



Langages et compilation - CS444 - Esisar 4A IR&C

Examen 2021-22 - **NE PAS ENLEVER LES AGRAFES**

Durée totale : 1 heure 30

Consignes :

- **Noircir ou bleuir** la/les cases du QCM, sans dépasser !
- Pour corriger les réponses au QCM : effacer proprement la case.
- Dans les parties **rédigées**, les carrés gris (prof) sont pour la correction, merci de ne rien écrire dedans.
- Le barème est **indicatif**.
- Vous pouvez dégrafer et garder la feuille d'accompagnement.

Notez vos NOM et Prénom ici :

.....

1 Questions rapides

Question 1 ♣ (1 point) Parmi les étapes suivantes du cycle de vie du logiciel, quelles sont celles qui font partie du *front-end* d'un compilateur ?

- | | |
|--|--|
| <input type="checkbox"/> Le coloriage du graphe de conflits. | <input type="checkbox"/> L'analyse syntaxique |
| <input type="checkbox"/> Le typage | <input type="checkbox"/> La génération de code |
| <input type="checkbox"/> L'analyse lexicale | |

Question 2 ♣ (1 point) Dans notre compilateur du cours, quelles parties faudrait-il modifier si l'on voulait changer de machine cible ?

- | | |
|---|--|
| <input type="checkbox"/> Le typeur | <input type="checkbox"/> conflits |
| <input type="checkbox"/> L'allocation de registre | <input type="checkbox"/> La génération de code |
| <input type="checkbox"/> L'algorithme de coloriage du graphe de | <input type="checkbox"/> L'analyse syntaxique |

Question 3 ♣ (1 point)

Extrait d'un tutorial Java officiel :

In Java SE 7 and later, any number of underscore characters (`_`) can appear anywhere between digits in a numerical literal. This feature enables you, for example, to separate groups of digits in numeric literals, which can improve the readability of your code.

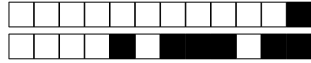
```
long creditCardNumber = 1234_5678_9012_3456L;  
long socialSecurityNumber = 999_99_9999L;
```

Quel(s) aspect(s) du compilateur Java sont impactés par cette nouvelle fonctionnalité ?

- | | |
|---|--|
| <input type="checkbox"/> L'analyse lexicale | <input type="checkbox"/> La table des symboles |
| <input type="checkbox"/> L'analyse sémantique | <input type="checkbox"/> La génération de code |
| <input type="checkbox"/> L'allocation de registre | <input type="checkbox"/> L'analyse syntaxique |

Question 4 (1 point) Si on compare l'ensemble des grammaires LL(1) et l'ensemble des grammaires LR(1) :

- | | |
|--|--|
| <input type="checkbox"/> LR(1) est inclus dans LL(1) | <input type="checkbox"/> LL(1) est inclus dans LR(1) |
| <input type="checkbox"/> Ils ont une intersection non nulle, mais aucun n'est inclus dans l'autre. | <input type="checkbox"/> LL(1) et LR(1) sont disjoints |



2 Grammaires, analyse syntaxique, arbres

On considère la grammaire G suivante :

- E -> E+T
- E -> T
- T -> (E)
- T -> id

2.1 Analyse Syntaxique

On s'intéresse à la réalisation d'un analyseur SLR(1).

Question 5 (2 points) Quel est l'arbre de **dérivation** pour l'expression **a+b+(c)** ? Quel est l'arbre de syntaxe abstrait correspondant ? On dessinera les deux arbres côte à côte en précisant bien les différences.

0 1 2 3 4 5 Prof

L'arbre de dérivation possède des noeuds interne qui sont les non terminaux des règles utilisées. L'AST ne possède pas ces terminaux, et le sucre syntaxique ainsi que les parenthèses sont supprimés.

Question 6 (1 point) Calculez les SUIV de E et de T.

0 1 2 3 4 5 Prof

.....
.....
.....

Question 7 (1 point) On considère la grammaire augmentée G' correspondant à G. Calculez l'ensemble d'items I0 défini comme la "fermeture" de l'ensemble d'items $\{E' \rightarrow \cdot E\}$

0 1 2 3 4 5 Prof

.....
.....
.....



