Algorithmique - Correction du TD3

IUT 1ère Année

18 décembre 2012

1 Les boucles (suite)

Exercice 1. Ecrire un algorithme qui reçoit en entrée un nombre entier de 1 à 10 et affiche en sortie la table de multiplication de ce nombre. Par exemple, si l'algorithme reçoit le nombre 7, il affichera la table :

- $1 \times 7 = 7$
- $2 \times 7 = 14$
- ...
- $10 \times 7 = 70$

Algorithme 1: Table de Multiplication

```
variables
    entier i,n

début
    lire n
    pour i de 1 à 10 faire
        afficher i " fois " n " est égal à " i × n
fin
```

Exercice 2. A la naissance de Marie, son grand-père Nestor, lui ouvre un compte bancaire. Ensuite, à chaque anniversaire, le grand père de Marie verse sur son compte 100 €, auxquels il ajoute le double de l'âge de Marie. Par exemple, lorsqu'elle a deux ans, il lui verse 104 €. Ecrire un algorithme qui permette de determiner quelle somme aura Marie lors de son n-ième anniversaire.

Algorithme 2: Compte de Marie

```
variables
    entier compte,age

début
    compte ← 0
    pour age de 1 à n faire
        compte ← compte + 100 + (2 ∗ age)
    afficher "Le compte de Marie au n-ième aniversaire est " compte
fin
```

Exercice 3. La population des Sims Alpha est de 10,000,000 d'habitants et elle augmente de 500,000 habitants par an. Celle des Sims Beta est de 5,000,000 habitants et elle augmente de 3% par an. Ecrire un algorithme permettant de déterminer dans combien d'années la population de Sims Beta dépassera celle des Sims Alpha.

Algorithme 3: Populations alpha et beta

```
variables
    entier années,alpha,beta

début
    alpha ← 10 000 000
    beta ← 5 000 000
    années ← 0
    tant que beta ≤ alpha faire
        années ← années + 1
        alpha ← alpha + 500 000
        beta ← beta ∗ 1.03
    afficher "Il faut " années " années pour que la population de beta dépasse celle de alpha"
fin
```

Exercice 4. Corriger le programme C++ suivant afin de résoudre le problème suivant :

- Données : un nombre entier positif n
- Résultat : le résultat de la suite harmonique : $\sum_{i=1}^{n} \frac{1}{i}$

Algorithme 4 – Suite Harmonique

```
#include <iostream>
using namespace std;

int main()
{
int i,n;
float somme = 0;
cout << "Entrer le nombre entier: ";
cin >> n;
for(i = 1; i <= n; i++)
    somme = somme + 1.0/i;
cout << "Le résultat est: " << somme << endl;
return 0;
}
```

Exercice 5. Construire un algorithme permettant d'évaluer vos chances de gagner dans l'ordre ou dans le désordre au tiercé, quarté ou quinté. De manière formelle, le problème est le suivant :

- Données : un nombre p de chevaux partants et un nombre $j \in \{3,4,5\}$ de chevaux joués
- Résultat : la probabilité de gagner au jeu dans l'ordre, et la probabilité de gagner au jeu dans le désordre

Rappel : les formules habituelles de comptage sont données dans la table ci-jointe.

Nombre de possibilités de construire une liste ordonnée, avec répétitions, de j éléments parmi p	p^j
Nombre de possibilités de construire une liste ordonnée, sans répétition, de j éléments parmi p	$\frac{p!}{(p-j)!}$
Nombre de possibilités de construire un ensemble non ordonné, sans répétition, de j éléments parmi p	$\frac{p!}{(p-j)!j!}$

Note : dans la correction on utilise la fonction factorielle déjà définie en cours et en TD. N'hésitez pas à *réutiliser* les fonctions ou procédures que vous avez déjà construites.

Algorithme 5: Tiercé

```
variables
    entier p, j

début
    afficher "Chevaux partants : "
    lire p
    afficher "Chevaux joués : "
    lire j
    afficher "Probabilité de gagner dans l'ordre : " fact(p − j)/fact(p)
    afficher "Probabilité de gagner dans le désordre : " fact(p − j) ∗ fact(j)/fact(p)
fin
```

2 Les tableaux

Exercice 6. Corriger l'algorithme en pseudo-code suivant afin de résoudre le problème suivant :
- Données : deux vecteurs **p** et **q** dans un espace (Euclidien) à 3 dimensions
- Résultat : la somme des vecteurs **p** + **q**

Algorithme 6: Somme De Vecteurs

```
variables
    réel p[3]
    réel q[3]
    réel r[3]

début
    pour i ← 0 à 2 faire
        r[i] ← p[i] + q[i]
fin
```

Exercice 7. Ecrire un algorithme permettant de résoudre le problème suivant :
- Données : deux vecteurs **p** et **q** dans un espace (Euclidien) à 3 dimensions
- Résultat : le produit scalaire de **p** et **q**

Algorithme 7: Produit Scalaire

```
variables
    réel p[3]
    réel q[3]
    réel v

début
    v ← 0
    pour i ← 0 à 2 faire
        v ← v + (p[i]*q[i])
    afficher v
fin
```

Exercice 8. Pour sa naissance, la grand-mère de Gabriel place une somme de 1000 € sur son compte épargne rémunéré au taux de 2.25% (chaque année le compte est augmenté de 2.25%). Développer un algorithme permettant d'afficher un tableau sur 20 ans associant à chaque anniversaire de Gabriel la somme acquise sur son compte.

