POLYTECHNIQUE MONTRÉAL

Questionnaire examen final

LOG3210

Sigle du cours

Identification de l'étudiant(e)		
Nom :	**Prénom :**	
Signature :	**Matricule :**	**Groupe :**

Réservé

Sigle et titre du cours	*Groupe*	*Trimestre*
LOG3210 – Éléments de langages et compilateurs	**Tous**	**20163**
Professeur	***Local***	***Téléphone***
Ettore Merlo, responsable	**M-4105**	**5758 / 5193**

Jour	*Date*	*Durée*	*Heures*
Lundi	**12 décembre 2016**	**2 h 30**	**9 h 30 à 12 h 00**

Documentation	*Calculatrice*	
☐ Aucune	☒ Aucune	**Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.**
☒ Toute	☐ Toutes	
☒ Voir directives particulières	☐ Non programmable	

Directives particulières

💻 Joindre l'énoncé à votre cahier d'examen en inscrivant votre nom et votre matricule.

Important

Cet examen contient **6** questions sur un total de **9** pages **(excluant cette page)**

La pondération de cet examen est de **50** **%**

Vous devez répondre sur : ☐ le questionnaire ☐ le cahier ☒ les deux

Vous devez remettre le questionnaire : ☒ oui ☐ non

L'étudiant doit honorer l'engagement pris lors de la signature du code de conduite.

Question 1 – Conversion de types (15 points)

1.1) Considérez les déclarations suivantes :

byte i, j, k;

char c, d, e;

int x, y, z;

double a, b, c;

short u, v, w;

Écrivez le code à trois adresses correspondant aux expressions suivantes en utilisant les **figures 1.1** et **1.2** :

1.2.1) m = k + j

1.2.2) n = x + i

1.2.3) o = u + d

$$
\begin{aligned}
E \rightarrow E_1 + E_2 \quad \{\ & E.type = max(E_1.type, E_2.type); \\
& a_1 = widen(E_1.addr, E_1.type, E.type); \\
& a_2 = widen(E_2.addr, E_2.type, E.type); \\
& E.addr = \textbf{new}\ Temp\,(); \\
& gen(E.addr\ '='\ a_1\ '+'\ a_2);\ \}
\end{aligned}
$$

Figure 1.1

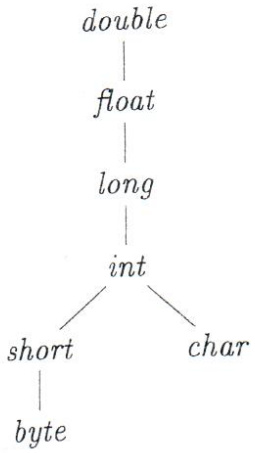


Figure 1.2

m = k + j :

k est un byte, j est un byte. Aucun cast de nécessaire. Le code généré est donc

t1 = k + j

n = x + i :

x est un int, i est un byte. E.type = max(int,byte) = int. Le code généré est donc :

t1 = x + (int) i

o = u + d :

u est un short, d est un char. E.type = max(short, char) = int. Le code généré est donc :

t1 = (int) u + (int) d

Question 2 – Génération de code intermédiaire (20 points)

En considérant les règles sémantique en Figure 6.19, 6.36 et 6.37 aux pages 3 et 4, écrivez le code intermédiaire à 3-adresses correspondant au programme suivant :

```
1 : x = 10
2: z = 0
3 : if (x > 0)
4 :    while ( x > 0)
5:   {
6 :      z = z + y
7 :      x = x + 1
8:   }
9 : else
10 :      z = y
```

Figure 2.1

PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathbf{id} = E\ ;$	$S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ - E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

Figure 6.19: Three-address code for expressions

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

Figure 6.36: Syntax-directed definition for flow-of-control statements.

PRODUCTION	SEMANTIC RULES
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow\ !\ B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}'\ B.true)$ $\mid\mid gen('\texttt{goto}'\ B.false)$
$B \rightarrow$ **true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$ **false**	$B.code = gen('\texttt{goto}'\ B.false)$

Figure 6.37: Generating three-address code for booleans

```
        x = 10
        z = 0
        if x > 0 goto L2
        goto L3
L2:L5:  if x > 0 goto L6
        goto L1
L6:     t1 = z + y
        z = t1
L4 :    t2 = x + 1
        x = t2
        goto L5
        goto L1
L3:     z = y
L1:     (fin du programme)
```

Notez qu'il serait possible d'optimiser grandement ce code, mais pas avec le SDD donné !

