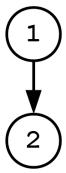
Chapitre 1

Séquentialité et variables

Un ordinateur est une machine et en tant que telle, elle ne fait que ce qu'on lui demande. Les programmeurs ont pour tâche de faire faire à ces machines exactement ce qu'ils veulent qu'elles fassent et dans le bon ordre ni plus, ni moins, ni autre chose. Pour cela, ils écrivent des programmes qui seront exécutés par l'ordinateur.

Un programme est une suite d'instructions effectuées l'une à la suite de l'autre, de la première à la dernière, de manière séquentielle. Chaque instruction est un ordre simple donné à l'ordinateur. Lors de l'exécution d'un programme, l'ordinateur commence par exécuter la première instruction de ce programme. Ce n'est que lorsqu'elle est terminée qu'il passe à la seconde. S'il y a une troisième instruction, il attend d'avoir terminé la deuxième pour l'exécuter.

Par exemple, l'instruction `affichage(1);` correspond à l'ordre donné à l'ordinateur d'afficher 1 à l'écran. Il s'agit d'un ordre simple, mais qui peut être combiné à d'autres pour produire des résultats (un peu) plus complexes. Par exemple, cette instruction suivie de `affichage(2);` forme un programme avec deux instructions (les numéros dans les cercles correspondent aux numéros de ligne du programme) :




```
1 affichage(1);
2 affichage(2);
```

qui affichera 1 puis affichera 2 **à la suite** de 1. Voici l'affichage obtenu :

```
1 12
```

Cet exemple est évidemment simpliste et peut paraître très éloigné des programmes utilisés habituellement. Mais même pour réaliser des applications complexes (animation et rendu 3D, par exemple), ce sont toujours les mêmes principes simples qui sont à l'œuvre. Au lieu d'instructions qui se contentent d'afficher un nombre à l'écran, on utilisera des instructions toujours aussi simples à écrire, mais dont les effets seront plus spectaculaires. Le travail du programmeur n'est pas plus complexe pour autant. Ainsi, au lieu du programme précédent, on aura des instructions comme

```
1 pivoter(0.2); // pour faire tourner une scène 3D de 0.2 degrés
2 afficher(0.1); // pour demander un rendu 3D de la scène en moins de 0.1 seconde
```

dont l'impact visuel sera bien plus important, mais dont la complexité pour le programmeur reste comparable.

À retenir : Construire des programmes informatiques, c'est toujours se demander quelles instructions exécuter, et dans quel ordre.

▶ Sur ce thème : **Exercice 1, Question 1.1, TD 1**

1.1 Structure d'un programme

L'écriture d'un programme doit suivre des **règles très précises** [1]. A la moindre erreur d'écriture, l'ordinateur ne comprend plus les instructions à effectuer.

Chaque instruction doit se terminer par le caractère `;`. De plus, le programme possède un début défini par

```
int main()
{
```

et une fin définie par

```
  return 0;
}
```

Toutes les instructions que l'ordinateur doit effectuer doivent se trouver entre le début et la fin. Ainsi, l'exemple précédent complet est

```
int main()
{
  affichage(1);
  affichage(2);

  return 0;
}
```

Les espaces (sauf pour séparer les mots) et les sauts de lignes entre les instructions n'ont pas d'importance. Le même programme pourrait être écrit comme

```
int main(){affichage(1);affichage(2);return 0;}
```

Cependant, pour une meilleure lisibilité du code, il est recommandé d'écrire une seule instruction par ligne et d'*indenter* les instructions par rapport au début et à la fin du programme. Indenter signifie insérer des espaces pour décaler légèrement vers la droite les instructions.

Il est possible d'insérer des commentaires pour expliquer les instructions du programme. Ces commentaires ne sont pas destinés à l'ordinateur (celui-ci ne les lit pas) mais aux personnes lisant le code. Il existe deux manières de commenter du code. En utilisant les caractères `//`, on indique que tout ce qui suit sur la ligne (avant le retour chariot) est un commentaire. Pour écrire un commentaire sur plusieurs lignes, il suffit de l'insérer entre `/*` et `*/`. À titre d'exemple, voici le code précédent commenté

```
//Début du programme
int main()
{
  affichage(1); // On affiche un à l'écran
  /* On affiche 2 grâce à
     l'instruction suivante */
  affichage(2);

  return 0; //Fin du programme
}
```

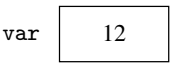
1. Les règles dépendent du langage dans lequel est écrit le programme. Le langage choisi pour ce cours est une version du langage C++ simplifiée. Le but de ce cours n'est pas d'apprendre le C++ mais l'algorithmique.

