Soutien informatique : langage C

Sergey Kirgizov

“When in doubt, use brute force”

– Ken Thompson

Aujourd'hui

- Notes historiques
- Littérature
- Hello world
- Structure
- Types
- Variables
- Affichage
- Bibliothèques
- Branchements
- Boucles
- Blocs
- Instructions imbriquées
- Fonctions
- Tableau, chaînes de caractères et pointeurs
- Hello world II
- Pour la suite

Notes historiques

Les premières versions d'Unix one été écrites en langage assembleur.

Les premières versions d'Unix one été écrites en langage assembleur.

Assembleur est un langage de niveau très bas. Il est assez difficile à utiliser.

Les premières versions d'Unix one été écrites en langage assembleur.

Assembleur est un langage de niveau très bas. Il est assez difficile à utiliser.

Donc, les auteurs ont décidé de créer un nouveau langage afin d'aider eux-mêmes à faire évaluer le système Unix.

Les premières versions d'Unix one été écrites en langage assembleur.

Assembleur est un langage de niveau très bas. Il est assez difficile à utiliser.

Donc, les auteurs ont décidé de créer un nouveau langage afin d'aider eux-mêmes à faire évaluer le système Unix.

Finalement...

Les premières versions d'Unix one été écrites en langage assembleur.

Assembleur est un langage de niveau très bas. Il est assez difficile à utiliser.

Donc, les auteurs ont décidé de créer un nouveau langage afin d'aider eux-mêmes à faire évaluer le système Unix.

Finalement...

Ils ont créé un outil qui a dépassé son objectif initial. Aujourd'hui, C est un des langages parmi les plus populaires.

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial."

– Wikipedia

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial."
– Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial."
– Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran
- ...

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran
- ...
- 1951 — Superplan, Heinz Rutishauser, ETH Zürich

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran
- ...
- 1951 — Superplan, Heinz Rutishauser, ETH Zürich
- 1948 – Plankalkül, Konrad Zuse

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

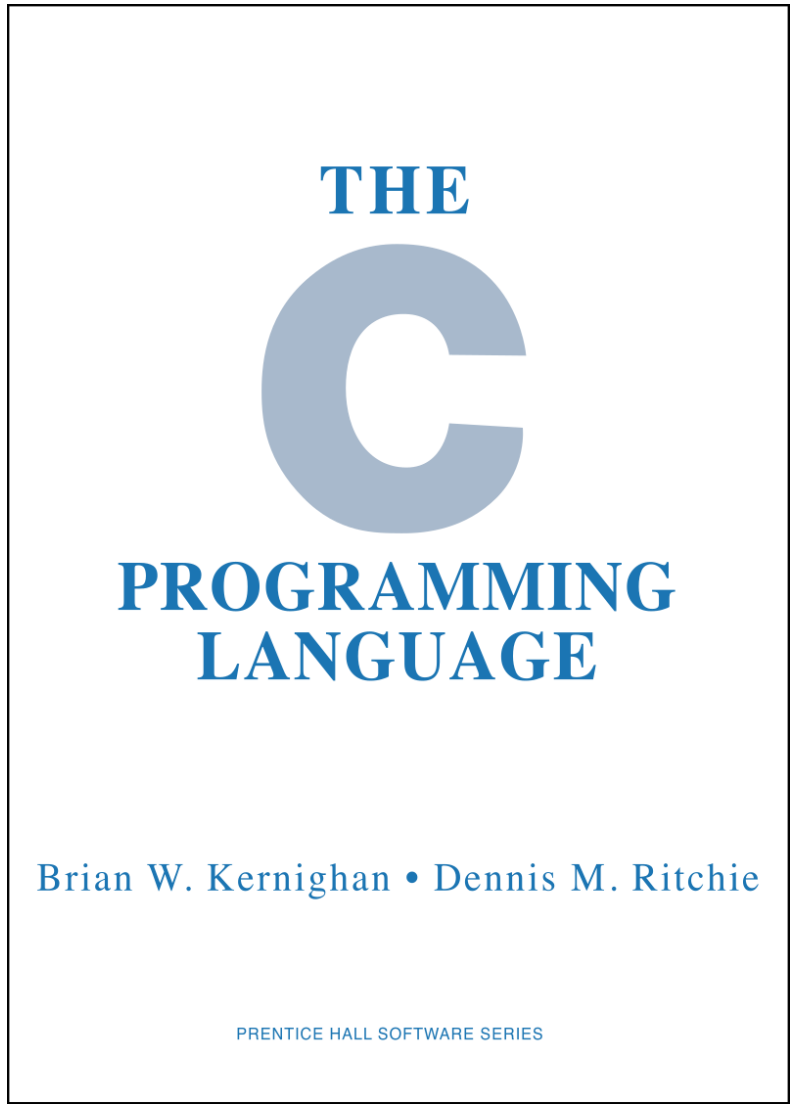
- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran
- ...
- 1951 — Superplan, Heinz Rutishauser, ETH Zürich
- 1948 – Plankalkül, Konrad Zuse
- 1879 – Begriffsschrift, Gottlob Frege