Question 3 – Optimisations locales (20 points)

Considérez le bloc de base suivant :

1: e = a + b

2: d = b – d

3: c = d – c

4: a = a + b

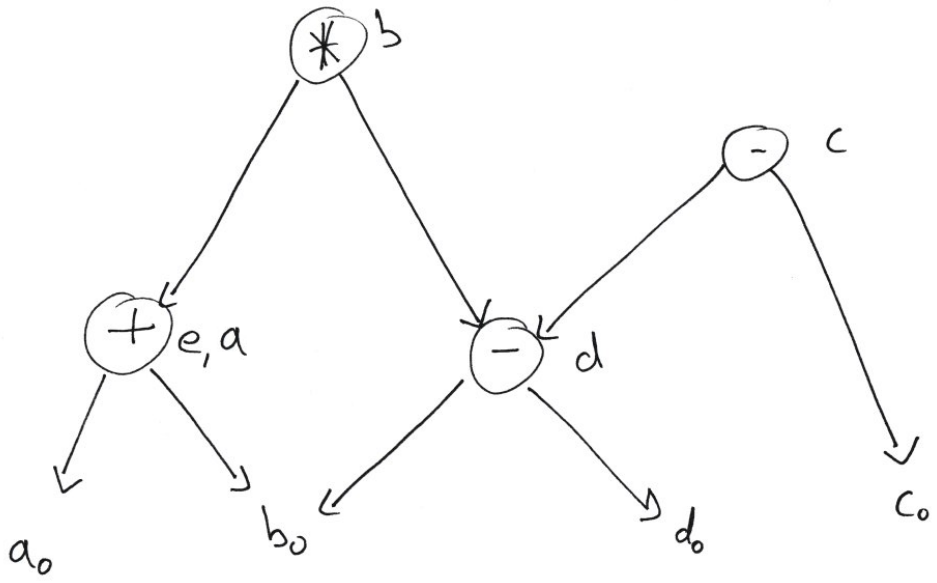
5: b = e * d

3.1) DESSINEZ le graphe orienté sans cycle (DAG) correspondant au bloc de base et aux sous-expressions.

Commençons par le tableau des définitions les plus récentes :

Variable	a	b	c	d	e
1: e = a + b	a0	b0	c0	d0	e0
2: d = b – d	a0	b0	c0	d0	1
3: c = d – c	a0	b0	c0	2	1
4: a = a + b	a0	b0	3	2	1
5: b = e * d	4	b0	3	2	1
fin	4	5	3	2	1

Ensuite, le DAG :



3.2) CALCULEZ la fonction « PROCHAINE_UTILISATION » (« NEXT-USE ») pour chaque variable utilisée avant et après chaque instruction à 3-adresses en supposant qu'à la sortie du bloc de base le NEXT-USE soit celui indiqué. REMPLISSEZ le **tableau 3.1**.

a	b	c	d	e
1	1	3	2	/

Prochaine utilisation ("NEXT-USE")

1: e = a + b

a	b	c	d	e
4	2	3	2	5

Prochaine utilisation ("NEXT-USE")

2: d = b – d

a	b	c	d	e
4	4	3	3	5

Prochaine utilisation ("NEXT-USE")

3: c = d – c

a	b	c	d	e
4	4	21	5	5

Prochaine utilisation ("NEXT-USE")

4: a = a + b

a	b	c	d	e
/	/	21	5	5

Prochaine utilisation ("NEXT-USE")

5: b = e * d

a	b	c	d	e
/	14	21	16	9

Prochaine utilisation ("NEXT-USE")

Tableau 3.1

3.3) En utilisant les réponses des points précédents, est-ce qu'il y a des sous-expressions en commun qui pourraient être éliminées ? Lesquelles ?

Oui : il y a un nœud commun à e et a, qui sont tous les deux égaux à a0 + b0. Comme a est mort à la fin du code, on peut éliminer la ligne 4.

3.4) En utilisant les réponses des points précédents, est-ce qu'il y a du code mort (inutile) qui pourrait être éliminé ? Lequel ?

Oui : a est mort à la fin du code, la ligne 4 est donc du code mort. On l'a cependant déjà éliminée au point précédent.

