Projet de compilation

Rapport final (PCL2)

Loïc Bertrand
Timon Fugier
Tony Zhou
Zineb Ziani El Idrissi

TELECOM Nancy 2ème Année
2019 – 2020

Table des matières

1 Contrôles sémantiques **3**
1.1 Déclaration . . . 3
1.1.1 de Variable . . . 3
1.1.2 de Procédure . . . 3
1.1.3 de Tableau . . . 4
1.1.4 de Label . . . 4
1.1.5 de Switch . . . 4
1.2 Affectation . . . 5
1.2.1 dans une Variable . . . 5
1.2.2 dans un Tableau . . . 5
1.3 Appel . . . 5
1.3.1 de Procédure . . . 5
1.3.2 de Tableau . . . 6
1.3.3 de Label (goto) . . . 6
1.3.4 de Switch (goto) . . . 6
1.4 Expressions . . . 6
1.4.1 Arithmétiques . . . 6
1.4.2 Relationnelles . . . 7
1.4.3 Logiques . . . 7
1.5 Blocs . . . 7
1.5.1 Simples . . . 7
1.5.2 boucles For . . . 8
1.5.3 déclaration If . . . 8
1.5.4 expression If . . . 8

2 Génération de code **9**
2.1 Stratégie adoptée . . . 9
2.1.1 Conception . . . 9
2.1.2 Évaluation des expressions et gestion des registres . . . 9

2.1.3 Stockage des tableaux 10
2.1.4 Chaînage dynamique et statique 11
2.2 Exemples de code généré 12
2.2.1 Déclaration de procédure 12
2.2.2 Appel de procédure et calcul du chaînage statique 13
2.2.3 Utilisation des switchs et labels 14
2.2.4 Déclaration d'un tableau 15
2.2.5 Affectation dans un tableau 16

3 Gestion de projet **17**
3.1 Fiche projet 17
3.2 Compte-rendus rédigés 17
3.3 Diagramme de Gantt 17
3.3.1 Partie « contrôles sémantiques » 17
3.3.2 Partie « génération de code » 18
3.4 Évaluation de la répartition du travail 18
3.5 Répartition des tâches au sein du groupe 19
3.5.1 Partie « contrôles sémantiques » 19
3.5.2 Partie « génération de code » 20

Bibliographie **21**

Annexes **22**
A Compte-rendus de réunion rédigés pour la partie « contrôles sémantiques » 22
B Compte-rendus de réunion rédigés pour la partie « génération de code » . . 34
C Estimation du temps de travail pour la partie « contrôles sémantiques » . . 53
D Estimation du temps de travail pour la partie « contrôles sémantiques » . . 54

1 Contrôles sémantiques

Cette partie liste les erreurs sémantiques détectées par notre compilateur.

1.1 Déclaration

1.1.1 de Variable

```
integer a, b
```

Erreurs :

— **Redéclaration :** le nom d'une des variables est déjà utilisé pour un autre symbole déclaré dans le même environnement.

Actions :

— **Stockage :** créer et ajouter les variables dans la table des symboles courante avec leur nom et leur type.

1.1.2 de Procédure

```
real procedure foo(x, y);
value x; integer y; real x;
begin
  foo := 3.14159
end
```

Erreurs :

— **Redéclaration :** le nom de la procédure est déjà utilisé pour un autre symbole déclaré dans le même environnement.
— **Mauvaise valeur :** un élément de la partie `value` ne correspond pas à un paramètre de la procédure.
— **Mauvaise précision de type :** un élément de la spécification de types ne correspond pas à un paramètre de la procédure.
— **Paramètre sans type :** un paramètre de la procédure n'a aucun type spécifié.
— **Retours non exhaustifs :** la procédure pourrait ne jamais retourner une valeur, par exemple, s'il n'y a qu'une seule affectation à la variable de retour, dans une branche conditionnelle.

— **Retour dans une procédure sans type de retour :** dans une procédure sans type de retour, on ne peut pas utiliser une variable du nom de la procédure.

Actions :

— **Stockage :** créer et ajouter la procédure dans la table des symboles courante avec son nom, son type de retour et ses paramètres (type, nom, passés pas valeur/nom).
— **Bloc :** vérifications sémantiques sur les instructions à l'intérieur du bloc de la procédure.

1.1.3 de Tableau

```
real array numbers[a:3,0:10]
```

Erreurs :

— **Redéclaration :** le nom du tableau est déjà utilisé pour un autre symbole déclaré dans le même environnement.
— **Mauvais type de borne :** une borne n'est ni un entier, ni un réel.
— **Bornes mal ordonnées :** un intervalle (pour une dimension) du tableau est mal ordonné (contrôle possible uniquement sur les intervalles avec bornes constantes).

Actions :

— **Stockage :** créer et ajouter le tableau dans la table des symboles courante avec son nom, son type de contenu et ses bornes si elles sont définies par des constantes.

1.1.4 de Label

```
apple:
```

Erreurs :

— **Redéclaration :** le nom du label est déjà utilisé pour un autre symbole déclaré dans le même environnement.

Actions :

— **Stockage :** créer et ajouter le label dans la table des symboles courante avec son identifiant.

1.1.5 de Switch

```
switch fruit := apple, banana
```

Erreurs :

— **Redéclaration :** le nom du switch est déjà utilisé pour un autre symbole déclaré dans le même environnement.

— **Label non accessible :** un label du switch n'est pas accessible depuis la table des symboles courante.

Actions :

— **Stockage :** créer et ajouter le switch dans la table des symboles courante avec son identifiant et la liste de ses labels.

1.2 Affectation

1.2.1 dans une Variable

```
n := 12
```

Erreurs :

— **Non déclarée :** la variable affectée n'est pas déclarée à portée.
— **Types incompatibles :** le type droit ne peut pas être affecté au type gauche.

1.2.2 dans un Tableau

```
numbers[3,5] = 3.14
```

Erreurs :

— **Non déclaré :** le tableau affecté n'est pas déclarée à portée.
— **Types incompatibles :** le type droit ne peut pas être affecté au type de contenu du tableau.
— **Indices non réels/entiers :** les indices donnés ne sont ni des réels, ni des entiers.
— **Indices hors bornes :** un indice dépasse les bornes définies par des constantes.

1.3 Appel

1.3.1 de Procédure

```
x := foo(42, 3.14, "hey")
```

Erreurs :

— **Non déclarée :** la procédure appelée n'est pas déclarée à portée.
— **Mauvais nombre de paramètres :** le nombre de paramètres passés à la procédure n'est pas égal au nombre de paramètres définis à la déclaration.
— **Types incompatibles :** le type d'un paramètre passé à la procédure ne peut pas être affecté au type défini pour ce paramètre à la déclaration de la procédure.

Actions :

— Retourner le type de l'expression.

1.3.2 de Tableau

```
x := numbers[3,5]
```

Erreurs :

- **Non déclarée :** le tableau appelé n'est pas déclaré à portée ou l'identifiant n'est pas celui d'un tableau.
- **Indices non réels/entiers :** les indices donnés ne sont ni des réels, ni des entiers.
- **Indices hors bornes :** un indice dépasse les bornes définies par des constantes.

Actions :

- Retourner le type de l'expression.

1.3.3 de Label (goto)

```
goto banana
```

Erreurs :

- **Non déclaré :** le label appelé n'est pas déclaré à portée ou l'identifiant n'est pas celui d'un label.

1.3.4 de Switch (goto)

```
goto fruits[1]
```

Erreurs :

- **Non déclaré :** le switch utilisé n'est pas déclaré à portée ou l'identifiant n'est pas celui d'un switch.
- **Indices non réels/entiers :** l'indices donné n'est ni un réel, ni un entier.
- **Indices hors bornes :** l'indice est inférieur à 1 ou supérieur au nombre de labels du switch.

1.4 Expressions

1.4.1 Arithmétiques

```
3 * (x + n) - 7 - y ** 8 / 9
```

Erreurs :

- **Types incompatibles :** une opération arithmétique ne peut être effectuée que si le type à droite est compatible avec celui de gauche.

— **Division par zéro :** toute division par zéro génère une erreur sémantique et lance une exception *DivisionByZeroException*.

Actions :

— Retourner le type de l'expression.

1.4.2 Relationnelles

```
x < 45
```

Erreurs :

— **Types incompatibles :** une comparaison ne peut être effectuée que si les deux entités sont comparables voire compatibles (par exemple un entier et un réel, deux réels)

Actions :

— Retourner le type booléen.

1.4.3 Logiques

```
x < 12 /\ x >= 9 \/ 4 + z <= 8
```

Erreurs :

— **Types incompatibles :** une opération logique ne peut être effectuée que si les deux entités sont des expressions booléennes.

Actions :

— Retourner le type booléen.

1.5 Blocs

1.5.1 Simples

```
begin
  integer a;
  a := 6;
  outinteger(1, a)
end
```

Actions :

— Création d'une table de symbole fille.
— Stockage de toutes les exceptions attrapées dans le bloc dans la liste d'exceptions.

1.5.2 boucles For

```
for i := 0 step 2 until n do ...
for i := 3, 7, 9 do ...
for i := 6  while i < 3 do ...
```

Erreurs :

- **Mauvais type de condition :** la condition doit être de type booléen.
- **Mauvais type de pas :** le pas doit être de type entier ou réel seul.
- **Mauvais type de borne :** la borne doit être de type entier ou réel seul.
- **Label inaccessible :** un label est inaccessible lorsqu'il est déclaré dans une boucle for.

1.5.3 déclaration If

```
if x < 8 then outstring(1, "ok");
if y > 9 then
  begin
    integer a = y + 9;
    outinteger(1, a);
  end
else outstring(1, "no");
```

Erreurs :

- **Mauvais type de condition :** la condition doit être de type booléen.

1.5.4 expression If

```
if x < 8 then 12 else 8
```

Erreurs :

- **Mauvais type de condition :** la condition doit être de type booléen.
- **Types incompatibles :** les types de chaque branche doivent être compatibles (ex : string / entier sont incompatibles)
- **Mauvais type de retour :** l'expression If doit obligatoirement retourner un résultat qui n'est pas de type `VOID`.

Actions :

- Retourner le type de l'expression le plus spécifique possible (ex : integer / real ⇒ real mais integer / real ⇒ integer).

2 Génération de code

2.1 Stratégie adoptée

2.1.1 Conception

Nous avons réutilisé le patron de conception « visiteur » et notre interface `ASTVisitor` pour parcourir l'AST, cette fois-ci pour générer du code assembleur. L'évaluation d'une sous-instruction se fait donc simplement au moyen du code : `instruction.accept(this)`.