.....
.....

Question 8 (1 point) Calculez l'ensemble d'items I3 défini par l'opération "transition" appliquée à I0 et au terminal '('.

0 1 2 3 4 5 Prof

.....
.....
.....
.....

Question 9 (1 point) On suppose qu'on a construit l'automate LR(0), puis la table de l'analyseur SLR(1) donnée ci-dessous.

| | Action | | | | | Succ. | |
|---|--------|----|----|----|-----|-------|---|
| | + | (|) | id | \$ | E | S |
| 0 | | d3 | | d4 | | 1 | 2 |
| 1 | d5 | | | | acc | | |
| 2 | r2 | | r2 | | r2 | | |
| 3 | | d3 | | d4 | | 6 | 2 |
| 4 | r4 | | r4 | | r4 | | |
| 5 | | d3 | | d4 | | | 7 |
| 6 | d5 | | d8 | | | | |
| 7 | r1 | | r1 | | r1 | | |
| 8 | r3 | | r3 | | r3 | | |

Qu'est-ce qu'une grammaire SLR(1)? Peut-on affirmer que G l'est?

0 1 2 3 4 5 Prof

.....
.....
.....

Question 10 (2 points) Montrez le déroulement de l'analyse SLR(1) de la chaîne a + (b + c) (où a, b et c sont des id bien sûr).

0 1 2 3 4 5 Prof

.....
.....
.....
.....
.....
.....



exemple est une *expression* qui s'évaluera en 42 si x vaut 0 et 70 sinon. On se restreint au test d'égalité à 0.

Question 13 (1 point) Modifier la grammaire précédente pour ajouter cette extension.

0 1 2 3 4 5 Prof

.....
.....
.....

On considère maintenant que l'on a réalisé la construction de l'arbre de syntaxe abstrait et que nos expressions sont conformes à la grammaire abstraite suivante (celle du cours, avec notre extension).

| | |
|----------------|-----------------------|
| $e ::= c$ | <i>constant</i> |
| x | <i>variable</i> |
| $e + e$ | <i>addition</i> |
| $e \times e$ | <i>multiplication</i> |
| $ifz(e, e, e)$ | <i>ifzero (NEW)</i> |

La grammaire abstraite des instructions demeure inchangée (cf feuille d'accompagnement).

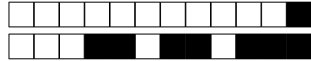
Question 14 (1 point) En s'inspirant des règles de typage fournies dans la feuille d'accompagnement, écrire une règle de typage pour ce nouveau constructeur d'expression.

0 1 2 3 4 5 Prof

.....
.....
.....
.....

Question 15 (2 points) En utilisant votre règle de typage précédente et celles de la feuille d'accompagnement, montrer que l'affectation $y := ifzero(x - 8) \text{ then } 18 \text{ else } 42 + x$ est bien typée sous l'environnement $\Gamma : x \mapsto int, y \mapsto int$. On fera attention à bien faire un arbre de preuve correct.

0 1 2 3 4 5 Prof



Question 16 (2 points) Sur le modèle de la règle de génération de code des expressions, écrire une règle de génération de code pour ce nouveau constructeur d'expression. *On justifiera soigneusement la réponse!*

0 1 2 3 4 5 Prof

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

3 Une attribution

On considère la grammaire :

```
start -> tree
tree -> Node(int, treelist)
tree -> int

treelist -> tree treelist
treelist -> eps
```

Cette grammaire permet de représenter des arbres n-aires, par exemple Node (42 12 1515 17) représente l'arbre de racine 42 avec 3 fils 12, 1515, 17.

Question 17 (2 points) Écrire une attribution qui permet de décider si un arbre reconnu par la grammaire est binaire (une feuille est un arbre binaire, un noeud est un arbre binaire ssi il a deux 2 fils qui sont des arbres binaires) **On précisera bien le type des attributs propagés.**

0 1 2 3 4 5 Prof

.....
.....
.....
.....
.....
.....
.....
.....



