# Chapitre 2

# L'ordinateur prend des décisions

## 2.1 Programmes maladroits et programmes impossibles

Dans les programmes que nous avons vus jusqu'à présent, l'ordinateur utilisait la puissance de son processeur ou la fidélité de sa mémoire pour afficher les résultats d'un calcul ou une donnée qu'il avait pu mémoriser. Examinons le schéma du programme :

- lire les données
- calculer le résultat (étape qui saute s'il s'agit de simplement restituer/réutiliser une donnée
- afficher des données (identiques ou nouvelles)

Ce schéma ne peut subir la moindre variation. Il serait pourtant intéressant de pouvoir introduire des variantes. Soit parce que nous voulons choisir le traitement, soit parce que le traitement prévu n'est pas adapté. Dans nos exercices, nous avons vu les deux possibilités. Nous avons été obligé d'écrire plusieurs programmes pour pouvoir calculer la somme ou la différence, ou un autre programme qui effectue tous les types d'opération. Nous devons alors changer de programme pour chaque type de calcul ou accepter que l'ordinateur nous donne des résultats qui ne nous intéressent pas en lui faisant effectuer toutes les opérations.

Il reste encore un cas plus gênant, c'est lorsqu'on décide de faire réaliser des divisions à l'ordinateur. Celui-ci n'a aucun moyen d'empêcher une division par zéro qui lui serait imposée par un utilisateur ignorant du problème ou simplement distrait.

Voyons à quoi ressemblerait un programme de division.

```
1  Variables Flottantes fDividende, fDiviseur
2  Variable Flottante fQuotient
3
4  Début du traitement
5    /* Lecture des données */
6    Afficher "Donnez le dividende de l'opération "
7    Lire fDividende
8    Afficher "Donnez le diviseur de l'opération "
9    Lire fDiviseur
10   /* Calcul */
11   fQuotient <- fDividende / fDiviseur
12   /* Affichage du résultat */
13   Afficher "Le quotient vaut : " & fQuotient
14 Fin du traitement
```

D'une manière évidente, l'instruction de la ligne 11 pose problème. Exécutée par le processeur, cette instruction provoque un plantage du programme lorsque le diviseur est nul. Cer-

tains compilateurs, dont GNU CC, produisent un résultat qui s'affiche `inf`.

```
Donnez le dividende de l'opération 12
Donnez le diviseur de l'opération 0
Le quotient vaut : inf
```

Il existe bien sûr la solution d'afficher une mise en garde pour l'utilisateur, mais rien ne dit qu'il en tiendra compte :

```
Afficher "Donnez le diviseur de l'opération (différent de 0)"
Lire fDiviseur
```

Une meilleure solution consiste à tester si le diviseur n'est pas nul et à ne poursuivre le traitement que si cette condition se réalise. Voici ce que devient le programme :

```
Variables Flottantes fDividende, fDiviseur
Variable Flottante fQuotient

Début du traitement
  Afficher "Donnez le dividende de l'opération "
  Lire fDividende
  Afficher "Donnez le diviseur de l'opération "
  Lire fDiviseur
  /* On ne calcule le quotient que si le diviseur n'est pas nul */
  Si fDiviseur Différent de 0 Alors
     fQuotient <- fDividende / fDiviseur
     Afficher "Le quotient vaut : " & fQuotient
  Fin de si
Fin du traitement
```

On peut regrettez que dans ce cas le programme se contente de ne rien faire. Il est possible de programmer d'autres instructions qui ne se réaliseront que si la condition est fausse :

```
/* On ne calcule le quotient que si le diviseur n'est pas nul */
Si fDiviseur Différent de 0 Alors
   fQuotient <- fDividende / fDiviseur
   Afficher "Le quotient vaut : " & fQuotient
/* Si le diviseur est nul, on signale que la division est impossible */
Sinon
   Afficher "Une division par 0 est impossible"
Fin de si
```

Voici un exemple d'exécution de la version finale du programme :

```
jacques@naxos:~/testpmp> ./division
Donnez le dividende de l'opération 555
Donnez le diviseur de l'opération 5
Le quotient vaut : 111.000000
jacques@naxos:~/testpmp> ./division
Donnez le dividende de l'opération 555
Donnez le diviseur de l'opération 0
Une division par 0 est impossible
```

## 2.2 Alternatives simples

Nous venons de voir un moyen efficace de produire des résultats adaptés aux circonstances.