Nous utilisons une classe `Assembly` pour faciliter l'écriture du code assembleur :
— gestion des procédures imbriquées,
— méthodes pour les instructions régulièrement utilisées,
— formatage esthétique du code assembleur.

La classe `UniqueReference` nous permet de générer des labels assembleurs uniques, à partir d'un nom donné. Ainsi, si l'on demande deux labels avec le nom `for_end`, cette classe nous génère les labels `for_end_` et `for_end_1`.

2.1.2 Évaluation des expressions et gestion des registres

Nous avons fait le choix d'empiler systématiquement la valeur résultante de l'évaluation d'une expression. Ainsi, évaluer l'expression `5 * 2 + 6` met le résultat `16` sur la pile. De même, évaluer l'expression `if true then f(5) else 3 * a` met la valeur résultante sur la pile.

Par exemple, évaluer l'expression `1 + 4 * 2 - 5` effectue les opérations décrites sur la Figure 2.1 dans la pile.

Cette méthode récursive de mise de pile de chaque expression n'est pas la plus optimisée. Pour réduire un peu le code produit, notre code assembleur subit une optimisation

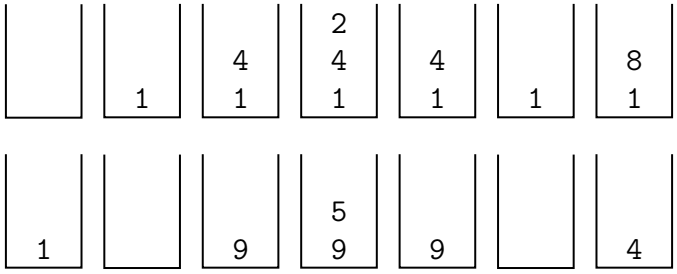

FIGURE 2.1 – Étapes d'empilements dépilements lors de l'évaluation de l'expression `1 + 4 * 2 - 5`.

```
STW R1, -(SP)       // Push value in R1 on the stack
LDW R1, (SP)+       // Pop value from the stack into R1
```

FIGURE 2.2 – Empilement et dépilement consécutifs depuis et vers un même registre, supprimé de notre code assembleur.

basique : les empilement-dépilement consécutifs, depuis et vers un même registre (voir Figure 2.2) sont tous supprimés, puisqu'ils sont inutiles. Celui nous donne le code assembleur présenté en Figure 2.3.

```
LDW R1, #1          // Load int value 1
STW R1, -(SP)       // Put it on the stack
LDW R1, #4          // Load int value 4
STW R1, -(SP)       // Put it on the stack
LDW R1, #2          // Load int value 2
LDW R2, (SP)+       // Pop second value from the stack into R2
MUL R1, R2, R1      // Mul first and second value
LDW R2, (SP)+       // Pop second value from the stack into R2
ADD R1, R2, R1      // Add first and second value
STW R1, -(SP)       // Push resulting value on the stack
LDW R1, #5          // Load int value 5
LDW R2, (SP)+       // Pop second value from the stack into R2
SUB R2, R1, R1      // reduce first value from the second
STW R1, (BP)-4      // Store value into variable 'a'
```

FIGURE 2.3 – Code assembleur optimisé pour calculer l'expression `1 + 4 * 2 - 5` et la ranger dans une variable locale.

Concernant la sauvegarde des registres en pile, nous l'effectuons d'une manière que l'on pourrait qualifier de *lazy*, c'est-à-dire, seulement lorsqu'il est nécessaire de ne pas perdre leur valeur. En pratique, toute évaluation d'une sous-instruction est susceptible de changer la valeur des registres. Nous effectuons donc les sauvegardes nécessaires avant chaque évaluation de sous expression (exemple en Figure 2.4).

```
// We don't want to lose @impl :
asm.push("R3", "Save @impl on stack");
index.accept(this); // Evaluate sub-expression
                    // and push its value on stack
asm.pop("R1", "Pop index value into R1");
asm.pop("R3", "Pop @impl into R3");
```

FIGURE 2.4 – Sauvegarde de registre avant évaluation de sous-expression.

2.1.3 Stockage des tableaux

À la déclaration d'un tableau dynamique et multidimensionnel, nous stockons dans la pile l'adresse d'implémentation du tableau (addresse de la première valeur située dans le

tas) ainsi que les couples borne inférieur-supérieure correspondant à chaque dimension. Dans le tas, les valeurs sont stockées de manière contiguë, comme présenté sur le schéma en Figure 2.5. Dans ce schéma (et le suivant), `SP0` correspond à la valeur initiale du pointeur de pile, les `BPi` sont les différentes valeurs des chaînages dynamiques et les `SCi` sont les différentes valeurs de chaînages statiques. `HP0` est la valeur initiale du pointeur de tas.

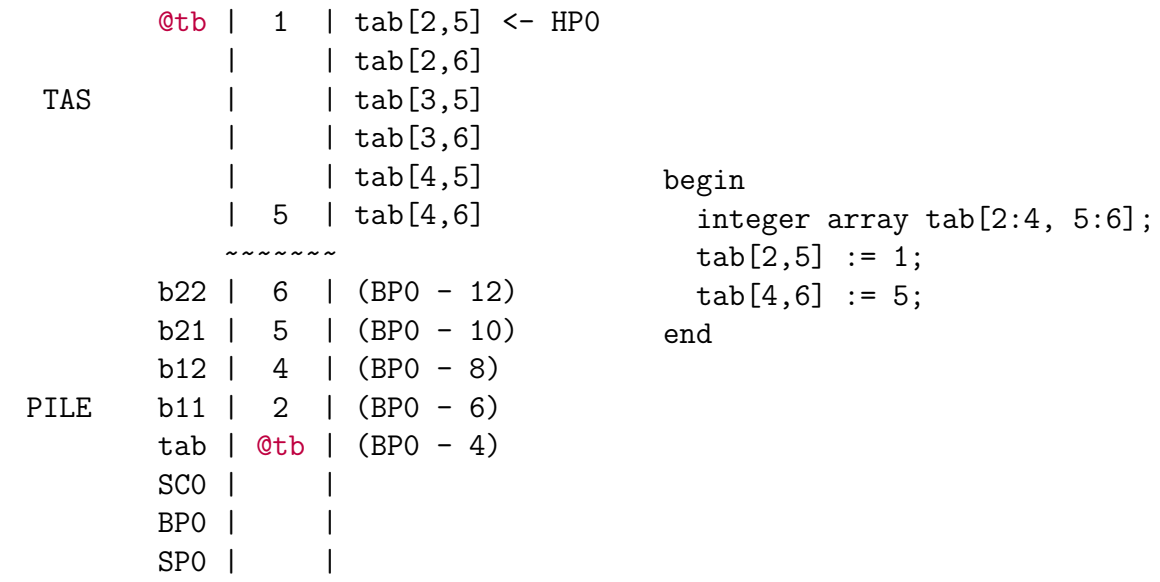

FIGURE 2.5 – Stockage des tableaux dynamiques multidimensionnels dans la pile et dans le tas.

2.1.4 Chaînage dynamique et statique

La Figure 2.6 montre comment sont stockés les chaînages statique et dynamique en pile. Le chaînage dynamique est toujours l'adresse de la base de l'environnement précédent. Le chaînage statique est calculé par l'appelant, en fonction du niveau d'imbrication courant et du niveau d'imbrication de la procédure appelée.

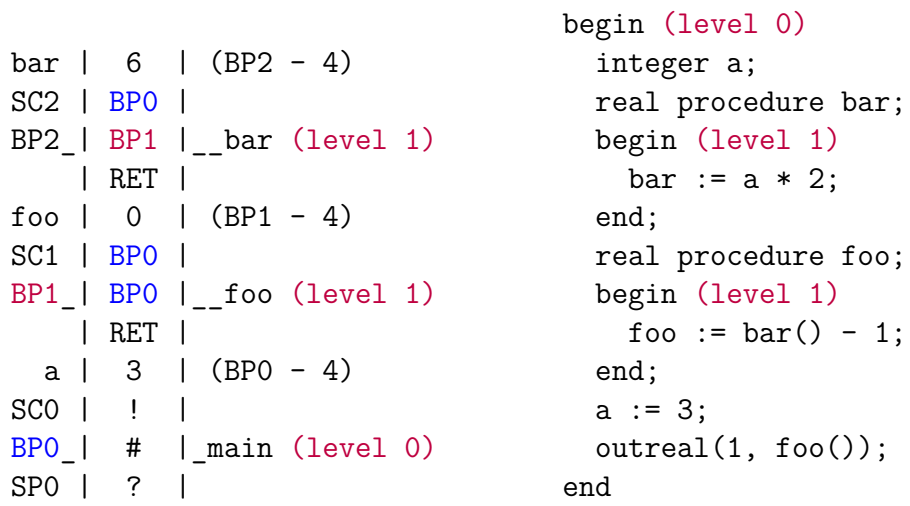

FIGURE 2.6 – Chaînages dynamique et statique dans la pile

2.2 Exemples de code généré

2.2.1 Déclaration de procédure

```
integer procedure proc;
begin
  proc := 3;
end
```

```
proc_ // Declaration of procedure 'proc'

    // Prepare procedure environment
    STW BP, -(SP)       // Push old BP on stack (dynamic chaining)
    LDW BP, SP          // New BP is the current SP
    STW R1, -(SP)       // Push SC computed by the caller (static chaining)

    // Put result variable 'proc' on the stack
    LDW WR, #0          // Initialize it with 0
    STW WR, -(SP)       // And place it on the stack

    // Assignment proc := 3;
    LDW R1, #3          // Load int value 3
    STW R1, (BP)-4      // Store value into 'proc'

    // Store return value into R1
    LDW R1, (BP)-4      // Load value of 'proc' into R1

    // End environment
    LDW SP, BP          // Return to base of current environment
    LDW BP, (SP)+       // Pop previously saved base pointer (dynamic chaining)
    RTS                 // Return to caller
```

La valeur du chaînage statique est calculée par le programme appelant et placée dans `R1`. Cette valeur est récupérée par la procédure appelée. La valeur de retour d'une procédure est placée dans une variable du nom de la procédure, déclarée implicitement. À la fin de la procédure, la valeur de cette variable est placée dans `R1` pour être récupérée dans l'appelant.

