# “Traitement des ruptures de séquence”

Soutien algo/prog AS 2009

4 novembre 2009

## 1 Notion de rupture

Beaucoup de traitements sur les tableaux reviennent à un parcours simple. Voici par exemple un tableau des ventes de voitures d'occasion dans un garage, trié par vendeur :

| Vendeur | Marque | Montant |
|---|---|---|
| albert | peugeot | 1200 |
| bernard | citroen | 3400 |
| bernard | renault | 5100 |
| charles | peugeot | 45000 |
| charles | mercedes | 12000 |

Totaliser les ventes, compter le nombre de voitures vendues d'un modèle donné, etc. , tout ceci se traite de la même façon : un parcours séquentiel du tableau et un bout de code qui cumule les montants, incrémente un compteur si une condition est remplie.

Mais parfois ça se complique un peu. Supposons que l'on veuille sortir le total des ventes, ou le nombre des ventes, par vendeur.

Le tableau étant classé par vendeur, il suffit de faire un parcours séquentiel, mais il faut tenir compte de la ligne précédente : par exemple en traitant le seconde ligne, il faut remarquer qu'on a changé de vendeur, donc que le total définitif d'albert est 1200, et qu'on commence à totaliser pour bernard. C'est ce qu'il faudra faire à chaque fois qu'on détecte une **rupture de séquence**.

En gros, ceci revient à traiter les données par groupes, concernant un même vendeur.

| Vendeur | Marque | Montant |
|---|---|---|
| albert | peugeot | 1200 |
| bernard | citroen | 3400 |
| bernard | renault | 5100 |
| charles | peugeot | 45000 |
| charles | mercedes | 12000 |

Techniquement ce n'est pas très compliqué, mais il faut penser qu'on ne peut remarquer une rupture de séquence qu'après avoir lu le premier élément du groupe suivant (ou être arrivé à la fin). On a donc un petit décalage, qui amène souvent une erreur classique : l'oubli du dernier élément...

# 2 Exercice

Pour vous familiariser avec cette situation de programmation, vous allez écrire une fonction qui, à partir d'un tableau ordonné de nombres,

par exemple `10, 10, 20, 20, 20, 30 , 40, 40`,

renverra deux tableaux :

- la liste des valeurs, sans répétition : `10, 20, 30, 40`
- le nombre de répétitions : `2, 3, 1, 2`

Vous trouverez un début de projet dans le sous-répertoire
`Soutien-AS-2009/Exos-rupture-sequence/src`
de `/net/exemples/`, avec Makefile, entêtes, tests unitaires, etc.

# 3 Sources

## 3.1 Entêtes des fonctions

```
1  // source "fonctions.hpp"
2
3  // ------------------------------------------------------------------
4  // fonction memeContenu
5  // indique si les n premiers éléments de t1[] et t2[] sont
6  // identiques
7
8  bool memeContenu(int t1[], int t2[], int n);
9
10 // ------------------------------------------------------------------
11 // fonction copieSansRepetition
12 //
13 // copie les n premiers élements du tableau t[] (ordonné) dans valeurs[]
14 // en supprimant les répétitions
15 // retourne le nombre d'élements mis dans t2
16
17 // exemple : si t[] contient   10,20,20,20,20,30,40,40
18 // valeurs[] contiendra 10,20,30,40
19 // et la fonction retournera 4
20
21 int copieSansRepetition(int t[], int n, int valeurs[]);
22
23 // ------------------------------------------------------------------
```

```
// fonction compterRepetitions
//
// copie les n premiers élements du tableau t[] (ordonné) dans valeurs[]
// en supprimant les répétitions
// met dans repetitions[] le nombre de répetitions des éléments de t2
// retourne le nombre d'élements mis dans t2 / t3

// exemple : si t[] contient   10,20,20,20,20,30,40,40
// valeurs[] contiendra 10,20,30,40
// repetitions[] contiendra 1,4,1,2
// et la fonction retournera 4


int compterRepetitions(int t[], int n, int valeurs[], int repetitions[]);

// --
```

## 3.2 Code des fonctions (bouchons)

```
// source "fonctions.cc"

#include "fonctions.hpp"


// fonctions utilitaires sur les tableaux
// M Billaud, 4/11/09

// ------------------------------------------------------------------
// fonction memeContenu
// indique si les n premiers éléments de t1[] et t2[] sont
// identiques

bool memeContenu(int t1[], int t2[], int n)
{
  return true; // bouchon

  // wikipedia http://fr.wikipedia.org/wiki/Bouchon_(informatique)
  // En informatique, un bouchon est un code qui n'effectue aucun
  // traitement et retourne toujours le même résultat. Un bouchon sert
  // d'alternative temporaire à un code qui n'est pas utilisable parce
  // qu'il n'est pas encore codé ou qu'il est en cours d'évolution.

}
```

```
26
   // ------------------------------------------------------------------------
28 // fonction copieSansRepetition
   //
30 // copie les n premiers élements du tableau t[] (ordonné) dans valeurs[]
   // en supprimant les répétitions
32 // retourne le nombre d'élements mis dans t2

34 // exemple : si t[] contient   10,20,20,20,20,30,40,40
   // valeurs[] contiendra 10,20,30,40
36 // et la fonction retournera 4

38 int copieSansRepetition(int t[], int n, int valeurs[])
   {
40   return -1; // bouchon
   }
42

44 // ------------------------------------------------------------------------
   // fonction compterRepetitions
46 //
   // copie les n premiers élements du tableau t[] (ordonné) dans valeurs[]
48 // en supprimant les répétitions
   // met dans repetitions[] le nombre de répetitions des éléments de t2
50 // retourne le nombre d'élements mis dans t2 / t3

52 // exemple : si t[] contient   10,20,20,20,20,30,40,40
   // valeurs[] contiendra 10,20,30,40
54 // repetitions[] contiendra 1,4,1,2
   // et la fonction retournera 4
56

58 int compterRepetitions(int t[], int n, int valeurs[], int repetitions[])
   {
60   return -1; // bouchon
   }
62

   // --
```