1.2 Variables

Les instructions d'un programme permettent de traiter des données (notes d'un étudiant, noms de famille, adresses électroniques *etc.*) qui doivent pouvoir être modifiées au fur et à mesure que le programme se déroule. L'ordinateur stocke donc chacune de ces informations dans sa mémoire à un endroit bien identifié, grâce à des variables.

Les *variables* associent un nom à un espace dans la mémoire. Cet espace mémoire contient une valeur qui peut changer au cours de l'exécution du programme. Dans le code, on accède à cet espace mémoire grâce au nom de la variable.

Voici un exemple de variable. L'espace mémoire est représenté par une boîte (rectangle). Le nom de la variable associé à cet espace mémoire est `var`. La valeur contenue (pour l'instant) dans cette variable est 12.



Remarque : On utilise le terme variables par opposition aux constantes.

1.2.1 Noms de variables

Selon la syntaxe utilisée dans ce cours, le nom d'une variable commence par une lettre minuscule (a à z) ou majuscule (A à Z), ou bien par le caractère souligné (_). Pour la suite de son nom, on peut utiliser des lettres minuscules ou majuscules, des soulignés et des chiffres (0 à 9). Voici quelques exemples de noms de variables valides :

```
var_1
_var1
prix_article
tempCapteur
```

Attention : Le nom d'une variable ne contient ni espace ni d'accent.

▶ Sur ce thème : **Exercice 1, Question 1.2, TD 1**

1.2.2 Types de données

Dans chaque variable, on ne peut stocker qu'un certain type de valeurs. On présente dans ce qui suit des types de base qui pourront être utilisés ultérieurement.

Entiers

Le type **int** (*integer*) permet de représenter un entier relatif. Voici quelques exemples de valeurs (constantes) de type `int` :

```
-35
2048
```

Remarque : Le type `int` ne permet pas de représenter les grands entiers (en valeur absolue). La valeur absolue maximale qui peut être représentée dépend du nombre de bits utilisés pour coder le type `int`. Dans tous les cas, il peut y avoir dans certains cas un *dépassement de capacité*, c'est-à-dire que l'on essaie de représenter un entier trop grand pour le type `int`. Ces dépassements de capacité sont parfois à l'origine d'erreurs dans le programme.

Réels

Le type **double**, correspondant aux nombres à virgule flottante (aussi appelés *nombres flottants*), représente les réels. Voici quelques valeurs de type `double` :

```
-12.56
3.141592653589793
```

Remarque : Les nombres flottants peuvent aussi être représentés avec le type `float`. Cependant, pour des raisons de compatibilité, il est préférable d'utiliser le type `double`.

Remarque : Le type `double` ne permet pas de coder tous les réels. En effet, ces derniers sont en nombre infini et le type `double` ne peut stocker qu'un nombre fini (mais très grand) de valeurs. Par conséquent, un arrondi doit parfois être fait. Ceci peut parfois engendrer des erreurs dans le programme.

Caractères

Le type **char** (*charactere*) permet de représenter un caractère. Une valeur de type `char` est indiquée entre apostrophes. Voici quelques exemples de valeurs de type `char` :

```
'a'
'1'
' '
```

Remarque : En réalité, chaque caractère est associé à un numéro entier, bien déterminé, qui est consigné dans une table. Cette table est nommée table *ASCII*. Par exemple, l'ordinateur représente le caractère `'a'` par le code ASCII 97. Il existe certains caractères spéciaux qui sont précédés de l'antislash. Ainsi `'\n'` est le caractère correspondant au saut de ligne. Par ailleurs, la table ASCII ne contient pas de caractère accentué.

Attention : Il ne faut pas confondre `'1'` et `1`. Le premier est un caractère associé au code ASCII 49 alors que le deuxième est l'entier 1.

Chaînes de caractères

Une *chaîne de caractères* (`string`) est une suite de caractères de longueur arbitraire délimitée par des guillemets `"`.
Attention : Chaque caractère compte, ainsi que la *casse* (majuscules/minuscules). Les chaînes de caractères suivantes sont donc toutes différentes :

```
"HelloWorld"
"Hello World"
"hello world"
"12.56"
```

Remarque : S'il n'y a pas de caractère, on parle de *chaîne vide*, notée `""`. Cette chaîne est différente de la chaîne `" "` qui est une chaîne contenant un caractère (espace).

