# TD : preuves et terminaisons

## 1 Terminaison

**Exercice 1.** 1. On considère la fonction

```
void bouclefor0(int n){
   for (int i = n; i>=0; i--)
     printf("i=%d\n",i);
}
```

Etudier la terminaison.

*Démonstration.* Le variant est $i$. □

2. On considère la fonction

```
void bouclefor1(int n){
  for (int i = 0; i<n; i++)
     printf("i=%d\n",i);
}
```

Etudier la terminaison.

*Démonstration.* Le variant est $n - i$ □

3. On considère la fonction

```
void bouclefor2(int n){
  for (int i = 0; i<n; i++)
     i=i-1;
     printf("i=%d\n",i);
  }
}
```

Etudier la terminaison.

*Démonstration.* $i$ est égal à -1 à la fin de chaque tour de boucle.
Il ne peut donc jamais atteindre de valeur strictement positive. □

4. On considère la fonction

```
void bouclewhile(int n){
  int i=0;
  while(i<n){
     i=i-2;
     printf("i=%d\n",i);
  }
}
```

Etudier la terminaison.

*Démonstration.* Supposons que les `int` sont écrits sur 32 bits. Dans certaines implémentations le comportement des entiers est cyclique.
A la fin du premier tour, $i$ vaut -2, puis $-4$ etc.
Si $n = 2^{31} - 1$ (max int), $i$ n'est jamais plus grand que $n$ puisqu'il vaut successivement

$$0, -2, -4, \ldots, -2^{-31}, 2^{31} - 2$$

et $i$ n'est jamais strictement plus grand que □

5. On considère la fonction

```c
int g(int x);

void bouclefor3(int n){
   for (int i = 0; i<n; i++)
      printf("i=%d\n",g(i));
}
```

Etudier la terminaison.

*Démonstration.* Elle dépend bien sûr de celle de $g$. □

**Exercice 2.** On considère la fonction

```c
void f(int p){
  int c=0;
  while (p > 0){
    if (c == 0){
      p = p - 2;
      c = 1;
       }
    else{
      p = p + 1;
       c = 0;
       }
  }
}
```

On numérote les passages dans la boucle à partir de 1. Le passage 0 désigne ce qui se passe avant la boucle. Les variables du programme sont indicées selon le tour de boucle : $p_0$ désigne la valeur de `p` juste avant la boucle ; $p_i$ la valeur de `p` à la fin du passage $i$.

On cherche un variant $v_i$ sous forme de combinaison linéaire entière de $p_i$ et $c_i$. On veut donc trouver $a, b$ dans $\mathbb{N}$ telle que la suite

$$(ap_i + bc_i)_{i\in\mathbb{N}} \text{ soit strictement décroissante.}$$

1. Expliquer pourquoi aucune des variables de ce programme n'est un variant.
2. Ecrire une fonction

```c
int cl(int a, int b, int p0, int res[100])

```

qui prend en paramètre un nombre $p_0$ et remplit le tableau `res` (assez gros) avec les CL $ap_i + bc_i$ jusqu'à trouver $p = 0$. Elle renvoie le nombre de passages dans la boucle.

3. Ecrire une fonction

```
bool check (int n, int tab[n])
```

qui prend en paramètre un nombre $n$ et un tableau et indique si ce tableau est rangé par ordre décroissant strictement.

4. Ecrire une fonction

```
bool find(int ab[2])
```

qui lance `cl` jusqu'à trouver un couple $a, b$ telle que la suite $(ap_i + bc_i)_i$ soit strictement décroissante.

5. A partir de ce résultat établir la preuve de terminaison de `f`.

*Démonstration.* — les fonctions sont données plus bas

— $2p + 3c$ semble un bon candidat.

Notations : On note $c_i$ le contenu de **c** à l'étape $i$ et $p_i$ le contenu de **p** à la fin du passage $i$ dans la boucle. $p_0$ et $c_0$ sont les valeurs avant l'entrée dans la boucle.

On cherche le variant comme une combinaison linéaire de $p_i$ et $c_i$.

— **On vérifie que lorsque le variant est négatif l'algorithme s'arrête**.

Si $2p_i + 3c_i \leq 0$ alors $2p_i \leq -3c_i$ et comme $c_i \geq 0$ (si on en doute, le montrer par récurrence), on trouve $2p_i \leq 0$ donc $p_i \leq 0$ et le programme s'arrête.

— Cas de base : **Avant d'entrer dans la boucle** :

Si $p_0 \leq 0$, on n'entre pas dans la boucle et le programme termine.

On suppose $p_0 \in \mathbb{N}^*$. Donc $2p_0 + 3c_0 = 2p_0 > 0$ avant l'entrée dans la boucle.

