

### Bref historique

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

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thomson 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)

### Bref historique

- 1972 : début du développement du langage C par Dennis Ritchie et Ken Thomson 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. Stroustrup, 1983), C# (Microsoft, 2000), Go (Google, 2007), Rust (Mozilla, 2010)

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.
- Le langage C est **statiquement typé** c'est à dire qu'une variable appartient à un type défini à la déclaration et durant toute sa durée de vie.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.
- Le langage C est **statiquement typé** c'est à dire qu'une variable appartient à un type défini à la déclaration et durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.



# C1 Introduction au langage C

## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.
- Le langage C est **statiquement typé** c'est à dire qu'une variable appartient à un type défini à la déclaration et durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- **⚠** Le standard précise un certain nombre de **comportements indéfinis**, c'est à dire de programmes dont le résultat est imprévisible.

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.
- Le langage C est **statiquement typé** c'est à dire qu'une variable appartient à un type défini à la déclaration et durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- **⚠** Le standard précise un certain nombre de **comportements indéfinis**, c'est à dire de programmes dont le résultat est imprévisible.
- Plus **proche de la machine** que bien d'autres langages de haut niveau, ce qui induit une certaine efficacité.

# C1 Introduction au langage C


## 2. Caractéristiques du C

### Quelques aspects du C

- Langage **impératif** : séquence d'instructions exécutées par l'ordinateur pour modifier l'état du programme. C n'est ni orienté objet, ni fonctionnel.
- Les variables sont **mutables** c'est à dire qu'elles peuvent changer de valeur pendant l'exécution.
- Le langage C est **statiquement typé** c'est à dire qu'une variable appartient à un type défini à la déclaration et durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- ⚠ Le standard précise un certain nombre de **comportements indéfinis**, c'est à dire de programmes dont le résultat est imprévisible.
- Plus **proche de la machine** que bien d'autres langages de haut niveau, ce qui induit une certaine efficacité.
- Souvent utilisé pour le développement de systèmes d'exploitation, de pilotes de périphériques, de logiciels embarqués.


### Compilation

Le langage C est **compilé** :

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



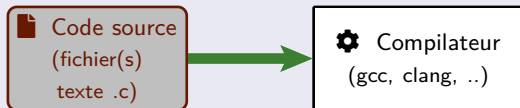
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.

# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



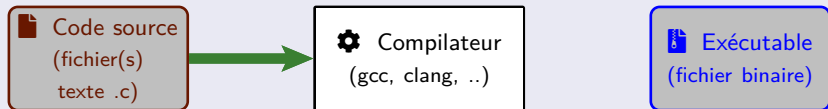
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien

# 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 avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien

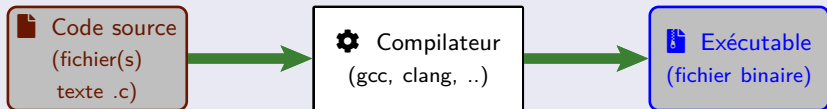


# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



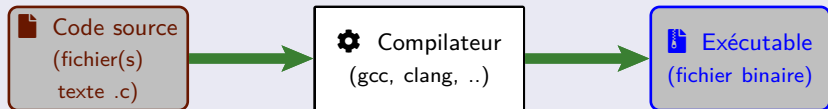
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien
- 3 Une compilation sans erreur (mais éventuellement des *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 avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien
- 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 `gcc hello.c`, l'exécutable produit s'appelle par défaut `a.out`, on peut modifier ce nom avec l'option `-o`. Par exemple : `gcc -o hello.exe 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  }
```

Appel aux fonctions **standard** d'entrées et de sorties (**i**nput et **o**utput) de la libc.

# 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 une fonction `main` par laquelle l'exécution du programme commence.

### Programme minimal

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

Avant le nom d'une fonction on trouve le type de variable qu'elle renvoie (ici `int`) et après entre parenthèses, les arguments éventuels de la fonction (ici aucun).

### 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 } ). Les instructions doivent se terminer par un point virgule ;. 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.



# 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, qui indique traditionnellement que le programme se termine sans erreurs.

### Exemple de boucle

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

Déclaration de la variable `somme` de type `int` et initialisation à zéro. A noter qu'on peut déclarer une variable sans l'initialiser.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

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

Une variable dont la valeur ne sera pas modifiée peut être déclaré avec `const`.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

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

Une variable dont la valeur ne sera pas modifiée peut être déclaré avec `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      int somme = 0;
5      const int nmax = 100;
6      for (int i=1;i<=nmax;i=i+1)
7          {somme = somme + i;}
8      printf("1+2+...+100 = %d\n",somme);
9      return 0;}
```