Question 4 – Environnements d'exécution (15 points)

Considérez un environnement d'exécution par enregistrement d'allocation des fonctions actives sur une pile.

Considérez le langage Java, en faisant l'hypothèse que les classes imbriquées ne soient pas permise.

4.1) Est qu'un environnement d'exécution par pile serait toujours nécessaire pour ce Java restreint ? POURQUOI ? EXPLIQUEZ.

Oui, car l'absence de classes imbriquées ne change en rien l'appel à des méthodes de classes standard.

4.2) Quelles seraient les règles de portée (scoping) dans un tel langage ? (Suggestions : discutez des variables statiques et locales)

On aurait deux niveaux de portée : les variables locales, accessibles uniquement dans l'environnement d'exécution local, et les variables globales, accessibles partout.

4.3) Est-ce que les liens d'accès (access link) et le lien de contrôle (control link) seraient toujours nécessaires dans un tel langage Java restreint ? POURQUOI ? EXPLIQUEZ.

Oui, les liens d'accès seraient toujours nécessaires, car on a deux niveaux de portée.

Les liens de contrôle seraient encore nécessaires par les appels standard aux méthodes.

Question 5 – Allocation de registres par coloriage de graphe (20 points)

Considérez le code machine avec registres *symboliques* suivant :

```
1:    LD    Rw, w
2:    LD    Ra, a
3:    LD    Rb, b
4:    LD    Rz, z
5:    LD    Ry, y
6:    ADD   Rx, Rz, Ry
7:    SUB   Ry, Rw, Ra
8:    SUB   Ry, Ry, Rb
9:    ADD   Ra, Ry, Rx
10:   SUB   Ry, Rw, Rd <= Il s'agit de Ra, il n'y a pas de Rd
```

ATTENTION : Interprétez le code à 3-adresses OP R1, R2, R3 comme R1 = R2 OP R3

5.1) CALCULEZ l'ensemble des registres vifs (« LIVE ») avant après chaque instruction pour le code machine ci-haut et REMPLISSEZ le **tableau 5.1**.

IMPORTANT : Considérez qu'aucun registre ne soit **VIF** après l'instruction 8.

LIVE: [ ]

1: LD Rw, w

LIVE: [Rw]

