

MATHIEU NEBRA
MATTHIEU SCHALLER

PROGRAMMEZ AVEC LE LANGAGE C++

TOUTE LA PUISSANCE DU LANGAGE C++
EXPLIQUÉE AUX DÉBUTANTS



Issu du célèbre
Site du Zéro
www.siteduzero.com



www.siteduzero.com



Sauf mention contraire, le contenu de cet ouvrage est publié sous la licence :
Creative Commons BY-NC-SA 2.0

La copie de cet ouvrage est autorisée sous réserve du respect des conditions de la licence
Texte complet de la licence disponible sur : <http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>

Simple IT 2011 - ISBN : 978-2-9535278-5-8

Chapitre 4

Utiliser la mémoire

Difficulté : 

Jusqu'à présent, vous avez découvert comment créer et compiler vos premiers programmes en mode console. Pour l'instant ces programmes sont très simples. Ils affichent des messages à l'écran... et c'est à peu près tout. Cela est principalement dû au fait que vos programmes ne savent pas interagir avec leurs utilisateurs. C'est ce que nous allons apprendre à faire dans le chapitre suivant.

Mais avant cela, il va nous falloir travailler dur puisque je vais vous présenter une notion fondamentale en informatique. Nous allons parler des **variables**.

Les variables permettent d'utiliser la mémoire de l'ordinateur afin de stocker une information pour pouvoir la réutiliser plus tard. J'imagine que vous avez tous déjà eu une calculatrice entre les mains. Sur ces outils, il y a généralement des touches **M+**, **M-**, **MC**, etc. qui permettent de stocker dans la mémoire de la calculatrice le résultat intermédiaire d'un calcul et de reprendre ce nombre plus tard. Nous allons apprendre à faire la même chose avec votre ordinateur qui n'est, après tout, qu'une grosse machine à calculer.



Qu'est-ce qu'une variable ?

Je vous ai donné l'exemple de la mémoire de la calculatrice parce, que dans le monde de l'informatique, le principe de base est le même. Il y a quelque part dans votre ordinateur des composants électroniques qui sont capables de contenir une valeur et de la conserver pendant un certain temps. La manière dont tout cela fonctionne exactement est très complexe.

Je vous rassure tout de suite, nous n'avons absolument pas besoin de comprendre comment cela marche pour pouvoir, nous aussi, mettre des valeurs dans la mémoire de l'ordinateur. Toute la partie compliquée sera gérée par le compilateur et le système d'exploitation. Elle n'est pas belle la vie ?

La seule et unique chose que vous ayez besoin de savoir, c'est qu'une **variable** est une partie de la mémoire que l'ordinateur nous prête pour y mettre des valeurs. Imaginez que l'ordinateur possède dans ses entrailles une grande armoire (figure 4.1). Cette dernière possède des milliers (des milliards !) de petits tiroirs ; ce sont des endroits que nous allons pouvoir utiliser pour mettre nos variables.



FIGURE 4.1 – La mémoire d'un ordinateur fonctionne comme une grosse armoire avec beaucoup de tiroirs

Dans le cas d'une calculatrice toute simple, on ne peut généralement stocker qu'un seul nombre à la fois. Vous vous doutez bien que, dans le cas d'un programme, il va falloir conserver plus d'une chose simultanément. Il faut donc un moyen de différencier les variables pour pouvoir y accéder par la suite. Chaque variable possède donc un **nom**. C'est en quelque sorte l'étiquette qui est collée sur le tiroir.

L'autre chose qui distingue la calculatrice de l'ordinateur, c'est que nous aimerions pouvoir stocker des tas de choses différentes, des nombres, des lettres, des phrases, des images, etc. C'est ce qu'on appelle le **type** d'une variable. Vous pouvez vous imaginer cela comme étant la forme du tiroir. En effet, on n'utilise pas les mêmes tiroirs pour stocker des bouteilles ou des livres.

Les noms de variables

Commençons par la question du nom des variables. En C++, il y a quelques règles qui régissent les différents noms autorisés ou interdits.

- les noms de variables sont constitués de lettres, de chiffres et du tiret-bas `_` uniquement ;
- le premier caractère doit être une lettre (majuscule ou minuscule) ;
- on ne peut pas utiliser d'accents ;
- on ne peut pas utiliser d'espaces dans le nom.