Algorithme 8: Compte de Gabriel

```
variables
    réel compte[21], i

début
    compte[0] ← 1000
    pour i ← 1 à 20 faire
        compte[i] ← compte[i] * 1.0225
fin
```

Exercice 9. Felix est un fermier qui dispose d'un couple de shadoks capables de se reproduire à vitesse phénoménale. Un couple de shadocks met deux mois pour grandir ; à partir du troisième mois, le couple de shadocks engendre une paire de nouveaux shadocks (qui mettront deux mois pour grandir et donc trois mois pour engendrer une nouvelle paire, etc.). Et surtout, les shadoks ne meurent jamais !

D'après cet exercice le nombre de couples de shadoks F_n à chaque mois n obéit à la loi :

- $F_1 = 1$
- $F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$

Développer un algorithme permettant de construire le tableau des couples depuis le premier jusqu'au 20ème mois.

Algorithme 9: Suite de Fibonacci

```
variables
    réel couples[20]

début
    couples[0] ← 1
    couples[1] ← 1
    pour i ← 2 à 19 faire
        couples[i] ← couples[i − 1] + couples[i − 2]
fin
```

Exercice 10. Corriger le programme C++ suivant afin de résoudre le problème suivant :

- Données : un tableau de 100 entiers, une valeur entière x
- Résultat : le nombre d'occurrences de x dans le tableau

Algorithme 10 – Nombre d'ccurrences

```
#include <iostream>
using namespace std;

int main()
{
int tableau[100];
int i,x,occurrences;

cout << "Entrer votre valeur: ";
cin >> x;
i = 0;
occurrences = 0;
for(i = 0; i < 100; i++)
    occurrences = occurrences + (x == tableau[i]);
cout << occurrences << endl;
return 0;
}
```

Exercice 11. Nous souhaitons développer un algorithme permettant de rechercher un élément dans un tableau de 100 entiers en partant des deux extrémités. Dans cette perspective, corriger le programme C++ suivant.

Algorithme 11 – Recherche Bipolaire

```
#include <iostream>
using namespace std;

int main()
{
int tableau[100];
int i,j,x;
bool trouve;

cout << "Entrer votre valeur: ";
cin >> x;
i = 0;
j = 99;
trouve = 0;
do
    {
    trouve = (tableau[i] == x) || (tableau[j] == x);
    i++;
    j--;
    }
while(!trouve && i <= j);
cout << trouve << endl;
return 0;
}
```

Exercice 12. Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : un tableau *tableau* contenant 100 entiers
- Résultat : "vrai" si les entiers sont consécutifs et "faux" sinon

Rappel : deux entiers x et y sont consécutifs si et seulement si $y = x + 1$.

Algorithme 12: Eléments consécutifs

```
variables
    entier tableau[100], i
    booléen consécutifs

début
    consécutifs ← vrai
    i ← 0
    tant que (consécutifs = vrai) et (i < 99) faire
        consécutifs ← tableau[i + 1] = tableau[i] + 1
        i ← i + 1
    afficher consécutifs
fin
```

Exercice 13. Ecrire un algorithme permettant de résoudre le problème suivant :
- Données : un tableau *tableau* contenant 100 entiers
- Résultat : "vrai" si le tableau est trié du plus petit au plus grand et "faux" sinon

Algorithme 13: Test du tri

```
variables
    entier tableau[100], i
    booléen trié

début
    trié ← vrai
    i ← 0
    tant que (trié = vrai) et (i < 99) faire
        trié ← tableau[i] ≤ tableau[i + 1]
        i ← i + 1
    afficher trié
fin
```

Exercice 14. Ecrire un algorithme permettant de saisir 100 valeurs et qui les range au fur et à mesure dans un tableau.

Algorithme 14: Tri à la volée (qui est une forme de tri par insertion)

```
variables
    entier tableau[100], i, j, x
    booléen positionné

début
    pour i de 0 à 100 faire
        afficher "Entrez votre valeur : "
        lire x
        j ← i
        tant que (j > 0) et (tableau[j − 1] > x) faire
            tableau[j] ← tableau[j − 1]
            j ← j − 1
        tableau[j] ← x
fin
```

Exercice 15. Ecrire un algorithme qui inverse l'ordre d'un tableau des 100 entiers triés. En d'autres termes, si le tableau est trié du plus petit au plus grand, alors l'algorithme retourne le tableau trié du plus grand au plus petit ; réciproquement, si le tableau est trié du plus grand au plus petit, alors l'algorithme retourne le tableau trié du plus petit au plus grand.