2.2.2 Appel de procédure et calcul du chaînage statique

```
begin
  integer a;

  real procedure bar;
  begin
    bar := a * 2
  end;

  real procedure big;
  begin
    real procedure foo;
    begin
      foo := bar() - 1
    end;
    big := foo()
  end;

  a := 3;
end
```

```
// Subtraction

// Call procedure 'bar'

// Compute static chaining
LDW R1, BP          // Put current BP into R1
ADQ -2, R1          // Point to static chaining
LDW R1, (R1)        // Go up by one environment
ADQ -2, R1          // Point to static chaining
LDW R1, (R1)        // Go up by one environment
JSR @bar_           // Call procedure 'bar'
STW R1, -(SP)       // Save procedure result on the stack
LDW R1, #1          // Load int value 1
LDW R2, (SP)+       // Pop second value from the stack into R2
SUB R2, R1, R1      // reduce first value from the second
STW R1, (BP)-4      // Store value into 'foo'
```

Lors d'un appel de procédure, on calcule le chaînage statique. On part de l'adresse de la base de l'environnement courant (`BP`) et on remonte les chaînages statiques n fois, où n est le niveau d'imbrication du block courant auquel on soustrait le niveau d'imbrication du block dans laquelle est définie la procédure appelée. Dans cet exemple, l'appel de `bar` est effectué dans un bloc d'imbrication 3, et `bar` est définie dans un bloc d'imbrication 1. Donc $n = 3 - 1 = 2$, on remonte de 2 chaînages statiques.

2.2.3 Utilisation des switchs et labels

```
switch s := carotte, chaussette, fraise;
goto s[3];
carotte:;
chaussette:;
fraise:;
```

```
    // Switch 's' declaration
    JMP #END_s_-$-2     // jump to the end of the label declaration of the
                        // switch 's'

BEGIN_s_ // The beginning of the switch declaration
    JMP #carotte_-$-2   // jump to the carotte_ Label
    JMP #chaussette_-$-2  // jump to the chaussette_ Label
    JMP #fraise_-$-2    // jump to the fraise_ Label

END_s_ // The end of the switch declaration
    LDW R1, #3          // Load int value 3
    STW R1, -(SP)       // Put it on the stack

    // goto s[3];
    LDW R1, (SP)+       // Pop first value from the stack into R1
    JGT #4              // Jump to the end of this loop if the bounds are corrects
    JMP #index_oob-$-2  // Print error out of bound message and exit
    ADI R1,R3,#-3       // check if the bounds are corrects
    JLE #4              // Jump to the end of this loop if the bounds are corrects
    JMP #index_oob-$-2  // Print error out of bound message and exit
    LDW R2, #4          // Put int value 4 into R2
    MUL R1, R2, R1      // R1 becomes 4*R1 so the index is ok
    ADQ -4, R1          // R1 becomes R1-4 so the index is perfect
    LDW R2, #BEGIN_s_   // stockage of the BEGIN address
    ADD R2, R1, R1      // Put address of correct jump into R1
    JEA (R1)            // jump to the R1 element of the 's' switch

carotte_ // create label carotte_
chaussette_ // create label chaussette_
fraise_ // create label fraise_
```

L'utilisation des labels est gérée par un ensemble de sauts inconditionnels. Lors de leur déclaration, une étiquette est créée. Les instructions goto permettent de déterminer l'adresse du label correspondant.

En ce qui concerne le switch, sa déclaration génère des instructions de sauts vers les labels qu'il contient. Le calcul de la valeur de l'indice du switch permet alors de déterminer le label sur lequel le goto a été utilisé. Une vérification des valeurs d'indice a été implémentée.

2.2.4 Déclaration d'un tableau

```
begin
  integer array t[1:2]
end
```

```
    // Array declaration integer array t[1:2];
    STW HP, -(SP)       // Store the origin of the array in the heap onto the
                        // stack
    LDQ 1,R7            // reset R7
    STW R7, -(SP)       // Save R7 on stack
    LDW R1, #1          // Load int value 1
    STW R1, -(SP)       // Put it on the stack
    LDW R5, (SP)+       // Pop first bound value into R5
    LDW R1, #2          // Load int value 2
    STW R1, -(SP)       // Put it on the stack
    LDW R2, (SP)+       // Pop second bound value into R2
    LDW R7, (SP)+       // Restore R7
    STW R5, -(SP)       // Store the first bound in the stack
    STW R2, -(SP)       // Store de the second bound in the stack
    SUB R2,R5,R6        // put the result of first bound - last bound in R6
    JGE #t_0-$-2        // Jump to the end of this loop if the bounds are correct
    JMP #index_oob-$-2  // Print error out of bound message and exit

t_0 // create the end of the 0 loop
    ADI R6,R6,#1        // R6 now contain the number of element in the index
    MUL R6,R7,R7        // R7 is updated to contain the number of elements in the
                        // array
    LDW R3,#0           // charge R3 avec la valeur par default

t_ // create label t_
    STW R3, (HP)+       // Store default value in and heap the increment heap
                        // pointer
    ADQ -1,R7           // number of elements left to initialize
    JNE #t_-$-2         // if we still have elements to initialize we loop
```

Pour les tableaux, on utilise le tas pour leur stockage. On sauvegarde l'adresse d'implémentation et les bornes dans la pile. Il faut d'abord calculer le nombre d'éléments dans le tableau. Pour tout élément, on charge une valeur initiale (0 pour un tableau d'entiers), on incrémente le stack pointer et on décrémente le nombre d'éléments à initialiser. Des mécanismes de vérification d'indice sont présents.

2.2.5 Affectation dans un tableau

```
begin
  integer array t[1:2];
  t[1]:=1;
end
```

```
// Array assignment t[1]:=1;
LDW R1, #1          // Load int value 1
LDW R3, (BP)-4      // R3 <- @impl
LDW R4, (BP)-6      // Lower bound
LDW R5, (BP)-8      // R5 <- Upper bound for the index
SUB R5,R1,R9        // just to check if the bound are correct
JGE #sup1_end_-$-2  // Jump to the end of this loop if the bounds are correct
JMP #index_oob-$-2  // Print error out of bound message and exit

// Compute shift and store
ADD R10,R10,R10     // R10 *2 because 1 elt size is 2
ADD R3, R10, R10    // @impl + element shift : @element
STW R10, -(SP)      // Save @element on stack
LDW R1, #1          // Load int value 1
LDW R10, (SP)+      // Pop @element into R10
STW R1, (R10)       // Store value at @element
```

Pour récupérer un élément d'un tableau, on se réfère à l'adresse d'implémentation dans la pile. Ensuite, on calcule le déplacement nécessaire dans le tas à l'aide des bornes, également empilées. Enfin, avec la bonne adresse, on stocke l'élément dans la bonne case.

3 Gestion de projet

3.1 Fiche projet

3.2 Compte-rendus rédigés

Les compte-rendus de réunions rédigés pour les parties « contrôles sémantiques » et « génération de code » sont respectivement en **Annexe A** et en **Annexe B** . Ils contiennent de nombreuses informations sur notre démarche, les étapes de conception et les décisions prises.

3.3 Diagramme de Gantt

3.3.1 Partie « contrôles sémantiques »

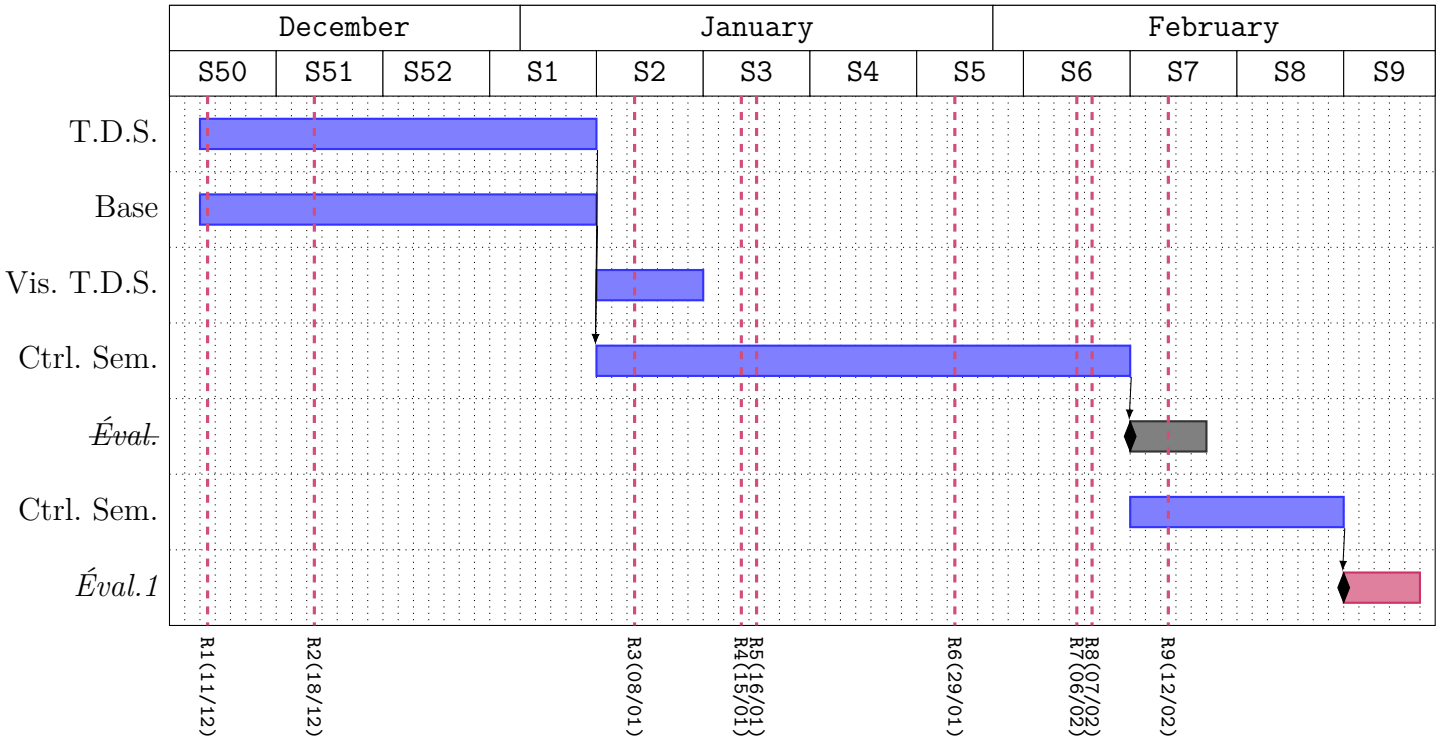

FIGURE 3.1 – Diagramme de Gantt (2e partie du projet)