2: LD Rw, w <= C'est évidemment Ra, a

LIVE: | Rw, Ra |

3: LD Rb, b

LIVE: | Rw, Rb, Ra |

4: LD Rz, z

LIVE: | Rw, Rb, Ra, Rz |

5: LD Ry, y

LIVE: | Rw, Rb, Ra, Rz, Ry |

6: ADD Rx, Rz, Ry

LIVE: | Rw, Rx, Rb, Ra |

7: SUB Ry, Rw, Ra

LIVE: | Rw, Ry, Rx, Rb |

8: SUB Ry, Ry, Rb

LIVE: | Rw, Ry, Rx |

9: ADD Ra, Ry, Rx

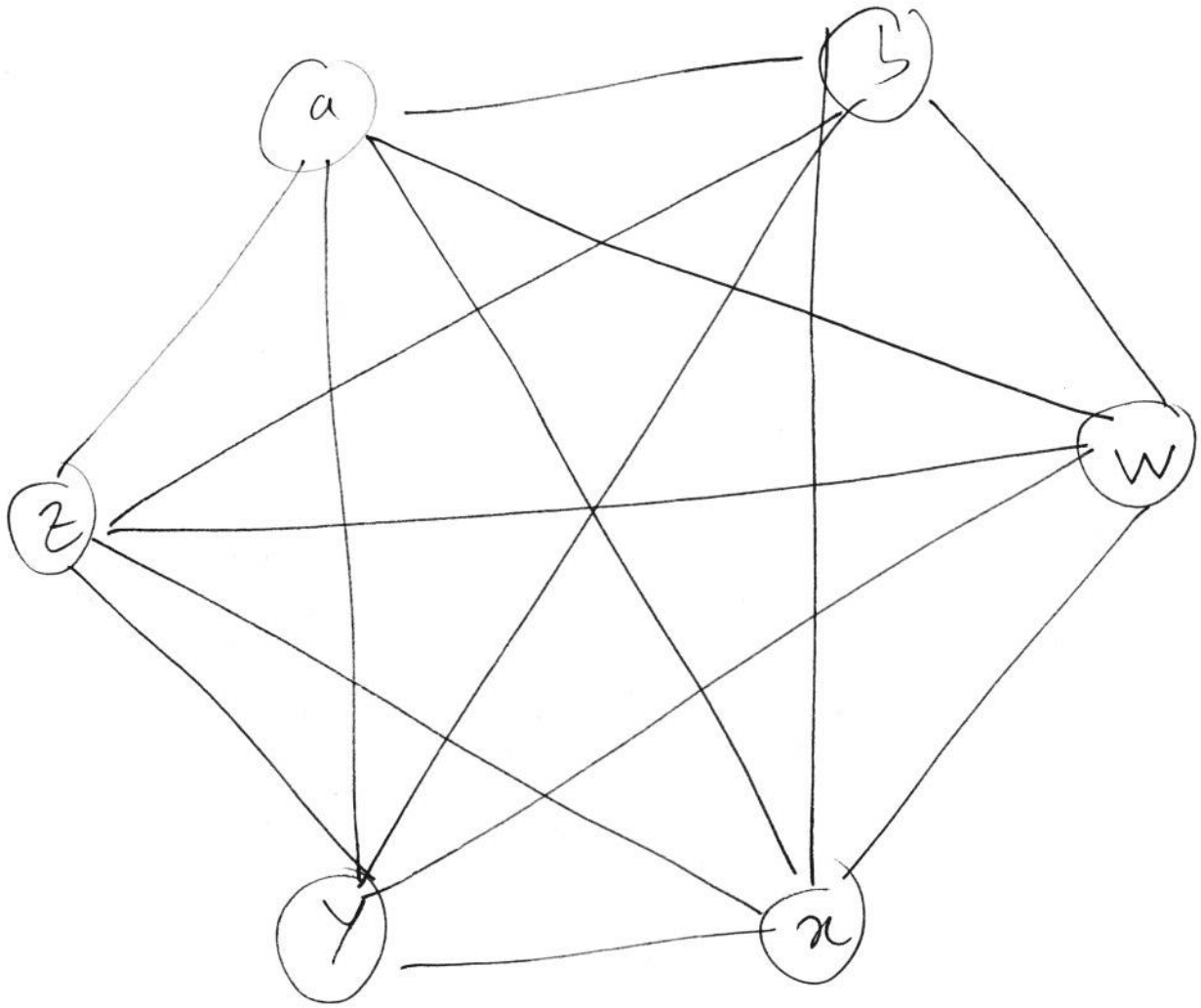
LIVE: | Rw, Ra |

10: SUB Ry, Rw, Rd <= Ra

LIVE: | Ry |

Tableau 5.1

5.2) DESSINEZ le graphe d'interférence de registres pour le code machine ci-haut.



5.3) Est-ce que 5 couleurs associées à des registres seraient suffisants à colorier le graphe de la réponse 5.2 ? Donnez le coloriage à 5 couleurs, si possible.

Non, pour faire un coloriage à cinq couleurs, il faudrait déplacer un LD à un endroit astucieux (là, ils sont tous les six à la suite, ce qui est inutile).

Question 6 – Ramasse-miette (10 points)

CONSIDÉREZ le code suivant :

```
void f(paramClasse par) {
        c1 o1 = new c1(); // 200 koctets
        par.o2 = new c2(); // 100 koctets
        c3 o3 = new c3(); // 500 koctets
main () {
        paramClasse p = new paramClasse(); // 40 koctets
        …
        // point A
        f(p);
        …
        // point B
}
```

6.1) Dessinez l'état (libre / occupée) de la mémoire au point A.

Occupée	Libre
p (40 ko)	

6.2) Dessinez l'état (libre / occupée) de la mémoire au point B dans les cas suivants :

6.2.1) sans ramasse-miettes;

Occupée				Libre
p (40 ko)	o1 (200 ko)	p.o2 (100 ko)	o3 (500 ko)	

6.2.2) ramasse-miettes par décompte des références;

Les références à o1 et o3 ont été supprimées en sortant de f, mais la mémoire est fragmentée.

Occupée	Libre	Occupé	Libre	Libre
p (40 ko)	(200 ko)	p.o2 (100 ko)	(500 ko)	

6.2.3) ramasse-miettes marqué et compacté (« mark and compact ») déclenché avant le point B.

Idem, mais la fragmentation est nulle grâce au compactage.

Occupée		Libre
p (40 ko)	p.o2(100 ko)	