— Hérédité. **A la fin de l'étape $i$ :**

Si $p_i \leq 0$, alors on sort de la boucle (pas de passage $i + 1$) et le programme termine.

On suppose $p_i > 0$.

On compare $2p_i + 3c_i$ avec $2p_{i+1} + 3c_{i+1}$.

1. Si $c_i = 0$, alors $c_{i+1} = 1$, $p_{i+1} = p_i - 2$ et $2p_i + 3c_i = 2p_i$.

$$0 \leq 1 \leq 2p_{i+1} + 3c_{i+1} = 2p_i - 4 + 3 = 2p_i - 1 < 2p_i = 2p_i + 3c_i.$$

2. Si $c_i = 1$, alors $c_{i+1} = 0$, $p_{i+1} = p_i + 1$ et $2p_i + 3c_i = 2p_i + 3$.

$$0 \leq 2p_i \leq 2p_{i+1} + 3c_{i+1} = 2p_i + 2 + 3 \times 0 = 2p_i + 2 < 2p_i + 3 = 2p_i + 3c_i.$$

— La quantité $2p_i + 3c_i$ est strictement décroissante. Et lorsqu'elle est négative, on sort de la boucle. Le programme termine.

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <stdbool.h>


/*----------AFFICHAGE----------*/
void affiche_tab(int n, int tab[n]){
  for (int i=0; i<n; i++){
    printf("%d, ", tab[i]);
  }
  printf("\n");
}


void f(int p){// p positif
  int c=0;
  while (p > 0){
    if (c == 0){
      p = p - 2;
      c = 1;
       }
    else{
      p = p + 1;
      c = 0;
       }
  }
}

int g(int x){
  return 3;
}


void bouclefor0(int n){
   for (int i = n; i>=0; i--)
    printf("i=%d\n",i);
}


void bouclefor1(int n){
   for (int i = 0; i<n; i++)
    printf("i=%d\n",i);
}

void bouclewhile(int n){// ne termine pas
  for (int i = 0; i<n; i++){
    i=i-2;
    printf("i=%d\n",i);
  }
}
int g(int x);

void bouclefor3(int n){
```

```
55    for (int i = 0; i<n; i++)
56      printf("i=%d\n",g(i));
57  }
58
59
60  int cl(int a, int b, int p, int res[100]){//p>0
61    int c=0, cpt = 1;
62    res[0]=a*p+b*c;
63    while (p > 0 && cpt<100){
64      if (c == 0){
65        p = p - 2; c = 1;}//if c==0
66      else{p = p + 1; c = 0;}//else
67      res[cpt] = a*p +b*c; cpt++;
68    }//while
69    return cpt;// nb de CL calculées
70  }
71
72  void bench_cl(){
73    int res[100];
74    int n = cl(1,2,15, res);
75    printf("nb de tours : %d : ",n);
76    affiche_tab(n,res);
77
78    n = cl(2,3,15, res);
79    printf("nb de tours : %d : ",n);
80    affiche_tab(n,res);
81  }
82
83  bool check(int n, int res[n]){
84    for (int i=0; i<n-1; i++){
85      if (res[i]<=res[i+1])
86        return false;
87    }
88    return true;
89  }
90
91  void bench_check(){
92    int res[100];
93    int n = cl(1,2,15, res);
94    printf("cl (1,2,15, res) donne un tableau decroissant strict : %d\n",
95     check(n,res));
96
97    n = cl(2,3,15, res);
98    printf("cl (2,3,15, res) donne un tableau decroissant strict : %d\n",
99     check(n,res));
100 }
101
102 // renvoie true si une CL est trouvée
103 bool find(int coefs [2]){
104   int res[100];
105   for (int a=1; a<10; a++){
106     for (int b=1; b<10; b++){
107       int n = cl(a,b,15,res);
108       printf("a=%d, b=%d;",a,b);
109       if (check(n,res)){
110       coefs[0]=a; coefs[1]=b;
111       return true;
```

```
112     }//if
113           printf("NOK\n");}//for b
114   }//for a
115    printf("\n"); return false;// pas trouvé de CL
116 }
117
118 void test_find(){
119   int res[2];
120   bool trouve = find(res);
121   if (trouve)
122     printf("test=%d : %dp+%dc semble strict. decr.\n",
123      trouve, res[0], res[1]);
124   else
125     printf("test=%d : pas de CL trouvée\n",
126      trouve);
127 }
128
129 int main(){
130   test_find();
131   //bench_check();
132   //bench_cl();
133   //bouclefor1(4);
134   //bouclewhile(4);
135
136   return 0;
137 }
```

Listing 1 – Code des fonctions pour trouver le variant

□