Haute École de Bruxelles
École Supérieure d'Informatique

TD Boucles

Distribué sous licence Creative Commons Paternité - Partage à l'Identique 2.0 Belgique (http://creativecommons.org/licenses/by-sa/2.0/be/)
Les autorisations au-delà du champ de cette licence peuvent être obtenues à http://www.heb.be/esi - mcoduti@heb.be

Résumé

Ces exercices ont pour but de vérifier que vous avez fixé les structures alternatives qui permettent de conditionner des parties d'algorithmes, de code.

1 Les boucles

1.1 Compréhension d'algorithme

Pour ces exercices, nous vous demandons de comprendre des algorithmes donnés.

Compréhension

Que vont-ils afficher ?

```
— module boucle1 ()
      x : entier
      x ← 0
      tant que x < 12 faire
        x ← x+2
      fin tant que
      afficher x
  fin module
```

```
— module boucle2 ()
      ok : booléen
      x : entier
      ok ← vrai
      x ← 5
      tant que ok faire
        x ← x+7
        ok ← x MOD 11 ≠ 0
      fin tant que
      afficher x
  fin module
```

```
— module boucle3 ()
      ok : booléen
      cpt, x : entiers
      x ← 10
      cpt ← 0
      ok ← vrai
      tant que ok ET cpt < 3 faire
        si x MOD 2 = 0 alors
          x ← x+1
          ok ← x < 20
        sinon
          x ← x+3
```

```
        cpt ← cpt + 1
      fin si
    fin tant que
    afficher x
fin module
```

```
module boucle4 ()
    pair, grand : booléens
    p, x : entiers
    x ← 1
    p ← 1
    faire
      p ← 2*p
      x ← x+p
      pair ← x MOD 2 = 0
      grand ← x > 15
    jusqu'à ce que pair OU grand
    afficher x
fin module
```

```
module boucle5 ()
    i, x : entiers
    ok : booléen
    x ← 3
    ok ← vrai
    pour i de 1 à 5 faire
      x ← x+i
      ok ← ok ET (x MOD 2 = 0)
    fin pour
    si ok alors
      afficher x
    sinon
      afficher 2 * x
    fin si
fin module
```

```
module boucle6 ()
    i, j, fin : entiers
    pour i de 1 à 3 faire
      fin ← 6 * i - 11
      pour j de 1 à fin par 3 faire
        afficher 10 * i + j
      fin pour
    fin pour
```

```
fin module
```

1.2 Compréhension de codes Java

Instructions répétitives

Quelles instructions répétitives sont correctes parmi les suivantes ? Expliquez pourquoi les autres ne le sont pas.

- ❏ proposition 1

```
While ( condition ) {
    // instructions
}
```

- ❏ proposition 2

```
do while ( condition ) {
    // instructions
}
```

- ❏ proposition 3

```
while ( true ) {
    // instructions
}
```

- ❏ proposition 4

```
while ( true ) do {
    // instructions
}
```

- ❏ proposition 5

```
FOR ( int i=0; i<=10; i=i+2 ) DO {
    // instructions
}
```

- ❏ proposition 6

```
for ( int i=0; i<=10; i=i+2 ) {
    // instructions
}
```

- ❏ proposition 7

```
for ( int i=0; i<=10; i=i+2 ) do {
    // instructions
}
```

- ❏ proposition 8

```
for ( int i=9; i>=0; i=i-2 ) {
    // instructions
}
```

Activité 'remplir les blancs'

Quel opérateur de comparaison Java représente la relation suivante ?

1. "est égal à" ? ____
2. "est différent de" ? ____

Quel opérateur booléen Java représente l'opérateur logique suivant ?

1. le ET : ____
2. le OU : ____
3. le NON : __

Expérience

Indiquez l'affichage obtenu par ce code.

Compréhension

Que vont-ils afficher ?

```
public class Boucles {

    public static void main ( String[] args ) {
        int facteur;
        final int VALEUR = 3;

        for (facteur = 1 ; facteur <= 10 ; facteur++){
            System.out.print(facteur*VALEUR+" ");
        }
        System.out.println();
    }
}
```

Exercice Tant que

Écrivez en Java l'algorithme suivant.

```
MODULE Test

    nb, produit : Entier
    produit ← 1

    LIRE nb
    TANT QUE nb ≠ 0 FAIRE
```

```
        produit ← produit * nb
        LIRE nb
    FIN TANT QUE
    AFFICHER produit

FIN MODULE
```

Exercice Pour

Écrivez en Java l'algorithme suivant.

```
MODULE Test

    nb: Entier
    i : Entier

    LIRE nb
    POUR i DE 1 A nb FAIRE
        AFFICHER i
    FIN POUR

FIN MODULE
```

1.3 À vous de jouer...

Voici quelques conseils pour vous guider dans la résolution de tels problèmes :

- il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- résolvez le problème via quelques exemples précis ;
- mettez en évidence les variables **«données »**, les variables **«résultats »** et les variables de travail ;
- n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé ;
- déclarez ensuite les variables (et leur type) qui interviennent dans chaque module ; les noms des variables risquant de ne pas être suffisamment explicites.
- Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.
- Pour la partie Java, dessinez l'arborescence des fichie, vous travaillerez dans un package `g12345.Boucles`.

Vous me copierez 100 fois...

Il est classique de demander comme punition de copier 100 fois la phrase "Je ne bavarderai pas avec mon voisin". Phrase précédée à chaque fois du numéro de la phrase.

Écrivez le code java correspondant ainsi que la javadoc.

Un nombre est-il divisible par 9 ?

En base dix, on peut facilement vérifier si un nombre est divisible par 9 : on calcule la somme de ses chiffres et l'on regarde si cette somme est encore divisible par 9 c'est-à-dire, si l'on itère le processus, si la dernière somme donne 9.

Écrivez un algorithme permettant de dire si un nombre donné est divisible par 9, en appliquant la méthode décrite ci-dessus.

Écrivez le code java correspondant ainsi que la javadoc.

Si le nombre entré est négatif, vous lancerez une exception qui sera gérée dans votre méthode `main`.

Suite de positifs

Écrivez un algorithme qui lit une suite de nombres positifs entrés au clavier et affiche le maximum, le minimum, leur somme et la moyenne.

La fin de la suite de nombre sera signifiée par une valeur sentinelle que vous choisirez judicieusement.

Écrivez le code java correspondant ainsi que la javadoc.

La conjecture de Goldbach

La conjecture de Goldbach est une assertion mathématique non démontrée qui s'énonce comme suit :

Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

Écrivez un module `isPremier` qui reçoit un nombre entier `n` et qui retourne vrai si ce nombre est premier et faux sinon.

Si le nombre entré est négatif, vous lancerez une exception qui sera gérée dans votre méthode `main` en Java.

Écrivez un module `goldbach` qui reçoit en paramètre un nombre entier pair `p` supérieur à 3 et qui retourne vrai s'il est la somme de 2 nombres premiers

et faux sinon. Si le `p` reçu n'est supérieur à 3, votre programme générera une erreur qui sera gérée dans votre méthode `main` en Java.

Écrivez un module, `principal`, qui lit un nombre et vérifie que tous les nombres pairs inférieurs à ce nombre sont la somme de 2 nombres premiers. Ce module affiche vrai ou faux selon le cas.

Mettez en évidence les variables **«données »**, les variables **«résultats »** et les variables de travail ;

Écrivez le code java correspondant ainsi que la javadoc.

Pour plus d'exercices, révisez ici (`www.heb.be/esi/InitBoucle/fr/../../TDBoucle/fr/html/Exercices_learningObject3.html`)