"C was created by Dennis Ritchie at Bell Labs in the early 1970s as an augmented version of Ken Thompson's B. Another Bell Labs employee, Brian Kernighan, had written the first C tutorial." – Wikipedia

- 1969 — B, Ken Thompson with Dennis Ritchie, Bell Labs
- 1967 — BCPL (Basic CPL), Martin Richards, University of Cambridge
- 1963 — CPL (Combined Programming Language), Mathematical Laboratory, University of Cambridge, Christopher Strachey, David Barron et al.
- ALGOL 60 (Algorithmic Language 1960)
- ALGOL 58, Fortran
- ...
- 1951 — Superplan, Heinz Rutishauser, ETH Zürich
- 1948 – Plankalkül, Konrad Zuse
- 1879 – Begriffsschrift, Gottlob Frege
- ...

Littérature

Un livre classique de Brian Kernighan et Dennis Ritchie.

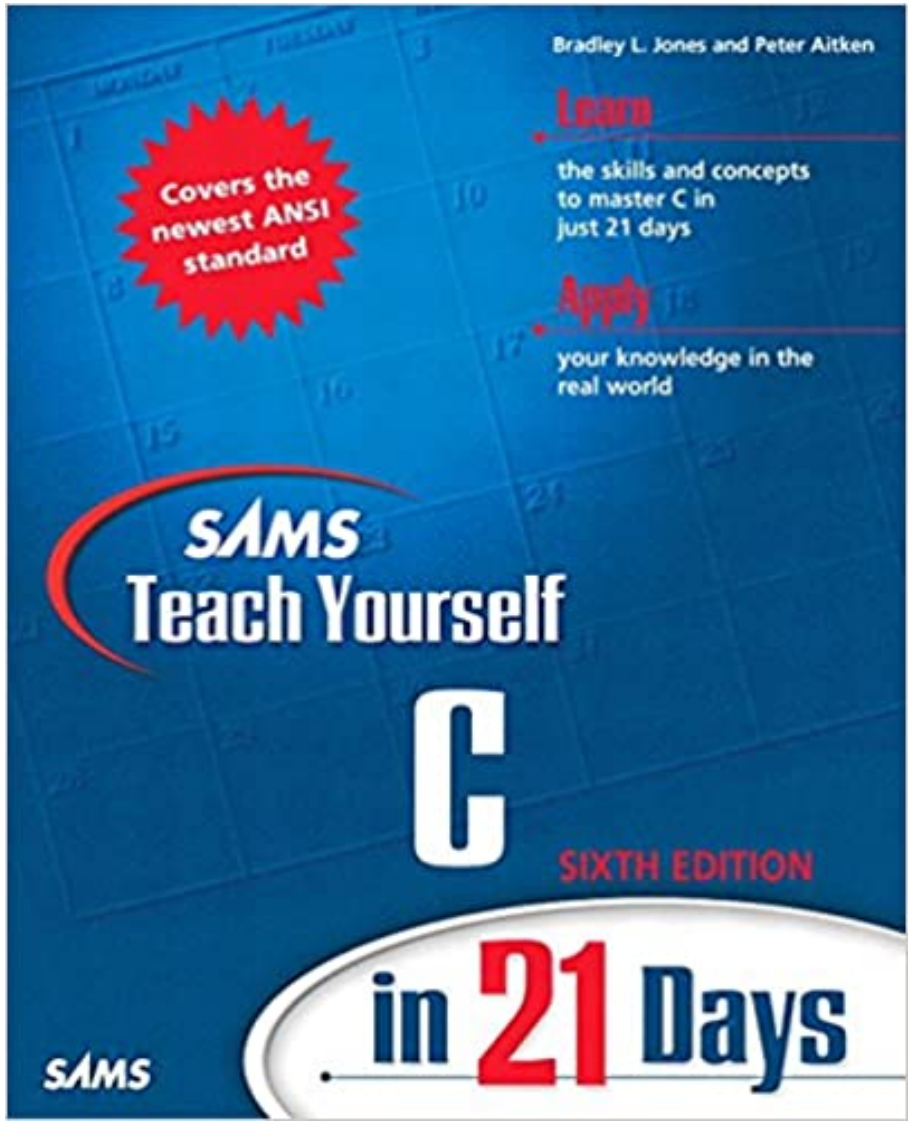


1978

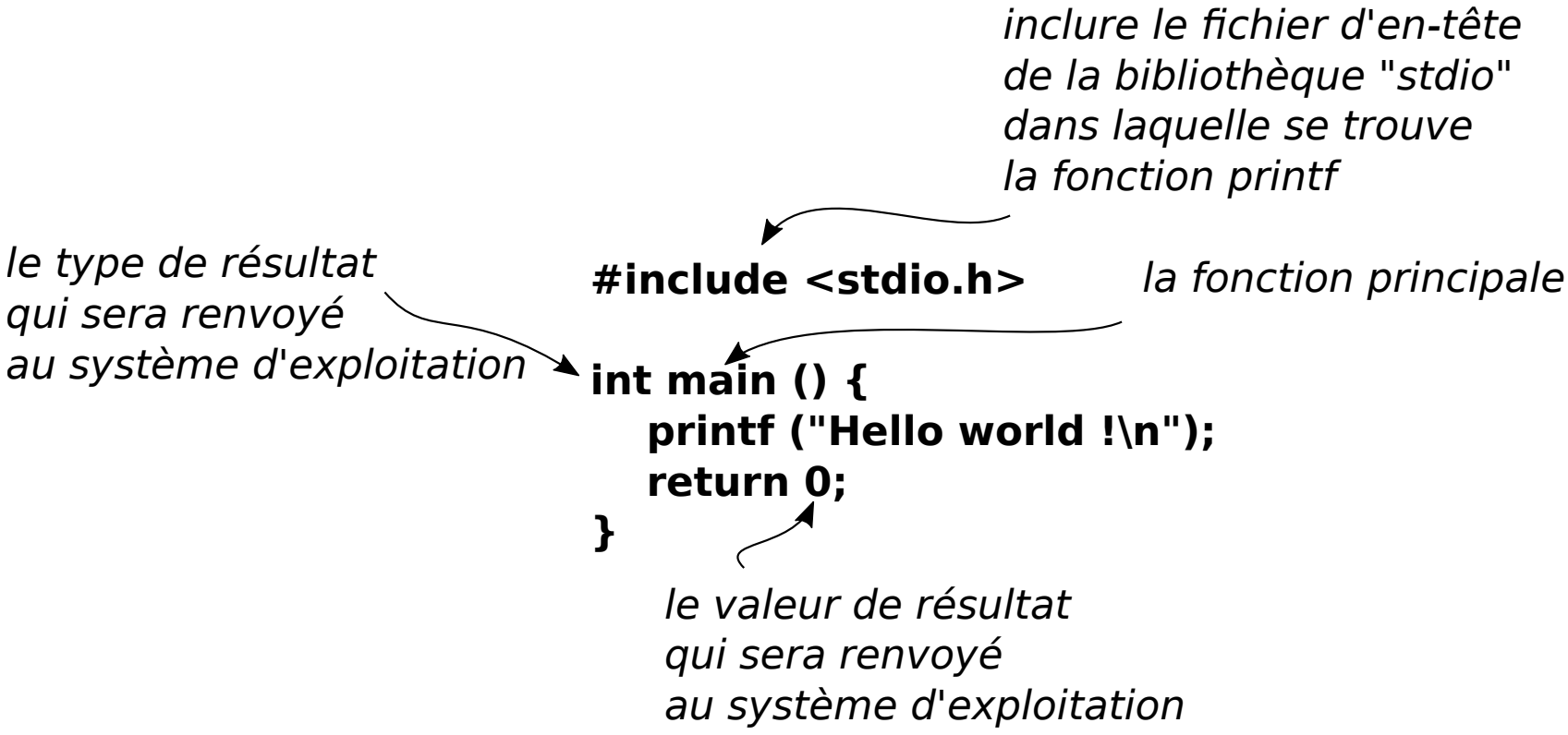
Littérature sur le net

- Cours d'Anne Canteaut
  `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/`
- Wikibook `https://en.wikibooks.org/wiki/C_Programming`
- Livre de Mike Banahan, Declan Brady et Mark Doran
  `https://publications.gbdirect.co.uk/c_book/`
- Standard ISO pour C
  `https://web.archive.org/web/20181230041359/http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf` **(534 pages)**

Un livre de Bradley L. Jones et Peter Aitken



Hello world



Comment tester ?