Attention : `"a"` et `'a'` ne représentent pas la même chose. La première valeur est une chaîne de caractères contenant un seul caractère alors que la seconde est un caractère. De même, `"1"` et `1` sont de type différents, le premier est une chaîne de caractère alors que le second est un entier.

1.2.3 Déclaration de variables

Les variables doivent être *déclarées* avant de pouvoir être utilisées. Pour cela, on doit indiquer le nom de chaque variable précédé de son type. L'exemple

```
1 int main()
2 {
3   int compteur;
4   char lettre;
5   string s;
6   return 0;
7 }
```

montre les déclarations d'une variable entière `compteur`, d'une variable caractère appelée `lettre` et d'une variable `s` de type chaîne de caractères.

Attention : Lorsque l'on déclare une variable, elle prend par défaut en général la valeur correspondant aux bits stockés à l'emplacement mémoire. **On ne donc peut absolument pas présager de la valeur d'une variable qui n'a pas été initialisée**.

1.2.4 Affectation

L'opération principale mettant en œuvre les variables est l'affectation, qui permet de changer la valeur d'une variable. Une affectation est notée avec le symbole `=`. À gauche de ce symbole, on place le nom de la variable dont on veut changer la valeur, et à droite on définit la valeur que doit prendre la variable. **Ce symbole n'est pas l'égalité mathématique.**

Par exemple, l'instruction ligne 4

```
1 int main()
2 {
3   int x;               // Déclaration de la variable x (de type entier)
4   x=2;                 // Affectation de la valeur entière 2 à la variable x
5   return 0;
6 }
```

change la valeur de `x` en 2.

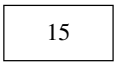
Notez bien que `2=x` est incorrect. 2 est un nombre constant et ne change pas de valeur. Ce n'est pas une variable et **on ne peut donc pas le mettre à gauche de l'opérateur d'affectation**.

Attention : Lors d'une affectation, la valeur qui est contenue dans la variable avant affectation est écrasée. Il n'est donc plus possible d'y accéder.

Dans l'exemple précédent, la nouvelle valeur est directement spécifiée. On aurait également pu placer à droite du symbole `=` une expression qui représente une valeur. Dans ce cas, l'opérateur d'affectation commence alors par évaluer l'expression à droite du signe `=`, (quelle que soit la complexité du calcul) avant de placer le résultat dans l'espace mémoire réservé à la variable. Ainsi, dans l'exemple suivant :

```
int main()
{
  int x;
  x=(1+2)*5;
  return 0;
}
```

l'instruction `x=(1+2)*5;` calcule d'abord la valeur de l'expression `(1+2)*5` avant d'affecter le résultat du calcul à la variable nommée `x`, qui prend par conséquent la valeur 15.



Dans l'évaluation d'une expression, chaque nom de variable est remplacé par sa valeur. Si `x` vaut 2, par exemple, alors `(x*3)` vaut 6. Ainsi, à l'issue de l'algorithme suivant :

```
1 int main()
2 {
3   int x;
4   x=2;
5   int y;
6   y=x*3;
7   return 0;
8 }
```

x vaut 2 et y vaut 6.

Remarque : On peut, dans une même instruction, déclarer une variable et l'*initialiser*, c'est-à-dire lui affecter une première valeur. Ainsi, les lignes 3 et 4 du code précédent peuvent être remplacées par `int x = 2;`. Ceci est recommandé afin d'éviter d'utiliser par la suite des variables non initialisées.

Il est important de noter qu'une variable ne peut être utilisée que si elle a été définie précédemment. Ainsi, le programme suivant est correct :

```
1 int main()
2 {
3   int x = 0;
4   int y = 2;
5   int z = (x+y)+2;
6   return 0;
7 }
```

alors que le programme suivant ne l'est pas

```
1 int main()
2 {
3   int x = 1;
4   int z = x+y;
5   return 0;
6 }
```

car la variable y utilisée lors de l' instruction de la ligne 4 n'a pas été définie préalablement. Selon le principe de séquentialité, ajouter la ligne 5 comme dans le programme

```
1 int main()
2 {
3   int x = 1;
4   int z = x+y;
5   int y = 3;
6
7   return 0;
8 }
```

ne resoudrait pas le problème.

