crée des étiquettes début et fin qui limitent chaque composant de la liste switch. Chaque étiquette porte le nom du switch et sa position dans la liste.

— S'il s'agit d'un label, il commencera par l'étiquette qui porte sa position dans liste, ensuite d'un JMP #nomlabel-$-2, enfin de l'étiquette qui porte sa position et on ajoute un "end"

— S'il s'agit d'un If, il fera les même étiquette sauf qu'il va ajouter le code assembleur qui correspond au if, et le code qui correspond à la génération des labels.

— Lors de l'appel du switch avec un goto, on fait un JMP #nom_du_switch_indice_-$-2, qui va aller directement au label correspondant.

5 Difficultés rencontrées :

5.1 Problèmes techniques :

Parmi les problèmes qu'on a rencontré :

Le premier était notre difficulté a lié le temps que nous devions consacrer projet afin de respecter les délais au reste du travail demandé en 2ème année ainsi qu'aux autres projets. C'est pour cela, qu'on a fixé des réunions hebdomadaires. Notre deuxième problème était l'arrêt des études à cause du virus. On a dû s'organiser autrement pour travailler ensemble en se partageant les tâches et discuter sur l'avancement des parties qu'on devait faire.

5.2 Apports de chaque membre :

Apport individuel en heures :

Tâche	**Ilham**	**Jaber**	**Walid**	**Souhail**
Erreurs sémantiques/Tables des symboles	**70h**	**74h**	**78h**	**72h**
Génération de code	**86h**	**88h**	**84h**	**82h**
Deboggage	**66h**	**62h**	**62h**	**68h**
Rapport, recherche et autres	**10h**	**6h**	**6h**	**8h**
Total	**232h**	**230h**	**230h**	**230h**

6 Conclusion et perspective :

6.1 Apport du projet :

Ce projet nous a permis de découvrir et d'apprendre tous les aspects liés à la compilation. À partir de faire une analyse lexicale et ainsi de savoir comment une grammaire d'un langage est rédigée. Ensuite, cela nous a permis d'analyser le langage de manière plus approfondie afin de détecter les erreurs sémantiques et construire une TDS avec le langage JAVA. Enfin, nous avons fait une génération de code qui nous a poussé à appliquer ce que nous avons appris pendant le cours PFSI et ainsi voir comment un bout de code est généré en code assembleur.

6.2 Conclusion :

Le but de ce projet était de nous apprendre à construire un compilateur. Cela nous a permis d'appliquer ce qu'on a appris et de lier les différents modules qu'on a suivi au cours de notre cursus. À travers ce projet, nous avons enrichi nos connaissances à savoir le côté technique et le côté gestion de projet. Du fait que ce projet a été fait en grande partie de manière autonome.

7 Réunions :

Réunion 14

Minutes for Vendredi 27 décembre 2019

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour :

1. Terminer le rapport de la première phase.
2. Lister les contrôles sémantiques à faire
3. Discuter les étapes à suivre pour la construction de la TDS.

Informations échangées :

1. S'accorder sur les parties importantes de l'AST qui ne seront plus présentes dans le rapport.
2. Discuter les contrôles sémantiques qu'on a déjà listé au cours de la réalisation de la grammaire.
3. Séparer la grammaire en 4 parties afin de permettre à chacun d'effectuer des recherches et ainsi lister les contrôles sémantiques liés à sa partie.

Décisions :

1. Terminer le rapport.
2. Regrouper les contrôles sémantiques.
3. Se documenter sur la construction de la TDS

To do list :

Description	Responsable(s)	Délai
Contôle sémantique de program à formal_parameter_list	Walid	Samedi 4 Janvier
Contôle sémantique de de value_part à if_clause	Souhail	Samedi 4 Janvier
Contôle sémantique de boolean_expression1 à boolean_factor	Jabeer	Samedi 4 Janvier
Contôle sémantique de boolean_secondary à WS	Ilham	Samedi 4 Janvier

Next Meeting: Samedi 4 Janvier 2020

Réunion 15

Minutes for Samedi 4 janvier 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Finaliser le rapport.
2. Regrouper les contrôles sémantiques.
3. Partager les contrôles sémantiques à faire.
4. Discuter la construction de la TDS et les paramètres qui seront affichés

Informations échangées :

1. Effectuer les dernières rectifications sur le rapport.
2. Regrouper les contrôles sémantiques rédigés par chaque membre du groupe et discuter sur ceux qui sont importants et liés.
3. Séparer les contrôles sémantiques en 4 parties.
4. Début de la construction de la TDS.