- Compilation : `gcc -Wall code-source.c -o nom-d-executable`
- Exécution : `./nom-d-executable`
- Voir le valeur de résultat : `echo $?`

Structure d'un programme C

```
DIRECTIVES AU PRÉPROCESSEUR

DÉCLARATIONS DE VARIABLES EXTERNES

FONCTIONS SECONDAIRES

int main () {
  DÉCLARATIONS DE VARIABLES INTERNES

  INSTRUCTIONS
}
```

Types

Types principaux

- `int` — nombre entier (4 octets)
- `char` — symbole ASCII (1 octet), exemple 'z'
- `float` — nombre en virgule flottante
- `double` — nombre en virgule flottante, double précision

On peut former les expressions avec les opérations suivates

- arithmétiques : `+`, `-`, `*`, `/`, `%` (modulo)
- groupement avec les parenthèses, par exemple (1+3)/(1+1)
- comparison : `==`, `!=`, `>`, `<`, `>=`, `<=`
- logiques : `!` (négation), `&&` (et), `||` (ou)
- bit à bit : `>>` (décalage à droite),
  `<<` (décalage à gauche),
  `&` (et), `|` (ou), `^` (xor), `~` (négation)
- conversion de types : `(NOM_DE_TYPE)EXPRESSION`

Conversion de types

Attension au conversion de types !

- 1/2 c'est zéro, car 1 et 2 ont le type int, ce type n'y pas de virgules.

Conversion de types

Attension au conversion de types !

- `1/2` c'est zéro, car 1 et 2 ont le type int, ce type n'y pas de virgules.

- `((float)1)/2` c'est 0.5, car `((float)1)` est un float et float/int = float

Conversion de types

Attension au conversion de types !

- 1/2 c'est zéro, car 1 et 2 ont le type int, ce type n'y pas de virgules.

- `((float)1)/2` c'est 0.5, car `((float)1)` est un float et float/int = float

Hiérarchie de types : char < int < float < double

char + int = int
char + float = float
int + double = double
etc...

Variables

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un “_”. Elle peut comporter des lettres, des chiffres et le caractère “_”.

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`
  ou bien
  `NOM_DE_TYPE NOM_DE_VARIABLE1, NOM_DE_VARIABLE2, ...;`

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`
  ou bien
  `NOM_DE_TYPE NOM_DE_VARIABLE1, NOM_DE_VARIABLE2, ...;`
  Exemple :

```
int a;
int a2;
int b_a_c, c, d;
```

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`
  ou bien
  `NOM_DE_TYPE NOM_DE_VARIABLE1, NOM_DE_VARIABLE2, ...;`
  Exemple :

```
int a;
int a2;
int b_a_c, c, d;
```

- Affectation :
  `NOM_DE_VARIABLE = EXPRESSION;`

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`
  ou bien
  `NOM_DE_TYPE NOM_DE_VARIABLE1, NOM_DE_VARIABLE2, ...;`
  Exemple :

```
int a;
int a2;
int b_a_c, c, d;
```

- Affectation :
  `NOM_DE_VARIABLE = EXPRESSION;`

```
v = 10 + 4;
```

Variables

Un nom de variable est une séquence de caractères. Elle doit commence par une lettre (majuscule ou minuscule) ou un "_". Elle peut comporter des lettres, des chiffres et le caractère "_".

- Déclaration :
  `NOM_DE_TYPE NOM_DE_VARIABLE;`
  ou bien
  `NOM_DE_TYPE NOM_DE_VARIABLE1, NOM_DE_VARIABLE2, ...;`
  Exemple :

```
int a;
int a2;
int b_a_c, c, d;
```

- Affectation :
  `NOM_DE_VARIABLE = EXPRESSION;`

```
v = 10 + 4;
```

- Déclaration-initialisation :

```
double a = 100;
char l = 'a';
```

Les mots-clefs réservés pour le langage

Un identificateur est un nom de variable ou de fonction. Il doit commence par une lettre (majuscule ou minuscule) ou un “_”. Il peut comporter des lettres, des chiffres et le caractère “_”.
Les mots suivants ne peuvent pas être utilisés comme identificateurs :

```
auto const double float int short struct unsigned
break continue else for long signed switch void case
default enum goto register sizeof typedef volatile
char do extern if return static union while
```

Exemple avec commentaires

```
/* Ceci est un commentaire
   sur deux lignes
   ou plus */
// Commentaire jusqu'à la fin de la ligne

#include <stdio.h>

int main () {
  int v;
  v = 10 + 4;
  v = v + v;
  v++;
  // v est égal à 29 maintenant
  return 0;
}
```

Astuces

```
    1 c'est "vrai" (true)
    0 c'est "faux" (false)
-----------------------------------------
    a++             ⇔  a = a + 1;
    a--             ⇔  a = a - 1;
    ++a             ⇔  a = a + 1;
    --a             ⇔  a = a - 1;

    i = a++;        ⇔   i = a;
                        a = a + 1;

    i = ++a;        ⇔  a = a + 1;
                       i = a;


    i += a;         ⇔   i = i + a;
    i -= a;         ⇔   i = i - a;
    i *= a;         ⇔   i = i * a;
    i /= a;         ⇔   i = i / a;
```