Le mieux est encore de vous donner quelques exemples. Les noms `ageZero`, `nom_du_zero` ou encore `NOMBRE_ZEROS` sont tous des noms valides. `AgeZéro` et `_nomzero`, en revanche, ne le sont pas.

À cela s'ajoute une règle supplémentaire, valable pour tout ce que l'on écrit en C++ et pas seulement pour les variables. Le langage fait la différence entre les majuscules et les minuscules. En termes techniques, on dit que C++ est *sensible à la casse*. Donc, `nomZero`, `nomzero`, `NOMZERO` et `NomZeRo` sont tous des noms de variables différents.



Pour des questions de lisibilité, il est important d'utiliser des noms de variables qui décrivent bien ce qu'elles contiennent. On préférera donc choisir comme nom de variable `ageUtilisateur` plutôt que `maVar` ou `variable1`. Pour le compilateur, cela ne fait aucune différence. Mais, pour vous et pour les gens qui travailleront avec vous sur le même programme, c'est très important.

Personnellement, j'utilise une « convention » partagée par beaucoup de programmeurs. Dans tous les gros projets regroupant des milliers de programmeurs, on trouve des règles très strictes et parfois difficiles à suivre. Celles que je vous propose ici permettent de garder une bonne lisibilité et surtout, elles vous permettront de bien comprendre tous les exemples dans la suite de ce cours.

- les noms de variables commencent par une minuscule ;
- si le nom se décompose en plusieurs mots, ceux-ci sont collés les uns aux autres ;
- chaque nouveau mot (excepté le premier) commence par une majuscule.

Voyons cela avec des exemples. Prenons le cas d'une variable censée contenir l'âge de l'utilisateur du programme.

- `AgeUtilisateur` : non, car la première lettre est une majuscule ;
- `age_utilisateur` : non, car les mots ne sont pas collés ;
- `ageutilisateur` : non, car le deuxième mot ne commence pas par une majuscule ;
- `maVar` : non, car le nom ne décrit pas ce que contient la variable ;
- `ageUtilisateur` : ok.

Je vous conseille fortement d'adopter la même convention. Rendre son code lisible et facilement compréhensible par d'autres programmeurs est très important.

Les types de variables

Reprenons. Nous avons appris qu'une variable a un nom et un type. Nous savons comment nommer nos variables, voyons maintenant leurs différents types. L'ordinateur aime savoir ce qu'il a dans sa mémoire, il faut donc indiquer quel type d'élément va contenir la variable que nous aimerions utiliser. Est-ce un nombre, un mot, une lettre? Il faut le spécifier.

Voici donc la liste des types de variables que l'on peut utiliser en C++.

Nom du type	Ce qu'il peut contenir
<code>bool</code>	Une valeur parmi deux possibles, vrai (<code>true</code>) ou faux (<code>false</code>).
<code>char</code>	Un caractère.
<code>int</code>	Un nombre entier.
<code>unsigned int</code>	Un nombre entier positif ou nul.
<code>double</code>	Un nombre à virgule.
<code>string</code>	Une chaîne de caractères, c'est-à-dire un mot ou une phrase.

Si vous tapez un de ces noms de types dans votre IDE, vous devriez voir le mot se colorer. L'IDE l'a reconnu, c'est bien la preuve que je ne vous raconte pas des salades. Le cas de `string` est différent, nous verrons plus loin pourquoi. Je peux vous assurer qu'on va beaucoup en reparler.



Ces types ont des limites de validité, des bornes, c'est-à-dire qu'il y a des nombres qui sont trop grands pour un `int` par exemple. Ces bornes dépendent de votre ordinateur, de votre système d'exploitation et de votre compilateur. Sachez simplement que ces limites sont bien assez grandes pour la plupart des utilisations courantes. Cela ne devrait donc pas vous poser de problème, à moins que vous ne vouliez créer des programmes pour téléphones portables ou pour des micro-contrôleurs, qui ont parfois des bornes plus basses que les ordinateurs. Il existe également d'autres types avec d'autres limites mais ils sont utilisés plus rarement.

Quand on a besoin d'une variable, il faut donc se poser la question du genre de choses qu'elle va contenir. Si vous avez besoin d'une variable pour stocker le nombre de personnes qui utilisent votre programme, alors utilisez un `int` ou `unsigned int`, ; pour stocker le poids d'un gigot, on utilisera un `double` et pour conserver en mémoire le nom de votre meilleur ami, on choisira une chaîne de caractères `string`.



Mais à quoi sert le type `bool`? Je n'en ai jamais entendu parler.