À retenir : Pour définir une variable, il faut impérativement la déclarer et lui affecter une valeur.

Une source importante d'erreurs en programmation est la non-maîtrise de la séquentialité des instructions. Les instructions s'exécutent dans l'ordre où elles sont écrites, les éventuelles modifications de variables se font dans le même ordre.

▶ Sur ce thème : **Exercice 1, Question 1.3, TD1**

1.3 Entrées/sorties

1.3.1 Affichage à l'écran

L'instruction `affichage` permet d'afficher une valeur à l'écran. Elle permet d'afficher indifféremment des nombres ou des chaînes de caractères. C'est un opérateur qui réalise une sortie car elle fait « sortir » des valeurs de l'ordinateur vers l'utilisateur.

Pour afficher une valeur, il faut utiliser l'instruction `affichage(valeur);`. La valeur à afficher peut être une constante, la valeur d'une variable ou un calcul. Ainsi

```
1  int main()
2  {
3    int nombre = 23;
4    affichage("Le nombre est : ");
5    affichage(nombre);
6    affichage("\nLe suivant est : ");
7    affichage(nombre + 1 );
8    affichage('\n');
9    return 0;
10 }
```

affichera

```
1 Le nombre est : 23
2 Le suivant est : 24
```

Pour accélérer l'écriture du code, il est possible d'enchaîner les affichages dans une même instruction `affichage` en séparant les valeurs à afficher par des virgules. Le même affichage que l'exemple précédent est alors donné par

```
1 affichage("Le nombre est : ", nombre, "\nLe suivant est : ", nombre + 1, '\n');
```

► Sur ce thème : **Exercices 1, Questions 1.4 à 1.8 et 2, TD1**

1.3.2 Saisies au clavier

Il est possible, durant l'exécution du programme, de demander à l'utilisateur de saisir des valeurs pour des variables. Ceci se fait grâce à l'instruction `saisie(variable);`, où `variable` est la variable que l'on souhaite affecter avec la valeur saisie au clavier. Voici un exemple de programme demandant à l'utilisateur son prénom et son âge.

```
1  int main()
2  {
3    string prenom;
4    int age;
5    affichage("Veuillez saisir votre prenom : ");
6    saisie(prenom);
7    affichage("Veuillez saisir votre age : ");
8    saisie(age);
9    affichage("Vous vous appelez ", prenom, " et vous avez ", age, " ans.\n");
10   return 0;
11 }
```

La lecture des caractères se fait jusqu'à l'appui de la touche "Entrée". La valeur correspondant aux caractères saisis par l'utilisateur est stockée dans la variable. Ainsi, pour l'instruction `saisie(age);`, si l'utilisateur appuie sur les touches 2, 0 puis "Entrée", la variable `age` contiendra après cette instruction l'entier 20.

Remarque : Lors de la conversion des caractères saisis par l'utilisateur en un nombre, l'ordinateur s'arrête à la première erreur. Si l'utilisateur a saisi pour son âge la valeur `12abc345`, la valeur affectée à la variable `age` est 12. Si l'utilisateur a saisi `abc6`, la valeur de `age` est alors 0.

▶ Sur ce thème : **Exercice 3, TD1**

1.4 Opérations et conversions sur les nombres

Il est possible d'appliquer plusieurs opérations sur les nombres (`int, double`) et de convertir une valeur d'un type en un autre type.

1.4.1 Conversion de type de nombres

La *conversion de type* (encore appelée *cast*) permet de convertir une **valeur** d'un type en sa représentation dans un autre type (avec perte éventuelle d'informations). La conversion explicite de type s'effectue en faisant précéder la valeur (ou variable) à convertir par le type cible entre parenthèses. Par exemple,

```
1  int main()
2  {
3    affichage( 3.5 );
4    affichage('\n');
5    affichage( (int) 3.5 );
6    affichage('\n');
7    double var = -3.8;
8    affichage( (int) var );
9    return 0;
10 }
```

affiche

```
1 3.5
2 3
3 -3
```

Remarque : Appliquer une conversion de type sur une variable **ne modifie pas** sa valeur. Ainsi, à la fin du programme, `var` vaut toujours `-3.8`.
Attention : Transformer un `double` en un `int` engendre une perte de données puisque les décimales sont tronquées.