Affichage

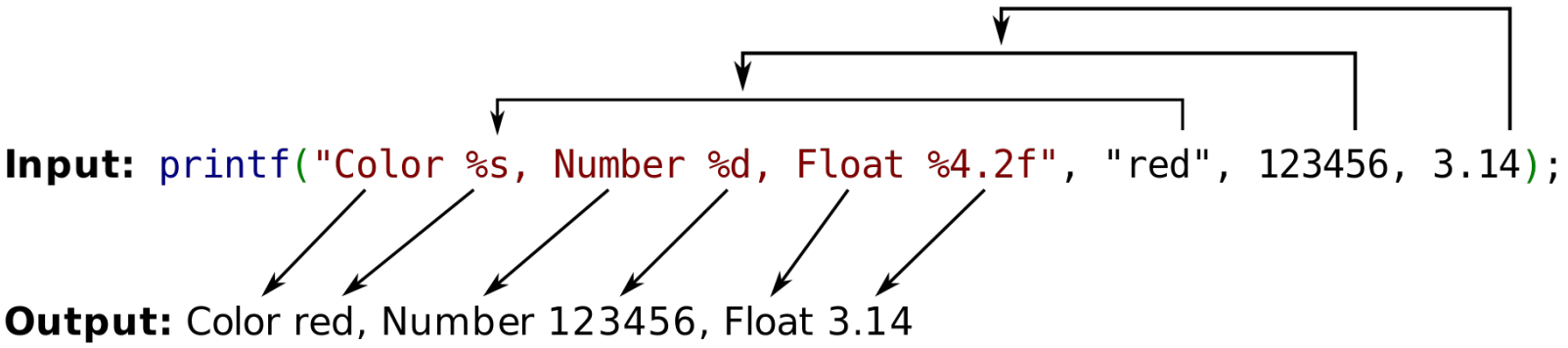
Fonction d'affichage : printf

Exemple :

```
char a = 'b';
printf ("ZZZZ %c %x %d ZZZZ \n", a, a, a);
// ça va afficher: ZZZZ b 61 97 ZZZZ
```

Type	Lettre de format
int	%d
float/double	%f
char	%c
string (char*)	%s
pointeur	%p
entier hexadécimal	%x

Documentation



- man 3 printf
- `https://fr.wikipedia.org/wiki/Printf`
- Pour la saisie de données formatées : la fonction `scanf`

Bibliothèques

Bibliothèques de fonctions

- C standard library, **libc**
- **FFmpeg**, traitement de flux audio ou vidéo
- **libjpeg**, gestion (ouverture, sauvegarde, gestion des marqueurs, etc.) des images au format JPEG
- **lubusb**, contrôler le transfert de données vers et depuis des périphériques USB
- **SDL**, pour créer des applications multimédias, les jeux vidéo, les démos graphiques, les émulateurs, etc
- **libastral**, interface pour les flux de données astrales
- etc, etc, etc

Opérations sur les fichiers

Fonctions fopen, putc, getc, fclose, fprintf, fscanf, ...

```
#include <stdio.h>

int main () {
   FILE* f;

   // ouvrir le fichier pour l'écriture (w)
   f = fopen("file.txt", "w");
   putc('A', f);
   fclose(f);

   return 0;
}
```

Branchements

if, la forme générale

```
if (CONDITION)
   INSTRUCTION
```

ou bien

```
if (CONDITION)
   INSTRUCTION
else
   INSTRUCTION
```

où `CONDITION` exprime une condition logique et `INSTRUCTION` est

- soit une seule ligne se terminant par un point-virgule ;
- soit un bloc de lignes délimitées par les accolades et séparées pas les points-virgules.

```
{
  quelques;
  lignes;
}
```

if

Exemples :

```
if (a > 0) {
  a = a - 10;
}
```

```
if (a > 0) printf ("positif \n");
```

```
if (a > 0)
  printf ("positif \n");
```

```
if (a > 0) {
  a = a - 10;
} else {
  printf ("Le nombre a = %d ", a);
  printf ("est negatif \n");
}
```

if

```
if (a > 0) a = a - 10;
else {
  printf ("Le nombre a = %d ", a);
  printf ("est negatif \n");
}

if (a == 1) {
  pritnf ("hop")
} else if (a == 2) {
  printf ("hop-hop")
} else if (b == 3) {
  printf ("hop-hop-hop")
} else {
  printf ("Les Shadoks")
}
```