### 3.3 Programme de test

```
1 // source "test_fonctions.cc"
```

```
3 // tests unitaires

5 #include <cassert>
  #include <iostream>
7
  #include "fonctions.hpp"
9
  using namespace std;
11

13 void test_memeContenu()
  {
15   cout << "- fonction memeContenu() ... ";
     int t1 [] = { 1, 2 ,3, 4};
17   int t2 [] = { 1, 2, 5, 4 };

19   assert( memeContenu(t1,t2,0) );
     assert( memeContenu(t1,t2,1) );
21   assert( memeContenu(t1,t2,2) );
     assert(!memeContenu(t1,t2,3) );
23   assert(!memeContenu(t1,t2,4) );

25   cout << "OK" << endl;
  }
27
  void test_copieSansRepetition()
29 {
     int t1 [] = { 10, 10, 10, 20, 30, 30, 40};
31   int r[] = {10, 20, 30, 40};
     int v[10];
33
     cout << "- fonction copieSansRepetition() ... ";
35
     assert (( copieSansRepetition(t1,0,v) == 0));
37   assert (( copieSansRepetition(t1,1,v) == 1)
             && memeContenu(v, r, 1));
39   assert (( copieSansRepetition(t1,2,v) == 1)
             && memeContenu(v, r, 1));
41   assert (( copieSansRepetition(t1,3,v) == 1)
             && memeContenu(v, r, 1));
43   assert (( copieSansRepetition(t1,4,v) == 2)
             && memeContenu(v, r, 2));
```

```
45    assert (( copieSansRepetition(t1,5,v) == 3)
              && memeContenu(v, r, 3));
47    assert (( copieSansRepetition(t1,6,v) == 3)
              && memeContenu(v, r, 3));
49    assert (( copieSansRepetition(t1,7,v) == 4)
              && memeContenu(v, r, 4));
51
      cout << "OK" << endl;
53  }


55
    void test_compterRepetitions()
57  {
      int t1 [] = { 10, 10, 10, 20, 30, 30, 40}; // exemple
59    int r[] = {10, 20, 30, 40};  // resultats attendus
      int v[10];
61    int n[10];


63
      cout << "- fonction compterRepetitions() ... ";
65
      assert (( compterRepetitions(t1,0,v,n) == 0));
67
      int n1[] = { 1 };
69    assert (( compterRepetitions(t1,1,v,n) == 1)
              && memeContenu(v, r, 1)
71            && memeContenu(n,n1,1));

73    int n2[] = { 2 };
      assert (( compterRepetitions(t1,2,v,n) == 1)
75            && memeContenu(v, r, 1)
              && memeContenu(n, n2, 1));
77
      int n3[] = { 3 };
79    assert (( compterRepetitions(t1,3,v,n) == 1)
              && memeContenu(v, r, 1)
81            && memeContenu(n, n3, 1));

83    int n4[] = { 3, 1};
      assert (( compterRepetitions(t1,4,v,n) == 2)
85            && memeContenu(v, r, 2)
              && memeContenu(n, n4, 2));
87
```

```
    int n5[] = { 3, 1, 1};
89  assert (( compterRepetitions(t1,5,v,n) == 3)
            && memeContenu(v, r, 3)
91          && memeContenu(n, n5, 3));

93  int n6[] = { 3, 1, 2};
    assert (( compterRepetitions(t1,6,v,n) == 3)
95          && memeContenu(v, r, 3)
            && memeContenu(n, n6, 3));
97
    int n7[] = { 3, 1, 2, 1};
99  assert (( compterRepetitions(t1,7,v,n) == 4)
            && memeContenu(v, r, 4)
101         && memeContenu(n, n7, 7));

103 cout << "OK" << endl;
  }
105
  int main()
107 {
    cout << "Tests unitaires du module 'fonctions.cc'" << endl;
109 test_memeContenu();
    test_copieSansRepetition();
111 test_compterRepetitions();
  }
113
  // --
```

### 3.4 Makefile

```
CXXFLAGS=-Wall -pedantic
2
EXEC=test_fonctions
4
all: $(EXEC)
6
fonctions.o: fonctions.cc fonctions.hpp
8
test_fonctions.o: test_fonctions.cc fonctions.hpp
10
test_fonctions: test_fonctions.o fonctions.o
12      $(LINK.cc) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

```
14
  pretty:
16          astyle --style=gnu *.cc *.hpp

18 clean:
          rm -f *.o *~ *.orig
20
  mrproper: clean
22          rm $(EXEC)
```