C'est ce qu'on appelle un **booléen**, c'est-à-dire une variable qui ne peut prendre que deux valeurs, vrai (`true` en anglais) ou faux (`false` en anglais). On les utilise par exemple pour stocker des informations indiquant si la lumière est allumée, si l'utilisateur

a le droit d'utiliser une fonctionnalité donnée, ou encore si le mot de passe est correct. Si vous avez besoin de conserver le résultat d'une question de ce genre, alors pensez à ce type de variable.

Déclarer une variable

Assez parlé, il est temps d'entrer dans le vif du sujet et de demander à l'ordinateur de nous prêter un de ses tiroirs. En termes techniques, on parle de **déclaration de variable**.

Il nous faut indiquer à l'ordinateur le type de la variable que nous voulons, son nom et enfin sa valeur. Pour ce faire, c'est très simple : on indique les choses exactement dans l'ordre présenté à la figure 4.2.

TYPE NOM (VALEUR);

FIGURE 4.2 – Syntaxe d'initialisation d'une variable en C++

On peut aussi utiliser la même syntaxe que dans le langage C (figure 4.3).

TYPE NOM = VALEUR ;

FIGURE 4.3 – Syntaxe d'initialisation d'une variable, héritée du C

Les deux versions sont *strictement équivalentes*. Je vous conseille cependant d'utiliser la première pour des raisons qui deviendront claires plus tard. La deuxième version ne sera pas utilisée dans la suite du cours, je vous l'ai présentée ici pour que vous puissiez comprendre les nombreux exemples que l'on peut trouver sur le web et qui utilisent cette version de la déclaration d'une variable.



N'oubliez pas le point-virgule (;) à la fin de la ligne ! C'est le genre de choses que l'on oublie très facilement et le compilateur n'aime pas cela du tout.

Reprenons le morceau de code minimal et ajoutons-y une variable pour stocker l'âge de l'utilisateur.

```
#include <iostream>
using namespace std;

int main()
{
    int ageUtilisateur(16);
    return 0;
}
```

Que se passe-t-il à la ligne 6 de ce programme ? L'ordinateur voit que l'on aimerait lui emprunter un tiroir dans sa mémoire avec les propriétés suivantes :

- il peut contenir des nombres entiers ;
- il a une étiquette indiquant qu'il s'appelle `ageUtilisateur` ;
- il contient la valeur 16.

À partir de cette ligne, vous êtes donc l'heureux possesseur d'un tiroir dans la mémoire de l'ordinateur (figure 4.4).



FIGURE 4.4 – Un tiroir dans la mémoire de l'ordinateur contenant le chiffre 16

Comme nous allons avoir besoin de beaucoup de tiroirs dans la suite du cours, je vous propose d'utiliser des schémas un peu plus simples (figure 4.5). On va beaucoup les utiliser par la suite, il est donc bien de s'y habituer tôt.

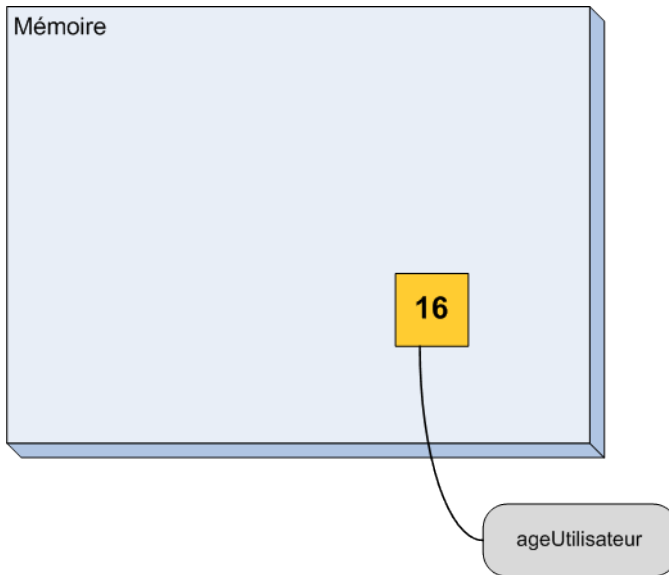


FIGURE 4.5 – Schéma de l'état de la mémoire après la déclaration d'une variable

Je vais vous décrire ce qu'on voit sur le schéma. Le gros rectangle bleu représente la mémoire de l'ordinateur. Pour l'instant, elle est presque vide. Le carré jaune est la zone de mémoire que l'ordinateur nous a prêtée. C'est l'équivalent de notre tiroir. Il contient, comme avant, le nombre 16 et on peut lire le nom `ageUtilisateur` sur l'étiquette qui y est accrochée. Je ne suis pas bon en dessin, donc il faut un peu d'imagination, mais le principe est là.

