

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thompson aux laboratoires Bell parallèlement à la création du système d'exploitation UNIX.

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thompson aux laboratoires Bell parallèlement à la création du système d'exploitation UNIX.
- 1978 : première édition du livre "The C programming language" (Kernighan & Ritchie)

# C1 Introduction au langage C

## 1. Historique

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thompson aux laboratoires Bell parallèlement à la création du système d'exploitation UNIX.
- 1978 : première édition du livre "The C programming language" (Kernighan & Ritchie)
- 1983 : première standardisation du langage par l'ANSI qui assure la compatibilité et la portabilité entre différentes plateformes. La dernière standardisation date de 2018 (C18)

# C1 Introduction au langage C

## 1. Historique

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thompson aux laboratoires Bell parallèlement à la création du système d'exploitation UNIX.
- 1978 : première édition du livre "The C programming language" (Kernighan & Ritchie)
- 1983 : première standardisation du langage par l'ANSI qui assure la compatibilité et la portabilité entre différentes plateformes. La dernière standardisation date de 2018 (C18)
- A partir de 1983 : développement de plusieurs dérivés de C, parmi lesquels C++ (B. Stroustup, 1983), C# (Microsoft, 2000), Go (Google, 2007), Rust (Mozilla, 2010)

# C1 Introduction au langage C

## 1. Historique

### Et aujourd'hui ...

- Le C reste un des langages les plus utilisés au monde, en juillet 2025 le classement de l'index TIOBE était le suivant :

1	Python	27 %
2	C++	9.8 %
3	C	9.7 %
4	Java	8.8 %
5	C#	4.87 %

# C1 Introduction au langage C

## 1. Historique

### Et aujourd'hui ...

- Le C reste un des langages les plus utilisés au monde, en juillet 2025 le classement de l'index TIOBE était le suivant :

1	Python	27 %
2	C++	9.8 %
3	C	9.7 %
4	Java	8.8 %
5	C#	4.87 %

- Le C est notamment très utilisé pour la programmation système, le développement d'applications embarquées, de *drivers* et les systèmes d'exploitation.

# C1 Introduction au langage C

## 1. Historique

### Et aujourd'hui ...

- Le C reste un des langages les plus utilisés au monde, en juillet 2025 le classement de l'index TIOBE était le suivant :

1	Python	27 %
2	C++	9.8 %
3	C	9.7 %
4	Java	8.8 %
5	C#	4.87 %

- Le C est notamment très utilisé pour la programmation système, le développement d'applications embarquées, de *drivers* et les systèmes d'exploitation.
- Pour de multiples raisons, Le C est l'un des langages les plus rapides (certains nouveaux langages comme *Rust* ont des performances similaires).

### Instructions et expressions

- En informatique, on distingue :
  - Les **instructions** qui modifient l'état des variables du programme. Par exemple, en Python `a = 5`, est une instruction qui modifie la variable `a` en lui donnant la valeur 5.
  - Les **expressions** qui *renvoient* une valeur après un processus d'évaluation. Par exemple, en Python `2**10` est une expression qui après évaluation renvoie 1024.



### Instructions et expressions


- En informatique, on distingue :
    - Les **instructions** qui modifient l'état des variables du programme. Par exemple, en Python `a = 5`, est une instruction qui modifie la variable `a` en lui donnant la valeur 5.
    - Les **expressions** qui *renvoient* une valeur après un processus d'évaluation. Par exemple, en Python `2**10` est une expression qui après évaluation renvoie 1024.
  - Le C est typiquement un langage **impératif**, c'est à dire qu'un programme est une séquence d'instructions exécutées par l'ordinateur. C n'est ni orienté objet, ni fonctionnel.
- ⚠ Cependant, certaines instructions du C sont aussi des expressions dans le sens où elles renverront une valeur.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :


 Code source  
(fichier(s)  
texte .c)

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :

 **Code source**  
(fichier(s)  
texte .c)

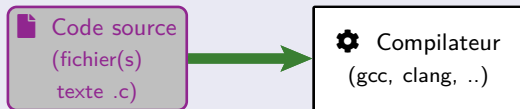
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



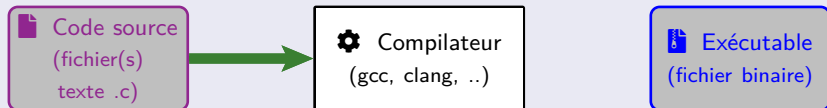
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissements (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions des liens

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



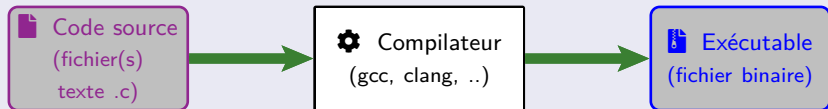
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissements (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions des liens

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



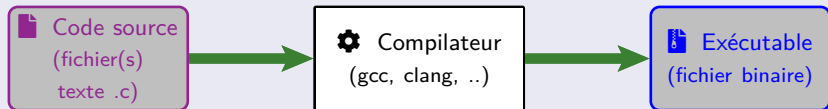
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissements (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions des liens
- 3 Une compilation sans erreur (mais éventuellement des *warning*) produit un exécutable.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissements (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions des liens
- 3 Une compilation sans erreur (mais éventuellement des *warning*) produit un exécutable.
- 4 Les erreurs dans l'exécution ne feront pas référence aux instructions du code source.



# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

Si le fichier texte s'appelle `hello.c`, on lance la compilation avec le compilateur `gcc` avec la commande :

```
gcc hello.c
```

le fichier exécutable produit s'appelle par défaut `a.out`, on peut modifier ce nom avec l'option `-o` du compilateur. Par exemple :

```
gcc -o hello hello.c
```

# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

Le langage C est doté d'une *bibliothèque standard* de fonctions appelée **libc** qui permet de réaliser des opérations courantes : entrées sorties, gestion de fichiers, gestion de la mémoire, ...

On inclut à la ligne 1, les fonctions **standard** d'entrées et de sorties (**input** et **output**) de la libc. Ce qui permettra plus loin d'utiliser la fonction `printf`.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

Un programme C contient *obligatoirement* une fonction appelée `main` par laquelle l'exécution du programme commence. L'absence de fonction `main` dans un programme C produit une erreur à la compilation : « référence indéfinie vers `main` ». Les fonctions en C sont définies avec la syntaxe suivante :

`<type de la valeur renvoyée> <nom>(<type et nom des paramètres>)`

Ici la fonction s'appelle `main`, renvoie un entier `int` et n'a pas d'arguments.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

Les blocs d'instructions sont délimités par des accolades : { et }, ici c'est donc le bloc d'instruction de la fonction `main`. A noter que les espaces, sauts de ligne et indentation sont ignorés par le compilateur, mais sont nécessaires pour une bonne lisibilité.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

L'instruction `printf` permet d'afficher dans le terminal. On notera les guillemets ("") pour délimiter une chaîne de caractères et le caractère `\n` pour indiquer un retour à la ligne. Une instruction en C se termine par un point virgule `;`.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Programme minimal

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6      return 0;
7  }
```

L'instruction `return` quitte la fonction en renvoyant la valeur donnée. Ici, on renvoie 0 (on rappelle que la fonction doit renvoyer un entier), qui indique traditionnellement que le programme se termine sans erreurs.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int somme = 0;
6      const int nmax = 100;
7      for (int i = 1; i <= nmax; i = i + 1)
8      {
9          somme = somme + i;
10     }
11     printf("1+2+...+100 = %d\n", somme);
12     return 0;
13 }
```

Déclaration de la variable `somme` de type `int` et initialisation à zéro.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int somme = 0;
6      const int nmax = 100;
7      for (int i = 1; i <= nmax; i = i + 1)
8      {
9          somme = somme + i;
10     }
11     printf("1+2+...+100 = %d\n", somme);
12     return 0;
13 }
```

Une variable dont la valeur ne sera pas modifiée peut être déclarée en faisant précéder son type de `const`.

On peut aussi utiliser une directive de précompilation : `#define NMAX 100`



# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int somme = 0;
6      const int nmax = 100;
7      for (int i = 1; i <= nmax; i = i + 1)
8      {
9          somme = somme + i;
10     }
11     printf("1+2+...+100 = %d\n", somme);
12     return 0;
13 }
```

On remarque que la boucle `for` est de la forme `for (<init>; <fin>; <incr>)`.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int somme = 0;
6      const int nmax = 100;
7      for (int i = 1; i <= nmax; i = i + 1)
8      {
9          somme = somme + i;
10     }
11     printf("1+2+...+100 = %d\n", somme);
12     return 0;
13 }
```

L'affichage en C utilise un système de *format*. Ici, on veut afficher un `int` dans la réponse, on utilise `%d` appelé *spécificateur de format* dans `printf` à l'emplacement souhaité.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de fonction et d'instruction conditionnelle

```
1  #include <stdio.h>
2  // S(n) = n/2 si n est pair et 3*n+1 sinon
3  int syracuse(int n)
4  {
5      if (n % 2 == 0) //(n est l'opérateur modulo, == le test d'égalité)
6      {
7          return n / 2;
8      }
9      else
10     {
11         return 3 * n + 1;
12     }
13 }
14 int main()
15 {
16     int n = 42;
17     printf("Syracuse(%d) = %d \n",n, syracuse(n));
18 }
```

Une ligne de commentaire commence avec `//`, un commentaire multiligne est encadré par `/*` et `*/`

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de fonction et d'instruction conditionnelle

```
1  #include <stdio.h>
2  // S(n) = n/2 si n est pair et 3*n+1 sinon
3  int syracuse(int n)
4  {
5      if (n % 2 == 0) //(n est l'opérateur modulo, == le test d'égalité)
6      {
7          return n / 2;
8      }
9      else
10     {
11         return 3 * n + 1;
12     }
13 }
14 int main()
15 {
16     int n = 42;
17     printf("Syracuse(%d) = %d \n",n, syracuse(n));
18 }
```

Signature (ou prototype) de la fonction.

❗ En C, les paramètres sont passés par **valeur** (on dit aussi par **copie**).

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de fonction et d'instruction conditionnelle

```
1  #include <stdio.h>
2  // S(n) = n/2 si n est pair et 3*n+1 sinon
3  int syracuse(int n)
4  {
5      if (n % 2 == 0)  //(n est l'opérateur modulo, == le test d'égalité)
6      {
7          return n / 2;
8      }
9      else
10     {
11         return 3 * n + 1;
12     }
13 }
14 int main()
15 {
16     int n = 42;
17     printf("Syracuse(%d) = %d \n",n, syracuse(n));
18 }
```

Instruction conditionnelle : on exécute le bloc qui suit la condition si celle-ci est vérifiée et sinon le bloc qui suit le **else** (s'il est présent). Noter les parenthèses autour de la condition.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de fonction et d'instruction conditionnelle

