Essayer un programme

1 Quand l'évidence nous trompe

On donne trois nombres `x, y` et `z`. On demande d'en échanger les valeurs de façon que `x` devienne le plus petit des trois et `z` le plus grand, ou en d'autres termes que `x, y, z` soient ordonnés, croissants. L'évidence est aidée par les connaissances : *je sais échanger les valeurs de deux variables* `a` *et* `b` *:*

$$\{\, t \leftarrow a\,;\; a \leftarrow b\,;\; b \leftarrow t\,\}$$

Je sais aussi échanger deux nombres s'ils ne sont pas dans le bon ordre, c'est-à-dire les ordonner :

$$\textbf{Si } a > b \textbf{ alors } \{\, t \leftarrow a\,;\; a \leftarrow b\,;\; b \leftarrow t\,\}$$

Je veux que les trois nombres soient ordonnés. Je vais les ordonner deux à deux de toutes les manières possibles. Or il y a trois façons de prendre les trois nombres deux à deux : `x, y` *;* `y, z` *et* `x, z`*. Voici donc mon programme*[1] *:*

```
01
02 #include <stdlib.h>           /* Reference a des informations de base */
03 void main(argc, argv)         /* Pour executer depuis la ligne de cmd */
04     int argc;                 /* Nombre d'arguments n1 n2 n3          */
05     char *argv[];             /* Les elements de la ligne de commande */
06 {                            /* Ca commence ici                      */
07     int x, y, z ;             /* Pour ranger les arguments et         */
08     int t ;                   /* une variable temporaire              */
09     if (argc != 4)            /* S'il n'y a pas le bon nombre d'arg.  */
10     {                         /* Signaler l'erreur et sortir          */
11         printf("usage: %s <x> <y> <z>, x, y et z entiers\n", argv[0]);
12         exit(1);
13     }                         /* Pas d'erreur, on commence le traitement */
14     x = atoi(argv[1]);        /* Mettre les arguments dans les variables */
15     y = atoi(argv[2]);
16     z = atoi(argv[3]);
17
18     if (x > y) { t = x; x = y; y = t; }
19     if (y > z) { t = y; y = z; z = t; }
20     if (x > z) { t = x; x = z; z = t; }
21
22     printf("%d < %d < %d\n", x, y, z);   /* Fournir le resultat */
23     exit(0);                             /* Sortie sans erreur  */
24 }
```

Je l'essaie avec `x = 3`*,* `y = 1` *et* `z = 2`*.*

[1] Si, au point ou vous en êtes, certaines lignes vous semblent incompréhensibles, dites-vous que seules les lignes 18 à 20 sont importantes

Comme `x = 3 > y = 1` *je les échange :*

$$x = 1 \;\; y = 3 \;\; z = 2$$

Maintenant, comme `y = 3 > z = 2`*, je les échange :*

$$x = 1 \;\; y = 2 \;\; z = 3$$

Enfin, comme `x = 1 < z = 3`*, je ne fais rien et j'ai le résultat cherché.* Voilà un bon programme, facilement trouvé et bien construit.

2 Essayer le programme

Après avoir rentré ce programme dans l'ordinateur, nous faisons des essais avec les triplets suivants :

```
5 3 8
6 5 5
3 8 5
2 3 2
```

Le programme fonctionne correctement, nous avons fait beaucoup d'essais pour un aussi petit programme. Malheureusement, **il est faux**. Comment le découvrir ?

> **Essayer un programme peut servir à montrer qu'il contient des erreurs**
> **jamais qu'il est juste**

Dans cet exemple, il est aisé de trouver un cas montrant que le programme est faux. `x = 2, y = 3` et `z = 1` donne en sortie : `2 < 1 < 3`.

Que faire ? Une seule réponse possible :

> **Raisonner pour que votre programme soit juste par construction**

3 Corriger l'erreur

3.1 Localiser l'erreur

Trop souvent, on s'acharne sur un morceau de programme où l'on croit détecter l'erreur ; trop souvent, un programme comporte des sections délicates, pour lesquelles on a laborieusement

élaboré un algorithme difficile. Quand l'erreur apparaî t, on a naturellement tendance à mettre en cause la partie difficile, suspectant quelque défaillance dans le raisonnement, alors que c'est dans les parties faciles que l'attention s'est relâchée et que l'erreur s'est glissée.