Ne nous arrêtons pas en si bon chemin. Déclarons d'autres variables.

```
#include <iostream>
using namespace std;

int main()
{
    int ageUtilisateur(16);
    int nombreAmis(432);      //Le nombre d'amis de l'utilisateur

    double pi(3.14159);

    bool estMonAmi(true);    //Cet utilisateur est-il mon ami ?

    char lettre('a');

    return 0;
}
```

Il y a deux choses importantes à remarquer ici. La première est que les variables de type `bool` ne peuvent avoir pour valeur que `true` ou `false`, c'est donc une de ces deux valeurs qu'il faut mettre entre les parenthèses. Le deuxième point à souligner, c'est que, pour le type `char`, il faut mettre la lettre souhaitée entre apostrophes. Il faut écrire `char lettre('a');` et pas `char lettre(a);`. C'est une erreur que tout le monde fait, moi le premier.



Il est toujours bien de mettre un commentaire pour expliquer à quoi va servir la variable.

Je peux donc compléter mon schéma en lui ajoutant nos nouvelles variables (figure 4.6).

Vous pouvez évidemment compiler et tester le programme ci-dessus. Vous constaterez qu'il ne fait strictement rien. J'espère que vous n'êtes pas trop déçus. Il se passe en réalité énormément de choses mais, comme je vous l'ai dit au début, ces opérations sont cachées et ne nous intéressent pas vraiment. En voici quand même un résumé chronologique.

1. votre programme demande au système d'exploitation de lui fournir un peu de mémoire;
2. l'OS¹ regarde s'il en a encore à disposition et indique au programme quel tiroir utiliser;
3. le programme écrit la valeur 16 dans la case mémoire;
4. il recommence ensuite pour les quatre autres variables;

1. Operating System ou, en français, système d'exploitation.

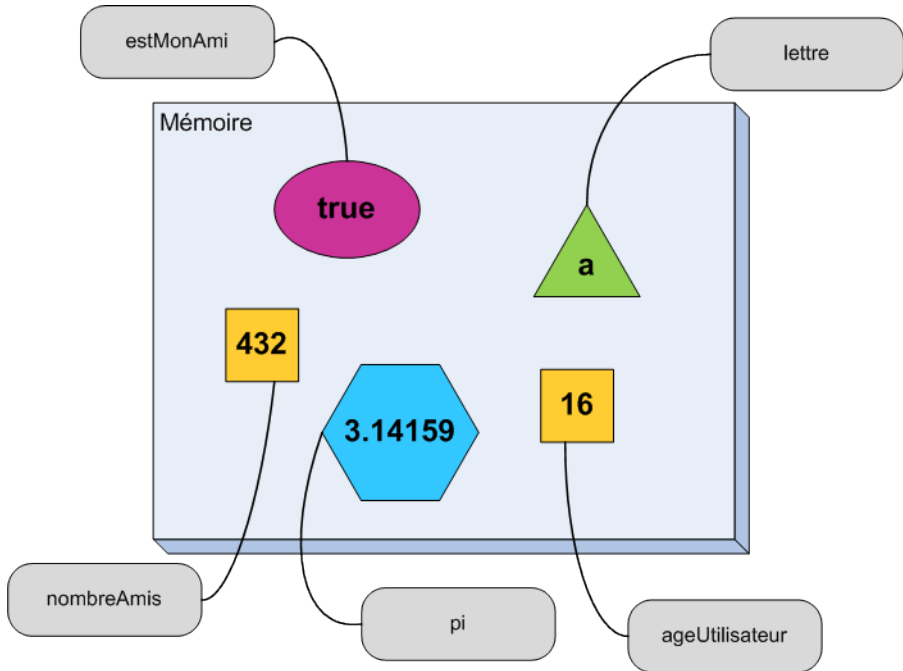


FIGURE 4.6 – Schéma de l'état de la mémoire après plusieurs déclarations

5. en arrivant à la dernière ligne, le programme vide ses tiroirs et les rend à l'ordinateur.