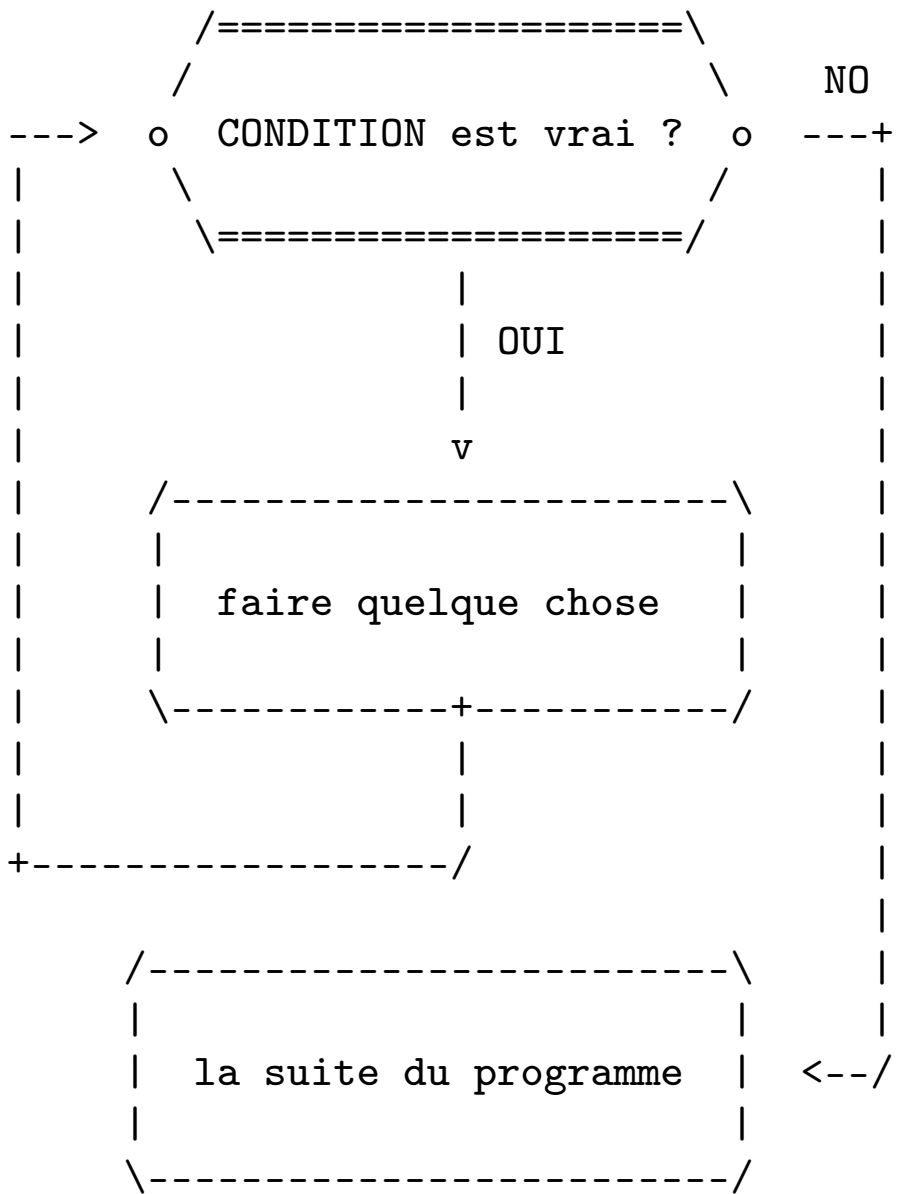
switch

```
switch (n) {
    case 1:
        // si n = 1;
        faire;
        qqch;
        break;
    case 2:
        // si n = 2;
        faire;
        qqch;
        break;
    default:
        // sinon
        faire; autre;
        chose;
}
```

Boucles

while

C'est comme réfléchir avant de faire



La syntaxe :

```
while (CONDITION) {
  faire;
  quelque;
  chose;
}
// la suite
...
```

