Télécom Nancy 2A

Projet de Compilation

Projet Compil : Compte-rendu Partie II

Auteurs :
M. Mathias OBERTI
M. Yann BOUGEARD
M. Marcus REIS DE MORAIS
M. Warren LATA

Encadrant :
Suzanne Colin
Sebastien Da-Silva
Gerald Oster

Table des matières

1 Introduction

L'objectif de ce projet est de réaliser un compilateur pour le langage de programmation ALGOL60. Nous avons utilisé ANTLR pour écrire la grammaire ainsi que pour générer l'arbre syntaxique. Nous avons ensuite réalisé l'analyse sémantique et la génération du code assembleur au format MicroPIUP/ASM en language Java. Ce rapport présente l'implémentation de la table des symboles et des contrôles sémantiques ainsi que la génération du code assembleur. Il fait suite à un précédent rapport concernant la spécication de la grammaire et réalisation de l'analyse lexicale et syntaxique.

2 Analyse sémantique

2.1 Table des symboles

La table des symboles est composée de 4 types symboles : les fonctions, les variables, les étiquettes et les arrays. Les symboles sont stockés dans une HashMap<String,Symbol>, la clé est leur nom. Dans la classe des symbolfonction, on prend comme attribut une linkedList<VariableSymbol> pour stocker les paramètres, et chaque table des symboles qui contient les symboles de sont environnement. Chaque instance de tabledesSymbole est composée d'un attribut Hashmap<Integer,tabledesSymboles> pour stocker les diérents blocks. Enn les diérentes TDS sont empilées dans une Stack<>. Les relations entres les différentes régions sont représentées par des relations père/fils.

2.2 Notion d'environnement en algol

La table des symboles est divisée en sous-tables correspondant à une région du programme. les structures suivantes représentent une nouvelle région :

- Le corps d'un BEGIN.
- Le corps d'une fonction.
- Le corps d'une boucle for.

2.3 Exemple de programme et leur tds

```
BEGIN
INTEGER i;
INTEGER a;
INTEGER b;
INTEGER temp;
  INTEGER PROCEDURE FIBO(N);
        VALUE N;
        INTEGER N;
        BEGIN
            a := 0;
            b := 1;
            FOR i:=1 STEP 1 UNTIL N DO
                BEGIN
                    temp := a + b;
                    a := b;
                    b := temp;
                    outinteger(1,temp);
            END
        END;
  FIBO(8)
END
```

FIGURE 1 – Test Complexe

```
Root|NAME                |OFFSET|SYMB          |TYPE          |
|N                  |-4    |VAR           |INTEGER       |
|print              |0     |FUNC          |VOID          |
|FIBO               |0     |FUNC          |INTEGER       |
|i                  |2     |VAR           |INTEGER       |
|a                  |4     |VAR           |INTEGER       |
|b                  |6     |VAR           |INTEGER       |
|temp               |8     |VAR           |INTEGER       |

FIBO|NAME                |OFFSET|SYMB          |TYPE          |
|N                  |-4    |VAR           |INTEGER       |
```

FIGURE 2 – Test Complexe

```
Root|NAME                |OFFSET|SYMB          |TYPE
|TESTLABEL          |0     |LABEL         |
|print              |0     |FUNC          |VOID          |
|Wawa               |0     |LABEL         |
|L                  |2     |VAR           |INTEGER       |
|i                  |4     |VAR           |INTEGER       |
|k                  |6     |VAR           |INTEGER       |
```

FIGURE 3 – Test Complexe

```
BEGIN
    INTEGER n;
    INTEGER m;
    INTEGER pt;
    INTEGER i;
    n := 100;
    m := 5;
    pt := 0;

    FOR i := 0 STEP 1 UNTIL n DO
        BEGIN
            IF (i / m EQUAL 0) THEN
            BEGIN
                pt := pt + 1;
                outinteger(1, i)
            END
        END;
    outinteger(1, pt)
END
```

