

Programmation C / C++

# #2 Basiques (1)

par David Albert

# Table des matières

## 01 Entrées / Sorties

std::cin. std::cout. scanf(). printf().

## 02 Types de variable

Entiers. Nombres à virgule flottante. Cast.

## 03 Déclaration de variables

Affectation de base. Convention de nommage.

## 04 Porter des variables

Portée d'une variable. Espace de noms.

## 05 Opérateurs de base

Opérateurs arithmétiques. Opérateurs logiques. Priorité.

## 06 Instructions de contrôle

Conditions. Boucles.

# 01

## Entrées / Sorties

# Entrées et Sorties

Pour lire des entrées saisies au clavier (depuis l'entrée standard **stdin**), on utilisera :

- `scanf()` en C ou C++
- `cin` en C++

Pour afficher des résultats à l'écran (sur la sortie standard **stdout**), on utilisera :

- `printf()` en C ou C++
- `cout` en C++

i

Il existe d'autres types d'entrées/sorties (fichier, réseau, base de données, ...) pour les programmes. Nous les verrons plus tard.

# Entrées et Sorties

## Exemple

### Exemple C++

```
#include <iostream>

int main (int argc, char **argv)
{
    int i;
    std::cout << "Entrez un entier : ";
    std::cin >> i;
    std::cout << "i = " << i << std::endl;

    return 0;
}
```

### Exemple C

```
#include <iostream>

int main (int argc, char **argv)
{
    int i;
    printf("Entrez un entier : ");
    scanf("%d", &i);
    printf("La value de i est %d", i);

    return 0;
}
```

# 02

## Types de variables

# Types de variables

## Types entiers

Les types entiers :

- **bool** : false ou true → booléen (seulement en C++ )
- **unsigned char** : 0 à 255 → entier très court (1 octet ou 8 bits)
- **char** : -128 à 127 → idem mais en entier relatif
- **unsigned short** : 0 à  $2^{16}$  → entier court (2 octets ou 16 bits)
- **short** :  $-2^{15}$  à  $2^{15} - 1$  → idem mais en entier relatif
- **unsigned int** : 0 à  $2^{32} - 1$  → entier sur 4 octets
- **int** :  $-2^{31}$  à  $2^{31} - 1$  → idem mais en entier relatif
- **unsigned long** : 0 à  $2^{32} - 1$  → entier sur 4 octets ou plus
- **long** :  $-2^{31}$  à  $2^{31} - 1$  → idem mais en entier relatif
- **unsigned long long** : 0 à  $2^{64} - 1$  → entier sur 8 octets
- **long long** :  $-2^{63}$  à  $2^{63} - 1$  → idem mais en entier relatif

# Types de variables

## Types à virgule flottante

Les types à virgule flottante :

- **float** : environ 6 chiffres de précision et un exposant qui va jusqu'à  $\pm 10^{\pm 38}$   
→ Codage IEEE754 sur 4 octets
- **double** : environ 10 chiffres de précision et un exposant qui va jusqu'à  $\pm 10^{\pm 308}$   
→ Codage IEEE754 sur 8 octets
- **long double** – → Codé sur 10 octets



# Types de variables

## Les constantes

Les constantes :

- celles définies pour le préprocesseur. Il n'y a aucun typage de la constante.

```
#define PI 3.1415 /* en C traditionnel */
```

- celles définies pour le compilateur : c'est une variable typée en lecture seule.

```
const double PI = 3.1415; // en C++ et en C ISO
```

i

Par convention, les noms des constantes sont en majuscules.

Exemple : `MAX_PLAYERS`, `HEIGHT`, `WIDTH`, etc..

# Conversion de type (cast)

## Implicite / explicite

i

Le compilateur ne peut appliquer des opérateurs qu'à des opérandes de même type.

**Exemple** : il n'existe pas d'addition pour :  $2 + 1.5$  car 2 est un entier et 1.5 est un flottant.

Il existe deux types de conversions :

1. **sans perte** :  $\text{int} \rightarrow \text{float}$  (2 devient 2.0)  
ces conversions sont automatiquement réalisées par le compilateur
2. **avec perte** :  $\text{float} \rightarrow \text{int}$  (1.5 devient 1)  
ces conversions doivent être explicitées par le programmeur

# Conversion explicite

## Conversion explicite

### Ancien opérateur (C/C++)

On utilise l'opérateur de **cast** en précisant le type entre parenthèses devant la variable à convertir (C/C++).

