be-OI 2011 **Demi-finale** 16 Février 2011	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

Olympiades belges d'Informatique (durée : 3h maximum)

Ce document est le questionnaire de la demi-finale des Olympiades belges d'Informatique 2011 pour **la catégorie supérieur**. Il comporte neuf questions qui doivent être résolues en **3h au maximum**. Chaque question est accompagnée d'un temps de résolution, donné à titre purement indicatif.

Notes générales (à lire attentivement avant de répondre aux questions)

1. N'indiquez votre nom, prénom et école **que sur la première page**. Sur toutes les autres pages, vous ne pouvez écrire que dans les **cadres prévus** pour votre réponse. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille, sans quoi votre réponse ne pourra être corrigée.

2. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, ... sont **interdits**.

3. Vos réponses doivent être écrites **au stylo ou au bic** bleu ou noir. Pas de réponses laissées au crayon. Si vous désirez des feuilles de brouillon, demandez-en auprès d'un surveillant.

4. Vous **devez** répondre aux questions ouvertes en **pseudo-code** ou dans l'un des **langages de programmation autorisés** (Java, C, C++, Pascal, Python et PHP). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation. Sauf mention contraire, vous ne pouvez utiliser aucune fonction prédéfinie à l'exception de `max(a,b)`, `min(a,b)` et `pow(a,b)` qui calcule a^b.

5. Dans l'ensemble de cet énoncé, si a et b sont des entiers, `a / b` et `a % b` représentent respectivement la division entière et le reste de la division entière de a par b. Par exemple, si a et b valent respectivement 14 et 3, `a / b` vaut 4 et `a % b` vaut 2.

6. Les tableaux sont indicés de 0 à $n-1$, où n correspond à leurs tailles. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ indique une boucle qui se répète tant que $i \leq b$ pour i initialisé à a et incrémenté de k à la fin de chaque itération.

7. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants. Toute question logistique peut leur être posée. A des fins d'égalité entre les participants des différents centres, vous ne **pouvez pas** poser de question **de compréhension** aux surveillants. Toute erreur dans l'énoncé doit être considéré comme faisant partie de l'épreuve.

8. Les participants **ne peuvent en aucun cas quitter leur place** pendant l'épreuve, par exemple pour aller aux toilettes ou pour fumer une cigarette. Selon le lieu où vous présentez l'épreuve, il peut vous être interdit de manger ou boire durant cette dernière.

9. Vous avez **exactement trois heures** pour répondre à ce questionnaire. Un **aide-mémoire** sur le pseudo-code est fourni en annexe à la dernière page.

Bonne chance !

Questionnaire demi-finale supérieur

be-OI 2011 Mercredi 16 février 2011	**Réservé**
Auteur : *D. Leroy*	

Question 1 – Échauffement (10 min)

Pour les trois sous-questions suivantes, une réponse correcte rapporte 1 point, une abstention vaut 0 point et une mauvaise réponse est sanctionnée par $-0{,}5$ point. Pour les questions à choix multiples, vous ne devez choisir qu'**une seule réponse**. Cochez la case de votre choix. Si vous vous êtes trompé, noircissez la case erronée pour annuler votre choix.

Q1(a) Soit n un entier strictement positif et soit l'algorithme suivant :

```
count ← 0
for (i ← 0 to n − 1 step 1)
{
    for (j ← 0 to n − 1 step 1)
    {
        if (i < j) { count ← count − 1 }
        else { count ← count + 1 }
    }
}
```

Quelle est la valeur de la variable `count` après exécution de l'algorithme ?

Q1(a)	☐ $n(n-1)$	☐ n^2	☐ n	☐ 0

Q1(b) Soit l'algorithme suivant :

```
i ← 0
while (i ≤ 15)
{
    i ← 2 * i + 1
}
```

Quelle est la valeur de la variable i après exécution de l'algorithme ?

Q1(b) **(une valeur)**

. . . . . .