### 2.2.1 Traitement subordonné à une condition

Lorsqu'une série d'instructions ne doivent se réaliser que sous certaines conditions, on les met entre parenthèses au moyen des commandes `Si` et `Fin de si`. Il faut bien sûr exprimer la condition de réalisation. Cette condition sera, dans la majorité des cas, le résultat d'une comparaison. Comme en mathématique, nous disposerons de six opérateurs de comparaison. Pour chacun d'eux, je propose une version en français et une version abrégée [1].

| Opérateur | | Exemple |
|---|---|---|
| **Égal à** | **?=**<br>== | fDelta **Égal à** 0 |
| **Différent de** | **!=** | fDelta **Différent de** 0 |
| **Inférieur à** | **<** | fDelta **Inférieur à** 0 |
| **Supérieur à** | **>** | fDelta **Supérieur à** 0 |
| **InfOuEgal** | **<=** | fDelta **InfOuEgal** 0 |
| **SupOuEgal** | **>=** | fDelta **SupOuEgal** 0 |

La condition s'exprime après le mot `Si`. Pour bien séparer l'expression de la condition et les instructions qu'elle contrôle, nous plaçons le mot `Alors`. Les instructions à effectuer viennent après (par clarté, nous les décalons légèrement vers la droite). Il faut évidemment indiquer la dernière instruction dépendant de la condition, puisque les instructions suivantes s'exécuteront toujours. On place ainsi la commande `Fin de Si`.

```
Lire fQuotient
Si fDiviseur Différent de 0 Alors
    fQuotient <- fDividende / fDiviseur
    Afficher "Le quotient vaut : "
    Afficher fQuotient
Fin de si
/* Suite du programme */
```