Décisions :

1. Procéder au codage des contrôles sémantiques.
2. Mettre en place deux classes où la première fera le parcours de l'AST et la deuxième fera l'affichage.
3. Mettre en code les classes liées aux tokens dont on aura besoin pour le contrôle et le parcours.
4. Effectuer les tests collectifs lors de la prochaine réunion.

To do list :

Description	Responsable(s)	Délai
ArrayParamSymbol- ArraySymbol	Ilham	Samedi 11 Janvier
EnumTypeTDS- -LabelSymbol ParameterSymbol	Souhail	Samedi 11 Janvier
ProcedureParameterSymbol- ProcSymbol	Walid	Samedi 11 Janvier
SymboleTable - SymbolType- VarSymbol	Jabeer	Samedi 11 Janvier

Next Meeting: Samedi 11 Janvier 2020

Réunion 16

Minutes for Samedi 18 janvier 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Rendre le rapport final.
2. Vérifier si la To-Do list réalisée précédemment a été bien effectuée.
3. Tester l'affichage de la TDS et le parcours de l'arbre en groupe.

Informations échangées :

1. Rectifier les erreurs qui sont été générées lors du regroupement des codes et lors du parcours de l'arbre.
2. Faire un premier affichage de la TDS.
3. Effectuer les premiers tests sémantiques.
4. Définir les classes nécessaires pour le contrôle sémantique.
5. Définir trois classes principaux :
   - Class TDSPRinter qui s'occupera de l'affichage de la TDS.
   - Class TreeParser qui fera le parcours de l'AST.
   - Class SemanticAnalyser qui regroupera les contrôles sémantiques.

Décisions

(a) Terminer les contrôles sémantiques.

(b) Effectuer les tests pour rectifier les erreurs.

(c) Se préparer pour le démonstration du 10 février.

Next Meeting: Samedi 25 Janvier 2020

Réunion 17

Minutes for Samedi 25 janvier 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Tester les contrôles sémantiques au niveau du statement et de l'assignment.
2. Tester si l'affichage de la TDS et le parcours ne génére pas des erreurs.

Informations échangées :

1. Corriger les erreurs générées par le parcours au niveau de la fonction travel_statement et travel_assignement.
2. Vérifier si les contrôles sémantiques sont corrects.
3. Afficher plusieurs TDS liées à chaque bloc.

Décisions

1. Terminer le reste des contrôles sémantiques.
2. Effectuer les tests pour vérifier si les erreurs sont détectées.

Next Meeting: Samedi 01 Février 2020

Réunion 18

Minutes for Samedi 01 Février 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Tester les contrôles sémantiques au niveau de statement et assignement.
2. Tester si l'affichage de la TDS et le parcours ne génèrent pas d'erreurs.

Informations échangées :

1. Corriger les erreurs générées des contrôles sémantiques, des fonctions analyse_FOR et analyse_Switch.
2. Vérfier si les contrôles sémantiques sont corrects.
3. Afficher plusieurs TDS liées à chaque bloc.

Décisions

1. Terminer le reste des contrôles sémantiques.
2. Effectuer les tests pour vérifier si les erreurs sont détéctées.

Next Meeting: Samedi 08 Février 2020

Réunion 19

Minutes for Samedi 08 Février 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Effectuer les tests sémantiques afin de rectifier les erreurs générées.
2. Vérifier si l'affichage de la TDS est juste.