Q1(c) Laquelle des expressions suivantes est-elle équivalente à "*a et b sont impairs*" :

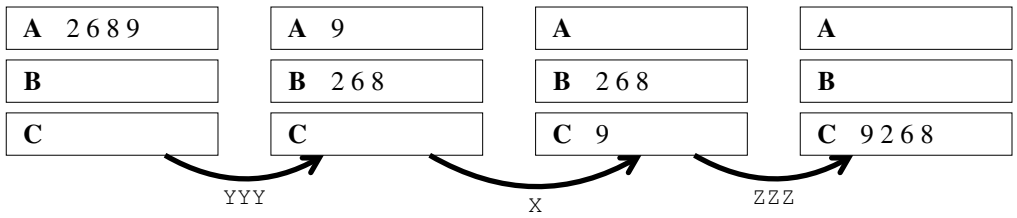
☐	`not (a % 2 = 0 or b % 2 = 0)`
☐	`a % 2 = 0 or b % 2 = 0`
☐	`a % 2 = 0 and b % 2 = 0`
☐	`not (a % 2 = 0 and b % 2 = 0)`

Question 2 – Mélange de cartes (10 min)

Vous vous trouvez face à trois tables que nous allons nommer A, B et C. Sur la table A se trouve une ligne de cartes, sur lesquelles figure à chaque fois un seul chiffre, faisant ainsi apparaitre un nombre. Votre but consiste à bouger les cartes vers la table C, de sorte à y créer le plus grand nombre possible. La **seule opération** que vous pouvez faire consiste à **prendre la carte à gauche** d'une table et la **placer à droite** sur une autre table. Vous avez trois possibilités :

- `X` : Prend sur A et place sur C ;
- `Y` : Prend sur A et place sur B ;
- `Z` : Prend sur B et place sur C.

Par exemple, supposons la situation de départ `2689`. Le schéma ci-dessous représente les mouvements `YYYXZZZ` qui permettent d'obtenir le résultat `9268` sur la table C. Si vous pensez qu'il s'agit du plus grand nombre qu'il est possible d'obtenir, vous devrez écrire `268` comme solution dans la case appropriée (les trois derniers chiffres du nombre obtenu).



Pour les trois situations initiales suivantes (c'est-à-dire le contenu de la table A, les autres étant vides), trouvez le plus grand nombre qu'il est possible d'obtenir et écrivez dans la case de réponse les trois derniers chiffres de ce nombre.

Q2(a) `3 5 9 7 6 2`

Q2(a) **(trois chiffres)**

. . . . . .

Q2(b) `9 1 2 8 4 5 7`

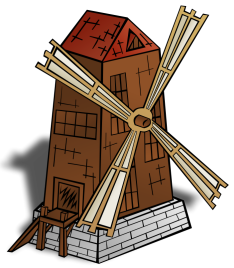
Q2(b) **(trois chiffres)**

. . . . . .

Q2(c) `8 6 4 1 9 7 5 3`

Q2(c) **(trois chiffres)**

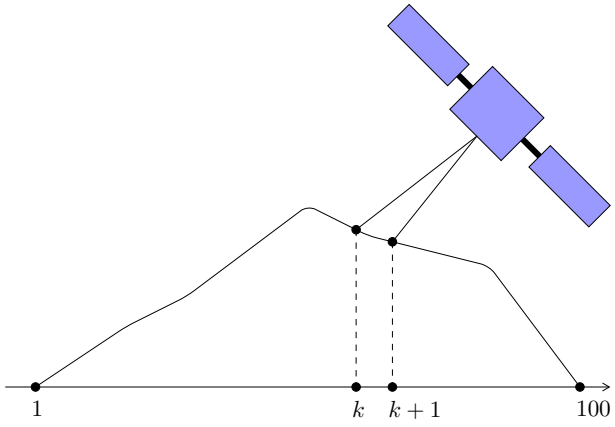
. . . . . .



Question 3 – Vive le vent ... (15 min)

Le Ministère de l'Environnement fait appel à vous pour déterminer la position optimale de son nouveau site d'éoliennes. Le site doit prendre place sur le point culminant d'une colline. Il vous reste à déterminer ce point. On vous informe que la colline a deux versants sans plateaux (de gauche à droite, la colline commence donc par croitre strictement jusqu'à atteindre son sommet pour ensuite décroitre strictement, sans qu'il n'y ait jamais le moindre plat).

Pour vous aider dans votre tâche, le Ministère met à votre disposition un satellite capable de déterminer l'inclinaison de la colline à une certaine abscisse. Le satellite peut être positionné en 100 points différents le long de l'axe des abscisses. Vous pouvez indiquer au satellite une abscisse k ($1 \leq k < 100$) et le satellite vous indiquera si la hauteur de la colline en k est plus grande ou plus petite que la hauteur de la colline en $k+1$. Si $hauteur(k) < hauteur(k+1)$, la réponse du satellite sera strictement positive. Dans le cas contraire, la réponse sera strictement négative. Dans l'exemple ci-dessus, le satellite retournera une valeur strictement négative étant donné que la hauteur en k est plus grande que la hauteur en $k+1$.



Vous devez utiliser le satellite avec parcimonie. Chaque interrogation du satellite ayant un coût financier important (10 000 €) et étant donné le budget limité de 150 000 € mis à votre disposition par le Ministère, votre algorithme doit donc veiller à ne pas interroger le satellite plus de 15 fois, pour toute colline respectant l'énoncé. Un technicien du Ministère a ébauché l'algorithme suivant mais n'a malheureusement pu déterminer certaines de ses parties. Cet algorithme doit retourner l'abscisse du point culminant de la colline, complétez-le.

```
a ← 0
b ← 100
while ([...])                                          // Q3(a)
{
    amesurer ← [...]                                   // Q3(b)
    if (interrogation_satellite (amesurer) > 0)
    {
        a ← amesurer
    }
    else
    {
        b ← amesurer
    }
}
return a
```

Q3(a) **(une condition)**

. . . . . .

Q3(b) **(une expression)**

. . . . . .

Question 4 – Ceci n'est pas un sudoku (25 min)

Votre cousin Damien est venu vous rendre visite en apportant un petit jeu se présentant sous la forme d'une grille de cases contenant chacune un chiffre. Le but du jeu consiste à partir du coin supérieur gauche de la grille et à rejoindre le coin inférieur droit. Vous ne pouvez vous déplacer que vers la case à votre droite ou vers celle du dessous.

Vous commencez avec un score de 0. À chaque déplacement, votre score courant est divisé par deux (et arrondi vers le bas si nécessaire) puis la valeur de la case sur laquelle vous arrivez est ajoutée à votre score. Le but du jeu est d'atteindre le coin droit inférieur en faisant le score le plus bas possible.

Soit la grille suivante :

0	3	9	6
1	4	4	5
8	2	5	4
1	8	5	9

Le plus petit score possible qu'il est possible d'obtenir sur cette grille est de 12. Il est obtenu avec les mouvements suivants : B, D, B, D, D, B (avec B pour bas et D pour droite).

Quelle est le plus petit score possible pour chacune des grilles suivantes :

0	2	9	6
1	1	9	1
3	2	1	2
1	2	3	3

Q4(a)

9	2	2	2	2
3	9	2	2	2
3	3	9	3	3
3	3	2	9	3
3	3	2	0	9

Q4(b)

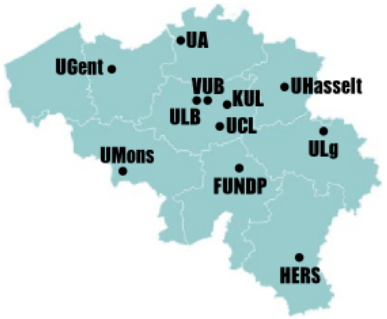
0	5	9	6	4
4	6	9	0	1
0	9	2	8	8
0	0	4	9	3
0	4	6	4	9

Q4(c)

0	3	9	6	4	2
0	3	9	6	4	2
0	6	9	0	1	1
0	9	2	9	0	1
0	0	4	4	5	6
0	4	6	8	0	9

Q4(d)

Question 5 – Où organiser la Finale ? (20 min)



Les demi-finales des Olympiades belges d'Informatique se déroulent dans plusieurs centres régionaux. Le comité des Olympiades a décidé que le lieu de la Finale serait revoté chaque année, pour prendre place parmi l'un des centres régionaux qui auront posé leur candidature. Lorsque plusieurs centres sont candidats, la règle établie par le comité est la suivante. La première fois, on prendra le centre régional le plus central, c'est-à-dire celui qui nécessite de parcourir, en moyenne, le moins de kilomètres en provenance des autres centres. Les années suivantes, la Finale sera organisée dans le centre régional le plus central, mais qui n'a pas encore organisé la Finale.

Le tableau suivant reprend les distances en kilomètres séparant les centres les uns des autres, et ce, pour quelques centres. Il faudra, par exemple, parcourir 75 km pour aller de Gent à Louvain-la-Neuve (LLN).

	Namur	LLN	Anvers	Leuven	Gent	Liège
Namur	0	38	107	47	104	66
LLN	38	0	73	23	75	84
Anvers	107	73	0	43	53	133
Leuven	47	23	43	0	71	67
Gent	104	75	53	71	0	138
Liège	66	84	133	67	138	0

Q5(a-b) Quels seront les 3e et 4e centres dans lesquels la Finale sera organisée, en supposant que les six centres présentés dans le tableau ci-dessus sont candidats tous les ans et qu'aucun de ceux-ci n'a encore organisé la finale ?

Q5(a) : 3e centre

Q5(b) : 4e centre

Pour résoudre ce problème de manière informatique, l'idéal serait d'avoir un algorithme qui trierait les différents centres par ordre décroissant en fonction du nombre de kilomètres moyen qui les séparent des autres centres. Voici justement une version de cet algorithme que vous pouvez compléter!

Dans cet algorithme, chaque centre est identifié par un numéro compris entre 0 et $N-1$. L'algorithme utilise un tableau, à deux dimensions, $N \times N$ nommé $distances$ et défini tel que pour chaque paire de centres i ($0 \leq i < N$) et j ($0 \leq j < N$), $distances[i][j]$ représente la distance entre le centre i et le centre j.

L'algorithme suivant a pour objectif de placer par ordre décroissant dans un tableau $rangs$ ($N \times 2$) sur chaque ligne l'identifiant d'un centre et le nombre de kilomètres moyen qui le sépare des **autres** centres. Ce dernier utilise l'algorithme $insere$ (non fourni) qui permet d'insérer le couple (i, m) dans la matrice $rangs$ à la position k ($0 \leq k < N$). Ainsi, après l'exécution de $insere(rangs, k, i, m)$, $rangs[k, 0]$ vaut i, $rangs[k][1]$ vaut m et tous les éléments qui se trouvaient dans $rangs$ à partir de k avant l'exécution de $insere$ ont été décalés d'une position vers les indices supérieurs.

```
Input  : distances, une matrice symmétrique carrée d'entiers positifs contenant les
         distances entre les centres régionaux. La matrice est de dimension N × N.
         N, un entier > 1, le nombre de centres.
         rangs, une matrice d'entiers (N × 2), initialisée avec rangs[i][1] = −1
         pour tout i.
Output : Pour tout i, rangs[i][0] et rangs[i][1] devront contenir respectivement l'identifiant
         du centre et le nombre moyen de kilomètres qui le sépare des autres centres.
         Les élément de la deuxième colonne de rangs doit être triée en ordre
         décroissant, c'est-à-dire, pour chaque ligne k de rangs (0 ≤ k < N − 1),
         rangs[k][1] ≥ rangs[k + 1][1].
         La première colonne doit contenir tous les centres une et une seule fois.
         L'algorithme ne renvoie rien.

for (i ← 0 to N − 1 step 1)
{
    s ← 0
    for (j ← 0 to N − 1 step 1)
    {
        s ← [...]                                                    // Q5(c)
    }
    m ← [...]                                                        // Q5(d)
    k ← 0
    while ([...])                                                    // Q5(e)
    {
        k ← k + 1
    }
    insere (rangs, k, i, m)
}
```

Complétez l'algorithme afin qu'il calcule ce qu'il faut.

Q5(c) **(une expression)**

. . . . . .

Q5(d) **(une expression)**

. . . . . .

Q5(e) **(une condition)**

. . . . . .

Question 6 – Le jeu des craies (10 min)


Mike Plante

Votre ami Martin vous a récemment appris un nouveau jeu. Trente craies sont déposées sur une table. Les deux joueurs jouent successivement en retirant de la table une, deux ou trois craies. Le joueur qui prend la dernière craie **perd** la partie. Martin parvient toujours à gagner contre vous lorsque qu'il commence la partie. Ci-dessous se trouve un algorithme à compléter dans le but de représenter la stratégie appliquée par Martin à chaque fois que c'est à lui de jouer.

```
Input  : remaining, un entier positif ≤ 30, le nombre de craies restant sur la table
         last, le nombre de craies retirées par l'adversaire de Martin au tour précédent
         qui vaut 1, 2 ou 3 (ou 0 si c'est le premier tour de la partie).
         Le scenario actuel permet à Martin d'être assuré de gagner si il joue de
         manière adéquate.
Output : le nombre de craies que Martin va retirer : 1, 2 ou 3
```

Quelle **expression mathématique** faut-il ajouter pour que l'algorithme suivant (ne comportant qu'une seule instruction) calcule le résultat attendu ?

```
return [...]                                                    // Q6base
```

Q6 base **(une expression)**

. . . . . .

Si vous ne trouvez pas de solution rapidement, passez à la page suivante pour une question alternative.

Question 6 (alternative)

Si vous ne parvenez pas à trouver une solution à l'algorithme utilisé par Martin, **et uniquement dans ce cas**, vous pouvez essayer de trouver celui utilisé par sa sœur Fanny, aussi efficace mais un peu plus long, en complétant l'algorithme suivant avec **une condition et deux expressions mathématiques** [1].

```
if ([...])                                                    // Q6alt(a)
{
    return [...]                                              // Q6alt(b)
}
else
{
    return [...]                                              // Q6alt(c)
}
```

Q6 alternative (a) **(une condition)**

. . . . . .

Q6 alternative (b) **(une expression)**

. . . . . .

Q6 alternative (c) **(une expression)**

. . . . . .

1. En cas de réponse correcte dans la case **Q6 base**, la réponse à la question **Q6 alternative** sera ignorée.

Question 7 – Parlez-vous la *Zorglangue* ? (20 min)

Claire et Manuel voulaient discuter entre eux sans que les gens ne puissent comprendre leur discussion. Jeunes et branchés, ils ont pensé au **verlan** qui consiste à inverser les syllabes d'un mot. En verlan, on dit "*chelou*" au lieu de louche. Ce jargon étant assez répandu, ils ont alors décidé d'utiliser la Zorlangue, inventée par Zorglub, personnage de la série de bande dessinée Spirou et Fantasio, et qui consiste à inverser l'ordre des lettres d'un mot. On ne dit alors plus louche, mais "*ehcuol*".

L'algorithme utilisé pour permettre de comprendre et de traduire les phrases écrites en "Zorglangue" est composé de deux parties. Le premier algorithme (Q7(a)) permet d'inverser les lettres d'un sous-tableau dans un tableau de caractères. Par exemple, si le tableau de caractères est `[v, i, v, e, , z, o, r, g, l, u, b, .]`, exécuter l'algorithme avec $debut = 5$ et $fin = 11$ aura pour effet de modifier le tableau en `[v, i, v, e, , b, u, l, g, r, o, z, .]`.

```
Input  : phrase, un tableau de caractères.
         debut et fin, entiers positifs (debut ≤ fin), indices du premier et du
         dernier caractère du mot.
Output : l'ordre des lettres du mot commençant à l'indice debut et se terminant
         à l'indice fin a été inversé, l'algorithme ne renvoie rien

function invert (phrase, debut, fin)
{
    [...]                                                          // Q7(a)
}
```

Écrivez complètement cet algorithme.

Q7(a) **(un algorithme)**

Question 7 (suite)

L'algorithme qui permet de résoudre le problème général va parcourir la phrase, et à chaque fois qu'un mot est identifié, faire appel à l'algorithme d'inversion de mot pour inverser le mot trouvé.

```
Input  : phrase, un tableau de caractères ne contenant que des caractères minuscules de
         l'alphabet, un point et des espaces. Les mots de la phrase sont séparés par un
         seul espace. Il n'y a pas d'espaces au début et à la fin de la phrase.
         La phrase se termine par un point.
         n, entier positif, le nombre de caractères de phrase.
Output : la phrase a été traduite en "Zorglangue". L'algorithme ne renvoie rien.

wordstart ← 0

for (i ← 0 to n − 1 step 1)
{
    if ([...])                                                    // Q7(b)
    {
        invert (phrase, wordstart, [...])                         // Q7(c)
        [...]                                                     // Q7(d)
    }
}
```

Quelles instructions manque-t-il à cet algorithme afin qu'il produise le résultat attendu ?

Q7(b) **(une condition)**

.....

Q7(c) **(une expression)**

.....

Q7(d) **(une instruction)**

.....

Question 8 – Anagrammes numériques (20 min)

Dans ce problème, nous considèrons que deux nombres sont des *anagrammes* s'ils sont composés d'exactement les mêmes chiffres, dans le même ordre ou dans un ordre différent. Par exemple, 121, 211, 112 et 121 sont des anagrammes alors que 411, 144 et 511 ne le sont pas.

Vous devez écrire un algorithme qui détermine si deux nombres entiers positifs, a et b, représentés sous forme de tableau de chiffres, sont des anagrammes. Vous pouvez déclarer de nouvelles fonctions tant que vous ne dépassez pas le cadre de réponse qui vous est alloué. Vous pouvez également déclarer de nouveaux tableaux d'entiers, dont les éléments sont initialisés à zéro, en utilisant la notation $newtab \leftarrow$ new_array($size$).

```
Input  : a, b, des tableaux de taille identique, n, dont chaque cellule contient un
         chiffre.
Output : true (vrai) si a et b sont des anagrammes, false (faux) sinon.
         Les tableaux a et b peuvent avoir été modifiés.
```

Aide

Afin de vous aidez, vous pouvez (ce n'est pas une obligation !) faire appel à la fonction suivante qui permet de trier les éléments d'un tableau par ordre croissant. La fonction parcourt le tableau en vue de retrouver, à chaque itération, l'élément suivant à placer afin de le trier. Par exemple, si t vaut $[1, 8, 3, 6, 2]$ avant l'appel à la fonction, après l'exécution de $tri(t, 5)$, celui vaut $[1, 2, 3, 6, 8]$.

```
Input  : tab, un tableau non-vide d'entiers positifs.
         n > 0, la taille du tableau.
Output : Les éléments du tableau tab ont été triés par ordre croissant.
         L'algorithme ne renvoie rien.

function tri (tab, n)
{
    for (i ← 0 to n − 1 step 1)
    {
        min ← i
        for (j ← i + 1 to n − 1 step 1)
        {
            if (tab[j] < tab[min])
            {
                min ← j
            }
        }

        temp ← tab[i]
        tab[i] ← tab[min]
        tab[min] ← temp
    }
}
```

Q8

Bonus

Vous pouvez obtenir un bonus à cette question si votre algorithme est **efficace**. Pour cela, vous devez proposer un algorithme pour lequel le nombre de fois que votre algorithme accède aux éléments des tableaux (c'est-à-dire, le nombre de $tableau[indice]$) est inférieur à $10 \times n$ pour les n strictement supérieurs à 10. Si vous utilisez la fonction tri donnée, les accès aux éléments du tableau faits par cette fonction doivent bien entendu être comptabilisés.

Question 9 – Préparer mon planning (30 min)

On souhaite réaliser un programme permettant de gérer un *agenda*. Les informations stockées dans l'agenda consistent en une série d'événements, ordonnés dans le temps. Chaque évènement est caractérisé par trois informations : une heure de début, une heure de fin et une chaine de caractères qui le décrit. On supposera que les événements ne se chevauchent jamais, par contre, il peut naturellement exister des plages libres entre deux événements consécutifs. Les événements peuvent également être ponctuels, ce qui signifie que l'heure de début et l'heure de fin coïncident. Voici un exemple d'agenda :

1. **De 10h à 11h** : Réunion
2. **De 12h à 13h** : Déjeuner
3. **De 13h à 15h** : Faire les courses
4. **À 17h** : Aller chercher les enfants à l'école

Comme on peut le voir, les événements 2 et 3 se suivent sans temps libre, et l'évènement 4 est ponctuel.

Concrètement, on choisit de représenter un agenda à l'aide de quatre tableaux `nom`, `duree`, `intervalle` et `suivant`, ainsi que de deux variables entières `premier` et `debut`. Chaque évènement sera représenté par les cases de même indice (disons i) des tableaux `nom`, `duree`, `intervalle` comme suit : `nom[i]` donnera le nom de l'évènement, `intervalle[i]` indiquera combien de temps (en heures) sépare le début de l'évènement de la fin du précédent (ce sera 0 pour le premier évènement), et enfin `duree[i]` indiquera la durée (en heures) de l'évènement (0 pour les événements ponctuels). Pour chaque évènement correspondant à l'indice i, le tableau `suivant` donnera l'indice de l'élément suivant, ou bien contiendra -1 s'il s'agit du dernier évènement. Enfin, la variable `premier` donnera l'indice du premier élément, et la variable `debut` donnera l'heure de début de ce premier évènement. Notons que ces deux variables valent -1 si l'agenda ne contient aucun évènement.

L'agenda donné en exemple ci-dessus pourra donc être représenté de la façon suivante :

`premier`	=	1					
`debut`	=	10					
`intervalle`	=	1	0		2	0	
`duree`	=	1	1		0	2	
`suivant`	=	4	0		-1	3	
`nom`	=	Déjeuner	Réunion		Enfants	Courses	

Comme la variable `premier` vaut 1, on sait que les cases `nom[1]`, `duree[1]` et `intervalle[1]` décrivent le premier évènement. Celui-ci commence à l'heure donnée par `debut` soit 10h. Il s'agit donc bien de la réunion, qui dure 1h, et se termine donc à 11h. La case `suivant[1]` contient 0, et le deuxième évènement est donc dans les cases d'indice 0. Comme `intervalle[1]` vaut 1, on voit que le second évènement (appelé "Déjeuner") commence à 11h + 1h = 12h. Il dure 1h, et se termine donc à 13h. Et ainsi de suite…

L'algorithme donné ci-dessous manipule un agenda stocké dans des tableaux, comme décrit ci-dessus. Cet algorithme réalise l'*insertion* d'un évènement supplémentaire dont le nom (paramètre n), ainsi que les heures de début et de fin (paramètres d et f) sont fournis. On indique également à l'algorithme à quel indice des tableaux l'évènement doit être stocké (paramètre $place$). L'algorithme fait l'hypothèse que l'évènement à insérer ne recouvre aucun autre évènement déjà présent dans l'agenda. On vous demande de le compléter.

```
Input  : n, nom de l'évènement
         d et f, heures de début et de fin de l'évènement (d ≤ f)
         place, indice (libre) dans les tableaux où l'évènement doit être stocké
Output : l'agenda a été mis à jour, l'algorithme ne renvoie pas de valeur

nom[place] ← n
duree[place] ← [...]                                                    // Q9(a)
if ([...])                                                              // Q9(b)
{
    if (premier ≠ −1)
    {
        intervalle[premier] ← debut − f
    }
    suivant[place] ← premier
    intervalle[place] ← 0
    premier ← place
    debut ← d
}
else
{
    i ← premier
    deb_i ← debut
    fin_avant_i ← 0
    while (deb_i ≤ d and [...])                                         // Q9(c)
    {
        avant_i ← i
        fin_avant_i ← deb_i + duree[i]
        i ← suivant[i]
        if (i ≠ −1)
        {
            deb_i ← fin_avant_i + intervalle[i]
        }
    }
    suivant[avant_i] ← place
    [...] ← i                                                           // Q9(d)
    intervalle[place] ← d − fin_avant_i
    if (i ≠ −1)
    {
        intervalle[i] ← [...]                                           // Q9(e)
    }
}
```

Quelle instructions manque-t-il à cet algorithme afin qu'il produise le résultat attendu ?

Q9(a) **(une expression)**

. . . . . .

Q9(b) **(une condition)**

. . . . . .

Q9(c) **(une condition)**

. . . . . .

Q9(d) **(une expression)**

. . . . . .

Q9(e) **(une expression)**

. . . . . .

Aide-mémoire pseudo-code

Les données sont stockées dans des variables et il est possible d'effectuer des opérations arithmétiques avec ces dernières. En plus des quatre opérateurs classiques ($+$, $-$, $\times$ et $/$), vous pouvez également utiliser $\%$ (voir définitions en première page). La valeur d'une variable est modifiée avec $\leftarrow$. Il y a des variables permettant de stocker des entiers, des nombres réels, des tableaux et des booléens : vrai (**`true`**) ou faux (**`false`**).

```
annee_naissance ← 1992
age ← 2011 − annee_naissance
```

On peut n'exécuter du code que si une certaine condition est vérifiée en utilisant une instruction **`if`** et éventuellement un autre code dans le cas contraire en ajoutant une instruction **`else`**. L'exemple suivant vérifie si une personne est majeure ou non et stocke la réponse dans la variable booléenne (vrai ou faux) $majeur$.

```
if (age ≥ 18)
{
    majeur ← true
}
else
{
    majeur ← false
}
```

Pour manipuler plusieurs éléments d'un coup, on utilise un tableau. Il s'agit d'une liste de valeurs qu'on va pouvoir accéder en utilisant les indices des éléments. Le premier élément de la liste est l'élément d'indice 0. Le second est celui d'indice 1 ... et le dernier est celui d'indice $N-1$ si N indique le nombre total d'éléments de la liste (sa taille). Par exemple, pour le tableau tab qui vaut `[5,9,12]`, tab`[0]` indique le premier élément (à savoir 5) et tab`[2]` indique le dernier (à savoir 12).

On peut exécuter une boucle en utilisant **`for`** (définition à la première page). L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.

```
sum ← 0
for (i ← 0 to N − 1 step 1)
{
    sum ← sum + tab[i]
}
```

On peut également exécuter une boucle avec **`while`** qui repète du code tant que sa condition est vraie. Dans l'exemple suivant, on va diviser un nombre entier positif N par 2, puis par 3, ensuite par 4 ... jusqu'à ce qu'il ne soit plus composé que d'un seul chiffre (c'est-à-dire qu'il est < 10).

```
d ← 2
while (N > 10)
{
    N ← N/d
    d ← d + 1
}
```