FIGURE 4 – Test Complexe

```
BEGIN
 INTEGER L;
 INTEGER i;
 INTEGER j;
 INTEGER k;
 INTEGER ARRAY narr[1:5];
 INTEGER ARRAY bb[1:5,1:5];
 INTEGER PROCEDURE SELECT(N);
        VALUE N;
        INTEGER N;
        BEGIN
           INTEGER S;
           outinteger(1,S);
           N :=N+1;
           FOR i := 0 STEP 1 UNTIL 5 DO
           BEGIN
            outinteger(1,N)
           END;
           outinteger(1,N)
        END;
  L:=12;
  i:=1;
  j:=2;
  j:=SELECT(L);
  FOR i := 1 STEP 1 UNTIL 5 DO
  BEGIN
  outinteger(1,L);
  IF (i LESS 3) THEN L:=11 ELSE L:=12;
  IF (i / 15 EQUAL 1) THEN j:=2
  END;
  narr[1]:= 17;
  L:=bb[1,2]*3;
  outinteger(1,L);
  L:=L+narr[1];
  outinteger(1,L);
  L:=L-3;
  outinteger(1,L)
END
```

FIGURE 5 – Test Complexe

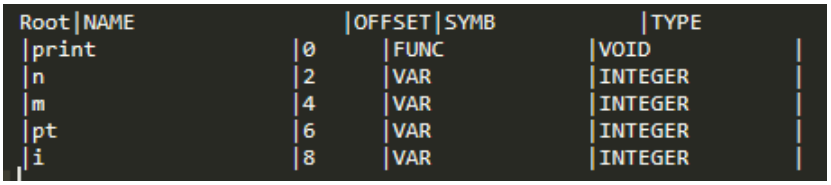
```
Root|NAME                |OFFSET|SYMB          |TYPE
|N                       |-4    |VAR           |INTEGER        |
|SELECT                  |0     |FUNC          |INTEGER        |
|print                   |0     |FUNC          |VOID           |
|L                       |2     |VAR           |INTEGER        |
|i                       |4     |VAR           |INTEGER        |
|j                       |6     |VAR           |INTEGER        |
|k                       |8     |VAR           |INTEGER        |
|narr                    |18    |ARRAY         |INTEGER        |
|bb                      |68    |MATRICE       |INTEGER        |

SELECT|NAME              |OFFSET|SYMB          |TYPE
|N                       |-4    |VAR           |INTEGER        |
|S                       |2     |VAR           |INTEGER        |
```

FIGURE 6 – Test Complexe

```
BEGIN
INTEGER L;
INTEGER i;
INTEGER k;
  TESTLABEL :
    BEGIN
        IF (i LESS 3) THEN L:=11 ELSE L:=12;
        k:=4;
       outinteger(1,L)
    END;
    Wawa :
    BEGIN
    FOR i := 1 STEP 1 UNTIL 5 DO
    BEGIN
    outinteger(1,L)
    END
    END
    ;
  L:=12;
  i:=1;
  outinteger(1,L);
  GOTO TESTLABEL;
  GOTO Wawa
END
```

FIGURE 7 – Test Complexe



Root	NAME		OFFSET	SYMB	TYPE	
	print	0		FUNC	VOID	
	n	2		VAR	INTEGER	
	m	4		VAR	INTEGER	
	pt	6		VAR	INTEGER	
	i	8		VAR	INTEGER	

FIGURE 8 – Test Complexe

2.4 Côntroles sémantiques implémentés

Lors du parcours de l'AST, on teste les instructions et soulève les erreurs sémantiques grâce à différents affichages. Voici la liste des contrôles sémantiques implémentés.

- "Redefining function" : vérifie qu'une fonction n'est pas redéfinie.
- "Redefining variable" : vérifie qu'une variable n'est pas redéfinie.
- "Redefining label" : vérifie qu'un label n'est pas redéfinie.
- "Redefining array" : vérifie qu'un array n'est pas redéfinie.
- "Affectation impossible car types incompatibles" : vérifie que le membre gauche et droits de l'assignation sont du même type.
- "Nom de variable "n'est pas défini" : vérifie que la variable à laquelle on veut affecter une valeur existe vraiment.
- "Redefining " + "nom de variable" : message renvoyé si la variable que l'on souhaite définir est déjà déclarée.
- "La fonction a été appelée avec le mauvais nombre de paramètres (" + realSize + " instead of " + size + ")." : vérifie que la fonction est appelée avec le bon nombre d'arguments.
- "The argument " + " is called with the wrong type + " instead of " : vérifie les types des arguments avec lesquels est appelée la fonction.
- "The parameter " + " is already defined in the function. : vérifie qu'on ne déclare pas deux fois le même argument.

Pour les for :

- "Redefining variable " : vérifie si la variable n'est pas deja déclarée.
- "Integer expected in for expression" : vérifie que la condition est de type int.