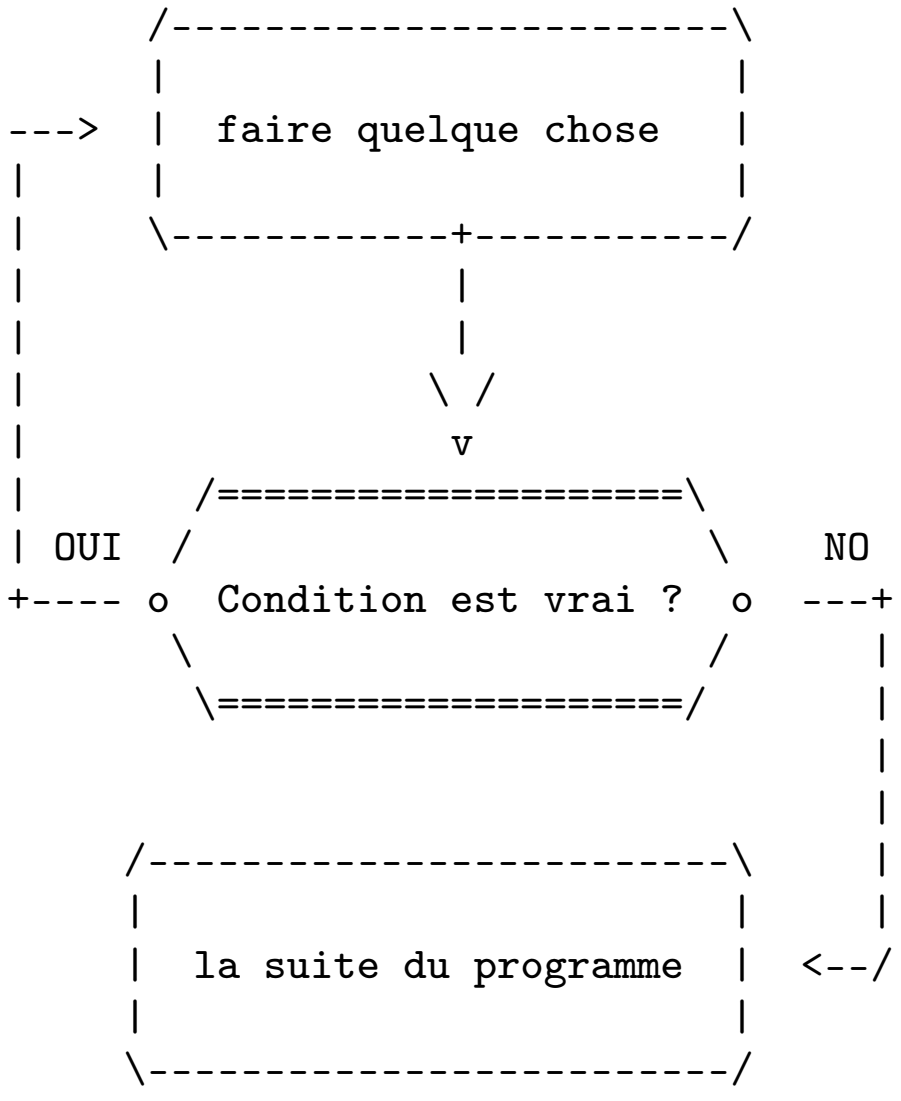
Exemple, boucle “while”

```
a = 0;
while (a < 5) {
  printf ("%d,", a);
  a++;
}
```

va afficher : 0,1,2,3,4,

do ... while

"do ... while ", c'est une boucle "jusqu'à ce que" à postcondition.
C'est comme faire avant de réfléchir.



La syntaxe :

```
do {
  faire;
  quelque;
  chose;
} while (condition);
// la suite
...
```

Exemple, boucle “do ... while”

```
a = 0;
while (a < 5) {
  printf ("%d,", a);
  a++;
}
```

va afficher : 0,1,2,3,4,

Exemple, boucle “do ... while”

```
a = 0;
while (a < 5) {
  printf ("%d,", a);
  a++;
}
```

va afficher : 0,1,2,3,4,

```
a = 0;
do {
  printf ("%d,", a);
  a++;
} while (a < 5);
```

va afficher : 0,1,2,3,4,

Exemple II, boucle “do ... while”

```
a = 7;
while (a < 5) {
  printf ("%d,", a);
  a++;
}
```

va afficher : [RIEN]

Exemple II, boucle “do ... while”

```
a = 7;
while (a < 5) {
  printf ("%d,", a);
  a++;
}
```

va afficher : [RIEN]