Note : dans la correction on utilise la fonction permuter déjà définie en cours et en TD. Rappelons qu'il ne faut pas hésiter à *réutiliser* les fonctions ou procédures que vous avez déjà construites.

Algorithme 15: Inversion de l'ordre d'un tableau

```
variables
    entier tableau[n], i

début
    pour i de 0 à n/2 faire
        permuter(tableau[i],tableau[n − i − 1])
fin
```

Exercice 16. Ecrire un algorithme qui calcule le plus grand écart dans un tableau d'entiers.
Rappel : l'écart entre deux entiers x et y est la valeur absolue de leur différence $|x-y|$.

Algorithme 16: Plus grand écart

```
variables
    entier tableau[n], i, min, max

début
    min ← +∞
    max ← −∞
    pour i de 0 à n − 1 faire
        si tableau[i] < min alors min ← tableau[i]
        si tableau[i] > max alors max ← tableau[i]
    afficher max − min
fin
```

Exercice 17 (*). Ecrire un algorithme de recherche dichotomique permettant de résoudre le problème suivant :
- Données : un tableau *tableau* contenant 1000 entiers (avec répétitions possibles) triés du plus petit au plus grand, ainsi qu'un entier x
- Résultat : l'index de la première occurrence de x dans le tableau s'il est présent, et -1 sinon.

Algorithme 17: Recherche dichotomique avec multiples occurrences

```
variables
    entier tableau[1000], gauche, droite, milieu,x

début
    lire x
    gauche ← 0
    droite ← 999
    répéter
        milieu ← (gauche + droite)/2
        si x > tableau[milieu] alors gauche ← milieu + 1
        si x < tableau[milieu] alors droite ← milieu − 1
        si x = tableau[milieu] alors droite ← milieu
    jusqu'à gauche ≥ droite
    si gauche = droite alors
        afficher gauche
    sinon
        afficher −1
fin
```

Exercice 18 (*). L'algorithme suivant *prétend* trier un tableau. Pensez-vous qu'il termine ? S'il termine effectivement, pensez-vous que le tableau final est bien trié ?

Algorithme 18: tri Bizzaroide

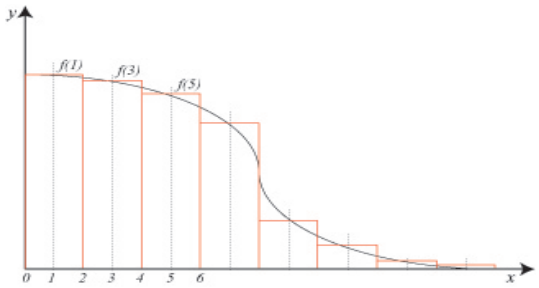
```
variables
    entier tableau[100]
    entier i, t
    booléen permuté

début
    répéter
        permuté ← faux
        pour i ← 1 à 99 faire
            si tableau[i − 1] > tableau[i] alors
                permuter(tableau[i − 1], tableau[i])
                permuté ← vrai
            fin
        fin
    jusqu'à permuté = faux
fin
```

L'algorithme termine effectivement : la boucle "répéter" s'arrête dès lors qu'il n'existe plus aucune paire $(i-1,i)$ telle que tableau$[i-1]$ > tableau$[i]$. Le tableau final est bien trié car, si ce n'était pas le cas, il existerait au moins une paire $(i-1,i)$ telle que tableau$[i-1]$ > tableau$[i]$. Pour information, il s'agit de l'algorithme du *tri à bulles* qui, en pratique, est moins efficace que le tri par insertion ou le tri par selection.

Exercice 19 (*). Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : un tableau f représentant une courbe : chaque couple $(x, f[x])$ du tableau correspond à l'abscisse et l'ordonnée de la courbe.
- Résultat : la surface de la courbe par la méthode des rectangles

La méthode des rectangles est illustrée dans la figure ci-dessous. Pour chaque entier impair t, les coordonnées du $t$ème rectangle sont données par $(t-1,0),(t+1,0),(t-1,f[t]),(t+1,f[t])$.



Algorithme 19: Surface d'une courbe (la fonction abs retourne la valeur absolue d'un réel)

```
variables
    entier f[n], t
    réel surfaceTotale, surfaceRectangle

début
    surfaceTotale ← 0
    pour t de 1 à n − 2 faire
        surfaceRectangle ← 2.0 ∗ abs(f[t])
        surfaceTotale ← surfaceTotale + surfaceRectangle
    fin
    afficher surfaceTotale
fin
```