Légende :

- T.D.S. : Classes permettant de construire une table des symboles
- Base : Classes de base permettant de construire l'analyseur sémantique
- Vis. T.D.S. : Visualisation de la table des symboles
- Ctrl. Sem. : Implémentation des contrôles sémantiques
- *Éval.1* : Évaluation des contrôles sémantiques (déplacée)

3.3.2 Partie « génération de code »

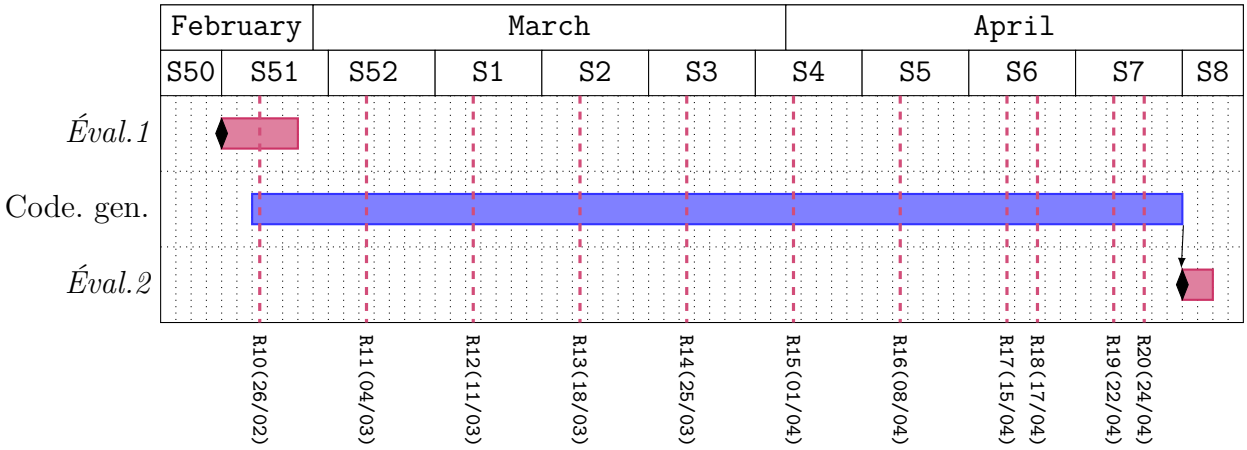

FIGURE 3.2 – Diagramme de Gantt (2e partie du projet)

Légende :

- *Éval.1* : Évaluation des contrôles sémantiques
- Code. gen. : Implémentation de la génération de code
- *Éval.2* : Évaluation de la génération de code

3.4 Évaluation de la répartition du travail

Ceci est une estimation (très) basse de notre temps de travail sur le projet. Le détail de cette estimation est disponible en **Annexe C** pour la partie « contrôles sémantiques » et en **Annexe D** pour la partie « génération de code » ;

Membre	Temps cumulé
Loïc	86h30
Tony	72h30
Timon	65h30
Zineb	65h30

3.5 Répartition des tâches au sein du groupe

3.5.1 Partie « contrôles sémantiques »

Élément du langage	Implémentation	Tests unitaires
Déclaration		
de Variable	Timon	Loïc
de Procédure	Loïc, Zineb	Tony, Zineb
de Tableau	Timon	Loïc
de Label	Zineb	Loïc
de Switch	Tony	Timon, Tony
Affectation		
dans une Variable	Loïc	Zineb
dans un Tableau	Timon	Timon
Appel		
de Procédure	Loïc, Zineb	Loïc, Zineb
de Tableau	Timon	Timon
de Label (goto)	Loïc	Loïc
de Switch (goto)	Timon	Zineb
Expressions		
Arithmétiques	Timon, Tony, Zineb	Loïc, Tony
Relationnelles	Timon, Tony, Zineb	Loïc, Tony
Logiques	Timon, Tony, Zineb	Loïc, Tony
Blocs		
Simples	Loïc	
boucles For	Tony	Loïc
déclaration If	Loïc, Timon	Zineb
expression If	Timon	Loïc, Zineb

3.5.2 Partie « génération de code »

Élément du langage	Implémentation	Tests unitaires
Déclaration		
de Variable	Loïc	Zineb, Loïc
de Procédure	Loïc	Loïc
de Tableau	Timon	Tony
de Label	Timon	Zineb
de Switch	Timon	Zineb
Affectation		
dans une Variable	Zineb	Timon
dans un Tableau	Tony	Tony
Appel		
de Procédure	Loïc	Loïc
de Tableau	Loïc	Tony
de Label (goto)	Timon, Loïc	Zineb, Loïc
de Switch (goto)	Timon, Loïc	Zineb
Expressions		
Arithmétiques	Timon, Loïc, Zineb, Tony	Tony
Relationnelles	Zineb, Tony	Tony
Logiques	Tony	Tony
Blocs		
Simples	Zineb, Timon	Zineb
boucles For	Tony, Loïc	Tony, Loïc
déclaration If	Loïc	Loïc
expression If	Zineb	Zineb

Bibliographie

[1] F. L. Bauer · A. S. Householder · F. W. J. Olver H. Rutishauser . K. Samelson · E. Stiefel. Description of ALGOL 60. http://www.algol60.org/docs/Rutishauser_Description_of_ALGOL_60_1967.pdf, 1967.

[2] hlevkin. Algol 60 Forever ! http://algol60.org/1home.htm, 2020.

[3] Shijinglu. Common Design Patterns in Antlr4 projects. https://shijinglu.wordpress.com/2015/01/22/common-design-patterns-in-antlr4-projects/, 2015.

[4] TerenceP and other contributors. Tree construction : « Using custom AST node types ». https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687090/Tree+construction, 2011.

Annexes

A Compte-rendus de réunion rédigés pour la partie « contrôles sémantiques »

Compte-rendu de réunion 1

Mercredi 11 décembre 2019, 13h-16h

Type : Réunion de lancement de PCL2

Lieu : Amphi Sud (tutorat) - Salle PI

Présents : Loïc Bertrand, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

— Rechercher comment concevoir la TDS (travaux existants, cours, tutorat...)
— Réaliser au minimum un diagramme de classe

Informations échangées :

— La classe `SymbolTable` doit posséder les méthodes `define(Symbol)` pour ajouter un `Symbol` à la table au moment d'une déclaration et `Symbol resolve(String)` pour récupérer un `Symbol` à partir du nom de son identificateur.
— Durant l'analyse sémantique, stocker la table des symboles correspondant à la région courante dans une variable (ou dans une pile).
— Pour les messages d'erreur à la compilation, utiliser un Logger.
— Exemple de table des symboles : `https://github.com/antlr/symtab`
— Visiteur : `https://jakubdziworski.github.io/java/2016/04/01/antlr_visitor_vs_listener.html`, « requires less code ».
— TreeAdaptor : `https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687090/Tree+construction`
— Écrire le code Java en anglais, pour être consistant avec le langage, éviter le franglais et les ambiguïtés causés par l'absence d'accents.

Actions à suivre :

— A-t-on besoin de la pile des régions ? Ou les liens père/fils entre tables de symboles suffisent-ils ?
— Placer les méthodes permettant de générer l'AST en PDF dans une autre classe que `App` (par exemple `eu.telecomnancy.PDFGenerator`).
— Faire un Gantt pour la seconde partie du projet
— Penser à utiliser les tests unitaires (ajouter JUnit à Gradle)

Date de la prochaine réunion : Semaine prochaine

Compte-rendu de réunion 2

20 décembre 2019, 18h-18h30

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Commencer à programmer la table des symboles et toutes les classes utiles à son parcours
- Planifier la suite (développement/tests unitaires/rapport PCL1)

Informations échangées :

- Explication du diagramme de classe :
  - Par défaut, ANTRL utilise des `CommonTree` pour construire l'AST. Pour notre compilateur, nous voulons construire des nœuds personnalisés, différents selon le « token » correspondant à chaque nœud. Ainsi, nous construisons un nœud par défaut `DefaultAST` qui hérite de `CommonTree` et des nœuds personnalisés (`RootAST`, `BlockAST`, `VarDecAST`...) qui héritent de `DefaultAST`.
  - Pour indiquer à ANTLR que nous souhaitons utiliser des nœuds personnalisés, nous créons un `ASTAdaptor` qui hérite de `CommonTreeAdaptor` et nous redéfinissons la méthode `create(Token)` qui doit retourner un nœ différent selon le type du token.
  - Maintenant que les nœuds ont des classes différentes, on peut parcourir l'arbre abstrait en utilisant le patron « visiteur ». On crée alors une interface `ASTVisitor`.
  - On implémente un `SemanticAnalysisVisitor` et une `SemanticException`.
- Peut-être ajouter les appels de procédure sans arguments (sans parenthèses) : `http://www.algol60.org/lego/procedure0.a60`

Actions à suivre :

- Implémenter la base des classes selon la distribution établie.

Date de la prochaine réunion : Semaine de la rentrée

Compte-rendu de réunion 3

8 janvier 2020, 14h-16h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand, Timon Fugier *(Secrétaire)*, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

— Commencer à programmer le visiteur pour pouvoir remplir la table des symboles
— Planifier la suite (développement/tests unitaires)

Informations échangées :

— Explication des tests réalisées.
— Mise en application basique via Gradle.
— Organisation et explication sur les méthodes à implémenter.
— Implementation de 5 méthodes dans le visiteur.

Actions à suivre :

— Implémenter la suite du visiteur.

Date de la prochaine réunion : Semaine suivante

Compte-rendu de réunion 4

15 janvier 2020, 16-18h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Zineb Ziani El Idrissi

Absents : Timon Fugier *(Excusé)*

Ordre du jour :

— Avancer dans le développement des contrôles sémantiques
— Validation des contrôles sémantiques concernant les labels et goto (ref : *Description of ALGOL 60*, partie 11.).
— Contrôles sémantiques sur les procédures
— Début des contrôles

Informations échangées :

— Les contrôles sémantiques pour les labels et goto semblent corrects
— Les contrôles sémantiques de base pour les procédures sont terminés mais non testés, il manque encore la vérification des valeurs retournées :

1. s'il y a une valeur retournée directement dans le bloc de la procédure, c'est bon ;
2. sinon, parcourir les `if`, les `for`, les `while`, les blocs imbriqués mais pas les procédures imbriquées ;
3. vérifier aussi qu'aucune variable dans la procédure (paramètres inclus) n'a le même nom que la procédure.