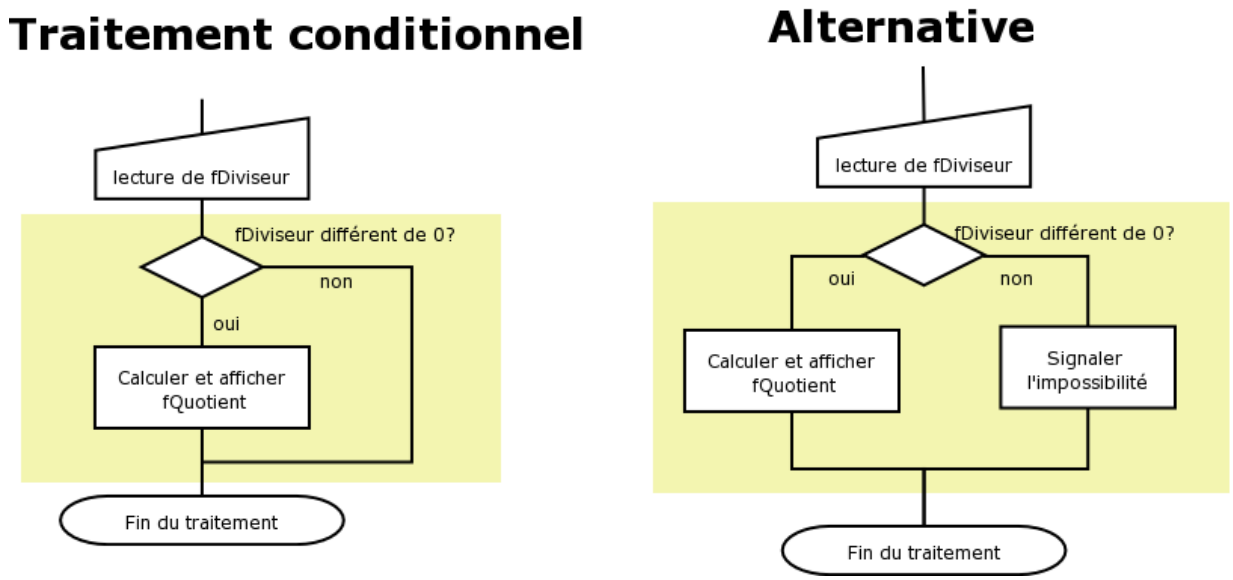
### 2.2.2 Traitements alternatifs

Lorsque la condition ne se limite pas à autoriser ou interdire l'exécution d'une série d'instructions, on se trouve devant une alternative : il est alors possible de définir deux séries d'instructions. La première série sera effectuée si la condition est vraie. La seconde ne le sera que si la condition est fausse. Dans ce cas, il faut bien prendre soin de séparer ces deux séries d'instructions. C'est le rôle du mot `sinon`. Les instructions à effectuer quand la condition est vraie se trouvent ainsi entre `Alors` et `Sinon`. Les instructions à effectuer quand la condition est fausse figurent entre `Sinon` et `Fin de Si`.

```
Si fDiviseur Différent de 0 Alors
    fQuotient <- fDividende / fDiviseur
    Afficher "Le quotient vaut : "
    Afficher fQuotient
```

1. On notera l'utilisation de ?= et != pour marquer l'égalité et l'inégalité. J'utiliserai personnellement la variante française, par souci de clarté. Pour des raisons techniques, j'ai préféré éviter l'emploi de =, qui risque de poser de gros problèmes en C. Pour rappel, les signes > ou < sont toujours orientés du côté du plus grand.

```
Sinon
    Afficher "Une division par 0 est impossible"
Fin de si
```




## 2.3 Alternatives complexes

L'ordinateur travaille avec une logique binaire, c'est pour cela que l'instruction conditionnelle comporte deux parties : les instructions à réaliser si la condition est vraie et celles à effectuer si la condition est fausse. La version de l'instruction conditionnelle sans traitement alternatif peut se réduire au cas complexe en spécifiant que le deuxième traitement se réduit à ne rien faire. On peut cependant facilement trouver des cas où il existe trois traitements alternatifs.

Si on veut examiner un nombre entier, il est possible de le caractériser au moyen de trois adjectifs :

- *positif* si le nombre est plus grand que 0
- *négatif* si le nombre est plus petit que 0
- *nul* s'il est égal à zéro

La structure alternative que nous venons d'envisager ne permet pas de considérer trois cas, puisque la condition est unique. Nous aurons donc le choix entre trois solutions, selon la condition envisagée [2] :

1. le nombre est plus grand que 0 : le nombre sera positif ou non positif (nul ou négatif)
2. le nombre est plus petit que 0 : le nombre sera négatif ou non négatif (nul ou positif)
3. le nombre est égal à 0 : le nombre sera nul ou non nul (positif ou négatif).

Arbitrairement, nous choisirons la troisième solution.

```
Variable Entière iNombre

Début du Traitement
   Afficher "Veuillez entrer un nombre "
   Lire iNombre
```

2. On pourrait encore envisager trois autres tests, puisque nous disposons de six comparateurs, mais ces trois autres ne correspondent à rien d'intuitif.

```
   Si iNombre Égale 0 Alors
      Afficher "Ce nombre est nul."
   Sinon
      Afficher "Ce nombre n'est pas nul."
   Fin de Si

Fin du Traitement
```

Pour améliorer ce programme, nous voudrions pouvoir distinguer les cas où le nombre est positif de ceux où il est négatif. Nous allons remplacer l'impression du deuxième message par un traitement plus complexe.

```
Si iNombre Égale 0 Alors
   Afficher "Ce nombre est nul."
Sinon
   Si iNombre < 0 Alors
     Afficher "Ce nombre est négatif."
   Fin de Si
   Si iNombre > 0 Alors
     Afficher "Ce nombre est positif."
   Fin de Si
Fin de Si
```

On peut simplifier ces deux instructions. En effet, quand le nombre n'est pas nul, s'il n'est pas négatif, il est forcément positif.

```
Si iNombre Égale 0 Alors
   Afficher "Ce nombre est nul."
Sinon
   Si iNombre < 0 Alors
     Afficher "Ce nombre est négatif."
   Sinon
     Afficher "Ce nombre est positif."
   Fin de Si
Fin de Si
```

En théorie, nous pouvons imbriquer nos instructions conditionnelles autant que nous voulons. Dans la pratique, nous devons tenir compte des limitations de l'esprit humain et considérer qu'au-delà de quatre imbrications, nous ne sommes plus vraiment capables de suivre.

