DEUG MIAS — IF121 — Corrigé du contrôle continu 2

Gilles Peskine

20/12/2002

1 Partie programmation

Voir à la fin le listing d'un programme complet.

Remarques générales

– Attention à ne pas confondre les types (`collier`, `statistiques`) avec les variables (`c`, `s`, `stats`, `col`).
– On crée un seul objet de type `collier`, dans `main`. Cet objet est passé en paramètre à chacune des autres fonctions.
– La fonction `main` n'était pas demandée, elle est juste dans le corrigé pour obtenir un programme complet.
– Attention aux bornes des boucles : pour parcourir un tableau de `M` éléments, on peut écrire `for(i=0;i<M;i++){...}` ou `for(i=0;i<=M-1;i++){...}`.
– Comme `M` est une constante valant `5`, on pouvait utiliser `5` au lieu de `M` dans le programme. Toutefois, l'énoncé disait que le nombre M de perles change de temps en temps : avec une constante, il suffit alors de changer la ligne `const int M = 5` au début du programme et de recompiler.
– La taille d'un collier étant une constante du programme (`M`), on n'a pas besoin de passer la taille en paramètre en même temps que le tableau lui-même.
– La constante `N` désigne le nombre total de couleurs.

Fonction `lire_collier`

– Il n'y a pas de moyen (simple) de déclarer en C++ qu'une variable ne prend que des valeurs entières dans un intervalle donné. On déclare donc `x` comme une variable de type `int` sans autre précision.
– Pour tester si `x` ne désigne pas une couleur, on pouvait écrire `if (x < 0 || x >= N) {...}` ou encore, puisque `N` valait 3, `if (x != 0 && x != 1 && x != 2 && x != 3) {...}`. Ne pas confondre le « et » (`&&`) avec le « ou » (`||`).
– Une fois qu'on a lu la couleur (`cin>>x;`) et râlé si l'utilisateur n'a pas entré un numéro de couleur valide, on stocke la valeur lue dans le tableau (`c[i]=x;`). Il faut faire ceci pour chaque place dans le collier (boucle `for`).
– Les tableaux sont toujours passés par référence en C++, donc l'appel `lire_collier(col)` dans la fonction `main` modifie la variable `col` de `main`.

Fonction `afficher_collier`

– `c` est passé en paramètre à la fonction. Il n'est pas question de le lire ! C'est le travail de `lire_collier`...
– La construction la plus naturelle pour afficher le nom de la couleur est le `switch`. Attention à ne pas oublier les `break` ! (Si on les omet, `gcc -Wall` affiche un avertissement.) On pouvait aussi utiliser une cascade de `if` :
```
if (c[i] == 0) { cout << "noir"; }
else if (c[i] == 1) { cout << "rouge"; }
else if (c[i] == 2) { cout << "vert"; }
else if (c[i] == 3) { cout << "bleu"; }
```
– On affiche une espace entre chaque couleur, c'est plus lisible.
– Il y a une espace en trop à la fin de la ligne, mais ça ne se voit pas...

Fonction `compter`

– Avant de travailler avec le tableau de statistiques `s`, on initialise toutes ses cases à 0. (L'énoncé rappelait bien qu'« il se peut que s contienne des valeurs bizarres ».)

- La deuxième boucle parcourt le collier `c`. Pour chaque perle, on ajoute 1 à la case de `s` correspondant à la couleur de la perle. Comme la couleur de la perle est `c[i]`, cette case es `s[c[i]]`.
- `s[c[i]] += 1;` s'écrit aussi `s[c[i]] = s[c[i]] + 1;` ou `s[c[i]]++;` ou `++s[c[i]];`.

Fonction `nombre_de_couleurs`

- La fonction `compter` était là pour servir dans `nombre_de_couleurs`. Comme les statistiques ne servent que pour `compter`, on définit dans `nombre_de_couleurs` une variable locale `stats` pour les stocker.
- La variable `n` sert à compter le nombre de couleurs distinctes dans le collier. Une couleur `j` est présente dans le collier si et seulement si `stats[j]` est non nul.

Questions subsidiaires

1. L'auteur du programme a quitté le fabricant de colliers et veut présenter son œuvre à un entretien d'embauche. Corriger la faute de grammaire commise dans `main` lorsque le collier est monochrome.
2. Modifier la fonction `afficher_collier` pour qu'elle sépare les couleurs par des virgules et non plus par une simple espace. La difficulté (petite) est qu'on ne veut pas afficher de virgule avant la première couleur ou après la dernière couleur.