```
1  #include <stdio.h>
2  // S(n) = n/2 si n est pair et 3*n+1 sinon
3  int syracuse(int n)
4  {
5      if (n % 2 == 0) //(n est l'opérateur modulo, == le test d'égalité)
6      {
7          return n / 2;
8      }
9      else
10     {
11         return 3 * n + 1;
12     }
13 }
14 int main()
15 {
16     int n = 42;
17     printf("Syracuse(%d) = %d \n",n, syracuse(n));
18 }
```

Appel à la fonction `syracuse`, c'est une copie du `n` définie dans le `main` qui est envoyé à la fonction, elle renvoie un entier.

# C1 Introduction au langage C

## 4. Types de base en C

### Types de base

Type	Opérations	Commentaires

# C1 Introduction au langage C

## 4. Types de base en C

### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>		



# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>	

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>intN_t</code> et <code>uintN_t</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int<math>N</math>_t</code> et <code>uint<math>N</math>_t</code>	$+$ , $-$ , $*$ , $/$ , $\%$  $++$ , $--$	Entiers signés ou non signés codés sur un minimum de 16 bits.  Entiers codés sur $N$ bits accessibles dans <code>stdint.h</code> ( $N = 8, 32$ ou $64$ ).

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int<math>N</math>_t</code> et <code>uint<math>N</math>_t</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.  Entiers codés sur $N$ bits accessibles dans <code>stdint.h</code> ( $N = 8, 32$ ou $64$ ).
<code>float</code> et <code>double</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgules flottantes en simple ou double précision de la norme IEEE754. Fonctions élémentaires dans <code>math.h</code>

# C1 Introduction au langage C

## 4. Types de base en C


### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int<math>N</math>_t</code> et <code>uint<math>N</math>_t</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.  Entiers codés sur $N$ bits accessibles dans <code>stdint.h</code> ( $N = 8, 32$ ou $64$ ).
<code>float</code> et <code>double</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgules flottantes en simple ou double précision de la norme IEEE754. Fonctions élémentaires dans <code>math.h</code>
<code>bool</code> : <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Évaluation « séquentielle » des expressions.

# C1 Introduction au langage C

## 4. Types de base en C

### Types de base


Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int<math>N</math>_t</code> et <code>uint<math>N</math>_t</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.  Entiers codés sur $N$ bits accessibles dans <code>stdint.h</code> ( $N = 8, 32$ ou $64$ ).
<code>float</code> et <code>double</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgules flottantes en simple ou double précision de la norme IEEE754. Fonctions élémentaires dans <code>math.h</code>
<code>bool</code> : <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Évaluation « séquentielle » des expressions.
<code>char</code>	<code>+</code> , <code>-</code>	Caractères noté entre quotes : <code>'</code> , uniquement ceux de la table ASCII. Caractère nul : <code>'\0'</code>



# C1 Introduction au langage C

## 4. Types de base en C

### Types de base

Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int<math>N</math>_t</code> et <code>uint<math>N</math>_t</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>  <code>++</code> , <code>--</code>	Entiers signés ou non signés codés sur un minimum de 16 bits.  Entiers codés sur $N$ bits accessibles dans <code>stdint.h</code> ( $N = 8, 32$ ou $64$ ).
<code>float</code> et <code>double</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgules flottantes en simple ou double précision de la norme IEEE754. Fonctions élémentaires dans <code>math.h</code>
<code>bool</code> : <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Évaluation « séquentielle » des expressions.
<code>char</code>	<code>+</code> , <code>-</code>	Caractères noté entre quotes : <code>'</code> , uniquement ceux de la table ASCII. Caractère nul : <code>'\0'</code>
<code>void</code>		Type vide, utilisé notamment pour les fonctions ne renvoyant rien.

### Exemples

- 1 Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.

### Exemples

- 1 Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.
- 2 Déclarer et initialiser `d` ayant comme valeur  $b^2 - 4ac$ , en supposant que `a`, `b` et `c` sont des variables flottantes existantes et initialisées.

### Exemples

- ❶ Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.
- ❷ Déclarer et initialiser `d` ayant comme valeur  $b^2 - 4ac$ , en supposant que `a`, `b` et `c` sont des variables flottantes existantes et initialisées.
- ❸ On suppose déjà déclarées deux variables booléennes `x` et `y`, écrire une expression booléenne correspondant à `x xor y`.

### Exemples

- 1 Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.
- 2 Déclarer et initialiser `d` ayant comme valeur  $b^2 - 4ac$ , en supposant que `a`, `b` et `c` sont des variables flottantes existantes et initialisées.
- 3 On suppose déjà déclarées deux variables booléennes `x` et `y`, écrire une expression booléenne correspondant à `x xor y`.
- 4 La déclaration de variable suivante est-elle correcte ? `char c = "a";`

### Exemples

- 1 Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.
- 2 Déclarer et initialiser `d` ayant comme valeur  $b^2 - 4ac$ , en supposant que `a`, `b` et `c` sont des variables flottantes existantes et initialisées.
- 3 On suppose déjà déclarées deux variables booléennes `x` et `y`, écrire une expression booléenne correspondant à `x xor y`.
- 4 La déclaration de variable suivante est-elle correcte ? `char c = "a";`
- 5 Quelle est selon vous la cause du message : *warning : 'return' with a value, in function returning void* lors d'une compilation ?

# C1 Introduction au langage C

## 4. Types de base en C

### Exemples

- 1 Déclarer deux variables entières `a` et `b` initialisées respectivement à 2024 et 137. Déclarer `c` ayant comme valeur le reste dans division euclidienne de `a` par `b`.
- 2 Déclarer et initialiser `d` ayant comme valeur  $b^2 - 4ac$ , en supposant que `a`, `b` et `c` sont des variables flottantes existantes et initialisées.
- 3 On suppose déjà déclarées deux variables booléennes `x` et `y`, écrire une expression booléenne correspondant à `x xor y`.
- 4 La déclaration de variable suivante est-elle correcte ? `char c = "a";`
- 5 Quelle est selon vous la cause du message : *warning : 'return' with a value, in function returning void* lors d'une compilation ?
- 6 On suppose que `a` et `b` sont des `int`, les instructions `if (b>0 && a/b < 1)` et `if (a/b<1 && b>0)` sont-elles rigoureusement équivalentes ? Sinon, donner des valeurs de `a` et `b` pour lesquelles elles ne produisent pas le même effet.

# C1 Introduction au langage C

## 4. Types de base en C

### Affichage et spécificateur de format

En C, l'affichage des variables se fait à l'aide de spécificateurs de format suivant le type de la variable

Type	Spécificateur
<code>char</code>	<code>%c</code>
<code>unsigned int</code> , <code>uint8_t</code> et <code>uint32_t</code>	<code>%u</code>
<code>int</code> , <code>int8_t</code> et <code>int32_t</code>	<code>%d</code>
<code>float</code>	<code>%f</code>
<code>double</code>	<code>%lf</code>
<code>uint64_t</code>	<code>%lu</code>
<code>int64_t</code>	<code>%ld</code>

### Exemple

Si `a` et `b` sont des `int`, écrire une instruction `printf` permettant d'afficher `a+b=` suivie de leur somme.



### Définition

- La **portée** d'une variable est la partie du programme dans laquelle cette variable est visible (on peut y faire référence).

### Définition

- La **portée** d'une variable est la partie du programme dans laquelle cette variable est visible (on peut y faire référence).
- La portée peut-être :

### Définition

- La **portée** d'une variable est la partie du programme dans laquelle cette variable est visible (on peut y faire référence).
- La portée peut-être :
  - **globale**, c'est-à-dire que la variable est accessible depuis tout le programme. En C, c'est le cas des variables déclarées en début de programme en dehors de tout bloc d'instructions.

### Définition

- La **portée** d'une variable est la partie du programme dans laquelle cette variable est visible (on peut y faire référence).
- La portée peut-être :
  - **globale**, c'est-à-dire que la variable est accessible depuis tout le programme. En C, c'est le cas des variables déclarées en début de programme en dehors de tout bloc d'instructions.
  - **locale** lorsque la variable est déclarée dans un bloc d'instruction alors sa portée est limitée à ce bloc. C'est le cas des paramètres d'une fonction ou d'une variable de boucle.

### Définition

- La **portée** d'une variable est la partie du programme dans laquelle cette variable est visible (on peut y faire référence).
- La portée peut-être :
  - **globale**, c'est-à-dire que la variable est accessible depuis tout le programme. En C, c'est le cas des variables déclarées en début de programme en dehors de tout bloc d'instructions.
  - **locale** lorsque la variable est déclarée dans un bloc d'instruction alors sa portée est limitée à ce bloc. C'est le cas des paramètres d'une fonction ou d'une variable de boucle.
- Lorsque deux variables ont le même identifiant, c'est la variable ayant la plus petite portée (celle définie dans le bloc le plus intérieur) qui est accessible.

### Exemples

- ① Dans le programme suivant, donner les portées de `maxn`, `n`, `somme`, `i`