Si l'erreur n'est pas là où vous la cherchez
c'est qu'elle est ailleurs...

Le travail le plus important est de localiser l'erreur. Vous avez la possibilité de mettre dans le programme des *instructions de surveillance*, que vous retirerez quand tout sera correct.

Prévoyez dans votre programme
des facilités de mise au point

```
#include <stdlib.h>


#define DEBUG  /* Je declare que je veux y voir quelque chose */


void main(argc, argv)
    int argc;
    char *argv[];
{
    int x, y, z ;
    int t ;
    if (argc != 4)
    {
        printf("usage: %s <x> <y> <z>, x, y et z entiers\n", argv[0]);
        exit(1);
    }
    x = atoi(argv[1]);
    y = atoi(argv[2]);
    z = atoi(argv[3]);


#ifdef DEBUG
printf ("Avant x = %d, y = %d, z = %d\n", x, y, z) ;
#endif


    if (x > y) { t = x; x = y; y = t; }


#ifdef DEBUG
printf ("Apres x et y  : x = %d, y = %d, z = %d\n", x, y, z) ;
#endif
```

```
    if (y > z) { t = y; y = z; z = t; }


#ifdef DEBUG
printf ("Apres y et z : x = %d, y = %d, z = %d\n", x, y, z) ;
#endif


    if (x > z) { t = x; x = z; z = t; }

    printf("%d < %d < %d\n", x, y, z);
    exit(0);
}
```

4 Corriger l'erreur

Ayant localisé l'erreur, deux cas sont possibles :

- C'est une faute de frappe ou d'étourderie : corrigez-la.
- C'est une faute plus profonde, et notamment une faute de raisonnement : n'essayez pas de la corriger immédiatement. Il y a trop de risques d'introduire ainsi une nouvelle erreur, peut-être plus grave que la précédente. N'insistez pas :

> **Corrigez tout de suite une faute ponctuelle.**
> **Pour une faute de raisonnement,**
> **réfléchissez d'abord,**
> **vous corrigerez plus tard...**

5 Comprendre le programme

Dans ce programme de trois lignes, aucune n'est fausse. C'est donc dans son principe que le programme est faux : il n'y a ni faute de frappe, ni étourderie... C'est le cas typique où il faut réfléchir ! Nous pouvons accompagner le déroulement de ce programme, comme nous accompagnons quelqu'un dans sa reflexion. Essayons de suivre dans le détail pour voir où il déraille. Nous ne pouvons le faire sur un cas particulier, nous savons en effet qu'il arrive que le programme fonctionne correctement. Il faut donc s'attaquer au cas particulier. Nous allons *donner du sens*, en *traduisant*, c'est-à-dire en *commentant*, en *reformulant*, sans **ajouter** d'**interprétation**, de **critique** ni de **supposition**. Nous n'allons pas, non plus, nous précipiter pour fournir une **solution**.

> **Il y a trois temps indispensables :**
>
> - **Le temps de voir**
> - **Le temps de comprendre**
> - **Le temps de conclure**

Le temps de voir

Voyons d'abord la première ligne. Ou bien `x` $\leq$ `y` et elle ne fait rien, donc elle laisse `x` $\leq$ `y`, ou bien `x > y` et elle échange ces valeurs, laissant `x < y`. Donc, dans tous les cas, à la fin de la ligne, `x` $\leq$ `y` :

$$\textbf{Si } x > y \textbf{ alors } \{\, t \leftarrow x\,;\ x \leftarrow y\,;\ y \leftarrow t\,\}$$

$$/* \ x \leq y \ */$$

Examinons maintenant l'effet de la deuxième ligne. Le test distingue deux cas :

1. `y` $\leq$ `z` rien n'est fait
2. `y > z` ces deux valeurs sont échangées

N'oublions pas que nous savons que `x` est plus petit que `y`. Reformulons donc ceci en explicitant cette connaissance :

1. `x` $\leq$ `y` et `y` $\leq$ `z`

   Nous voyons, dans ce cas que le résultat est déjà acquis : non seulement il n'y a pas à échanger `y` et `z`, mais en plus le travail est déjà fini.