4 Compilation de mini langage impératif

On considère l'instruction MiniC suivante : `if (x + y < 3) x = x + 2;`

Question 18 (2 points) En utilisant les règles de génération de code fournies, et en considérant la mémoire $y \mapsto temp_0, x \mapsto temp_1$, remplir les trous dans le code 3 adresses correspondant :^a.

0 1 2 3 4 5 Prof

```
# code généré
add temp_2, temp_1, temp_0
li temp_3, 3
li temp_4, 0
# compléter TODO
```

```
lbl_end_relational_3_main:
    beq temp_4, zero, lbl_else_2_main
    # code correspondant à x = x+2 TODO
```

```
# fin TODO à remplir
lbl_else_2_main:
lbl_end_if_1_main:
```

a. On rappelle que le code trois adresses a le même jeu d'instruction que la machine RISC-like du cours (cf feuille d'accompagnement), et il utilise des registres temporaires au lieu des registres physiques.

5 Dataflow et allocation de registre

On utilise dans cet exercice la machine RISC-like du cours (cf la feuille d'accompagnement). Un compilateur génère le code 3 adresses suivant : les chaînes $temp_i$ désignent des temporaires. Seuls les temporaires $temp_1$ et $temp_3$ sont supposés vivants à la sortie du code :

```
1  mv temp3 r3
2  mv temp1 r1
3  mv temp2 r2
4  li temp4 0
5  mv temp5 temp1 ; attention c'est temp5 <--- temp1
6  loop: add temp4 temp4 temp2
7      sub temp5 temp5 1
8      condjump (temp5, ">", 0, loop) ; test
9  mv temp6 temp4
10 mv temp8 temp3
11 print signed temp6
12 print signed temp8
```



Question 19 (1 point) Générer le code final pour la ligne 5 avec la stratégie d'allocation "tout en mémoire" (ie tous les registres temporaires sont alloués en mémoire, à des adresses différentes) en supposant un offset 2 pour $temp_5$ et 3 pour $temp_1$. On utilisera sp comme registre correspondant au bas de pile (attention la pile a des adresses décroissantes), s_6 et s_7 pour accéder aux éléments de pile.. La syntaxe des load et store est dans la feuille d'accompagnement attention à la taille des mots machine.

0 1 2 3 4 5 Prof

.....

.....

.....

.....

Question 20 (1 point) Remplir les cases du tableau en mettant une étoile si la variable considérée est vivante en entrée de la ligne considérée. on ne demande pas le détail des calculs.

0 1 2 3 4 5 Prof

| ℓ | $temp_1$ | $temp_2$ | $temp_3$ | $temp_4$ | $temp_5$ | $temp_6$ | $temp_7$ | $temp_8$ |
|--------|----------|----------|----------|----------|----------|----------|----------|----------|
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | | | |



Question 21 (1 point) Tracer le graphe d'interférences et le colorier avec l'algorithme du cours et 4 couleurs (0 = rouge, 1 = vert, 2 = bleu, 3 = noir). On écrira aussi la pile de coloriage en montrant clairement le haut de pile. On rappelle qu'il faut empiler les sommets de plus bas degré en premier, que les degrés sont mis à jour à chaque fois que l'on empile, et que si il y a un choix c'est le temporaire de numéro le plus petit qui est empilé en premier ($temp_1 < temp_2 < temp_3 < temp_4 < temp_5 < temp_6 < temp_7 < temp_8$). Enfin le coloriage se passe dans le sens inverse.

0 1 2 3 4 5 Prof

.....

.....

.....

.....