```
1  #include <stdio.h>
2
3  const int maxn = 10000;
4
5  double harmo(int n)
6  {
7      double somme = 0;
8      for (int i = 1; i <= n; i = i + 1)
9      {
10         somme = somme + 1.0 / i;
11     }
12     return somme;
13 }
14
15 int main()
16 {
17     double s = harmo(maxn);
18     printf("Somme = %f\n", s);
19     return 0;
20 }
```

# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Exemples

- ① Dans le programme suivant, donner les portées de `maxn`, `n`, `somme`, `i`

```
1  #include <stdio.h>
2
3  const int maxn = 10000;
4
5  double harmo(int n)
6  {
7      double somme = 0;
8      for (int i = 1; i <= n; i = i + 1)
9      {
10         somme = somme + 1.0 / i;
11     }
12     return somme;
13 }
14
15 int main()
16 {
17     double s = harmo(maxn);
18     printf("Somme = %f\n", s);
19     return 0;
20 }
```

- ② On définit une variable entière `i`, juste après la ligne 7, donner la portée de cette nouvelle variable. Le programme fonctionne-t-il encore correctement ?

### Conversion implicite de type

La ligne `double somme = 0;` est une **conversion implicite de type**. En effet, 0 est de type entier mais est converti en flottant pour être affecté à la variable `somme` qui est de type double.



# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Conversion implicite de type

La ligne `double somme = 0;` est une **conversion implicite de type**. En effet, 0 est de type entier mais est converti en flottant pour être affecté à la variable `somme` qui est de type double.

### Conversion explicite : *cast*

On aurait pu réaliser une **conversion explicite** ou *cast* en spécifiant le type de destination entre parenthèses : `double somme = (double) 0;`

# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Exemple

```
1  #include <stdio.h>
2
3  float division(int num, int den)
4  {
5      float res = num / den;
6      return res;
7  }
8
9  int main()
10 {
11     float deux_tiers = division(2, 3);
12     printf("2/3 = %f\n", deux_tiers);
13     return 0;
14 }
```

# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Exemple