Pour les tableaux :

- "la variable d'accès n'est pas de type INT" : vérifie que lorque l'on veut avoir accès à une case la variable est entier.
- "Le tableau n'existe pas " : vérifie que le tableau est déclaré.

Pour un while :

- "Int expected in while condition expression : vérifie que la condition est de type int.

Pour le body d'un for ou d'un while :

- "Void expected in while block expression" : vérifie que le type de retour du body est un void.

Pour un If :

- "Integer expected in if condition expression : vérifie que la condition est de type int.
- " Different block types" : vérifie que le then et le else renvoie le même type.
- "The logical connectives AND and OR should be used with two integers." : vérifie que les opérateurs AND et OR sont utilisés entre deux entiers.
- "Mathematical inequalities or comparisons should be done between two integers. " : vérifie que les comparaisons sont réalisées entre deux entiers.
- " Unary operation should be done with an integer :") : vérifie qu'une opération unaire de négation est faite sur un entier.

3 Compilation

Le principe de la génération de code est le suivant : On va sur un noeud ensuite on suit un schéma de traduction (que nous allons expliquer).

Nous avons pour notre implémentation en java une classe RegistreManager qui permet de gérer les registres (si on peut on utiliser un ou non). Quand on a besoin d'un registre on demande au registremanager. Si tous les registres sont utilisés, on enregistre les registres actuels dans la pile pour avoir accès à une mémoire plus grande.

Nous avons des registres par défaut : R0 pour les résultats, SP (stack pointer), BP (base pointer) et un registre usuelle WA qui nous sert pour la gestion des liens entre environnements pour retrouver le base pointer initial.

3.1 Schéma de traduction du noeud Begin

Lorque nous avons un noeud BEGIN, il peut y avoir des noeuds de déclaration à gauche puis des instructions.

- On réserve de la mémoire nécessaire sur la pile.
- On traite les noeuds DEC.
- On traite les noeuds suivant et on enlève la mémoire réservée.

3.2 Schéma de traduction du noeud PROCEDURE

- On commence par ajouter l'étiquette de la fonction.
- On empile le contenu du registre BP STW BP,-(SP).
- On charge contenu SP dans BP LDW BP, SP.
- On réserve sur SP la place nécessaire pour les variables.
- Ensuite on se déplace sur le derniers fils pour traiter les instructions(corps de la fonction).
- On réserve de la mémoire nécessaire sur la pile.
- On abandonne les informations locales ; SP pointe sur l'ancienne valeur de BP : LDW SP, (BP).
- On dépile ancienne valeur de BP dans BP ; SP pointe sur adresse de retour : LDW BP, (SP)+.
- On quitte le sous-programme avec RTS.

3.3 Schéma de traduction affectation

Dans notre AST nous avons des noeuds assigment pour les affectations. Le fils gauche correspond à la variable qui reçoit et à droite la variable qu'on affecte. on respecte le schémas suivant :

- On commence par déterminer si la variable correspond à l'environnement courant ou non : ainsi on utilisera soit le registre utilitaire WA (qui contiendra le base pointer du bon environnement) pour se déplacer au BP de l'environnement courant. (On vérifie si la variable appartient à l'environnement courant, si ce n'est pas le cas on charge WA avec le bon BP.)
- Ensuite on détermine l'offset de la variable.
- Ensuite on va à droite, la valeur de ce noeud est stockée dans R0.
- Enfin en écrit la valeur contenue dans R0 à l'adresse de la variable de gauche en s'y déplaçant.

3.4 Schéma de traduction d'un retour de fonction

On considère ici une instruction de type $NomProcedure := expr$, qui est l'assignation d'un retour de fonction. Ici, la procédure retournera la valeur de *expr*,*expr* pouvant être n'importe quelle expression logique, mathématique, valeur de variable etc.

- On vérifie dans la TDS que le membre gauche de l'assignation est bien le nom de la procédure englobante.
- On réserve un registre qui contiendra la valeur de retour.
- On récupère le déplacement permettant de retrouver la valeur de *expr* dans la pile. (C'est l'offset du résultat de l'expression.)
- On charge dans le registre précédemment réservé cette valeur avec LDW *Registre*, (BP)+ *Déplacement*.

3.5 Schéma de traduction If

- On commence la compilation avec un appel à la génération de code pour l'expression booléenne contenue dans le "IF" et on met le résultat sur un registre.
- On prend le résultat et on fait un "JMP" conditionnel au label du bloc intérieur du if si le résultat sauvegardé sur le registre montre que la condition du if a été satisfaite. Si la condition n'a pas été satisfaite, on fait un "JMP" inconditionnel au label de fin du "IF".