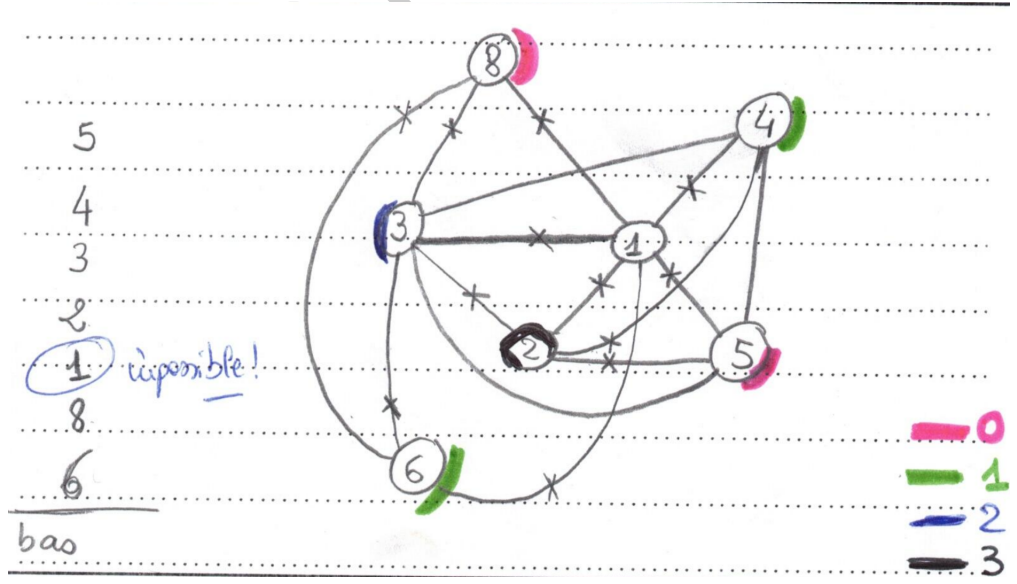
.....

.....

.....

.....

.....

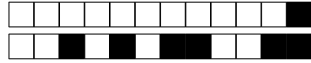


Question 22 (1 point) Montrer qu'il n'est pas possible d'utiliser 3 registres physiques uniquement.

0 1 2 3 4 5 Prof

.....

.....



On se propose maintenant de stocker la variable $temp_3$ en mémoire à l'adresse pointée par sp (offset 0), et d'allouer aux autres variables les registres suivants :

- variable $temp_8, temp_7, temp_2$: registre t_2
- variable $temp_6, temp_1, temp_5$: registre t_3
- variable $temp_4$: registre t_4

Question 23 (2 points) Par quelles instructions est remplacée la ligne 6 (et la ligne 10) du code 3 adresses dans le code finalement généré? On utilisera s_6 et s_7 pour réaliser les calculs intermédiaires.

0 1 2 3 4 5 Prof

.....
.....
.....
.....

DRAFT

Mini-while abstract syntax

Mini-while:

Boolean expr:

$$b ::= \begin{array}{l} true \\ false \\ b \text{ or } b \\ b \text{ and } b \\ \dots \end{array} \quad \begin{array}{l} constant \\ constant \\ or \\ and \end{array}$$

Numerical expressions:

$$e ::= \begin{array}{l} c \\ x \\ e + e \\ e \times e \\ \dots \end{array} \quad \begin{array}{l} constant \\ variable \\ addition \\ multiplication \end{array}$$

$$S(Smt) ::= \begin{array}{l} x := e \\ skip \\ S_1; S_2 \\ \text{if } b \text{ then } S_1 \text{ else } S_2 \\ \text{while } b \text{ do } S \text{ done} \end{array} \quad \begin{array}{l} assign \\ do \text{ nothing} \\ sequence \\ test \\ loop \end{array}$$

Typing

For mini-while Then a typing judgment for expressions is $\Gamma \vdash e : \tau \in Basetype = \{int, bool\}$. Statements have type void. $\bowtie \in \{+, *, -, /, \spadesuit\}$; $\spadesuit \in \{=, <, \leq, >, \geq\}$.