```
1  #include <stdio.h>
2
3  float division(int num, int den)
4  {
5      float res = num / den;
6      return res;
7  }
8
9  int main()
10 {
11     float deux_tiers = division(2, 3);
12     printf("2/3 = %f\n", deux_tiers);
13     return 0;
14 }
```

- Quel est le résultat de ce programme ? Pourquoi ?

# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Exemple

```
1  #include <stdio.h>
2
3  float division(int num, int den)
4  {
5      float res = num / den;
6      return res;
7  }
8
9  int main()
10 {
11     float deux_tiers = division(2, 3);
12     printf("2/3 = %f\n", deux_tiers);
13     return 0;
14 }
```

- Quel est le résultat de ce programme ? Pourquoi ?
- Comment afficher le résultat de la division décimale ?

# C1 Introduction au langage C

## 5. Portée des variables, conversion de types

### Exercice

Prévoir (et éventuellement observer) le résultat du programme suivant, expliquer.

```
1  #include <stdio.h>
2
3  void incremente(int n)
4  {
5      n = n + 1;
6  }
7
8  int main()
9  {
10     int n = 42;
11     incremente(n);
12     printf("n = %d\n", n);
13 }
```

### Définition

Un **comportement indéfini** (en anglais **undefined behavior** souvent abrégé en **UB**) est le résultat d'une suite d'instructions dont le résultat n'est pas spécifié par la norme du langage C.

Par conséquent, le résultat est alors ***totalement imprévisible*** et peut varier d'un compilateur ou d'un ordinateur à l'autre et même changer d'une exécution à l'autre sur un même ordinateur.

### Définition

Un **comportement indéfini** (en anglais **undefined behavior** souvent abrégé en **UB**) est le résultat d'une suite d'instructions dont le résultat n'est pas spécifié par la norme du langage C.

Par conséquent, le résultat est alors ***totalelement imprévisible*** et peut varier d'un compilateur ou d'un ordinateur à l'autre et même changer d'une exécution à l'autre sur un même ordinateur.

### Conséquences

Un comportement indéfini peut avoir des conséquences graves et doit donc être évité à tout prix. :

- Le programme peut planter (se terminer brutalement)
- Le programme peut produire un résultat incorrect
- Le programme peut fonctionner correctement mais de manière aléatoire, c'est à dire que le résultat peut varier d'une exécution à l'autre.

### Exemples de comportements indéfinis

- Accéder à une variable non initialisée.

```
1  # include <stdio.h>
2
3  int main()
4  {
5      int x;
6      printf("x=%d\n",x);
7  }
```



### Exemples de comportements indéfinis

- Accéder à une variable non initialisée.

```
1  # include <stdio.h>
2
3  int main()
4  {
5      int x;
6      printf("x=%d\n",x);
7  }
```

- Effectuer une division *entière* par zéro.

```
1  int main()
2  {
3      int x = 0;
4      int y = 42 / x;
5  }
```

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

- `-Wall` affichage de tous les *warning*

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

- `-Wall` affichage de tous les *warning*
- `-Wextra` affichage de *warning* supplémentaires

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

- `-Wall` affichage de tous les *warning*
- `-Wextra` affichage de *warning* supplémentaires
- on pourra éventuellement ajouter `-Wconversion` pour signaler les problèmes éventuels de conversion implicite

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

- `-Wall` affichage de tous les *warning*
- `-Wextra` affichage de *warning* supplémentaires
- on pourra éventuellement ajouter `-Wconversion` pour signaler les problèmes éventuels de conversion implicite

D'autre part, il est préférable de spécifier un nom pour l'exécutable produit grâce à l'option `-o`

# C1 Introduction au langage C

## 6. Comportement indéfini

### Remarques

Afin de repérer *dès la compilation* le maximum de problèmes potentiels, il est **très fortement recommandé** de toujours utiliser `gcc` avec les options :

- `-Wall` affichage de tous les *warning*
- `-Wextra` affichage de *warning* supplémentaires
- on pourra éventuellement ajouter `-Wconversion` pour signaler les problèmes éventuels de conversion implicite

D'autre part, il est préférable de spécifier un nom pour l'exécutable produit grâce à l'option `-o`

### Exemple

Pour compiler le programme `exemple.c`, la ligne de compilation devrait donc être :