Et tout cela sans que rien ne se passe du tout à l'écran ! C'est normal, on n'a nulle part indiqué qu'on voulait afficher quelque chose.

Le cas des strings

Les chaînes de caractères sont un petit peu plus complexes à déclarer mais rien d'insurmontable, je vous rassure. La première chose à faire est d'ajouter une petite ligne au début de votre programme. Il faut, en effet, indiquer au compilateur que nous souhaitons utiliser des **strings**. Sans cela, il n'inclurait pas les outils nécessaires à leur gestion. La ligne à ajouter est `#include <string>`.

Voici ce que cela donne.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
```

```
string nomUtilisateur("Albert Einstein");
return 0;
}
```

L'autre différence se situe au niveau de la déclaration elle-même. Comme vous l'avez certainement constaté, j'ai placé des guillemets autour de la valeur. Un peu comme pour les `char` mais, cette fois, ce sont des guillemets doubles (") et pas juste des apostrophes ('). D'ailleurs votre IDE devrait colorier les mots "Albert Einstein" d'une couleur différente du 'a' de l'exemple précédent. Confondre ' et " est une erreur, là encore, très courante qui fera hurler de douleur votre compilateur. Mais ne vous en faites pas pour lui, il en a vu d'autres.

Une astuce pour gagner de la place

Avant de passer à la suite, il faut que je vous présente une petite astuce utilisée par certains programmeurs. Si vous avez plusieurs variables du *même type* à déclarer, vous pouvez le faire sur une seule ligne en les séparant par une virgule (,). Voici comment :

```
int a(2),b(4),c(-1); //On déclare trois cases mémoires nommées a, b et c et
↳ qui contiennent respectivement les valeurs 2, 4 et -1

string prenom("Albert"), nom("Einstein"); //On déclare deux cases pouvant
↳ contenir des chaînes de caractères
```

Ça peut être pratique quand on a besoin de beaucoup de variables d'un coup. On économise la répétition du type à chaque variable. Mais je vous déconseille quand même de trop abuser de cette astuce : le programme devient moins lisible et moins compréhensible.

Déclarer sans initialiser

Maintenant que nous avons vu le principe général, il est temps de plonger un petit peu plus dans les détails.

Lors de la déclaration d'une variable, votre programme effectue en réalité deux opérations successives.

1. Il demande à l'ordinateur de lui fournir une zone de stockage dans la mémoire. On parle alors d'**allocation** de la variable.
2. Il remplit cette case avec la valeur fournie. On parle alors d'**initialisation** de la variable.

Ces deux étapes s'effectuent automatiquement et sans que l'on ait besoin de rien faire. Voilà pour la partie vocabulaire de ce chapitre.

Il arrive parfois que l'on ne sache pas quelle valeur donner à une variable lors de sa déclaration. Il est alors possible d'effectuer uniquement l'allocation sans l'initialisation. Il suffit d'indiquer le **type** et le **nom** de la variable sans spécifier de valeur (figure 4.7).

TYPE NOM ;

FIGURE 4.7 – Déclaration d'une variable sans initialisation

Et sous forme de code C++ complet, voilà ce que cela donne :

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string nomJoueur;
    int nombreJoueurs;
    bool aGagne;        //Le joueur a-t-il gagné ?

    return 0;
}
```



Une erreur courante est de mettre des parenthèses vides après le nom de la variable, comme ceci : `int nombreJoueurs()`. C'est incorrect, il ne faut *pas* mettre de parenthèses, juste le type et le nom.

Simple non ? Je savais que cela allait vous plaire. Et je vous offre même un schéma en bonus (figure 4.8) !

On a bien trois cases dans la mémoire et les trois étiquettes correspondantes. La chose nouvelle est que l'on ne sait pas ce que contiennent ces trois cases. Nous verrons dans le chapitre suivant comment modifier le contenu d'une variable et donc remplacer ces points d'interrogation par d'autres valeurs plus intéressantes.



Je viens de vous montrer comment déclarer des variables sans leur donner de valeur initiale. Je vous conseille par contre de *toujours initialiser* vos variables. Ce que je vous ai montré là n'est à utiliser que dans les cas où l'on ne sait vraiment pas quoi mettre comme valeur, ce qui est très rare.

Il est temps d'apprendre à effectuer quelques opérations avec nos variables parce que vous en conviendrez, pour l'instant, on n'a pas appris grand chose d'utile. Notre écran est resté désespérément vide.