Actions à suivre :

— Réfléchir à la nécessité de la pile des régions ouvertes.

Date de la prochaine réunion : 16 janvier 2019

Compte-rendu de réunion 5

16 janvier 2020, 13h-16h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Zineb Ziani El Idrissi

Absents : Timon Fugier *(Excusé)*

Ordre du jour :

- Écrire des tests unitaires pour valider/débugger nos contrôles sémantique
- Tester un compilateur Algol60 pour les labels et les goto

Informations échangées :

- Il est important d'utiliser des variables intermédiaires, avec des noms compréhensibles, pour pouvoir comprendre du code écrit par quelqu'un d'autre
- Nous avons écrit des tests unitaires pour la déclaration de procédure et les affectations
- Le compilateur de JvanKatwijk sur Github *(accessible ici)* considère les 4 programmes suivants comme valides :

```
begin
  goto x;
  begin x: end
end
```

```
begin
  begin x: end;
  goto x
end
```

```
begin
  x:;
  begin goto x end
end
```

```
begin
  begin goto x end;
  x:
end
```

- Nous avons ré-implémenté les contrôles sémantiques sur les labels et goto de telle sorte que les 4 cas décrits ci-dessus soient sémantiquement valides.
- Concernant les procédures, nous avons décider de forcer la dernière instruction d'un bloc de procédure à être un « return » (une affectation donc la partie gauche est le nom de la procédure).

Actions à suivre :

- Implémenter les contrôles concernant le retour de procédure.
- Ajouter des tests unitaires pour les procédures
- Implémenter les contrôles concernant les expressions mathématiques (Add, Minus, Div, IntDiv...)
- Ajouter des tests unitaires pour les affectations et les expressions mathématiques (`AssignmentTest.java`)

Date de la prochaine réunion : Pendant la semaine de ski ou la suivante

Compte-rendu de réunion 6

29 janvier 2020, 14h-19h30

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- ✓ Ajouter des tests unitaires pour les procédures (valeurs de retour et imbrications) (`ProcDecTest.java`)
- ✓ Implémenter les contrôles concernant le retour de procédure.

- ∽ Ajouter des tests unitaires pour les affectations et les expressions mathématiques et booléennes (`AssignmentTest.java`)
- ✓ Implémenter les contrôles concernant les expressions mathématiques (Add, Minus, Div, IntDiv...)

- ✓ Ajouter des tests pour les appels de procédure (bons types d'arguments, type de retour...) (`ProcCallTest.java`)
- ✓ Implémenter les contrôles sémantiques pour les appels de procédure

- ✓ Remplacer tous les `getChildAST` par `getChild` et supprimer la méthode `getChildAST` de la classe `DefaultAST`. Ensuite, `./gradlew test` et `commit`.

Informations échangées :

- — Il nous reste 12 jours de travail pour terminer les contrôles sémantiques
- — L'analyse syntaxique des expressions booléennes est à compléter

Actions à suivre :

- ✓ Tests unitaires pour les appels de procédures
- ✓ Remplacer les appels sur `typeCompat` par `Types.cannotDoArithmeticperation(a, b)`
- ✓ Décider ensemble des priorités opératoires des opérateurs booléens (modification minime dans la grammaire si besoin). Chercher dans *la documentation*.
- — Implémenter les contrôles sémantiques pour les tableaux (déclaration et affectation) + tests unitaires
- — Implémenter les contrôles sémantiques pour les boucles `while`
- ✓ Implémenter les contrôles sémantiques pour les boucles `for`
- ✓ Ajouter les nœuds correspondants aux opérateurs booléens :

```
Classe          -> Algol60Parser.NOM
-----------------------------------
LogicalValueAST -> LOGICAL_VALUE
AndAST          -> AND
OrAST           -> OR
ImplyAST        -> IMPLY
EquivalentAST   -> EQUIVALENT
GreaterThanAST  -> GREATER_THAN
LessThanAST     -> LESS_THAN
GreaterEqualAST -> GREATER_EQUAL
LessEqualAST    -> LESS_EQUAL
EqualAST        -> EQUAL
NotEqualAST     -> NOT_EQUAL
```

- ✓ Implémenter les contrôles sémantiques pour les opérateurs booléens
- — Ajouter des tests unitaires sur les opérations arithmétiques et booléennes (concordance des types).
- — Ajouter les instructions `pushTable()` et `popTable()` au bons endroits pour créér des tables des symboles dans les parties `then` et `else` est `IfStatement` (D'abord vérifier dans la documentation que ces constructions nécessitent la création d'un environnement).
- ✓ Revoir et tester les contrôles sémantiques du `IfStatement` pour gérer les expressions complexes (simplement appeler `accept(this)` sur la condition et vérifier qu'elle est booléenne).
- — Revoir les contrôles sémantiques des labels d'après la documentation.

Date de la prochaine réunion : Réunions de développement régulières, selon les disponibilités de chacun

Compte-rendu de réunion 7

6 février 2020, 10h-12h & 14h30-17h30

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou

Absents : Zineb Ziani El Idrissi *(Excusée)*

Ordre du jour :

- Modification de la grammaire concernant les boucles `for`.
- Modification des tests et contrôles sémantiques sur les déclarations de labels et les instructions `goto`.
- Implémentation des contrôles sémantiques sur les déclarations de tableaux et affectations de tableaux + tests.

Informations échangées :

- La grammaire a été modifiée pour forcer les déclarations (de variables et procédure) à être situées en début de bloc, avant toute autre type d'instruction.
- Les priorité opératoires ont été renforcées dans la grammaire.
- Les nouvelles structures de boucles `for` sont :

  ```
  for idf := n step n until n do ...
  for idf := n, n, n do ...
  for idf := n while b do ...
  ```

  où les `n` sont des expressions à valeurs entières ou réelles et `b` est une expression à valeur booléenne.
- La sémantique des déclarations de labels fonctionne désormais comme celle des déclarations de valeurs, à ceci près que l'on peut déclarer une label après l'avoir utilisé dans un `goto`. On ne peut pas faire un `goto` sur un label situé dans un sous bloc ou dans un `for` par exemple.
- Concernant les contrôles sur les tableaux, nous vérifions le respect de leurs dimensions (1D, 2D...) et le respect des intervalles de valeurs d'indices quand cela est possible (quand on a définit ces valeurs par des constantes entières).

Actions à suivre :

- ✓ Implémenter les contrôles sémantiques pour les tableaux (déclaration, appel et affectations)
- — Tests sur les tableaux (`ArrayAssignment` et `ArrayCall`)
- — Tests des contrôles sémantiques du `IfStatement` pour des expressions arithmétiques complexes.
- ∽ Tests pour les affectations et les expressions mathématiques et booléennes (concor-

dance des types et opérations).

Date de la prochaine réunion : Réunions de développement régulières, selon les disponibilités de chacun

Compte-rendu de réunion 8

7 février 2020, 10h-12h

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier

Absents : Zineb Ziani El Idrissi, Tony Zhou *(Excusés)*

Ordre du jour :

— Implémentation des contrôles sémantiques sur les appels de tableaux
— Tests et corrections sur les contrôles sémantiques des différents types de boucle `for`

Informations échangées :

— Une analyse de *test coverage* ont montré que nos tests unitaires couvrent 84% des classes, 74% des lignes de code et 76% des méthodes.
— Les contrôles sur les appels de tableaux sont près à être testés

Actions à suivre :

✓ Tests sur les tableaux (`ArrayAssignment` et `ArrayCall`)
✓ Tests des contrôles sémantiques du `IfStatement`
✓ Tests pour les affectations et les expressions mathématiques et booléennes (concordance des types et opérations).
✓ Tests pour les `IfExpression`
— Tests sur sur les constantes entières/réelles (max/min)
— `switch` statement
— Préparation de fichiers tests propres (programmes avec du sens) sans erreurs et d'autres avec erreurs multiples (grande diversité d'erreurs)

Date de la prochaine réunion : Réunions de développement régulières, selon les disponibilités de chacun

Compte-rendu de réunion 9

12 février 2020, 14-18h

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- ✓ Ajout du `switch` statement à la grammaire
- ✓ Contrôles sémantiques sur les `switch` statement
- ✓ Tests sur les déclarations et appels de `switch`
- ∽ Préparation de fichiers tests propres (programmes avec du sens) sans erreurs et d'autres avec erreurs multiples (grande diversité d'erreurs)

Informations échangées :

- Les tableaux et `switch` acceptent des réels en tant qu'indice. Leur valeur sera arrondie à l'entier le plus proche.

Actions à suivre :

- Tests sur les indices réels (sur les tableaux et les `switch`
- ∽ Préparation de fichiers tests propres (programmes avec du sens) sans erreurs et d'autres avec erreurs multiples (grande diversité d'erreurs)
- Nettoyage (vérification qu'il n'y a pas de `System.out.println` ni d'affichage de l'ast non désirés)

Date de la prochaine réunion : 26 février 2020

B Compte-rendus de réunion rédigés pour la partie « génération de code »

Compte-rendu de réunion 10

26 février 2020, 14-18h

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

— Fin de la deuxième partie sur l'analyse sémantique.
— Début de la partie trois sur la génération de code.
— Évaluation de notre analyse sémantique.

Informations échangées :

— Lecture de PDF de cours sur l'assembleur.
— Informations sur la structure à implémenter pour la suite.
— L'évaluation s'est bien passée. Elle a aussi permis de détecter un bug sur les appels récursifs de procédure imbriqués dans un bloc.
— Résolution du bug et tag v2.0.

Actions à suivre :

— Implémenter le design pattern visiteur pour la géneration de code.
— Lire et se renseigner sur le langage assembleur.
— Faire des codes assembleurs basiques.
— Ajouter l'attribut `offset` dans la classe `Symbol`
— Ajouter un argument au constructeur de `CodeGenerationVisitor` : un objet `Assembly` qui permet d'écrire et stocker le code assembleur facilement.

Date de la prochaine réunion : 4 mars 2020

Compte-rendu de réunion 11

4 mars 2020, 14-18h

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

— Implémenter le design pattern visiteur pour la génération de code.
— Lire et se renseigner sur le langage assembleur.
— Faire des codes assembleurs basiques.
— Ajouter l'attribut `offset` dans la classe `Symbol`
— Ajouter un argument au constructeur de `CodeGenerationVisitor` : un objet `Assembly` qui permet d'écrire et stocker le code assembleur facilement.