```
gcc exemple.c -o exemple.exe -Wall -Wextra
```

### Conditionnelle

- `if (condition) { instruction }`



### Conditionnelle

- `if (condition) { instruction }`
- `if (condition) { instruction } else { instruction }`

### Conditionnelle

- `if (condition) { instruction }`
- `if (condition) { instruction } else { instruction }`

⚠ Pas de ; après la condition !

### Exemple

Ecrire une fonction `compare` en C, prenant comme paramètre deux entiers `a` et `b` et renvoyant `-1` si `a < b`, `0` si `a = b` et `1` sinon.

### Exemples

Ecrire les instructions conditionnelles suivantes (les variables utilisées sont supposées déjà déclarées) :

- 1 Affiche "Ok" si `a` est positif.

### Exemples

Ecrire les instructions conditionnelles suivantes (les variables utilisées sont supposées déjà déclarées) :

- 1 Affiche "Ok" si `a` est positif.
- 2 Affecte `nb` à 2 si `d` est strictement positif, 1 si `d` est nul et 0 sinon.

### Exemples

Ecrire les instructions conditionnelles suivantes (les variables utilisées sont supposées déjà déclarées) :

- 1 Affiche "Ok" si `a` est positif.
- 2 Affecte `nb` à 2 si `d` est strictement positif, 1 si `d` est nul et 0 sinon.
- 3 Affecte `e` à 1 si `a` et `b` ont la même parité et 0 sinon.

### Boucles

- `for (init; fin; increment) { instruction }`  
permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle.

### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

On tolère `i++` pour l'incréméntation, on recommande fortement de ne *pas* utiliser cet opérateur dans un autre contexte.



### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

On tolère `i++` pour l'incrémentation, on recommande fortement de ne *pas* utiliser cet opérateur dans un autre contexte.

- `while (condition) { instruction }`

permet de répéter le bloc d'instruction tant que la condition est vérifiée.

### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

On tolère `i++` pour l'incréméntation, on recommande fortement de ne *pas* utiliser cet opérateur dans un autre contexte.

- `while (condition) { instruction }`  
permet de répéter le bloc d'instruction tant que la condition est vérifiée.
- Lorsqu'une boucle se trouve dans le corps d'une fonction, une instruction `return` a pour effet de quitter immédiatement cette boucle (et le corps de la fonction) et de revenir au point d'appel de la fonction.

### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

On tolère `i++` pour l'incrémentation, on recommande fortement de ne *pas* utiliser cet opérateur dans un autre contexte.

- `while (condition) { instruction }`  
permet de répéter le bloc d'instruction tant que la condition est vérifiée.
- Lorsqu'une boucle se trouve dans le corps d'une fonction, une instruction `return` a pour effet de quitter immédiatement cette boucle (et le corps de la fonction) et de revenir au point d'appel de la fonction.
- Une boucle peut être interrompue avec l'instruction `break`.

### Boucles

- `for (init; fin; increment) { instruction }`

permet de répéter le bloc d'instruction pour chaque valeur prise par la variable de boucle. Généralement utilisé sous la forme :

```
for (int i=0; i<n; i=i+1) { ... }
```

On tolère `i++` pour l'incrémentation, on recommande fortement de ne *pas* utiliser cet opérateur dans un autre contexte.

- `while (condition) { instruction }`  
permet de répéter le bloc d'instruction tant que la condition est vérifiée.
- Lorsqu'une boucle se trouve dans le corps d'une fonction, une instruction `return` a pour effet de quitter immédiatement cette boucle (et le corps de la fonction) et de revenir au point d'appel de la fonction.
- Une boucle peut être interrompue avec l'instruction `break`.
- ⚠ Pas de `;` après la condition.

### Exemple

Le type `char` correspond en fait à une valeur entière, les caractères imprimables vont de 32 (l'espace) à 127 (DEL). Sachant que l'affichage d'un caractère avec `printf` se fait à l'aide de `%c`

### Exemple

Le type `char` correspond en fait à une valeur entière, les caractères imprimables vont de 32 (l'espace) à 127 (DEL). Sachant que l'affichage d'un caractère avec `printf` se fait à l'aide de `%c`

- Ecrire une boucle `for` permettant d'afficher ces caractères.

### Exemple

Le type `char` correspond en fait à une valeur entière, les caractères imprimables vont de 32 (l'espace) à 127 (DEL). Sachant que l'affichage d'un caractère avec `printf` se fait à l'aide de `%c`

- Ecrire une boucle `for` permettant d'afficher ces caractères.
- Faire de même avec une boucle `while`.

### Exercices

Ecrire une boucle permettant :

- 1 d'afficher les 10 premiers multiples de 42.



### Exercices

Ecrire une boucle permettant :

- 1 d'afficher les 10 premiers multiples de 42.
- 2 d'afficher les entiers de 10 à 1 (dans cet ordre).

### Exercices

Ecrire une boucle permettant :

- 1 d'afficher les 10 premiers multiples de 42.
- 2 d'afficher les entiers de 10 à 1 (dans cet ordre).
- 3 de calculer la somme des entiers impairs de 1 à 999.

### Exercices

Ecrire une boucle permettant :

- 1 d'afficher les 10 premiers multiples de 42.
- 2 d'afficher les entiers de 10 à 1 (dans cet ordre).
- 3 de calculer la somme des entiers impairs de 1 à 999.
- 4 de déterminer le plus petit entier  $n$  tel que  $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} > 7$ .

### Tableaux

! *La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.*

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

! La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.

```
bool est_premier[1000]; //un tableau de 1000 booléens
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

! La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.  
`bool est_premier[1000]; //un tableau de 1000 booléens`
- On peut initialiser le tableau en donnant une liste de valeurs entre accolades.

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

! La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.  
`bool est_premier[1000]; //un tableau de 1000 booléens`
- On peut initialiser le tableau en donnant une liste de valeurs entre accolades.  
`double notes[4]={5.5, 12.0, 13.5, 7.0}; //un tableau de 4 flottants`

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

! La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.  
`bool est_premier[1000]; //un tableau de 1000 booléens`
- On peut initialiser le tableau en donnant une liste de valeurs entre accolades.  
`double notes[4]={5.5, 12.0, 13.5, 7.0}; //un tableau de 4 flottants`
- Les éléments sont numérotés à partir de 0



# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

❗ *La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.*

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.  
`bool est_premier[1000]; //un tableau de 1000 booléens`
- On peut initialiser le tableau en donnant une liste de valeurs entre accolades.  
`double notes[4]={5.5, 12.0, 13.5, 7.0}; //un tableau de 4 flottants`
- Les éléments sont numérotés à partir de 0
- On accède à un élément en donnant son numéro (son indice) entre crochet.