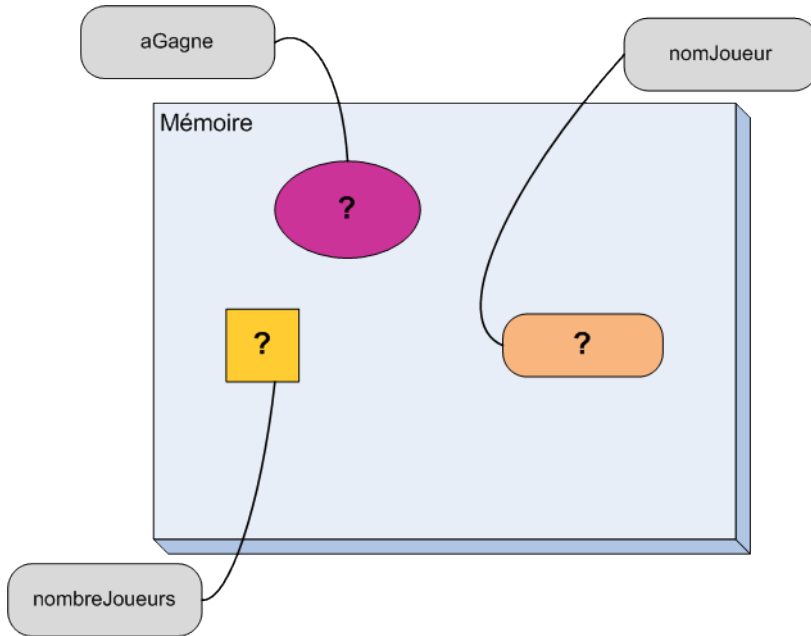


FIGURE 4.8 – La mémoire après avoir alloué 3 variables sans les initialiser

Afficher la valeur d'une variable

Au chapitre précédent, vous avez appris à afficher du texte à l'écran. J'espère que vous vous souvenez encore de ce qu'il faut faire.

Oui, c'est bien cela. Il faut utiliser `cout` et les chevrons (`<<`). Parfait. En effet, pour afficher le contenu d'une variable, c'est la même chose. À la place du texte à afficher, on met simplement le nom de la variable.

```
| cout << ageUtilisateur;
```

Facile non ?

Prenons un exemple complet pour essayer.

```
| #include <iostream>
| using namespace std;
|
| int main()
| {
|     int ageUtilisateur(16);
|     cout << "Votre age est : ";
|     cout << ageUtilisateur;
|     return 0;
| }
```

Une fois compilé, ce code affiche ceci à l'écran :

```
Votre age est : 16
```

Exactement ce que l'on voulait ! On peut même faire encore plus simple : tout mettre sur une seule ligne ! Et on peut même ajouter un retour à la ligne à la fin.



Pensez à mettre un espace à la fin du texte. Ainsi, la valeur de votre variable sera détachée du texte lors de l'affichage.

```
#include <iostream>
using namespace std;

int main()
{
    int ageUtilisateur(16);
    cout << "Votre age est : " << ageUtilisateur << endl;
    return 0;
}
```

Et on peut même afficher le contenu de plusieurs variables à la fois.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int qiUtilisateur(150);
    string nomUtilisateur("Albert Einstein");

    cout << "Vous vous appelez " << nomUtilisateur << " et votre QI vaut " <<
    ↵ qiUtilisateur << endl;
    return 0;
}
```

Ce qui affiche le résultat escompté.