2. `x` $\leq$ `y` et `y > z`

   On échange `y` et `z`. les **valeurs** sont maintenant disposées comme suit :

   $$a_1 \ a_3 \ a_2$$

   Nous savons que le premier est inférieur ou égal au troisième, et que le second est inférieur au troisième. En tenant compte du changement de nom des **contenus** dans ce deuxième cas :

   $$/* \ x \leq z \ et \ y < z \ */$$

Remettons tout ceci dans notre programme :

Si $x > y$ **alors** $\{ \ t \leftarrow x \ ; \ x \leftarrow y \ ; \ y \leftarrow t \ \}$
$/* \ x \leq y \ */$
Si $y \leq z$ **alors fini**
Si $y > z$ **alors** $\{ \ t \leftarrow y \ ; \ y \leftarrow z \ ; \ z \leftarrow t \ \}$
$/* \ x \leq z \ et \ y < z \ */$
Si $x > z$ **alors** $\{ \ t \leftarrow x \ ; \ x \leftarrow z \ ; \ z \leftarrow t \ \}$

Il est maintenant *évident* que la troisième ligne ne sert à rien. *Je sais* que `x` $\leq$ `z`. La question :

Si $x > z$ **alors**

recevra toujours la réponse **FAUX**, le troisième échange ne se fera jamais...

Le temps de comprendre

6 Actions et situations

Interprétons maintenant ce que nous venons de faire. Nous avions un programme dont nous savions qu'il était faux, mais non pourquoi. Pour le découvrir, nous avons explicité la situation créée par l'exécution de chaque ligne. Nous avons reformulé, non pas les actions, mais les états successifs du programme.

> **Pour comprendre un programme,**
> **explicitez les situations qu'il engendre**

La clef de cette affaire, c'est cette notion même de *situation*[2]. **Une instruction d'un programme commande une action dont l'effet est de modifier une situation. Un programme dit quelle suite d'actions fera passer de la situation initiale, celle des données, à la situation finale, celle des résultats.**

Le sens d'un programme est dans la traduction des actions en situations engendrées par son exécution.

Nous en tirons deux conséquences essentielles.

- Le sens d'un programme est dans la traduction en *situations*[3].

  > **Ne prenez pas le lecteur pour un imbécile.**
  > **Pas de commentaire paraphrasant les instructions du programme.**

  Par contre, un commentaire rappelant ce qu'était `x` me permettra de saisir qu'il est trop petit d'une unité, et donc pourquoi on l'augmente de un. Les meilleurs commentaires sont ceux qui précisent la situation réalisée en un point du programme.

  > **Un commentaire doit dire**
  > **la situation réalisée là où il est mis.**

- On ne peut décider de l'action à prendre que si l'on connaî t la situation dans laquelle on se trouve :

  > **Pour créer le programme**
  > **partir des situations**

[2]Faites le parallèle avec une conversation. Ce qui permet d'être efficace dans l'échange, c'est *l'écoute, la reformulation* et surtout la verbalisation des *états* et des *sentiments*.

[3]Certains mettent des commentaires annonçant ce qui va être fait :

$$/* \ on\ augmente\ x\ de\ 1 \ */$$

$$x \leftarrow x + 1$$

Je sais lire. Ce commentaire ne m'apprend rien !

C'est en sachant où l'on en est et où on veut aller, qu'on a une petite chance de trouver le chemin à suivre. Beaucoup de programmeurs, ayant à créer un programme, analysent le problème puis se demandent ce qu'ils vont faire. C'est vouloir deviner l'action en espérant tomber juste.

Ne vous demandez pas
que vais-je faire ?
Demandez vous plutôt
où en suis-je ?

7 Construire le programme

Comme nous savons comment fonctionne le programme, nous voyons que l'on a, après le deuxième échange,

$$\begin{array}{l}
\textbf{Si}\ x > y\ \textbf{alors}\ \{\, t \leftarrow x\,;\ x \leftarrow y\,;\ y \leftarrow t\,\} \quad (1)\\
/*\ \ x \leq y\ \ */\\
\textbf{Si}\ y \leq z\ \textbf{alors fini}\\
\textbf{Si}\ y > z\ \textbf{alors}\ \{\, t \leftarrow y\,;\ y \leftarrow z\,;\ z \leftarrow t\,\} \quad (2)\\
/*\ \ x \leq z\ et\ y < z\ \ */
\end{array}$$

On voit qu'après la ligne (2) `x` et `y` ne sont pas nécessairement dans le bon ordre. C'est d'ailleurs bien l'erreur que nous avons constatée. Il faut donc une troisième ligne pour ordonner `x` et `y`.