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Tableaux

! La notion de tableau en C est intimement liée à celle de pointeur. Ces derniers seront abordés plus tard, aussi on fait ici une présentation élémentaire des tableaux, en particulier, on s'interdit pour le moment d'écrire des fonctions qui renvoient un tableau.

- Un tableau se déclare en donnant sa longueur et le type de ses éléments.  
`bool est_premier[1000]; //un tableau de 1000 booléens`
- On peut initialiser le tableau en donnant une liste de valeurs entre accolades.  
`double notes[4]={5.5, 12.0, 13.5, 7.0}; //un tableau de 4 flottants`
- Les éléments sont numérotés à partir de 0
- On accède à un élément en donnant son numéro (son indice) entre crochet.  
`est_premier[0]; //Le premier élément du tableau est_premier`

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

 Quelques points de vigilance !

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### ⚠ Quelques points de vigilance !

- Un accès en dehors des bornes du tableau est un **comportement indéfini**

```
1  int tab[10];  
2  int x = tab[10] + 42; //UB !
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### ⚠ Quelques points de vigilance !

- Un accès en dehors des bornes du tableau est un **comportement indéfini**

```
1      int tab[10];  
2      int x = tab[10] + 42; //UB !
```

- On ne peut pas directement affecter un tableau.

```
1      int tab1[4] = {17, 11, 9, 4};  
2      int tab2 = tab1; // Produit un Warning !
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### ⚠ Quelques points de vigilance !

- Un accès en dehors des bornes du tableau est un **comportement indéfini**

```
1  int tab[10];  
2  int x = tab[10] + 42; //UB !
```

- On ne peut pas directement affecter un tableau.

```
1  int tab1[4] = {17, 11, 9, 4};  
2  int tab2 = tab1; // Produit un Warning !
```

- Une fonction recevant en argument un tableau ne peut pas en déterminer la taille, par conséquent elle doit également recevoir la taille du tableau en argument.

```
1  int t[5] = {4, 2, 6, 1, 8};  
2  int s = somme(t); // Impossible de déterminer la taille de t  
↪ dans la fonction somme
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

Ecrire une fonction `croissant` qui prend un argument un tableau et sa taille et renvoie `true` si le tableau est trié et `false` sinon.

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

Ecrire une fonction `croissant` qui prend un argument un tableau et sa taille et renvoie `true` si le tableau est trié et `false` sinon.

```
1  bool croissant(int tableau[], int taille)
2  {
3      for (int i = 0; i < taille - 1; i = i + 1)
4      {
5          if (tableau[i] > tableau[i + 1])
6          {
7              return false;
8          }
9      }
10     return true;
11 }
```



# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

Ecrire une fonction `echange` qui prend un argument un tableau et deux indices `i` et `j` ne renvoie rien et échange les éléments d'indice `i` et `j` de ce tableau.

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

Ecrire une fonction `echange` qui prend un argument un tableau et deux indices `i` et `j` ne renvoie rien et échange les éléments d'indice `i` et `j` de ce tableau.

```
1 void echange(int tableau[], int i, int j)
2 {
3     int temp = tableau[i];
4     tableau[i] = tableau[j];
5     tableau[j] = temp;
6 }
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

Ecrire une fonction `echange` qui prend un argument un tableau et deux indices `i` et `j` ne renvoie rien et échange les éléments d'indice `i` et `j` de ce tableau.

```
1 void echange(int tableau[], int i, int j)
2 {
3     int temp = tableau[i];
4     tableau[i] = tableau[j];
5     tableau[j] = temp;
6 }
```

Mais .... en C, les paramètres sont passés par valeur non ?

### Exercices

- 1 Déclarer un tableau tab de 20 entiers.

### Exercices

- 1 Déclarer un tableau `tab` de 20 entiers.
- 2 Déclarer un tableau `test` de 5 booléens initialisé aux valeurs `false`, `true`, `false`, `true`, `false`.

### Exercices

- 1 Déclarer un tableau `tab` de 20 entiers.
- 2 Déclarer un tableau `test` de 5 booléens initialisé aux valeurs `false`, `true`, `false`, `true`, `false`.
- 3 Déclarer un tableau de 100 entiers, écrire une boucle permettant d'initialiser `t[i]` à la valeur  $2*i$ .

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exercices

- 1 Déclarer un tableau `tab` de 20 entiers.
- 2 Déclarer un tableau `test` de 5 booléens initialisé aux valeurs `false`, `true`, `false`, `true`, `false`.
- 3 Déclarer un tableau de 100 entiers, écrire une boucle permettant d'initialiser `t[i]` à la valeur `2*i`.
- 4 Commenter le programme suivant :

```
1  int main()  
2  {  
3      int tab[3];  
4      printf("Valeur à l'indice 0 : %d\n", tab[0]);  
5      printf("Valeur à l'indice 3 : %d\n", tab[3]);  
6  }
```

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets `"`) sont des tableaux de caractères (type `char []`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.



# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets ") sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.
- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets `"`) sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.
- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

- Le module `string.h` fournit des fonctions usuelles de manipulation de caractères, notamment :

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets ") sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.

- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

- Le module `string.h` fournit des fonctions usuelles de manipulation de caractères, notamment :
  - `strlen` : renvoie la longueur de la chaîne de caractères

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets ") sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.
- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

- Le module `string.h` fournit des fonctions usuelles de manipulation de caractères, notamment :
  - `strlen` : renvoie la longueur de la chaîne de caractères
  - `strcpy` : copie une chaîne de caractères

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets `"`) sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.

- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

- Le module `string.h` fournit des fonctions usuelles de manipulation de caractères, notamment :
  - `strlen` : renvoie la longueur de la chaîne de caractères
  - `strcpy` : copie une chaîne de caractères
  - `strcat` : concaténation de chaînes de caractères

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Chaînes de caractères

- En C, les chaînes de caractères (notées entre guillemets ") sont des tableaux de caractères (type `char[]`) dont le dernier élément est le caractère spécial `'\0'` qui marque la fin de la chaîne.

- Par exemple `char exemple[] = "Hello !";` crée le tableau :

H	e	l	l	o		!	\0
0	1	2	3	4	5	6	7

- Le module `string.h` fournit des fonctions usuelles de manipulation de caractères, notamment :
  - `strlen` : renvoie la longueur de la chaîne de caractères
  - `strcpy` : copie une chaîne de caractères
  - `strcat` : concaténation de chaînes de caractères
- Contrairement à un tableau « classique », on peut donc connaître la longueur d'une chaîne de caractères, grâce à la présence du caractère sentinelle `'\0'` qui en indique la fin.

### Exemple

- 1 Quel affichage est produit par le programme suivant ?

# C1 Introduction au langage C

## 8. Tableaux à une dimension, chaînes de caractères

### Exemple

❶ Quel affichage est produit par le programme suivant ?

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char test[] = "langage c";
7      test[0] = 'L';
8      test[8] = 'C';
9      printf("%s \n",test);
10     printf("Longueur = %ld\n",strlen(test));
11 }
```

❷ Quel affichage est produit en remplaçant la ligne 8 par `test[3]='\0';` ?



# C1 Introduction au langage C

## 9. Tableaux multidimensionnels

### Définition

On peut aussi déclarer en C des tableaux à plusieurs dimensions, par exemple :

```
1  bool ex1[10][10] = {false}; // une matrice 10x10 de  
    ↪ booléens
```

L'accès à un élément se fait alors en donnant successivement les deux indices entre crochets :

```
1  ex1[3][5] = true;
```

Ce qui a été vu pour les tableaux à une dimension reste vrai (numérotation à partir de 0 et les accès hors bornes sont UB).

⚠ Quand un tableau multidimensionnel est passé à une fonction, toutes les dimensions sauf (éventuellement) la *première* doivent être figurées ! : Par exemple, la signature de la fonction suivante n'est *pas* correcte :

```
void affiche(bool m[][], int l, int c)
```

On devrait écrire :

```
void affiche(bool m[][10], int l, int c)
```

### Exemple

Ecrire en C une fonction de signature `void identite(int n)` qui ne prend pas d'argument et affiche la trice identité d'ordre  $n$  (c'est à dire qu'elle contient des 1 sur sa diagonale principale et des 0 partout ailleurs). On déclarera un tableau à deux dimensions dans la fonction et on le remplira correctement avant d'effectuer son affichage.

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un `spécificateur de format` (comme `printf`) qui permet de préciser le type de la variable attendue.

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un `spécificateur de format` (comme `printf`) qui permet de préciser le type de la variable attendue.
- On fera précéder la variable qui reçoit la valeur saisie au clavier du caractère `&`

# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un `spécificateur de format` (comme `printf`) qui permet de préciser le type de la variable attendue.
- On fera précéder la variable qui reçoit la valeur saisie au clavier du caractère `&`. Ce point sera expliqué plus loin dans le cours.

### Exemple

# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un `spécificateur de format` (comme `printf`) qui permet de préciser le type de la variable attendue.
- On fera précéder la variable qui reçoit la valeur saisie au clavier du caractère `&`. Ce point sera expliqué plus loin dans le cours.
- Cette fonction renvoie le nombre de valeurs correctement lues.

### Exemple

# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un **spécificateur de format** (comme `printf`) qui permet de préciser le type de la variable attendue.
- On fera précéder la variable qui reçoit la valeur saisie au clavier du caractère `&`. Ce point sera expliqué plus loin dans le cours.
- Cette fonction renvoie le nombre de valeurs correctement lues.

### Exemple

- Ecrire un programme qui demande à l'utilisateur de saisir au clavier deux entiers a et b puis affiche leur somme.



# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Fonction scanf

- La fonction `scanf` permet la saisie de valeurs des variables depuis le clavier.
- Elle prend en argument un **spécificateur de format** (comme `printf`) qui permet de préciser le type de la variable attendue.
- On fera précéder la variable qui reçoit la valeur saisie au clavier du caractère `&`. Ce point sera expliqué plus loin dans le cours.
- Cette fonction renvoie le nombre de valeurs correctement lues.

### Exemple

- Ecrire un programme qui demande à l'utilisateur de saisir au clavier deux entiers a et b puis affiche leur somme.
- Modifier ce programme pour que les valeurs saisies soient des flottants.

# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Correction

```
1  #include <stdio.h>
2
3  int somme(int n, int m){
4      return n+m;}
5
6  int main(){
7      int n,m,s;
8      printf("a=");
9      scanf("%d",&n);
10     printf("b=");
11     scanf("%d",&m);
12     s = somme(n,m);
13     printf("a+b=%d\n",s);
14     return 0;
15 }
```

# C1 Introduction au langage C

## 10. Saisie de valeurs au clavier

### Correction

```
1  #include <stdio.h>
2
3  float somme(float n, float m){
4      return n+m;}
5
6  int main(){
7      float n,m,s;
8      printf("a=");
9      scanf("%f",&n);
10     printf("b=");
11     scanf("%f",&m);
12     s = somme(n,m);
13     printf("a+b=%f\n",s);
14     return 0;
15 }
```