Exemple :

```
float a = 2.5;  
int b = (int)a; // force la variable a en int
```

### Nouveaux opérateurs (C++ uniquement)

- **static\_cast** : opérateur de transtypage à tout faire.
- **const\_cast** : opérateur spécialisé et limité au traitement des caractères const et volatile.
- **dynamic\_cast** : opérateur spécialisé et limité au traitement des downcast (transtypage descendant dans le cas d'héritage en POO).
- **reinterpret\_cast** : opérateur spécialisé dans le traitement des conversions de pointeurs.

Exemple :

```
float a = 2.5;  
int b = static_cast<int>(a); // force la variable a en int
```

# 03

## Déclaration de variables

# Qu'est-ce qu'une variable ?

## ♥ Définition - Variable

En informatique, les **variables** sont des symboles qui associent un nom (**l'identifiant**) à une **valeur**. Dans la plupart des langages, **les variables peuvent changer de valeur au cours du temps**.

De plus, les variables ont un **type** de valeur (int, bool, double, ...).

En C / C++, la déclaration d'une variable se fait avec l'opérateur d'allocation **=**

## Exemples :

```
char letter = 'A';           # variable de type entier (char)
int age = 23;                 # variable de type entier (int)
float moyenne = 10.8;         # variable de type flottant (float)
string prenom = "Jonathan";   # variable de type chaîne de caractères (string)
```

# Convention de nommage

## Nommage des variables

Par convention, un nom de variable commence par une lettre minuscule puis les différents mots sont repérés en mettant en majuscule la première lettre.

*Exemples :* `distance`, `distanceMax`, `consigneCourante`, `etatBoutonGaucheSouris`, `nbreDEssais`

## Nommage des constantes

Par convention, un nom de constante est en majuscule.

*Exemple :* `MAX_PLAYERS`, `HEIGHT`, `WIDTH`, ...

## Mots réservés

Les mots réservés sont les mots prédéfinis du langage C.  
Ils ne peuvent pas être réutilisés pour des identifiants.

*Exemples :* `for`, `do`, `while`, `if`, `return`, `void`, `extern`, `break`, `static`, ...

# 04

## Portée des variables

# Portée d'une variable

La portée (scope) d'un identifiant (variables, fonctions, ...) est l'étendue au sein de laquelle cet identifiant est lié.

En C/C++, la portée peut être globale (en dehors de tout bloc {})  
ou locale (au bloc {}).

```
int uneVariableGlobale; // initialisée par défaut à 0

int main(int argc, char* argv[])
{
    int uneVariableLocale; // non initialisée par défaut
    {
        int uneAutreVariableLocale; // non initialisée par défaut
    }
    // la variable i est locale bloc for :
    for(int i=0;i<10;i++) cout << i;
    return 0;
}
```

!

Des variables déclarées dans des blocs différents peuvent porter le même nom.



# Espace de noms

# 05

## Opérateurs de base

# Opérateurs arithmétiques

Permettre d'effectuer des calculs mathématiques sur et entre les variables.

Opérateur	Nom	Usage
+	addition	<code>a + b</code>
-	soustraction	<code>a - b</code>
*	multiplication	<code>a * b</code>
/	division	<code>a / b</code>
%	reste de la division	<code>a % b</code>

Ou encore d'effectuer opérations bits à bits sur les représentations binaires des variables.

Opérateur	Nom	Usage
<<	décalage à gauche	<code>a &lt;&lt; 3</code>
>>	décalage à droite	<code>a &gt;&gt; 10</code>
~	complément à 1	<code>~a</code>

# Opérateurs logiques

Permettre de comparer et réaliser des tests logiques entre des valeurs.