$$\begin{array}{c} \Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int \quad ct \in \mathbb{Z} \quad \Gamma(x) = \tau \\ \hline \Gamma \vdash e_1 \bowtie e_2 : int \quad \Gamma \vdash ct : int \quad \Gamma \vdash ct : bool \quad \Gamma \vdash x : \tau \\ \hline \Gamma \vdash x : \tau \quad \Gamma \vdash e : \tau \quad \Gamma \vdash b : bool \quad \Gamma \vdash S_1 : void \quad \Gamma \vdash S_2 : void \\ \hline \Gamma \vdash x := e : void \quad \Gamma \vdash \text{if } b \text{ then } S_1 \text{ else } S_2 : void \end{array}$$

$$\begin{array}{c} \Gamma \vdash b : bool \quad \Gamma \vdash S : void \quad \Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int \quad \Gamma \vdash S_1 : void \quad \Gamma \vdash S_2 : void \\ \hline \Gamma \vdash \text{while } b \text{ do } S \text{ done} : void \quad \Gamma \vdash e_1 \spadesuit e_2 : bool \quad \Gamma \vdash S_1; S_2 : void \end{array}$$

Mini-ISA of the course RISC Machine

- Arithmetical and logical instructions:

```
add t1, t0, 3      ; t1 ← - t0 + 3
addi t1, t1, 15   ; t1 ← - t1 + 15
sub t1, t2, t3     ; t1 ← - t2 - t3
```

- Conditional jumps can be made with:

```
loop:
; ...
bneqz t1, loop ; if t1 different from 0, jump to loop
```

or

```
loop:
; ...
bne t1, zero, loop ; if t1 different from register with value 0, jump
```

(*bneqz* is non zero, you also have *bgt*(>) and *ble*(≤))

- Unconditional jump to label :

```
j foo
; ...
f bar;
```

- Read from memory ($t_2 \leftarrow Mem[t_1 + offset]$) where *offset* is an immediate (constant!) **mult** of 8 (bytes).

```
ld t2, offset(t1)
; ...
; Write to memory (mem[t1 + offset] ← t2):
sd t2, offset(t1)
```

- Convention:** use t_i for “general purpose registers” and s_i for “reserved registers to spill”.

3 address code generation

$new_tmp : () \rightarrow \mathbb{N}$ and $new_label : () \rightarrow \mathbb{N}$.

$GenCodeExpr :$

| | |
|-------------|---|
| c | <pre> dest ← new_tmp () code.add(" li dest, c") return dest </pre> |
| x | <pre> # get the temporary associated to x. reg ← symbol_table[x] return reg </pre> |
| $e_1 + e_2$ | <pre> t1 ← GenCodeExpr(e.1) t2 ← GenCodeExpr(e.2) dest ← new_tmp() code.add(" add dest, t1, t2") return dest </pre> |
| $e_1 - e_2$ | <pre> t1 ← GenCodeExpr(e.1) t2 ← GenCodeExpr(e.2) dest ← new_tmp() code.add(" sub dest, t1, t2") return dest </pre> |
| $true$ | <pre> dest ← new_tmp () code.add(" li dest, 1") return dest </pre> |
| $e_1 < e_2$ | <pre> dest ← new_tmp () t1 ← GenCodeExpr(e1) t2 ← GenCodeExpr(e2) endrel ← new_label() code.add(" li dest, 0") # if t1 >= t2 jump to endrel code.add(" bge endrel, t1, t2") code.add(" li dest, 1") code.addLabel(endrel) return dest </pre> |

$GenCodeSmt :$

| | |
|-----------------------------|--|
| $x = e$ | <pre> dest ← GenCodeExpr(e) loc ← symbol_table[x] code.add(" mv loc, dest") </pre> |
| $S1; S2$ | <pre> # Just concatenate codes GenCodeSmt(S1) GenCodeSmt(S2) </pre> |
| $if\ b\ then\ S1\ else\ S2$ | <pre> ltest ← new_label() lendif ← new_label() t1 ← GenCodeExpr(b) # if the condition is false, jump to else code.add(" beq ltest, t1, 0") GenCodeSmt(S1) # then code.add(" j lendif") code.addLabel(ltest) GenCodeSmt(S2) # else code.addLabel(lendif) </pre> |
| $while\ b\ do\ S\ done$ | <pre> ltest ← new_label() lendwhile ← new_label() code.addLabel(ltest) t1 ← GenCodeExpr(b) code.add(" beq lendwhile, t1, 0") GenCodeSmt(S) # execute S code.add(" j ltest") # and jump to the test code.addLabel(lendwhile) # else it is done. </pre> |