Informations échangées :

1. Effectuer des tests avec des codes erronés afin de vérifier si le contrôle sémantique est bien appliqué.
2. Effectuer les dernières rectifications.

Décisions

1. Préparer les jeux de tests lors de la démonstration.

Next Meeting: Samedi 15 Février 2020

Réunion 20

Minutes for Samedi 15 Février 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Préparation de la soutenance.
2. Préparation des jeux de tests à présenter lors de la soutenance.

Informations échangées :

1. Vérifier les résultats de différents tests.
2. Effectuer les dernières rectifications.
3. Mettre en commun les tests de chacun pour chosir les programmes à exécuter lors de la soutenance.

Décisions

1. Garder un test final qui reprend l'ensemble des tests mis en place.
2. Se documenter sur la génération de code

Next Meeting: Samedi 28 Février 2020

Réunion 21

Minutes for Samedi 29 Février 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1.
2. Discussion sur la génération du code.
3. Discussion sur les remarques de la soutenance.
4. Visualisation des documents envoyés par S.Collin le 30 janvier sur la génération du code.

Informations échangées :

1. Définir les étapes pour la génération du code.
2. Se documenter sur l'assembleur.

Décisions

1. Définir les étapes de génération du code.
2. Revoir le cours de PfSI.
3. Se partager les tâches.

Next Meeting: Samedi 7 Mars 2020

Réunion 22

Minutes for Samedi 7 Mars 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Se partager les blocs de génération du code.

Informations échangées :

1. Commencer par définir comment on va travailler avec les registres et la pile.
2. Se partager la création du code.

Décisions

1. Terminer les différentes tâches.

To do list :

Description	Responsable(s)	Délai
Génération du code FOR	Walid	Samedi 21 Mars
Génération du code ARRAY	Souhail	Samedi 21 Mars
Génération du code des opérations arithmétiques	Jabeer	Samedi 21 Mars
Génération du code IF	Ilham	Samedi 21 Mars

Next Meeting: Samedi 21 Mars 2020

Réunion 23

Minutes for Samedi 21 Mars 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Vérifier l'avancement de la to-do list.
2. Se partager d'autres tâches.
3. Discuter le déroulement de la soutenance à distance.
4. Préparer les tests à faire pour la soutenance.

Informations échangées :

1. Regrouper les différents bouts de code de la to-do list.
2. Préparer le test à présenter lors de la soutenance.
3. Définir les prochaines étapes à faire.

Décisions

1. Préparer la soutenance.
2. Terminer les tâches à faire de la dernière to-do list.

To do list :

Description	Responsable(s)	Délai
Génération du code de l'affichage	Walid	Samedi 4 Avril
Génération du code ARRAY	Souhail	Samedi 4 Avril
Génération des fonctions	Jabeer	Samedi 4 Avril
Génération du code Switch	Ilham	Samedi 4 Avril

Next Meeting: Samedi 18 Avril 2020

Réunion 24

Minutes for Samedi 18 Avril 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Vérifier l'avancement de la to-do list.
2. Commencer la rédaction du rapport.
3. Regrouper les codes générées.

Informations échangées :

1. Regrouper le code pour effectuer les tests.
2. Corriger les erreurs générés.
3. Terminer la dernière to-do list.

Décisions

1. Commencer le rapport.
2. Terminer les dernières tâches.

Next Meeting: Mecredi 21 Avril 2020

Réunion 25

Minutes for Mardi 21 Avril 2020

Present: Souhail FAZOUANE, Jaber HMIZA, Walid REHIOUI, Ilham ANOUAR

Ordre du jour

1. Vérifier l'avancement de la to-do list.
2. Débuger les problèmes.
3. Rédiger le rapport.
4. Vérifier les niveaux d'avancement.

Informations échangées :

1. Faire des tests de niveau pour la soutenance compile.
2. Corriger les erreurs générés.
3. Terminer la dernière to-do list.
4. Commencer la rédaction du rappport.

Décisions

1. Terminer le rapport.
2. Terminer les dernières tâches.
3. Coder les programmes demandés pour le rendu.

Next Meeting: