

### 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)

# C1 Introduction au langage C

## 1. Historique

### 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. Stroustup, 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 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 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 durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- Le standard précise un certain nombres de **comportements indéfinis**, c'est à dire de programmes dont le résultat est imprévisible.

# 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 durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- Le standard précise un certain nombres 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 durant toute sa durée de vie.
- Equipé d'une librairie standard : la **libc**.
- Le standard précise un certain nombres 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,

# 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** :



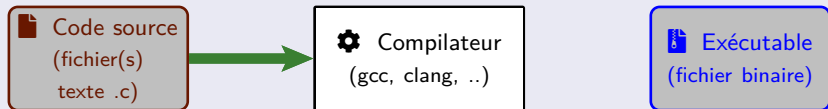
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien

# 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

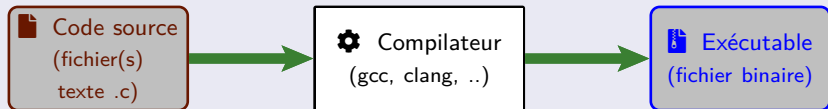


# C1 Introduction au langage C

## 2. Caractéristiques du C

### Compilation

Le langage C est **compilé** :



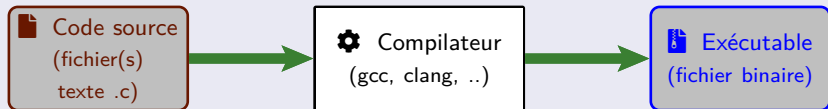
- 1 Les IDE comme VS Code signalent certaines erreurs dans le code.
- 2 La compilation peut produire des erreurs ou des avertissement (*warning*)  
La compilation se déroule en 4 étapes : préprocesseur, compilation, assemblage, éditions de lien
- 3 Une compilation sans erreur (mais éventuellement des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

# 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  }
```

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).

# 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 } ). 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.

# 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;}
```

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 `*/`

# 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;}
```

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**.



# 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;}
```

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

## 4. Définitions et types de base

### Types de base

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

# C1 Introduction au langage C

## 4. Définitions et types de base

### Types de base

| Type   | Opérations   | Commentaires   |
|--|--|--|
| <code>int</code> et <code>unsigned int</code><br><br><code>int<sub>N</sub>_t</code> et <code>uint<sub>N</sub>_t</code> | <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> | Entiers signés ou non signés codés sur un minimum de 16 bits.<br><br>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. Définitions et types de base

### Types de base

| Type   | Opérations   | Commentaires   |
|--|--|--|
| <code>int</code> et <code>unsigned int</code><br><br><code>int<sub>N</sub>_t</code> et <code>uint<sub>N</sub>_t</code> | <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> | Entiers signés ou non signés codés sur un minimum de 16 bits.<br><br>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>  </code> <code>&amp;&amp;</code> , <code>!</code>                           | Booléens accessibles dans <code>stdbool.h</code> . Évaluations paresseuses des expressions.  |
|  |  |  |

# C1 Introduction au langage C

## 4. Définitions et types de base

### Types de base

| Type   | Opérations  | Commentaires   |
|--|---|--|
| <code>int</code> et <code>unsigned int</code><br><br><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><br><br><code>++</code> , <code>--</code> , <code>+=</code> , <code>-=</code> | Entiers signés ou non signés codés sur un minimum de 16 bits.<br><br>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>  </code> <code>&amp;&amp;</code> , <code>!</code>  | Booléens accessibles dans <code>stdbool.h</code> . Évaluations paresseuses 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. Définitions et types de base

### Types de base

| Type   | Opérations  | Commentaires   |
|--|---|--|
| <code>int</code> et <code>unsigned int</code><br><br><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><br><code>++</code> , <code>--</code> , <code>+=</code> , <code>-=</code> | Entiers signés ou non signés codés sur un minimum de 16 bits.<br><br>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>  </code> <code>&amp;&amp;</code> , <code>!</code>  | Booléens accessibles dans <code>stdbool.h</code> . Evaluations paresseuses 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`.

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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>   | <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>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 :

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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.

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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.

### Remarques

Lorsque deux variables ont le même identifiant, c'est la variable ayant la plus petite portée qui est accessible.

# C1 Introduction au langage C

## 4. Définitions et types de base

### Exemples

Dans le programme suivant, donner les portées des variables `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;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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;}
```



# C1 Introduction au langage C

## 4. Définitions et types de base

### 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;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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. Définitions et types de base

### 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     printf("%d\n",(int)13.6 % 2);
11     return 0;}
```

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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     printf("%d\n",(int)13.6 % 2);
11     return 0;}
```

- Quel est le résultat de ce programme ? Pourquoi ?

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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     printf("%d\n",(int)13.6 % 2);
11     return 0;}
```

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

### Remarques

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

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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 fichier un nom pour l'exécutable produit grâce à l'option `-o`

# C1 Introduction au langage C

## 4. Définitions et types de base

### 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 fichier 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 -Wconversion
```

### Conditionnelle

- `if (condition) { instruction }`

### Exemple

### Conditionnelle

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

### Exemple

### Conditionnelle

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

### 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          {return -1;}
5      else if (a==b)
6          {return 0;}
7      else
8          return 1;
9  }
```



### Boucles

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

### Exemple

### Boucles

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

Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`

### Exemple

### Boucles

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

Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`

- `while (condition) { instruction }`

### Exemple

# C1 Introduction au langage C

## 5. Structures de contrôle

### Boucles

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

Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`

- `while (condition) { instruction }`
- Une boucle peut-être interrompue avec l'instruction `break`

### Exemple

# C1 Introduction au langage C

## 5. Structures de contrôle

### Boucles

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

Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`

- `while (condition) { instruction }`
- Une boucle peut-être interrompue avec l'instruction `break`

### 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`

# C1 Introduction au langage C

## 5. Structures de contrôle

### Boucles

- `for (init; fin; increment) { instruction }`  
Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`
- `while (condition) { instruction }`
- Une boucle peut-être interrompue avec l'instruction `break`

### 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.

# C1 Introduction au langage C

## 5. Structures de contrôle

### Boucles

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

Généralement utilisé sous la forme : `for (int i=0; i<n; i=i+1) { ... }`

- `while (condition) { instruction }`
- Une boucle peut-être interrompue avec l'instruction `break`

### 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);}}
```



# C1 Introduction au langage C

## 5. Structures de contrôle

### 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;}}
```

### Tableaux

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



Attention

# C1 Introduction au langage C

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

### Tableaux

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

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



Attention

# C1 Introduction au langage C

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

### Tableaux

- 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.



Attention

# C1 Introduction au langage C

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

### Tableaux

- 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*



Attention

# C1 Introduction au langage C

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

### Tableaux

- 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



### Attention

# C1 Introduction au langage C

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

### Tableaux

- 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.



### Attention

# C1 Introduction au langage C

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

### Tableaux

- 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`



### Attention



# C1 Introduction au langage C

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

### Tableaux

- 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`



### Attention

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

# C1 Introduction au langage C

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

### Tableaux

- 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`

### ! Attention

- Un accès en dehors des bornes du tableau est un **comportement indéfini**
- La gestion de la taille du tableau est de la *responsabilité du programmeur*. Il n'y a pas de fonctions permettant d'y accéder. En conséquence lorsqu'un tableau est passé en paramètre à une fonction on passe aussi sa taille.

### 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

## 6. 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      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;}
```

# C1 Introduction au langage C

## 6. 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

## 6. 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

## 6. 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 ?

### 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 |
|---|---|---|---|---|--|---|----|

# 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 |
|---|---|---|---|---|--|---|----|
- 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 |
|---|---|---|---|---|--|---|----|
- 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 |
|---|---|---|---|---|--|---|----|
- 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 |
|---|---|---|---|---|--|---|----|
- 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

### Exemple

Quel est l'affichage produit par le programme suivant ?

# C1 Introduction au langage C

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

### Exemple

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 }
```

### 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

# 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.
- 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

## 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.
- 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

## 7. 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

## 7. 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 }
```