```
Vous vous appelez Albert Einstein et votre QI vaut 150
```

Mais je pense que vous n'en doutiez pas vraiment. Nous verrons au prochain chapitre comment faire le contraire, c'est-à-dire récupérer la saisie d'un utilisateur et la stocker dans une variable.

Les références

Avant de terminer ce chapitre, il nous reste une notion importante à voir. Il s'agit des **références**. Je vous ai expliqué au tout début de ce chapitre qu'une variable pouvait être considérée comme une case mémoire avec une étiquette portant son nom. Dans la vraie vie, on peut très bien mettre plusieurs étiquettes sur un objet donné. En C++, c'est la même chose, on peut coller une deuxième (troisième, dixième, etc.) étiquette à une case mémoire. On obtient alors un deuxième moyen d'accéder à la *même* case mémoire. Un petit peu comme si on donnait un surnom à une variable en plus de son nom normal. On parle parfois d'**alias**, mais le mot correct en C++ est **référence**.

Schématiquement, on peut se représenter une référence comme à la figure 4.9.

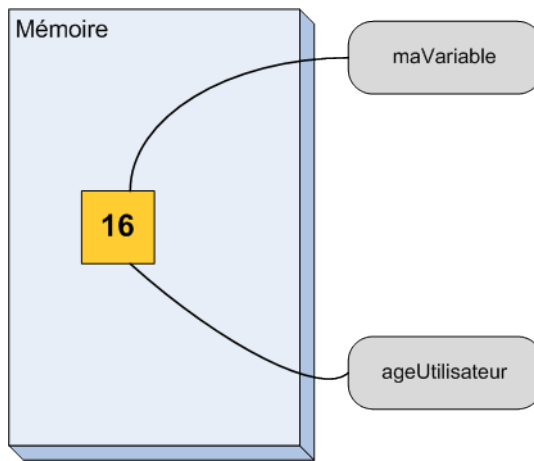


FIGURE 4.9 – Une variable et une référence sur cette variable

On a une seule case mémoire mais deux étiquettes qui lui sont accrochées.

Au niveau du code, on utilise une esperluette (&) pour déclarer une référence sur une variable. Voyons cela avec un petit exemple.

```
int ageUtilisateur(16); //Déclaration d'une variable.

int& maVariable(ageUtilisateur); //Déclaration d'une référence nommée
↪ maVariable qui est accrochée à la variable ageUtilisateur
```

À la ligne 1, on déclare une case mémoire nommée `ageUtilisateur` dans laquelle on met le nombre 16. Et à la ligne 3, on accroche une deuxième étiquette à cette case mémoire. On a donc dorénavant deux moyens d'accéder au même espace dans la mémoire de notre ordinateur.

On dit que `maVariable` *fait référence* à `ageUtilisateur`.



La référence doit impérativement être du même type que la variable à laquelle elle est accrochée! Un `int&` ne peut faire référence qu'à un `int`, de même qu'un `string&` ne peut être associé qu'à une variable de type `string`.

Essayons pour voir. On peut afficher l'âge de l'utilisateur comme d'habitude *et via* une référence.

```
#include <iostream>
using namespace std;

int main()
{
    int ageUtilisateur(18); //Une variable pour contenir l'âge de l'utilisateur

    int& maReference(ageUtilisateur); //Et une référence sur la variable
    ↪ 'ageUtilisateur'

    //On peut utiliser à partir d'ici
    //'ageUtilisateur' ou 'maReference' indistinctement
    //Puisque ce sont deux étiquettes de la même case en mémoire

    cout << "Vous avez " << ageUtilisateur << " ans. (via variable)" << endl;
    //On affiche, de la manière habituelle

    cout << "Vous avez " << maReference << " ans. (via reference)" << endl;
    //Et on affiche en utilisant la référence

    return 0;
}
```

Ce qui donne évidemment le résultat escompté.

```
Vous avez 18 ans. (via variable)
Vous avez 18 ans. (via reference)
```

Une fois qu'elle a été déclarée, on peut manipuler la référence comme si on manipulait la variable elle-même. Il n'y a *aucune différence* entre les deux.



Euh... Mais à quoi est-ce que cela peut bien servir ?

Bonne question ! C'est vrai que, dans l'exemple que je vous ai donné, on peut très bien s'en passer. Mais imaginez que l'on ait besoin de cette variable dans deux parties très différentes du programme, des parties créées par différents programmeurs. Dans une des parties, un des programmeurs va s'occuper de la déclaration de la variable alors que l'autre programmeur va juste l'afficher. Ce deuxième programmeur aura juste besoin

d'un accès à la variable et un alias sera donc suffisant. Pour l'instant, cela vous paraît très abstrait et inutile? Il faut juste savoir que c'est un des éléments importants du C++ qui apparaîtra à de très nombreuses reprises dans ce cours. Il est donc essentiel de se familiariser avec la notion avant de devoir l'utiliser dans des cas plus compliqués.

En résumé

- Une variable est une information stockée en mémoire.
- Il existe différents types de variables en fonction de la nature de l'information à stocker : `int`, `char`, `bool`...
- Une variable doit être déclarée avant utilisation. Exemple : `int ageUtilisateur(16);`
- La valeur d'une variable peut être affichée à tout moment avec `cout`.
- Les références sont des étiquettes qui permettent d'appeler une variable par un autre nom. Exemple : `int& maReference(ageUtilisateur);`