Opérateur	Nom	Exemple
==	égalité	a == b
!=	différence	a != b
< et <=	infériorité	a < b
> et >=	supériorité	a >= b
&&	ET	a && b
	OU	a    b
^^	OU EXCLUSIF	a ^^ b
!	NON	!a

# Opérateurs d'affectation

Pour mémoriser le résultat d'une opération dans une variable, on utilise l'opérateur d'affectation `=`.

Opérateur	Nom	Exemple	Equivalence
<code>=</code>	affectation classique	<code>a = (3 == 4)</code>	<code>-</code>
<code>++</code>	incrément de 1	<code>a++</code>	<code>a = a + 1</code>
<code>--</code>	décrément de 1	<code>a--</code>	<code>a = a - 1</code>
<code>+=</code>	addition	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	soustraction	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	multiplication	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	division	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	reste de la division	<code>a %= b</code>	<code>a = a % b</code>

# Priorité des opérateurs

Opérateurs	Description
::	résolution de porté
. -> [] () sizeof()	référence et sélection, parenthèses
! ~ ++ -- - +	unaires préfixés
* / %	multiplicatifs
+ -	additions
>> <<	décalages
< <= > >=	relations d'ordre
= = !=	égalité
&&	logique
? :	conditionnel (ternaire)
= += -= *=	affectation

i

## Note

On peut dans tous les cas forcer une évaluation allant contre les priorités définies en utilisant le parenthésage des expressions à évaluer en premier.

# 06

## Instructions de contrôle

# Instructions de contrôle

## Conditions (if / else)

### 1. if / else

```
if (cond) {  
    // code si vrai  
} else {  
    // code si faux  
}
```

### 2. if / else if / else

```
if (cond) {  
    // code si vrai  
} else if (cond2) {  
    // code si vrai  
} else if (cond3) {  
    // code si vrai  
} else {  
    // code si faux  
}
```

### Exemple

```
int temperature;  
scanf("%d", &temperature);  
if (temperature >= 100)  
{  
    printf("L'eau bout !");  
}
```

i

La notation **cond** dans les exemples ci-contre représente une expression quelconque qui renvoie un booléen.

Exemples :

- `if (true)`
- `if (is_winner)`
- `if (age < 30)`
- `if (age < 20 && name == "Mathéo")`



# Instructions de contrôle

## Conditions (switch)

### Exemple

```
switch (unite)
{
    case 'i':
        cout << longueur << " in == " << conversion * longueur << " cm\n";
        break;
    case 'c':
        cout << longueur << " cm == " << longueur / conversion << " in\n";
        break;
    default:
        cout << "Désolé, je ne connais pas cette unité " << unite << endl;
        break;
}
```

### Notes

- la valeur utilisée par le **switch()** doit être un entier, un char ou une énumération.
- on peut utiliser plusieurs **case** menant à la même instruction
- !!! Ne pas oublier les **break**

# Instructions de contrôle

## Boucles (do / while)

### 1. while ... do ...

```
while (cond) {  
    // ...  
    // code ici  
    // ...  
}
```

### 2. do ... while ...

```
do {  
    // ...  
    // code ici  
    // ...  
} while (cond);
```

### Exemple

```
while (motDePasse != secret || agePersonne <= 3)  
{  
    printf("Accès refusé\n");  
    scanf("%d %d", &agePersonne, &motDePasse);  
}
```

# Instructions de contrôle

## Boucles (for)

*for (expression1 ; expression2 ; expression3 ) instructions*

avec

- `expression1` est la condition de départ (initialisations).
- `expression2` est la condition de fin.
- `expression3` est l'incrément de boucle.

i

Les 5 étapes du déroulement d'une boucle for sont :

1. `expression1` est évaluée avant d'entrer dans le for
2. `expression2` est évaluée
3. si `expression2` est vrai, instructions est exécuté, sinon, on passe après la fin de la boucle et l'exécution de la boucle est finie
4. `expression3` est évaluée après l'exécution de instructions
5. on revient en 2

# Instructions de contrôle

## Boucles (for)

### Exemple 1

```
for (int i = 0; i < 100; i++) {  
    cout << i << '\n';  
}
```

### Exemple 2

```
for (int i = 100; i >= 0; i--) {  
    cout << i << '\n';  
}
```