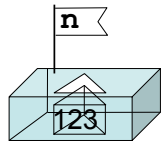
Les variables

Une variable possède 3 caractéristiques:

- Son **identificateur**, qui est le nom par lequel la donnée est désignée;
- Son **type**, qui définit de quel « genre » est la donnée contenue dans la variable;
- Sa **valeur**. Par exemple, si la donnée est un nombre, sa valeur pourra être 123 ou 3.14




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

int main()
{
  int n(4);
  int n_carre;

  n_carre = n * n;

  cout << "La variable n contient " << n << "." << endl;
  cout << "Le carre de " << n << " est " << n_carre << "." << endl;
  cout << "Le double de n est " << 2 * n << "." << endl;

  return 0;
}
```

Déclarations de variables

Les lignes:

```
int n(4);
int n_carre;
```

sont des **déclarations de variables**.

Une déclaration de variable permet de créer une variable.

Déclarations de variables

Les lignes:

type de la variable

```
int n(4);
int n_carre;
```

sont des **déclarations de variables**.

Une déclaration de variable permet de créer une variable.

Déclarations de variables

Les lignes:

type de la variable

identificateur, ou nom, de la variable.
Il est choisi par le programmeur et permet de référer la variable créée

```
int n(4);
int n_carre;
```

sont des **déclarations de variables**.

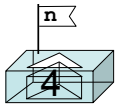
Une déclaration de variable permet de créer une variable.

Initialisation

En même temps qu'elle est déclarée, une variable peut être initialisée, c'est-à-dire lui donner une valeur avant de l'utiliser.

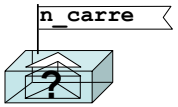
La ligne:

```
int n(4);
```



déclare donc une variable appelée `n` et lui donne la valeur 4.
Si une variable n'est pas initialisée, elle peut contenir n'importe quelle valeur!

```
int n_carre;
```

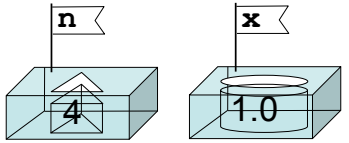


Il faut toujours initialiser une variable avant d'utiliser la valeur qu'elle contient !

type pour les valeurs entières: la variable `n` ne peut contenir que des valeurs entières

```
int n(4);
double x(1.0);
```

type pour les valeurs décimales: la variable `x` ne peut contenir que des valeurs décimales



Déclaration de variables

De façon générale, une déclaration de variable suit le schéma:

```
type identificateur(valeur_initiale);
```

ou éventuellement:

```
type identificateur;
```

N'oubliez pas le point-virgule ; à la fin

Une fois défini, le type de la variable ne peut plus changer.

Déclaration de variables

D'autres exemples de déclaration:

```
int m(1);
int p(1), q(0);
double x(0.1), y;
```

on peut déclarer plusieurs variables simultanément.
Ne pas en abuser

Noms de variables

Règles pour nommer les variables:

- Le nom doit être constitué uniquement de lettres et de chiffres (pas d'espace, ni de symboles !);
- Les accents ne sont pas autorisés;
- Le premier caractère est nécessairement une lettre;
- Le caractère souligné _ (*underscore*) est autorisé et considéré comme une lettre;
- Le nom ne doit pas être un mot-clé réservé par le langage C++;
- Les majuscules et les minuscules sont autorisées mais ne sont pas équivalentes. Les noms `ligne` et `Ligne` désignent deux variables différentes.

Exemples de noms valides:

`n_carre` `Total` `sousTotal98`

Exemples de noms invalides:

`n_carré` Contient un accent;
`n carre` Contient des espaces;
`n-carre` Contient le symbole – (moins);
`1element` Commence par un chiffre.

Types de variables

Les principaux types élémentaires sont:

- `int`, pour les valeurs entières (pour *integer*, entiers en anglais);
- `double`, pour les nombres à virgule, par exemple 0.5

et aussi:

- `unsigned int`: pour les entiers positifs;
- `char`: pour les caractères (A..Z etc.);
- …

Affectations

La ligne:

```
n_carre = n * n;
```

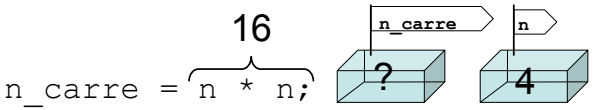
est une **affectation**.

Attention, ce **n'est pas** une égalité mathématique: Une affectation est une instruction qui permet de **changer** une valeur à une variable.

Affectations

L'exécution d'une affectation se décompose en deux temps :

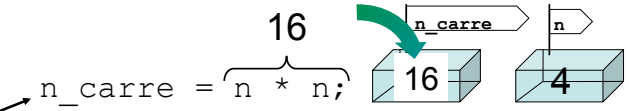
1 L'expression à droite du signe = est évaluée:
`n * n` est évaluée avec la valeur de `n` au moment de l'exécution.
L'étoile `*` représente la multiplication, `n * n` vaut donc 4 × 4 = 16



Affectations

L'exécution d'une affectation se décompose en deux temps :

2



la valeur de l'expression est stockée dans la variable à gauche du signe =
L'ancienne valeur de `n_carre` est perdue.

Affectations

De façon plus générale, une affectation suit le schéma:

nom_de_variable = ***expression;***

N'oubliez pas le point-virgule `;` à la fin

Une expression calcule une valeur, qui doit être de même type que la variable.

Exemples d'expression:

- `4`
- `n * n`
- `n * (n + 1) + 3 * n - 2`

Nous reviendrons sur les expressions un peu plus loin.

Attention: Ne confondez pas une affectation avec une égalité mathématique.

Toutes les deux utilisent le signe égal =, mais l'affectation est un mécanisme dynamique.

Par exemple, les deux instructions:

```
a = b;
b = a;
```

copie la valeur de `b` dans `a`

copie la valeur de `a` dans `b`

ne sont pas identiques.

En mathématiques:

$$b = a + 1$$

signifie que tout au long des calculs, a et b vérifieront toujours cette relation. Autrement dit, quel que soit a, b sera toujours égal à $a+1$

En C++:

```
a = 5;
b = a + 1;
a = 2;
```

donne à `b` la valeur de `a+1`, c'est-à-dire 6.

donne à `a` la valeur 2, sans que `b` ne soit modifiée!
`b` contient donc toujours 6.

On peut écrire aussi des affectations telles que:

```
a = a + 1;
```

Ce type d'affectation, où une variable apparaît de chaque côté du signe = permet de résoudre de nombreux problèmes.

Cette affectation signifie:
« calculer l'expression de `a + 1` et ranger le résultat dans `a`. Cela revient à augmenter de `1` la valeur de `a` »
Nous reviendrons sur ce point dans la suite.

Déclaration de constantes

Il peut arriver que la valeur d'une variable ne doive pas changer après l'initialisation. Dans ce cas, il faut ajouter le mot-clé `const` devant la déclaration:

```
const type identificateur(valeur_initiale);
```

Par exemple:

```
const double vitesse_de_la_lumiere(299792.458);
```

Dans ce cas, on ne peut plus modifier la variable:

```
vitesse_de_la_lumiere = 100; // erreur !
```