Lorsque l'on utilise l'opérateur de conversion, on parle d'une *conversion explicite*. Cependant, en cas de besoin, l'ordinateur effectue des *conversions implicites*, c'est-à-dire des conversions qui ne sont pas écrites dans le programme. Cependant, ces conversions implicites ne sont faites que s'il n'y a pas de perte d'information, c'est-à-dire si l'on convertit une valeur entière en un réel. Dans le cas contraire, le programme affiche le message `conversion to 'int' from 'double' may alter its value` et le programme s'arrête. Ainsi, le code

```
1  int main()
2  {
3    double a = 1.2;
4    int b = a;
5    affichage(b);
6    return 0;
7  }
```

ne fonctionne pas puisque l'on initialise la variable `b` de type `int` avec une valeur de type `double`.

Attention : La conversion (implicite ou explicite) ne se fait pas avec les chaînes de caractères. Il n'est donc pas possible de transformer un nombre en `string` ou inversement.

1.4.2 Opérations

Le tableau suivant récapitule les principales opérations possibles sur des nombres :

Expression	Résultat, `x` et `y` étant des nombres (`int` ou `double`)
`x + y`	somme de `x` et de `y`
`x - y`	différence de `x` et de `y`
`x * y`	produit de `x` et de `y`
`x / y`	division de `x` par `y`
`x % y`	reste de la division entière de `x` par `y`
`-x`	opposé de `x`
`abs(x)`	valeur absolue de `x`
`round(x)`	Arrondi de `x` à la valeur entière la plus proche
`floor(x)`	Partie entière inférieure de `x`
`ceil(x)`	Partie entière supérieure de `x`

Une opération s'applique uniquement sur des *nombres de même type*. Si les types sont différents (`int` et `double`), l'ordinateur effectue une conversion implicite de la valeur `int` en `double`. Ainsi, lors de l'opération `3 + 2.1`, la valeur 3 est transformée en `3.0`. Par ailleurs, le résultat d'une opération est **de même type** que les nombres sur lesquels s'applique cette opération. Au besoin, une conversion implicite est effectuée. Cela peut notamment poser des problèmes lors de la **division de deux valeurs (ou variables) entières** car le résultat est alors **un entier**. Typiquement, `3/2` vaut `1` et pas `1.5`. Pour obtenir la valeur `1.5`, il faut alors utiliser une conversion explicite de `int` en `double`.

Remarque : L'opérateur de conversion explicite est *prioritaire* par rapport aux opérations ci-dessus. Le résultat de `(double) 1/2` vaut donc `0.5` puisque la conversion de `1` en `double` se fait avant la division. Par conséquent, la division s'applique avec un `double` et un `int`, donc ce dernier est transformé en `double`. Le résultat est donc un `double`, soit `0.5`.

▶ Sur ce thème : **Exercices 1, Questions 1.9 et 1.10 et 4 à 8, TD1**

TD1 : Séquentialité et variables

Les étoiles associées à chaque exercice indiquent sa difficulté : facile (une étoile), moyen (deux étoiles) ou difficile (trois étoiles). De plus, les exercices marqués du symbole ✓ abordent les notions du chapitre et seront faits en priorité en TD. Le symbole ® signifie que cet exercice est donné en révision. Les exercices des contrôles continus seront similaires à ceux marqués de ce symbole.

✓ Exercice 1 : Test de compréhension*

Question 1.1 : [Séquences d'instructions] Qu'affichent les programmes suivants ?

```
int main()
{
  affichage(1);
  affichage(2);
  affichage(3);

  return 0;
}
```

```
int main()
{
  affichage(2);
  affichage(1);
  affichage(3);

  return 0;
}
```

Question 1.2 : [Noms de variables] Parmi ces exemples, seuls certains sont des noms de variable valides. Lesquels ?

- `x`
- `X1`
- `éric`
- `_eric`
- `t_42`
- `24_t`

Question 1.3 : [Séquentialité] Quelles sont les valeurs des variables à la fin du programme ?

```
int main()
{
  int a = 4;
  int b = 6;
  b = a + 12;
  int c = a + b;
  a = a + 1;
  b = b +c ;
  return 0;
}
```

Pour répondre à la question, dessiner les cases mémoires à chaque étape de l'exécution du programme.