On remarque que la boucle `for` est de la forme `for (init; fin; incr)`. Les opérateurs de comparaison en C sont `==`, `!=`, `<`, `>`, `<=` et `>=`.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple de boucle

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

On veut afficher un `int` dans la réponse, on utilise `%d` dans `printf` à l'emplacement souhaité.

# C1 Introduction au langage C

## 3. Exemples de programmes

### Exemple d'instruction conditionnelle

```
1  #include <stdio.h>
2  // S(n+1) = S(n)/2 si n est pair et 3S(n)+1 sinon
3  int syracuse(int n)    {
4      if (n%2 == 0)
5          {return n/2; }
6      else
7          {return 3*n+1; }}
8
9  int main()
10 { printf("Syracuse 25 = %d \n",syracuse(25));
11     return 0;}
```

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

### Exemple d'instruction conditionnelle

```
1  #include <stdio.h>
2  //  $S(n+1) = S(n)/2$  si  $n$  est pair et  $3S(n)+1$  sinon
3  int syracuse(int n)    {
4      if (n%2 == 0)
5          {return n/2; }
6      else
7          {return 3*n+1; }}
8
9  int main()
10 { printf("Syracuse 25 = %d \n",syracuse(25));
11     return 0;}
```

Définition d'une fonction `syracuse` qui prend comme paramètre un entier et renvoie un entier. C'est la **signature** de la fonction.

❗ En C, les paramètres sont passés par **valeur**.



### Exemple d'instruction conditionnelle

```
1  #include <stdio.h>
2  //  $S(n+1) = S(n)/2$  si  $n$  est pair et  $3S(n)+1$  sinon
3  int syracuse(int n)    {
4      if (n%2 == 0)
5          {return n/2; }
6      else
7          {return 3*n+1; }}
8
9  int main()
10 { printf("Syracuse 25 = %d \n",syracuse(25));
11     return 0;}
```

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.

### Types de base

Type	Opérations	Commentaires

### Types de base

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

### Types de base

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


### 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>	


### 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.

### 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.

### Types de base


Type	Opérations	Commentaires
<code>int</code> et <code>unsigned int</code>  <code>int</code> $N\_t$ et <code>uint</code> $N\_t$	<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 $64$ ).



# 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> , valeurs <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Evaluation « 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> , valeurs <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Evaluation « 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> , valeurs <code>true</code> ou <code>false</code> .	<code>  </code> , <code>&amp;&amp;</code> , <code>!</code>	Booléens accessibles dans <code>stdbool.h</code> . Evaluation « 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>

Pour indiquer l'absence de type, notamment pour les fonctions ne renvoyant rien (par exemple une fonction d'affichage) on utilise `void`.

### 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

- 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`.

### 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 Discuter l'effet de l'instruction conditionnelle `if (b>0 && a/b < 1)` suivant les valeurs des variables entières `a` et `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.
- 3 On suppose déjà déclarées deux variables booléennes `x` et `y`, écrire une expression booléenne correspondant à `x xor y`.
- 4 Discuter l'effet de l'instruction conditionnelle `if (b>0 && a/b < 1)` suivant les valeurs des variables entières `a` et `b`.
- 5 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 Discuter l'effet de l'instruction conditionnelle `if (b>0 && a/b < 1)` suivant les valeurs des variables entières `a` et `b`.
- 5 La déclaration de variable suivante est-elle correcte ? `char c = "a";`
- 6 Quelle est selon vous la cause du message : *warning : 'return' with a value, in function returning void* lors d'une compilation ?