Vous objecterez, peut-être, que cela est inutile puisque cela a déjà était fait en ligne (1). Mais la *situation* produite par la ligne (1) a été modifiée par la ligne (2).

Vous pouvez aussi vous demander si la ligne (3) ne va pas mettre en cause `z`. Non, car nous savons que `z` est le plus grand des trois nombres et qu'il est maintenant définitivement à sa place.

Voici le nouveau programme :

$$\begin{array}{l}
\textbf{Si}\ x > y\ \textbf{alors}\ \{\, t \leftarrow x\,;\ x \leftarrow y\,;\ y \leftarrow t\,\} \quad (1)\\
/*\ \ x \leq y\ \ */\\
\textbf{Si}\ y \leq z\ \textbf{alors fini}\\
\textbf{Si}\ y > z\ \textbf{alors}\ \{\, t \leftarrow y\,;\ y \leftarrow z\,;\ z \leftarrow t\,\} \quad (2)\\
/*\ \ x \leq z\ et\ y < z\ \ */\\
\textbf{Si}\ x > y\ \textbf{alors}\ \{\, t \leftarrow x\,;\ x \leftarrow y\,;\ y \leftarrow t\,\} \quad (3)
\end{array}$$

P1

Ce programme est juste, non parce que nous l'avons essayé, mais parce que nous connaissons

les situations qu'il engendre, nous savons qu'elles représentent tous les cas possibles et qu'elles aboutissent toujours à la suite ordonnée cherchée.

8 Travailler un programme

> [a]**Hâtez-vous lentement et sans perdre courage**
> **Ving fois sur le métier remettez votre ouvrage**
> **Polissez-le sans cesse et le repolissez. . .**
>
> [a]Boileau

Quand vous avez enfin obtenu un programme correct, il ne faut pas croire le travail fini. Nous allons voir sur le programme `P1` qu'il faut l'améliorer.

Le test `y > z` en ligne (2) est inutile.

Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (1)
$/* \ x \leq y \ */$
Si $y \leq z$ **alors fini**
$\{ t \leftarrow y ; y \leftarrow z ; z \leftarrow t \}$ (2) $\longleftarrow$ P2
$/* \ x \leq z \ et \ y < z \ */$
Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (3)

Inconvénient : La ligne (2) est toujours exécutée. Il est possible d'utiliser un **sinon** et voici une troisième version[4] :

Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (1)
$/* \ x \leq y \ */$
Si $\mathbf{y} \leq \mathbf{z}$ *alors fini sinon* $\{ \mathbf{t} \leftarrow \mathbf{y} ; \mathbf{y} \leftarrow \mathbf{z} ; \mathbf{z} \leftarrow \mathbf{t} \}$ (2) $\longleftarrow$ P3
$/* \ x \leq z \ et \ y < z \ */$
Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (3)

Remarquez que l'on peut améliorer la lisibilité en inversant le test :

Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (1)
$/* \ x \leq y \ */$
Si $y > z$ **alors** $\{ t \leftarrow y ; y \leftarrow z ; z \leftarrow t \}$ **sinon fini** (2) $\longleftarrow$ P4
$/* \ x \leq z \ et \ y < z \ */$
Si $x > y$ **alors** $\{ t \leftarrow x ; x \leftarrow y ; y \leftarrow t \}$ (3)

[4]On a regroupé les deux cas dans une même sélection

Le **sinon** de la ligne (2) est-il indispensable ? Non, car on ne rajoute qu'**un test inutile**, mais aucune action n'est effectuée. D'où le cinquième programme :

Si $x > y$ **alors** $\{\, t \leftarrow x\,;\; x \leftarrow y\,;\; y \leftarrow t\,\}$ (1)
$/*\ x \leq y\ */$
Si $y > z$ **alors** $\{\, t \leftarrow y\,;\; y \leftarrow z\,;\; z \leftarrow t\,\}$ (2) $\longleftarrow$ P5
$/*\ x \leq z$ *et* $y < z\ */$
Si $x > y$ **alors** $\{\, t \leftarrow x\,;\; x \leftarrow y\,;\; y \leftarrow t\,\}$ (3)

Vous voyez que le travail n'est pas fini au moment où le programme fonctionne sans erreur. La *mise-en-sens* a comme conséquence une amélioration du programme.