Question 1.4 : [Affichage de `string` constantes] L'instruction suivante est-elle correcte ? Efficace ?
`affichage("bon","jour");`

Question 1.5 : [Affichage d'un programme] Qu'affiche le programme suivant ?

```
int main()
{
  string hello = "bonjour";
  affichage("hello \n");
  affichage(hello, "\n");

  return 0;
}
```

Question 1.6 : [Séquentialité et valeur des variables] Qu'affiche le programme suivant ?

```
int main()
{
  int a=1;
  int b=2;
  a=b;
  b=a;
  affichage(a, '\n', b, '\n');

  return 0;
}
```

Question 1.7 : [Séquentialité et valeur des variables] Qu'affiche le programme suivant ?

```
int main()
{
  int a=1;
  int b=2;
  b=a;
  a=b;
  affichage(a, '\n', b, '\n');

  return 0;
}
```

Question 1.8 : [Échange des valeurs de deux variables] Que manque-t-il aux programmes précédents pour réaliser un échange entre les valeurs des variables a et b ? Écrire un programme qui réalise un tel échange.

Question 1.9 : [Types numériques] Complétez les pointillés avec le type de variable le plus adapté, c'est-à-dire permettant qu'il n'y ait aucune conversion implicite lors de l'affectation des variables.

```
int main()
{
       ..... a = 1.5;
       ..... b = 2;
       ..... c = a + 1;
       ..... d = b + 3;
       ..... e = (double) d;
       ..... f = c / b;
       ..... g = (int) c / b;

       return 0;
}
```

Question 1.10 : [Conversions] Indiquez l'affichage obtenu par l'exécution du programme suivant.

```
int main()
{
  affichage( 1/2, '\n');
  affichage( 1/2.0, '\n');
  affichage( (double) 1/2, '\n');
  affichage( 1/(double) 2, '\n');
  affichage( (double) (1/2), '\n');
  affichage( (double) (1.0/2), '\n');
  double a = 1/2;
  affichage( a , '\n');
  return 0;
}
```

✓ Exercice 2 : Code postal*

Considérons le programme suivant :

```
int main()
{
  int cp = 93;
  string nom = "Seine Saint-Denis";
  affichage( ??? );

  return 0;
}
```

Que faut-il mettre à la place des ? ? ? ? ? pour que le programme affiche « Le code postal de Seine Saint-Denis est 93. » ?

Exercice 3 : Somme de deux flottants*

Écrire un programme demandant à l'utilisateur de saisir deux nombres flottants (`double`) dans deux variables `nb1` et `nb2` et afficher la somme des deux nombres.

✓ Exercice 4 : Calcul de l'aire et du périmètre d'un rectangle*

Écrire un programme permettant de calculer l'aire et le périmètre d'un rectangle, les dimensions des largeur et longueur étant saisies par l'utilisateur.

Ⓡ Exercice 5 : Convertisseur euros/dollars*

Écrire un programme permettant d'afficher en dollars un montant saisi en euros par l'utilisateur, sachant qu'un dollar équivaut à 0.906355364 euros.

Ⓡ Exercice 6 : Soldes*

Écrire un programme permettant de calculer les prix durant les soldes. L'utilisateur doit saisir le prix initial et le pourcentage de réduction. Le programme affichera le prix après réduction.

Ⓡ Exercice 7 : Calcul de l'IMC*

L'indice de masse corpulaire (IMC) est une grandeur qui permet d'estimer la corpulence d'une personne[2]. Il se calcule à l'aide de la formule : poids (kg) / taille (m)^2. Écrire un programme permettant de calculer l'imc de l'utilisateur.

2. Définition donnée par Wikipedia

Exercice 8 : Échange de nombres sans variable intermédiaire**

Écrire un programme qui demande à l'utilisateur deux nombres, les affiche, les échange, et les ré-affiche après échange, mais sans utiliser de variable intermédiaire. On pourra utiliser une addition et une soustraction pour ne rien perdre.

Remarque : la méthode d'échange sans variable temporaire peut paraître plus intéressante que celle utilisant une variable intermédiaire. Cependant, cette méthode nécessite plus d'opérations (l'ordinateur doit effectuer des additions/soustractions). De plus, elle est moins générale car elle suppose que les variables à échanger sont de type numérique : elle ne fonctionnerait pas sur des chaînes de caractères par exemple. D'une manière générale, il y a souvent plusieurs manières de résoudre un problème. Le rôle du programmeur consiste à identifier le meilleur algorithme (le meilleur enchaînement d'instructions), compte tenu des hypothèses qu'il lui semble raisonnable de faire.