## Commentaires et notions de logique

Lors de l'utilisation de plusieurs alternatives, il peut être utile de placer des commentaires pour bien marquer les cas prévus. Ainsi, le programme ci-dessous pourrait s'écrire d'une manière plus lisible de la façon suivante :

```
1 Si iNombre == 0 Alors
2    Afficher "Ce nombre est nul."
3 Sinon // Nombre différent de 0
4    Si iNombre < 0 Alors
```

```
5       Afficher "Ce nombre est négatif."
6     Sinon // Nombre supérieur à 0
7       Afficher "Ce nombre est positif."
8     Fin de Si
9   Fin de Si
```

Notons bien que les commentaires de la ligne 3 et de la ligne 6 ne sont pas des conditions. La condition liée à l'exécution de la deuxième branche d'une alternative est **toujours implicite** et correspond à la négation de la condition originale.

Depuis des années, quand je demande « quel est la négation de la condition A > B ? », il s'élève toujours une voix pour dire « A < B ». Si Mohamed n'est pas plus grand que Rémy, cela ne signfie pas nécessairement qu'il est plus petit. Ils peuvent avoir la même taille. Rappelons donc des évidences :

| Condition | Négation |
|---|---|
| A == B | A != B |
| A < B | A >= B |
| A > B | A <= A |

## Vocabulaire

ALTERNATIVE, CONDITION, EXPLICITE, FAUX, IMBRICATION, IMPLICITE, OPÉRATEURS DE COMPARAISON, VRAI

## Exercices

**II-1.** Écrire un programme qui affiche le plus grand de deux nombres entrés au clavier.

**II-2.** Écrire un programme qui affiche dans l'ordre croissant deux nombres entiers entrés au clavier.

**II-3.** Les factures concernant les réparations dans des habitations se voient appliquer un taux de TVA variable :

– dans les maisons de plus de 15 ans, on applique un taux de TVA de 6%,
– dans les maisons récentes, un taux de TVA de 21%.

Écrire un programme qui calcule le prix total d'une facture dont on connaît le montant hors TVA et l'âge du bâtiment.

**II-4.** Modifier le programme précédent en simplifiant la procédure d'encodage. Il n'est en effet pas nécessaire d'encoder l'âge exact (parfois inconnu dans le cas d'un vieux bâtiment).

**II-5.** Écrire un programme qui calcule les racines d'une équation données, si c'est possible. Si vous êtes vraiment allergique à l'algèbre, vous pouvez vous limiter à écrire les expressions correspondant aux différentes formules.

Une équation du second degré possède la forme canonique suivante :
$ax^2 - bx + c = 0$
En calculer les racines revient à chercher la ou les valeurs de x qui rendent cette égalité vraie. Par exemple, l'équation $x^2-5x+6 = 0$ a pour racines les valeurs 3 et 2 parce que $3^2-5.3+6 = 0$ et $2^2 - 5.2 + 6 = 0$.
Rappelons qu'une telle équation se résout en calculant préalablement la valeur de $\delta$ qui vaut $b^2 - 4ac$. Trois cas peuvent se présenter :
– $\delta$ est positif et dans ce cas, l'équation admet deux solutions : $\frac{-b+\sqrt{\delta}}{2a}$ et $\frac{-b-\sqrt{\delta}}{2a}$
– $\delta$ est nul et l'équation possède une racine double $\frac{-b}{2a}$
– $\delta$ est négatif et l'équation n'admet pas de racines réelles
Les expressions ci-dessus peuvent être encodées sans trop de problèmes, à condition de se rappeler que les produits et les divisions nécessitent parfois l'usage de parenthèses. Ne pas oublier d'écrire tous les opérateurs.
A titre d'exemple, les équations suivantes ont des solutions réelles :

| | | | |
|---|---|---|---|
| $x^2 + x - 2 = 0$ | $x^2 - 8x + 7 = 0$ | $x^2 - 5x + 6 = 0$ | $x^2 - 8x + +16 = 0$ |