- Ensuite, on met le label du bloc intérieur du "IF" et on génère le code du bloc.
- Après le code du bloc, on met le label de fin du "IF", de façon de que le JMP inconditionnel saute tout le bloc.
- Si le "IF" a un bloc "ELSE" après, on génère le code du bloc "ELSE" à la suite du label de fin du "IF".

3.6 Schéma de traduction For

À chaque fois qu'on a un noeud "FOR" sur l'AST, on sait qu'il sera composé par une variable qu'on va utiliser pour la boucle, la liste d'éléments avec les valeurs pour affecter la variable, le "STEP" et/ou le "WHILE". Donc, on doit générer les expressions pour chaque élément sur cette liste. Pour faire cela, on suit le schéma suivant :

- On commence avec la génération de l'expression d'affectation de la variable qu'on utilisera pour le loop.
- Ensuite, on parcourt chaque élément sur la liste de valeurs qui affecteront la variable à chaque itération du loop :
  - Si l'élément de la liste est juste une valeur entière, on met cette valeur sur un registre, on sauve sur la pile la nouvelle valeur de la variable du "FOR" et on parcourt le bloc.
  - Si l'élément de la liste est une construction "STEP", on met la valeur initiale du "STEP" sur un registre, on affecte la variable avec cette valeur et ensuite, on met le label pour le début de la boucle. On fait un "CMP" pour comparer le valeur actuelle de la variable avec le valeur maximale qu'elle doit prendre sur la boucle et on utilise le "BGT" pour faire le saute à la fin de la boucle si le valeur actuel est plus grande que la maximale. Après cette comparaison, on génère le bloc qui sera parcouru et, à la fin du bloc, on incrémente la variable avec un ADD et on revient au label de début de la boucle. Postérieurement à ce code, on met le label de fin du "STEP".
  - Si l'élément est une construction "WHILE", on affecte la variable de la boucle avec le valeur précisé et on met le label de début du "WHILE" juste après. Ensuite, on parcourt l'expression booléenne et on fait un saut conditionnel au label à la fin du "WHILE" si l'expression booléenne n'est plus vrai. Après le code de ce saut, on génère le code du bloc qui sera parcouru et on revient au label de début de la boucle. À la fin de ce code, le label de fin du "WHILE" est écrit.
- On répète le procédure précédente pour chaque élément sur la liste d'affectation du "FOR".

3.7 Schéma de traduction labels

Les labels sont traités d'une manière similaire à celle des procédures, on va parcourir tout le code pour récupérer les labels et leur attribuer un environnement en début de génération de code. On procède donc de la sorte :

- On récupère le nom du label pour créer son espace mémoire.
- On traite les instructions stockées dans le bloc du label correspondant.
- On ajoute un RTS en fin de bloc pour fermer le label et revenir à l'environnement courant lors de l'éxécution du code.

On a ainsi établi des zones pour chaque label. Lors de l'utilisation d'un GOTO, il suffit "jump" à l'adresse mémoire du label correspondant.

4 Gestion de projet

4.1 Identifications des forces et menaces

Forces	Faiblesses
Avance du groupe	Travail à distance pour Yann Bougeard
Expérience de Marcus en Assembleur	Mathias n'a pas fait d'assembleur depuis 2 ans
Table des symboles et contrôles sémantiques solides	Complexité syntaxique de certaines instructions algol

Nous avons cherché à identifier les forces et faiblesses du groupe au moment d'aborder la partie délicate de la génération de code. Cela nous a permis entre autre de proposer une répartition des tâches intelligente qui tire partie des capacités de chacun.

4.2 Répartition des rôles

Nous avons donc décidé de la répartition des tâches suivantes pour la partie contrôles sémantiques :

Tâche	Lata	Reis De Morais	Oberti	Bougeard
Opérations				X
Labels				X
GOTO			X	
Assignements			X	
Déclarations hors label	X			
Ifs/For		X		
Appels de fonction et accès tableaux			X	
Linking et gestion des tds	X			

Et pour la partie génération de code :

Tâche	Lata	Reis De Morais	Oberti	Bougeard
Opérations				X
Labels/GOTO				X
Prints				X
Assignements			X	
Déclarations de fonctions et tableaux	X			
Appel de fonction	X			
Linking TDS et generateur de code	X			
Gestion accès tableau et matrice	X		X	
Visibilité des variables et gestion des registres	X	X		
Ifs		X		
For Step		X		
For Complexe		X		
Retours de fonction			X	

4.3 Charges de travail

Ci dessous sont présentées les charges de travail en heures par jalon et par membre.