Informations échangées :

— Implémenter la génération de code pour les éléments basiques

Actions à suivre :

— Finir ProcDec, ProcCall
— Remplacer de `SymbolTable::createChild` par une méthode appropriée pour re-parcourir tout l'ast
— Ajouter une méthode dans Assembly pour charger du code depuis un fichier

Date de la prochaine réunion : Mecredi 11 mars 2020

Compte-rendu de réunion 12

11 mars 2020, 14-18h

Type : Réunion de développement

Lieu : Salle PI

Présents : Loïc Bertrand, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

— Remplacer de `SymbolTable::createChild` par une méthode appropriée pour reparcourir tout l'ast
— Se mettre d'accord sur la position des éléments en pile et des `shift` de variables

Informations échangées :

— Les méthodes `pushTable()` et `popTable()` permettent désormais de reparcourir l'arborescence des tables des symboles, sans recréer des fils.
— Les variables locales auront un `shift` négatif (vers le haut) et les paramètre un `shift` positif (vers le bas de la pile).
— Toute évaluation d'expression (addition, multiplication, variable, constante etc) empile son résultat pour pouvoir le récupérer facilement et de manière récursive. Des optimisations futures pourront supprimer les redondances empilement-dépilement inutiles.

Actions à suivre :

— Opérations simples
— Déclarations de variables simples

Date de la prochaine réunion : Mecredi 11 mars 2020

Compte-rendu de réunion 13

18 mars 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Multiplication, soustraction et division (avec message d'erreur pour la division par zéro) sur le même modèle que l'addition.
- Déclaration de tableau
- Déclaration de booléen
- Déclaration de procédure

Informations échangées :

- L'ordre des opérations s'effectue pour l'instant de droite à gauche (`5-2-1` donne `4` au lieu de `2`)
- Comment gérer les déplacements de tableaux dans la pile ? (Si on les stocke en pile) $\Longrightarrow$ On devrait peut être stocker les *bornes et adresse du « 0 virtuel »* de chaque dimension dans la pile et stocker les valeurs du tableau dans le tas. La taille d'un tableau en pile serait donc de $2 * dim$ (pour calculer les déplacements dans la table des symboles).

Actions à suivre :

- Résoudre bug double affichage avec `outinteger()`
- Procédure `outboolean()`
- Fin de déclaration de procédure

Date de la prochaine réunion : Mercredi 18 mars 2020

Compte-rendu de réunion 14

25 mars 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Réfléchir au chaînage statique (commencer par faire un chaînage simplifié, identique au chaînage dynamique, donc valable uniquement avec 1 niveau d'imbrication et aucune récursivité)
- Implémenter les if-else
- Procédure `outboolean()`
- Opérations logiques sur les booléens (et/ou etc)
- Déclaration de procédure (faire fonctionner une procédure simple puis gérer la valeur de retour)
- Pour les tableaux, étant donné qu'ils sont dynamiques, on ne stocke dans la pile que les dimensions et un pointeur sur sa position dans le tas

Informations échangées :

- *Document concernant la génération de code*
- La procédure `outboolean()` n'est pas native en Algol60, elle peut donc être définie en langage Algol60 et non en assembleur.
- Sans chaînage statique, on devrait quand même pouvoir faire des appels récursifs, mais il ne faut pas utiliser de variables non-locales.

Actions à suivre :

- Implémenter les opérations de comparaison `<`, `<=`, `>`, `>=`, `=` entre nombres entiers (ou « réels »)
- Finir les opérations booléennes
- Écrire des tests unitaires ?
- Réfléchir à l'implémentation du chaînage statique (commencer par implémenter le chaînage statique à 1 niveau, égal au chaînage dynamique et récupérer la valeur les variables non-locales avec une boucle assembleur). Pour cela, ajouter une méthode `Variable::isNonLocal(SymbolTable)`.

Date de la prochaine réunion : Mercredi 1er avril 2020

Compte-rendu de réunion 15

1er avril 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Finir l'implémentation des `if-else`. [**Timon**]
- Implémenter les `if-expression` []
- Continuer à étudier l'implémentation des tableaux dynamiques (dans le tas). Commencer par tester avec les bornes initialisées avec des constantes entières. [**Tony**]
- Implémenter la division entière si ce n'est pas fait puis regarder s'il reste des opérateurs arithmétiques à implémenter. [**Zineb**]
- Revoir l'ordre opératoire dans la grammaire (sans casser le compilateur) [**Loïc**]
- Réfléchir au chaînage statique (commencer par faire un chaînage simplifié, identique au chaînage dynamique, donc uniquement capable de récupérer les variables non-locales de l'environnement immédiatement supérieur. Modifier les valeur initiales des `shift` en conséquence) []

Informations échangées :

- Notre rapide réunion avec Monsieur Oster nous a permis de confirmer que nous sommes dans les temps. Concernant le chaînage statique, nous allons recevoir plus d'informations de sa part pour savoir s'il est vraiment utile.
- Nous avons implémenté les Labels et les Goto.

Actions à suivre :

- Écrire des tests unitaires ? (si on a le temps)

Date de la prochaine réunion : Mercredi 8 avril 2020

Compte-rendu de réunion 16

8 avril 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- — Il nous reste moins de 3 semaines pour terminer le projet ! :'(
- — Les procédures stockent désormais leur résultat dans R1 pour bénéficier de l'optimisation de code. On aurait pu utiliser les registres de travail R0, R1, R2 etc au lieu de commencer à R1.
- ✓ `TODO(Zineb)` : Implémenter les `if-expression` en se basant sur le modèle des `if-statement`
- ✓ `TODO(Tony & Loïc)` : Mettre en place un chaînage statique simplifié (identique au chaînage dynamique) donc invalide lors d'appels récursifs et implémenter l'accès à la valeur des variables non locales (dans `visit(IdentifierAST)`)
- ✓ `TODO(Tony)` : Implémenter l'affectation dans des variables non locales en utilisant le chaînage statique (dans `visit(AssignmentAST)`)
- — `TODO` : Implémenter les tableaux (tout d'abord à 1 dimension), se référer au cours de traduction et à la documentation de `microPIUPK` (dans le jar)
- ✓ `TODO(Timon)` : Corriger l'implémentation des labels (en utilisant `uniqueReference.forName()`)
- ✓ `TODO(Timon)` : Implémenter les switchs
- ✓ `TODO(Loïc)` : Tests unitaires pour l'accès (et affichage) de variables locales et non locales (test du chaînage statique)
- ∽ `TODO(Tony)` : Tests unitaires pour les boucles `for`
- ∽ `TODO(Zineb)` : Tests unitaires pour les `if-expression`
- ✓ `TODO(Tony)` : Tests unitaires pour les opérateurs booléens (faciles à tester : tables de vérité)

Informations échangées :

- — La sauvegarde des registres ne semble pas nécessaire après réflexion. En effet, toute évaluation d'expression étant placée en pile, il n'y a pas de risque que ces valeurs soient perdues. Au lieu de sauvegarder les registres en pile, puis de les restaurer, on empile nos valeurs à ne pas perdre, directement.

— Structure actuelle de la pile avec des blocs simples :

```
                                  begin
                                    integer a, b;
  c |  3  | (BP1 - 2)               a := 1;
BP1 | PB0 | (ch. dyn.)              b := 2;
  b |  2  | (BP0 - 4)               begin
  a |  1  | (BP0 - 2)                 integer c;
BP0 |  #  | (ch. dyn.)               c := 3;
SP0 |  ?  |                          end
                                  end
```

Note : SP0 est le pointeur de pile initial (définit par STACK_ADRS). BP0 et BP1 sont les bases d'environnement. ? est la valeur initiale dans la case mémoire d'adresse STACK_ADRS et # est la valeur initiale du registre R13 (BP).

— Structure actuelle de la pile avec un appel de procédure :

```
res |  10  | (BP1 - 4)      begin
  f |  10  | (BP1 - 2)        integer procedure f(n);
BP1 | BP0 | (ch. dyn.)        value n; integer n;
    | RET |                  begin
  n |  5  | (BP1 + 4)          integer res; res := n * 2;
  b |  0  | (BP0 - 4)          f := result;
  a |  5  | (BP0 - 2)        end;
BP0 |  #  | (ch. dyn.)       integer a, b; a := 5; b := f(a);
SP0 |  ?  |                  end
```

Note : RET est l'adresse de retour, correspondant à la valeur du PC, soit l'adresse de la prochaîne instruction juste après le JSR.

— L'idée pour intéger le chaînage statique (SC), serait de commencer par empiler une deuxième fois la valeur du chaînage dynamique (BP) :

```
res |  10  | (BP1 - 6)
  f |  10  | (BP1 - 4)
SC1 | BP0 | (ch. stat.)
BP1 | BP0 | (ch. dyn.)
    | RET |
  n |  5  | (BP1 + 4)
  b |  0  | (BP0 - 6)  // shift des variables
  a |  5  | (BP0 - 4)  // décalé à -4
SC0 |  #  | (ch. stat.)
BP0 |  #  | (ch. dyn.)
SP0 |  ?  |
```

— Algorithme pour produire le code de recherche d'une variable non locale (on sait déjà que la variable est non locale) :

```
diff <- currentSymTable.level - definitionSymTable.level
asm( R1 <- BP-2 // R1 initialized with current SC )
for i in 0..diff-1 {
  asm( R1 <- (R1)-2 // R1 replaced by the value in R1-2 )
}

// in visit(AssignmentAST) :
asm( R1+shift <- anyValue // store value into variable location)

// in visit(IdentifierAST) :
asm( R1 <- (R1+shift) // load value of variable into R1 )
asm( push R1 on stack, as usual )
```

Note : `asm()` dénote du code assembleur (en pseudo-code) à ajouter au fichier.