Faisons donc une reformulation (mise-en-sens) de ce que nous avons fait. Les cinq programmes ordonnent une suite de trois nombres par la même méthode : On ordonne la paire `x`–`y` s'il y a lieu, puis on ordonne la paire `y`–`z` et enfin si nécessaire à nouveau, la paire `x`–`y`. Les cinq programmes réalisent le *même algorithme*. Mais `P1` évalue inutilement le test `y > z` dans certains cas. Dans `P5` aussi il y a un test inutile comme nous l'avons souligné en le construisant.

Vous objecterez que dans un programme aussi simple, un test inutile est sans importance. Mais,

1. Si ce travail de trois ligne est enfermé dans une boucle qui se répète 10000 fois...

   Ne faites pas de travail inutile

   Pour avoir des programmes efficaces, c'est en éliminant les petits calculs inutiles que vous aurez les meilleurs résultats.

2. Le programme `P4` révèle une maî trise complète de la part du programmeur. C'est parce qu'il sait à chaque instant où il en est exactement qu'il peut à chaque instant prendre la meilleure décision. Par contre le programme `P5` laissera toujours subsister un doute : le programmeur a-t-il vraiment compris ce qu'il a fait ? Ce programme est-il le fruit d'une réflexion solide ou le résultat d'un *bricolage* inavouable ?

9 Discuter les choix

Croyez-vous vraiment avoir épuisé le sujet ? Lorsque nous avons entamé cette étude, nous avons commencé par ordonner la première paire, le reste a été imposé par l'analyse de la *situation* qui en résulte. Il est extrêmement important de percevoir que ce choix est arbitraire. *Nous aurons a développer notre sens de l'observation.* Car, si nous ne percevons pas qu'il y a eu un choix, nous n'aurons pas la possibilité d'en discuter.

Remettons en cause ce choix. Il nous reste deux possibilités : `y`–`z` et `x`–`z`. Essayons `x`–`z`. Comme précédemment, nous posons la première action, puis nous étudions la situation engendrée :

$$\begin{array}{l}\textbf{Si } x > z \textbf{ alors } \{\, t \leftarrow x\,;\; x \leftarrow z\,;\; z \leftarrow t\,\} \\ /* \;\; x \leq z \;\; */\end{array}$$

Ainsi `x` et `z` sont bien ordonnés, mais on ne sait rien de `y`.

Comparons d'abord `x` et `y`.

- Si `x` $\leq$ `y`, cette paire est bien ordonnée. Nous avons alors $/* \;\; x \leq y$ *et* $x \leq z \;\; */$, `x` est le plus petit des trois, il est à sa place. Il faut maintenant mettre en ordre `y` et `z` si c'est nécessaire.
- Si maintenant `x > y`, nous avons alors `x` compris entre `y` et `z`, c'est-à-dire que l'ordre entre les trois éléments est connu, mais que `x` et `y` sont inversés.

D'où le programme :

$$\begin{array}{ll}\textbf{Si } x > z \textbf{ alors } \{\, t \leftarrow x\,;\; x \leftarrow z\,;\; z \leftarrow t\,\} & (1) \\ /* \;\; x \leq z \;\; */ & \\ \textbf{Si } x > y \textbf{ alors } \{\, t \leftarrow y\,;\; y \leftarrow x\,;\; x \leftarrow t\,\} & (2) \\ \textbf{Si } y > z \textbf{ alors } \{\, t \leftarrow y\,;\; y \leftarrow z\,;\; z \leftarrow t\,\} & (3)\end{array} \qquad \boxed{\text{P6}}$$

Ce programme ressemble à `P4`, trois tests, trois échanges. Excepté que si vous faites bien le compte, `P6` termine le tri en trois questions et deux échanges au maximum, alors que `P4` termine en trois questions et peut-être trois échanges, ce qui dans une boucle de 10000...