2 Partie théorique

L'algorithme

```
Fonction palindromique(Valeur c : tableau d'entiers de 0 à M - 1) Résultat booléen
Variable entière i
Début Fonction
    Pour i allant de 0 à (M div 2) - 1
        Si c[i] = c[M-1-i] alors retourner faux
    Fin Pour
    Retourner vrai
Fin Fonction
```

Remarques

- La fonction palindromique prend comme argument le tableau `c`. Il n'y a rien à lire au clavier : c'est la partie du programme qui *appelle* la fonction palindromique qui se charge de remplir le tableau, en lisant les valeurs au clavier ou par tout autre moyen.
- La variable i est interne à la fonction. On la déclare donc comme une variable locale, pas comme un paramètre (ce n'est pas à l'appelant de la fournir).
- On pouvait faire aller la boucle de 0 à $M-1$. La faire s'arrêter plus tôt est une optimisation. Si on fait aller la boucle jusqu'à $M-1$, la preuve est plus simple.
- En revanche, faire aller la boucle de 0 à M est une erreur : il y n'y a que M cases dans le tableau, pas $M+1$.

Preuve

Il est clair que cette fonction termine toujours (il n'y a pas de boucle tant que).

Un invariant pour cette boucle est : au début du corps de la boucle, on a $c[k] = c[M-1-k]$ pour tout entier k tel que $0 \leq k \leq i-1$. En effet, c'est trivial pour $i = 0$, et si c'est vrai en début d'exécution de la boucle, alors on a en plus $c[i] = c[M-1-i]$ à la fin de la boucle, donc $c[k] = c[M-1-k]$ pour tout k tel que $0 \leq k \leq i$.

Si l'exécution continue jusqu'à sortir normalement de la boucle, on a alors $c[k] = c[M-1-k]$ pour tout k tel que $0 \leq k \leq m$, où $m = (M \operatorname{div} 2) - 1$. On a également $c[M-1-j] = c[j]$ pour tout j tel que $M-1-m \leq j \leq M-1$ en posant $j = M-1-k$.

Si M est pair, alors $M = 2m+2$, et on a $c[k] = c[M-1-k]$ pour tout entier k tel que $0 \leq k \leq m$ ou $m+1 \leq k \leq M-1$, c'est-à-dire pour tout entier k tel que $0 \leq k \leq M-1$.

Si M est impair, alors $M = 2m + 3$, et on a $c[k] = c[M - 1 - k]$ pour tout entier k tel que $0 \leq k \leq m$ ou $m+2 \leq k \leq M-1$. Il reste à prouver que $c[m+1] = c[M-1-(m+1)]$, ce qui est vrai $(M-1-(m+1) = m+1)$.

Dans tous les cas, si la fonction retourne vrai, on a bien un palindrome. Il est d'autre part clair qu'une sortie anticipée n'a lieu que si le collier n'est pas palindromique, ce qui achève de prouver que le résultat retourné par la fonction est correct.

Listing

```
#include <iostream.h>
#include <stdlib.h>
const int M = 5;
const int N = 4;
typedef int collier[M];
typedef int statistiques[4];

void lire_collier(collier c) {
    int i, x;
    for (i = 0; i < M; i++) {
        cin >> x;
        if (x < 0 || x >= N) {
            cerr << "Mauvaise couleur!"
                 << endl;
            exit(1);
        }
        c[i] = x;
    }
}

void afficher_collier(collier c) {
    int i;
    for (i = 0; i < M; i++) {
        switch (c[i]) {
        case 0: cout << "noir"; break;
        case 1: cout << "rouge"; break;
        case 2: cout << "vert"; break;
        case 3: cout << "bleu"; break;
        }
        cout << " ";
    }
    cout << endl;
}

void compter(collier c, statistiques s) {
    int i, j;
    for (j = 0; j < N; j++) {
        s[j] = 0;
    }
    for (i = 0; i < M; i++) {
        s[c[i]] += 1;
    }
}

int nombre_de_couleurs(collier c) {
    int j;
    statistiques stats;
    int n = 0;
    compter(c, stats);
    for (j = 0; j < N; j++) {
        if (stats[j] != 0) {
          n += 1;
        }
    }
    return n;
}

int palindromique(collier c) {
    int i;
    for (i = 0; i <= M / 2 - 1; i++) {
        if (c[i] != c[M-1-i]) return 0;
    }
    return 1;
}

int main() {
    collier col;
    cout << "Entrer les " << M
         << " codes numériques des "
         << "couleurs du collier."
         << endl;
    lire_collier(col);
    cout << "Description du collier :"
         << endl;
    afficher_collier(col);
    cout << "Ce collier comporte "
         << nombre_de_couleurs(col)
         << " couleurs différentes."
         << endl;
    if (palindromique(col)) {
        cout << "Ce collier est un "
             << "palindrome." << endl;
    } else {
        cout << "Ce collier n'est pas un"
             << " palindrome." << endl;
    }
}
```