Actions à suivre :

— Implémenter le réel calcul du chaînage statique
— Tests unitaires supplémentaires
— Corriger et tester les boucles for (il faut qu'elles fonctionnent quel que soit le niveau d'imbrication)
— Corriger les labels et goto pour qu'ils dépilent l'environnement si nécessaire, autant de fois que nécessaire, avant d'effectuer le saut

Date de la prochaine réunion : Mercredi 15 avril 2020

Compte-rendu de réunion 17

15 avril 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- — Dernière ligne droite !
- ✓ `TODO(Tony)` : Tests unitaires pour les boucles `for`
- ✓ `TODO(Zineb)` : Tests unitaires pour les `if-expression`
- ✓ `TODO(Zineb)` : Tests unitaires pour les `label/goto` multiniveaux
- ✓ `TODO(Zineb)` : Tests unitaires pour les `switch/goto` multiniveaux
- ✓ `TODO(Loïc)` : Implémenter le réel calcul du chaînage statique
- ✓ `TODO(Tony)` : Corriger et tester les boucles for (il faut qu'elles fonctionnent quel que soit le niveau d'imbrication)
- — `TODO(Zineb)` : Corriger les labels et goto pour qu'ils dépilent autant d'environnements que nécessaire avant d'effectuer le saut (semble très rapide à implémenter)
- — `TODO(Non prioritaire)` : Corriger les switch et goto pour qu'ils dépilent autant d'environnements que nécessaire avant d'effectuer le saut (semble un peu plus difficile à implémenter)
- — `TODO(Timon)` : Implémenter les tableaux (tout d'abord à 1 dimension), se référer au cours de traduction et à la documentation de `microPIUPK` (dans le jar) pour savoir comment mettre des éléments dans le tas
- — `TODO(Non prioritaire)` : Simplifier Pow10 : calculer la valeur à la compilation et l'empiler, comme dans `visit(IntAST)`

Informations échangées :

- — Extension du `goto` sur des labels extérieurs au bloc :

```
begin                                répéter endEnvironment() n fois
  integer i;                         où n = currentLevel - labelLevel
  start:                               |
  outstring(1, "start");               |
  begin                                |
    i := i + 1;                        v
    if i < 5 then      |   LDW SP, BP
      goto start; <---|   LDW BP, (SP)+
  end                  |   JMP #start_-$-2
end
```

— Chaînage statique souhaité (sur un programme qui pose problème pour l'instant) :

```
                                  begin (level 0)
bar |  6  | (BP2 - 4)              integer a;
SC2 | BP0 |                        real procedure bar;
BP2_| BP1 |__bar (level 1)         begin (level 1)
    | RET |                          bar := a * 2;
foo |  0  | (BP1 - 4)              end;
SC1 | BP0 |                        real procedure foo;
BP1_| BP0 |__foo (level 1)         begin (level 1)
    | RET |                          foo := bar() - 1;
  a |  3  | (BP0 - 4)              end;
SC0 |  !  |                        a := 3;
BP0_|  #  |_main (level 0)         outreal(1, foo());
SP0 |  ?  |                      end
```

— Algorithme de calcul du chaînage statique actuel :

```
called_procedure_
    // Prepare environment (Assembly::newEnvironment))
    R1 <- BP  // Save a copy of old BP in R1
    push(BP)  // Push old BP on stack (dynamic chaining)
    BP <- SP  // New BP is the current SP
    push(R1)  // Push old BP on stack (static chaining)
```

— Algorithme de calcul du chaînage souhaité (seulement pour les appels de procédure, la mise en place des blocs anonymes reste inchangée) :

```
    // Caller (visit(ProcCallAST))
    push(arguments)

    R1 <- BP  // Put BP into R1
    int n = currentSymTable.level - proc.level;
    duplicate n times {
        R1 <- R1 - 2  // Put SC into R1
        R1 <- (R1)    // Put upper BP into R1
    }
    // We end up with the correct BP address

    jsr(called_procedure_)
    pop(arguments)

called_procedure_
    // Prepare environment (Assembly::newProcEnvironment))
    push(BP)  // Push old BP on stack (dynamic chaining)
    BP <- SP  // New BP is the current SP
    push(R1)  // Push SC computed by the caller
```

Actions à suivre :

— Penser à faire une liste des chose qu'on n'a pas eu le temps de faire (switch avec goto sur un label d'un autre niveau etc.)

Date de la prochaine réunion : Vendredi 17 avril 2020

Annexe :

Un programme avec trois niveaux d'imbrication différents :

```
                                    begin (level 0)
                                      integer a;
SP->bar |  6  | (BP3 - 4)
    SC3 | BP0 |                       real procedure bar;
BP->BP3_| BP2 |_bar  (level 1)        begin (level 1)
        | RET |                         bar := a * 2
    foo |  0  | (BP2 - 4)             end;
    SC2 | BP1 |
    BP2_| BP1 |_foo  (level 2)        real procedure big;
        | RET |                       begin (level 1)
    big |  0  | (BP1 - 4)               real procedure foo;
    SC1 | BP0 |                         begin (level 2)
    BP1_| BP0 |_big  (level 1)            foo := bar() - 1
        | RET |                         end;
      a |  3  | (BP0 - 4)               big := foo()
    SC0 |  !  |                       end;
    BP0_|  #  |_main (level 0)
    SP0 |  ?  |                       a := 3;
                                      outreal(1, big())
                                    end
```

Compte-rendu de réunion 18

17 avril 2020, 10-12h et 14-16h30

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand*(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Code :
  - Observation : `outinteger(1, 32767)` imprime le bon résultat mais `outinteger(1, 32768)` imprime `-32768`. On en déduit que les nombres entiers signés sont stockés sur 16 bits dans cet assembleur.
  - Info : Le signe '+' n'apparaît plus quand on affiche des nombres positifs
  - Info : maintenant que le chaînage statique est terminé, nous avons validé le niveau 3 d'implémentation.
  - `TODO(Timon)`: Continuer l'implémentation des tableaux dynamiques (mais non flexibles)
  - ✓ `TODO(Tony)`: Exécuter les tests unitaires en mode 'coverage' pour déceler les méthodes qui n'ont pas été testées
  - ✓ `TODO(Tony)`: Simplifier Pow10 : calculer la valeur directement à la compilation et l'empiler (ce ne sont que des constantes réelles ou entières).
  - `TODO(Non prioritaire)`: Implémenter `integer entier(real)`

- Rapport et compte-rendu :
  - ✓ `TODO(Tony)`: Compléter le tableau concernant les tâches effectuées par chacun d'entre nous
  - ✓ `TODO(Loïc & Zineb)`: Compléter la description des contrôles sémantiques
  - `TODO`: Partie 'Contrôles sémantiques', mettre des exemples simples et courts de programmes en Algol60 et leur traduction en code assembleur (enlever toutes les définitions de fonction non nécessaires, peut-être aussi ignorer les alias de registres non nécessaires).

Informations échangées :

- Éléments non implémentés ou partiellement implémentés :
  - `goto` sur un indice de switch correspondant à une étiquette qui n'est pas définie dans le même block que ce switch
  - Tableaux à dimensions multiples ? ? ? ? ? ? ? ? ?
  - Passage de paramètres par nom et gestion des réels (bonus)
  - Gestion des entrées clavier (non demandée)
  - Mot-clé `own` non implémenté

— Schéma de la pile après déclaration d'un tableau :

```
begin
  integer array tab[2:4, 5:6];
  tab[2,5] := 1;
  tab[4,6] := 5;
end

@tb |  1  | tab[2,5] (@tb + 0)  <- HP0
    |     | tab[2,6] (@tb + 2)
    |     | tab[3,5] (@tb + 4)
    |     | tab[3,6] (@tb + 6)     TAS
    |     | tab[4,5] (@tb + 8)
    |  5  | tab[4,6] (@tb + 10)
    ~~~~~~~
      tab
b22 |  6  | (BP0 - 12)
b21 |  5  | (BP0 - 10)
b12 |  4  | (BP0 - 8)
b11 |  2  | (BP0 - 6)              PILE
tab | @tb | (BP0 - 4)
SC0 |     |
BP0 |     |
SP0 |     |
```

Actions à suivre :

— Finir `ArrayCall` et `ArrayAssignment`
— Créer un programme de démonstration avec tous les éléments du langage (sauf les tableaux pour l'instant) : `src/test/resources/codegen/demo/prog1.alg`.
— Déplacer l'affichage du message d'erreur et l'arrêt du programme dans un 'fonction', sur le même modèle que la fonction `div0`. Le définir dans un fichier et l'ajouter dans la classe `PredefinedCode`. Appeler cette fonction `index_oob` par exemple, **sans underscore** à la fin de ce label. Il suffira alors de faire un `JMP #index_oob-$-2` pour afficher le message et quitter le programme.

Date de la prochaine réunion : Mercredi 22 avril 2020

Compte-rendu de réunion 19

22 avril 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- — Rappel : les accept(this) utilisent R1 => pas que R1 :'(
- ✓ `TODO(Timon)` : Question : Le traitement des erreurs d'indice dans les switch-goto est-il fait ?
- — Code :
  - ✓ `TODO(Loïc)` : Finir `ArrayAssignment` (sans vérification d'indices)
  - ✓ `TODO(Timon)` : Implémenter `ArrayCall` (sans vérification d'indices)
  - ✓ `TODO(Loïc)` : Nettoyer `ArrayDec` et déplacer l'affichage du message d'erreur dans une 'fonction', sur le même modèle que la fonction `div0`. Le définir dans un fichier et l'ajouter dans la classe `PredefinedCode`. Appeler cette fonction `index_oob` par exemple, **sans underscore** à la fin de ce label. Il suffira alors de faire un `JMP #index_oob-$-2` pour afficher le message et quitter le programme.
  - ✓ `TODO(Timon)` : Ajouter la vérification d'indice dans `ArrayAssignment`
  - ✓ `TODO(Timon)` : Ajouter la vérification d'indice dans `ArrayCall`
  - ∽ `TODO(Tony)` : Créer un programme de démonstration principal avec tous les éléments du langage (`src/test/resources/codegen/demo/main.alg`). L'ajouter aux tests unitaires (`DemoTest.java`)
  - — `TODO` : Créer un fichier `READ_ME` comme demandé dans le mail de Mme Collin.
  - ✓ `TODO(Loïc)` : Créer les scripts bash `./build.sh` qui génère un jar exécutable de notre compilateur, `./compile.sh fichier.alg` qui permet de compiler un fichier Algol60 en .iup et `./run.sh fichier.iup` qui permet d'exécuter le code avec microPIUPK.
  - ✓ `TODO(Zineb & Timon)` : Implémenter l'algorithme pour les indices indices de tableau à n dimensions avec des registres (Difficile).
- — Rapport :
  - — `TODO` : Partie 'Contrôles sémantiques', mettre des exemples simples et courts de programmes en Algol60 et leur traduction en code assembleur (enlever toutes les définitions de fonction non nécessaires, peut-être aussi ignorer les alias de registres non nécessaires).