Jalon	Lata	Reis De Morais	Oberti	Bougeard
Table des Symboles	25	20	14	10
Generation de code	40	25	10	20
Deboggage	20	13	10	15
Rapport	10	3	2	2
Total	95	61	36	57

4.4 Comptes rendus de réunion de la partie génération de code

Compil 2019-2020 - Réunion 1 de la génération de code

Minutes for February, 27, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Debriefing de l'évaluation des contrôles sémantiques
2. Répartition des tâches

Evaluation des sur la partie sémantique

Tout s'est bien passé, il faut juste corriger la portée de déclaration des variables. Le reste est solide, le groupe semble être dans la moyenne haute en terme de quantité des contrôles.

Répartition du travail

Il a été décidé de conserver dans les grandes lignes la répartition choisie pour les contrôles sémantiques, à savoir :

- Marcus M ;
  - Ifs et For
- Yann B ;
  - Opérations logiques et arithmétique ainsi que les GOTO
- Mathias O ;
  - Appels de fonctions et assignations
- Warren L ;
  - Déclarations

Next Meeting: March, 5, 2020

Points à évoquer

1. Peaufinage de la répartition et questions sur l'assembleur

Compil 2019-2020 - Réunion 2 de la génération de code

Minutes for March, 5, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Définition d'une strategie
2. Points de détail

Stratégie et discussion sur les "niveaux"

Le groupe s'est entendu sur les différents niveaux et leurs signification

Warren porte à l'attention du groupe que par expérience, mieux vaux ne pas prendre de retard et être régulier dans les commits

Discussion diverses sur l'implémentation du FOR, les tableaux et le fonctionnement de microPIUP pour mettre à niveau Marcus qui ne l'a jamais utilisé

La priorité est mise sur les déclarations, les assignations et les fonctions non récursives, pour avoir le niveau octroyant la moyenne. Le reste sera du bonus.

Next Meeting: March, 12, 2020

Points à évoquer

1. Avancement

Compil 2019-2020 - Réunion 3 de la génération de code

Minutes for March, 12, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement

Points techniques

Pour l'instant seul la moitié du groupe a touché au code, il s'agit donc surtout de clarifier certains points

Warren informe le groupe que le niveau 1 est en bonne voie. Il reste encore quelques cas délicats mais cela devrait être réglé rapidement.

Discussion diverses sur le mode batch de microPIUP et les déplacements.

Avec le confinement à venir et la fermeture de l'école, des membres du groupes sont en déplacement et le workflow va donc en être un peu perturbé.

Next Meeting: March, 17, 2020

Points à évoquer

1. Avancement

Compil 2019-2020 - Réunion 4 de la génération de code

Minutes for March, 17, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement

Avancement

Le niveau 1 est terminé et le 2 est en cours.

Mathias porte à l'attention de groupe que Warren a déjà très bien travaillé et qu'il est temps de faire chacun sa part.

Il reste quelques problèmes au niveau des opérations logiques.

Marcus pose quelques questions à Warren sur son code concernant les switch.

A faire :

- Marcus M ;
  - Ifs et For
- Yann B ;
  - Opérations logiques et arithmétique ainsi que les GOTO
- Mathias O ;
  - retours de fonction
- Warren L ;
  -

Next Meeting: March, 24, 2020

Points à évoquer

1. Avancement
2. Tableaux

Compil 2019-2020 - Réunion 5 de la génération de code

Minutes for March, 23, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement
2. Tableaux
3. Goto

Avancement

Le niveau 2 est terminé, il manque les etiquettes et la récusivité pour le niveau 3. Marcus doit encore finir les STEP.

Tableaux

Warren informe qu'il va se pencher sur les tableaux.

GOTO

Warren et Yann discutent de l'implémenation des GOTO et des labels : le principe est à peu près le même que pour les fonctions, il faut donc prendre modèle sur le code d'un appel de fonction.