### 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>char []</code> (chaîne de caractères)	<code>%s</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>

### 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  int maxn = 10000;
4
5  double harmo(int n){
6      double somme = 0;
7      for (int i=1;i<=n;i=i+1)
8          {somme = somme + 1.0/i;}
9      return somme;}
10
11 int main(){
12     double s = harmo(maxn);
13     printf("Somme = %f\n",s);
14     return 0;}
```

### Exemples

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

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

- ② On définit une variable entière `i`, juste après la ligne 6, donner la portée de cette nouvelle variable. Le programme fonctionne-t-il encore correctement ?

### Exemples

`maxn` est une variable globale

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

### Exemples

`n` est un paramètre de la fonction `harmo`

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

### Exemples

`somme` est déclarée dans la fonction `harmo`

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

### Exemples

`i` est locale à la boucle

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

### Exemples

s est locale au main

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

### 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 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

## 4. Types de base en C

### Exemple

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

### Exemple

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

- Quel est le résultat de ce programme ? Pourquoi ?

### Exemple

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

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

### 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 }
```

### 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
- `-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
- `-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

## 4. Types de base en C

### 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
- `-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.

### Correction de l'exemple

```
1 int compare(int a, int b)
2 {
3     if (a < b)
4     {
5         return -1;
6     }
7     else if (a == b)
8     {
9         return 0;
10    }
11    else
12        return 1;
13 }
```

### 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* utilisé 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'incrémement, 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'incrémement, 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'incrémement, on recommande fortement de ne *pas* utilisé 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'incrémement, 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`.

### Correction de l'exemple

- Avec une boucle `for`

```
1  #include <stdio.h>
2  int main() {
3      for (int i=32;i<128;i=i+1)
4          {printf("Code %d : %c \n",i,i);}}
```

### Correction de l'exemple

- Avec une boucle `for`

```
1  #include <stdio.h>
2  int main() {
3      for (int i=32;i<128;i=i+1)
4          {printf("Code %d : %c \n",i,i);}}
```

- Avec une boucle `while`

```
1  #include <stdio.h>
2  int main() {
3      int i = 32;
4      while (i<128) {
5          printf("Code %d : %c \n",i,i);
6          i = i + 1;}}
```

### 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

Écrire 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.

### 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
```

### 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.

### 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`

### 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

### 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.



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

### 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`
- ⚠ Un accès en dehors des bornes du tableau est un **comportement indéfini**
- On ne peut pas accéder à la taille d'un tableau en C. En conséquence, une fonction qui travaille sur un tableau doit aussi recevoir en argument la taille de ce dernier.

### 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.

### 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      for (int i=0; i<taille-1; i=i+1)  
3      {  
4          if (tableau[i]>tableau[i+1])  
5              {return false;}  
6      }  
7      return true;}
```

### Exemple

Écrire 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

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

### Exemple

Écrire 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

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

### Exemple

Écrire 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 initialisés aux valeurs `false`, `true`, `false`, `true`, `false`.

### Exercices

- 1 Déclarer un tableau `tab` de 20 entiers.
- 2 Déclarer un tableau `test` de 5 booléens initialisés 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  $2*i$ .

### Exercices

- 1 Déclarer un tableau `tab` de 20 entiers.
- 2 Déclarer un tableau `test` de 5 booléens initialisés 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

## 6. 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

## 6. 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

## 6. 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

## 6. 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

## 6. 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

## 6. 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

## 6. 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 est l'affichage produit par le programme suivant ?

### Exemple

- ① Quel est l'affichage 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 }
```

- ② Quel est l'affichage produit en remplaçant la ligne 8 par `test[3]='\0';` ?

### 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

## 7. 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



### 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

### 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.

### 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.

### 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 }
```

### 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 }
```