Informations échangées :

— Calcul du décalage dans un tableau à n dimensions (Timon) :

$$@[i_1, i_2 \dots i_N] = @impl + \sum_{i=1}^{N} \left[\left(\prod_{j=1}^{i-1} (sup_{N-j+1} - inf_{N-j+1} + 1) \right) * (index_{N-i+1} - inf_{N-i+1}) \right] * 2$$

```
exemple : integer array tab[2:4, 5:6]
mémoire : [(2,5), (2,6), (3,5), (3,6), (4,5), (4,6)]
           @impl   +2     +4     +6     +8    +10

tab[2, 5] = @impl + [(2-2)(6-5+1) + (5-5)] * 2 = @impl + 0
tab[3, 6] = @impl + [(3-2)(6-5+1) + (6-5)] * 2 = @impl + 6
tab[4, 6] = @impl + [(4-2)(6-5+1) + (6-5)] * 2 = @impl + 10

Ancien algorithme (ne fonctionne pas) :

push(base sack address of table)
R6 <- 0
for (int n = 0; n <= dims - 2; n++) {
    R5 <- nth index            // accept => SIDE EFFECTS :'(
    R2 <- (SP)-(1+n*2)*2       // l(n)
    R3 <- (SP)-(1+(n+1)*2)*2   // l(n+1)
    R4 <- (SP)-(1+(n+1)*2+1)*2 // u(n+1)
    R2 <- R5 - R2              // i(n) - l(n)
    R3 <- R4 - R3              // u(n+1) - l(n+1)
    R3 <- R3 + 1               // u(n+1) - l(n+1) + 1
    R2 <- R2 * R3
    R6 <- R6 + R2              // accumulator
}
R2 <- (dims-1)th index         // accept => SIDE EFFECTS :'(
R3 <- (SP)-(1+(dims-1)*2)*2    // l(k-1)
R2 <- R2 - R3                  // i(k-1) - l(k-1)
R2 <- R2 * 2
R1 <- @impl + R2
// R1 est l'adresse de la case demandée
```

Actions à suivre :

—

Date de la prochaine réunion : 24 avril 2020

Compte-rendu de réunion 20

24 avril 2020, 14-18h

Type : Réunion de développement à distance

Lieu : Discord

Présents : Loïc Bertrand *(Secrétaire)*, Tony Zhou, Timon Fugier, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Informations :
  - `Assembly::push` corrigé (bug '`STW STW...`')
  - `ArrayTest::testTwoDimArray` corrigé (tous les tests passent pour l'instant)
  - Ajout de `codegen/custom_bubble_sort.alg`, presque fonctionnel
  - Ajout de quelques sauvegardes de registres, exemple :

```
// We don't want to loose @impl :
asm.push("R3", "Save @impl on stack");
index.accept(this); // Push index on stack
asm.pop("R1", "Pop index into R1");
asm.pop("R3", "Pop @impl into R3");
```

- Code :
  - ∽ `TODO(Tony)` : Créer notre 'plus beau programme' de démonstration avec tous les éléments du langage (`src/test/resources/codegen/demo/main.alg`). L'ajouter aux tests unitaires (`DemoTest.java`)
  - ∽ `TODO(Tous)` : Créer 4 ou 5 autres programmes que notre correcteur pourra tester
  - ✓ `TODO(Loïc & Timon)` : Confirmer les calculs d'accès aux 'lower bounds' avec Timon.
  - ✓ `TODO(Timon)` : Tester l'accès aux tableaux non locaux et la vérification de leurs bornes

- Rapport :
  - `TODO(IMPORTANT)` : Partie 'Contrôles sémantiques', mettre des exemples simples et courts de programmes en Algol60 et leur traduction en code assembleur (enlever les éléments superflus du code assembleur)

Informations échangées :

- Programmes de démonstration :
  - `main.alg` : calculs vectoriels
  - `factorial.alg`
  - `visibility.alg` : accès aux variables non locales

— `bubble_sort.alg` : génération de nombres randoms puis bubble sort
—

C Estimation du temps de travail pour la partie « contrôles sémantiques »

Membre	Tâche	Date	Temps
Loïc	Participation réunion 1	11/12/2019	3h00
Tony	Participation réunion 1	11/12/2019	3h00
Zineb	Participation réunion 1	11/12/2019	3h00
Timon	Participation réunion 1	11/12/2019	3h00
Loïc	Recherche de possibilités de conception	11/12/2019	1h30
Loïc	Diagramme de Gantt PCL2	14/12/2019	0h30
Loïc	Complétion rapport PCL1 + Répartition travail	15/12/2019	1h00
Loïc	Implémentation de classes pour l'AST	20/12/2019	0h30
Loïc	Participation réunion 2	20/12/2019	0h30
Tony	Participation réunion 2	20/12/2019	0h30
Zineb	Participation réunion 2	20/12/2019	0h30
Timon	Participation réunion 2	20/12/2019	0h30
Loïc	Ajout de tests unitaires	27/12/2019	1h00
Zineb	Ajout de package et création de classes	30/12/2019	1h00
Timon	Ajout de classes pour l'AST	30/12/2019	1h00
Loïc	Séance PCL individuelle	07/01/2020	2h00
Loïc	Participation réunion 3	08/01/2020	2h00
Tony	Participation réunion 3	08/01/2020	2h00
Zineb	Participation réunion 3	08/01/2020	2h00
Timon	Participation réunion 3	08/01/2020	2h00
Loïc	Participation réunion 4	15/01/2020	2h30
Tony	Participation réunion 4	15/01/2020	2h30
Zineb	Participation réunion 4	15/01/2020	2h30
Loïc	Participation réunion 5	16/01/2020	4h00
Tony	Participation réunion 5	16/01/2020	4h00
Zineb	Participation réunion 5	16/01/2020	4h00
Loïc	Refactoring de visit(ProcDevAST)	17/01/2020	2h00
Loïc	Participation réunion 6	30/01/2020	4h30
Tony	Participation réunion 6	30/01/2020	4h00
Zineb	Participation réunion 6	30/01/2020	3h30
Timon	Participation réunion 6	30/01/2020	2h30
Timon	Ajout de classes pour l'AST (opérateurs)	02/02/2020	0h30
Loïc	Refactoring et amélioration des contrôles	02/02/2020	2h00
Tony	Modification de grammaire	05/02/2020	2h00
Timon	Adaptation de visit(IfStatementAST)	05/02/2020	0h30
Loïc	Réunion de développement 7 (matin)	06/02/2020	2h00
Tony	Réunion de développement 7 (matin)	06/02/2020	2h00
Timon	Réunion de développement 7 (matin)	06/02/2020	2h00
Loïc	Réunion de développement 7 (après-midi)	06/02/2020	3h00
Timon	Réunion de développement 7 (après-midi)	06/02/2020	3h00
Loïc	Réunion de développement 8 (matin)	07/02/2020	2h00
Timon	Réunion de développement 8 (matin)	07/02/2020	2h00

Timon	Écriture de tests (après-midi)	07/02/2020	1h00
Zineb	Écriture de tests (après-midi)	07/02/2020	1h30
Zineb	Écriture de tests	10/02/2020	0h30
Loïc	Participation réunion 9	12/02/2020	4h00
Tony	Participation réunion 9	12/02/2020	4h00
Zineb	Participation réunion 9	12/02/2020	4h00
Timon	Participation réunion 9	12/02/2020	4h00

D Estimation du temps de travail pour la partie « génération de code »

Membre	**Tâche**	**Date**	**Temps**
Loïc	Participation réunion 10	26/02/2020	4h00
Tony	Participation réunion 10	26/02/2020	4h00
Zineb	Participation réunion 10	26/02/2020	4h00
Timon	Participation réunion 10	26/02/2020	4h00
Loïc	Participation réunion 11	04/03/2020	4h00
Tony	Participation réunion 11	04/03/2020	4h00
Zineb	Participation réunion 11	04/03/2020	4h00
Timon	Participation réunion 11	04/03/2020	4h00
Loïc	Participation réunion 12	11/03/2020	4h00
Tony	Participation réunion 12	11/03/2020	4h00
Zineb	Participation réunion 12	11/03/2020	4h00
Timon	Participation réunion 12	11/03/2020	4h00
Loïc	Débuggage	15/03/2020	0h30
Loïc	Participation réunion 13	18/03/2020	4h00
Tony	Participation réunion 13	18/03/2020	4h00
Zineb	Participation réunion 13	18/03/2020	4h00
Timon	Participation réunion 13	18/03/2020	4h00
Loïc	Participation réunion 14	25/03/2020	4h00
Tony	Participation réunion 14	25/03/2020	4h00
Zineb	Participation réunion 14	25/03/2020	4h00
Timon	Participation réunion 14	25/03/2020	4h00
Zineb	If statement	25/03/2020	1h00
Timon	Code pour les blocs et création d'environnement	25/03/2020	1h00
Tony	Travail sur les booléens	25/03/2020	1h00
Zineb	Modifications de ProcDec	25/03/2020	0h30
Tony	Opérations booléennes	30/03/2020	1h00
Loïc	Participation réunion 15	01/04/2020	4h00
Tony	Participation réunion 15	01/04/2020	4h00
Zineb	Participation réunion 15	01/04/2020	4h00
Timon	Participation réunion 15	01/04/2020	4h00
Loïc	Correction de l'associativité opératoire	01/04/2020	1h00
Tony	Ajout de boucle while	02/04/2020	1h00
Loïc	Prévision de la séance suivante	02/04/2020	1h30
Loïc	Correction de bugs	02/04/2020	1h00

Loïc	Participation réunion 16	08/04/2020	4h00
Tony	Participation réunion 16	08/04/2020	4h00
Zineb	Participation réunion 16	08/04/2020	4h00
Timon	Participation réunion 16	08/04/2020	4h00
Tony	Ajout d'outinteger et début de procCall	11/04/2020	1h00
Zineb	Code pour Add et Int	11/04/2020	1h00
Loïc	Participation réunion 17	15/04/2020	4h00
Tony	Participation réunion 17	15/04/2020	4h00
Zineb	Participation réunion 17	15/04/2020	4h00
Timon	Participation réunion 17	15/04/2020	4h00
Loïc	Participation réunion 18	17/04/2020	4h30
Tony	Participation réunion 18	17/04/2020	4h30
Zineb	Participation réunion 18	17/04/2020	2h30
Timon	Participation réunion 18	17/04/2020	4h30
Loïc	Participation réunion 19	22/04/2020	4h00
Tony	Participation réunion 19	22/04/2020	4h00
Zineb	Participation réunion 19	22/04/2020	04h00
Timon	Participation réunion 19	22/04/2020	04h00
Loïc	Participation réunion 20	24/04/2020	04h00
Tony	Participation réunion 20	24/04/2020	04h00
Zineb	Participation réunion 20	24/04/2020	02h00
Timon	Participation réunion 20	24/04/2020	02h00