A faire :

- Marcus M ;
  - Fin des for
- Yann B ;
  - GOTO
- Mathias O ;
  - retours de fonction
- Warren L ;
  - Tableaux

Next Meeting: March, 27, 2020

Points à évoquer

1. Avancement
2. GOTO et Retours de fonction

Compil 2019-2020 - Réunion 6 de la génération de code

Minutes for March, 27, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement
2. Retours de fonctions
3. Goto

Avancement

Les GOTO et retours de fonctions avancent bien que lentement. Warren point le fait que ce n'est pas une notion facile mais qu'une fois les retours implémentés, la récusrivité devrait marcher assez vite.

Retours de fonction

Mathias fait part de la difficulté à faire fonctionner les retours en s'aidant du PDF envoyé par Madame Collin. Il informe qu'il a corrigé un problème dans la méthode getOffset qui engendrait une erreur lorsque la variable était un retour de fontion, à cause d'un cas non géré.

GOTO

Yann avance sur les GOTO et informe qu'il s'est mis à niveau sur le travail des autres et la structure du projet.

A faire :

- Marcus M ;
  - For avec plusieurs éléments sur la liste.
- Yann B ;
  - GOTO
- Mathias O ;
  - retours de fonction
- Warren L ;
  - Tableaux

Next Meeting: April, 1, 2020

Compil 2019-2020 - Réunion 6 de la génération de code

Minutes for March, 27, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement
2. Retours de fonctions
3. Goto

Avancement

Les GOTO et retours de fonctions avancent bien que lentement. Warren point le fait que ce n'est pas une notion facile mais qu'une fois les retours implémentés, la récusrivité devrait marcher assez vite.

Retours de fonction

Mathias fait part de la difficulté à faire fonctionner les retours en s'aidant du PDF envoyé par Madame Collin. Il informe qu'il a corrigé un problème dans la méthode getOffset qui engendrait une erreur lorsque la variable était un retour de fontion, à cause d'un cas non géré.

GOTO

Yann avance sur les GOTO et informe qu'il s'est mis à niveau sur le travail des autres et la structure du projet.

A faire :

- Marcus M ;
  - For avec plusieurs éléments sur la liste.
- Yann B ;
  - GOTO
- Mathias O ;
  - retours de fonction
- Warren L ;
  - Tableaux

Next Meeting: April, 1, 2020

Compil 2019-2020 - Réunion 8 de la génération de code

Minutes for April, 10, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Retours de fonctions
2. Goto
3. Autres

Retours de fonction

Mathias a réussi à régler le problème des retours de fonctions, qui venait d'un registre non unlocké.

GOTO

Yann a presque finit les GOTO.

FOR et Tableaux

Marcus finalise les FOR et pourra ensuite peaufiner la partie "Tableaux" de Warren.
A faire :

- Marcus M ;
  - FOR et Tableaux
- Yann B ;
  - GOTO
- Mathias O ;
  -
- Warren L ;
  -

Next Meeting: April, 17, 2020

Points à évoquer

1. Tests

Compil 2019-2020 - Réunion 9 de la génération de code

Minutes for April, 17, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement et tableaux
2. Tests

Avancement

Tout est plus ou moins finalisé, les fonctions recursives et les labels fonctionnent.

Tableaux

Les tableaux à une dimension fonctionnent mais pas les matrices. Warren et Mathias vérifient la formule d'accès à une case du tableau qui semble être bonne. Le problème venait en fait d'un verrouillage de registre

Mathias fait remarquer que l'accès aux tableaux ne fonctionne pas avec des bornes littérales.

Warren répond que c'est normal et que tout ne peut pas être implémenté

Test

Le groupe se réparti les tâches pour la fin du projet et les tests A faire :

- Marcus M ;
  - Tests divers
- Yann B ;
  - Tests divers
- Mathias O ;
  - Tests sur tableaux et matrices
- Warren L ;
  - Rapport

Next Meeting: April, 23, 2020

Points à évoquer

1. Dossier et git

Compil 2019-2020 - Réunion 10 de la génération de code

Minutes for April, 23, 2020

Present: M. Morais, Y. Bougeard, M. Oberti, W. Lata

Absent: *Aucun*

Ordre du jour

1. Avancement et tableaux
2. Tests

Avancement

Suite au mail de Madame Collin, il faut se répartir les dernières taches :

- Marcus M ;
  - Tests pour le rendu
- Yann B ;
  - Automatisation shell
- Mathias O ;
  - Nettoyage du dépot git et partie GDP du rapport
- Warren L ;
  - Tests pour le rendu