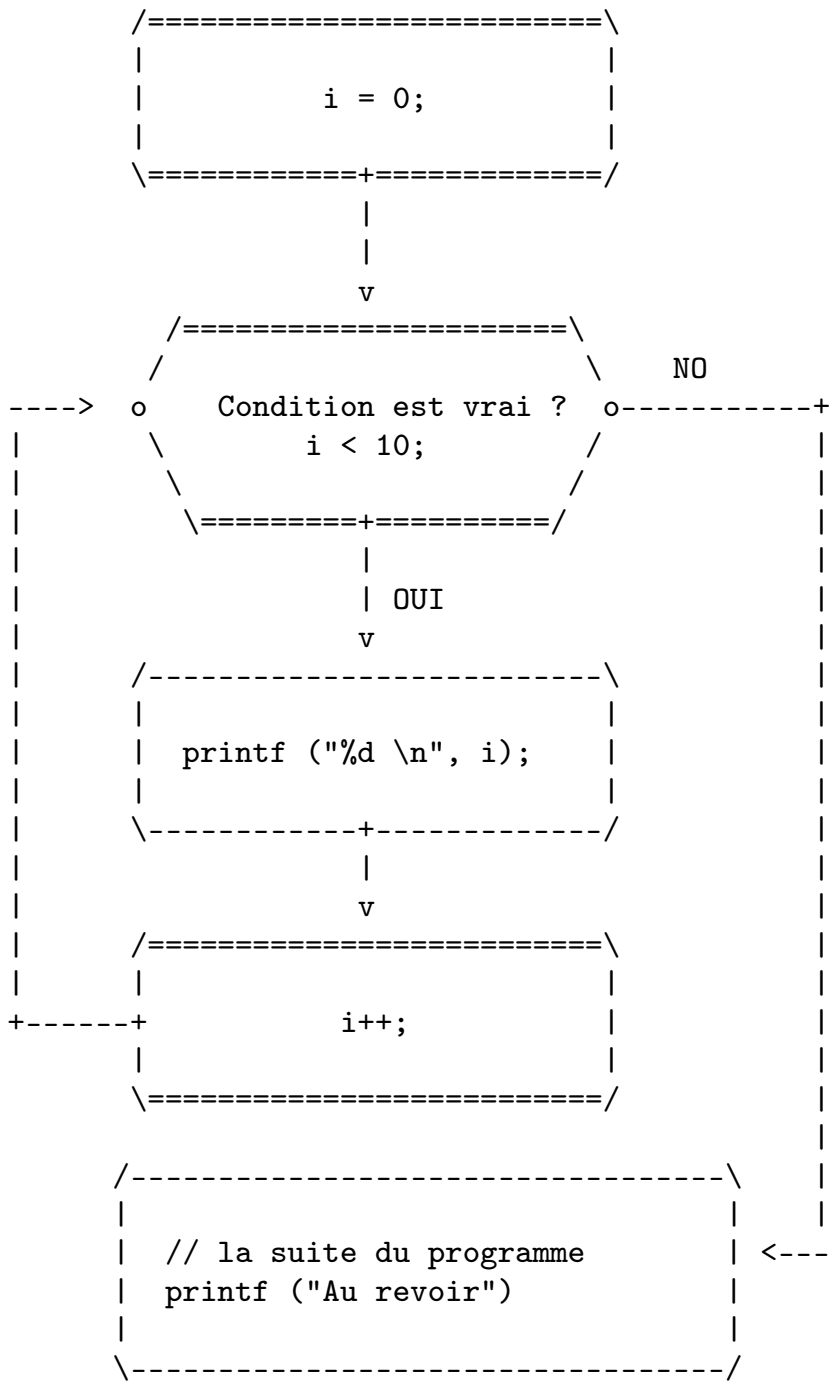
```
a = 7;
do {
  printf ("%d,", a);
  a++;
} while (a < 5);
```

va afficher : 7,

Boucle “for”

La syntaxe :

```
for (initialisation; condition; post-expression) {
  faire;
  quelques;
  choses;
}
```



Exemple :

```
int i;
for (i = 0; i < 10; i++) {
  printf ("%d \n", i);
}
printf ("Au revoir");
```

continue et break

Les instructions spéciales :

- `continue;`
  arrêter l'itération actuelle de boucle, passer à l'itération suivante de boucle.
- `break;`
  interrompre l'exécution de la boucle ou de l'instruction `switch`, et passer directement à la première instruction qui suit la boucle ou l'instruction `switch`.

continue et break

```
int j = 0;
while (j < 100) {
  j = j + 1;
  if (j % 2 == 0) {
    continue;
  }
  if (j < 30) {
    printf ("%d=", j);
  } else {
    printf ("%d\n", j);
    break;
  }
}
```

????

continue et break

```
int j = 0;
while (j < 100) {
  j = j + 1;
  if (j % 2 == 0) {
    continue;
  }
  if (j < 30) {
    printf ("%d=", j);
  } else {
    printf ("%d\n", j);
    break;
  }
}
```

????
1=3=5=7=9=11=13=15=17=19=21=23=25=27=29=31

Blocs entre les accolades

Blocs et variables locales

"Ce qui est déclaré dans {...} reste dans {...}"

Comparer :

```
int a = 100;
printf ("%d ", a);
{
  int a = 10;
  a = 20 + a;
  printf ("%d ", a);
}
printf ("%d \n", a);
```

???

```
int a = 100;
printf ("%d ", a);
{
  a = 20 + a;
  printf ("%d ", a);
}
printf ("%d \n", a);
```

???

Blocs et variables locales

"Ce qui est déclaré dans {...} reste dans {...}"

Comparer :

```
int a = 100;
printf ("%d ", a);
{
  int a = 10;
  a = 20 + a;
  printf ("%d ", a);
}
printf ("%d \n", a);
```

???
100 30 100

```
int a = 100;
printf ("%d ", a);
{
  a = 20 + a;
  printf ("%d ", a);
}
printf ("%d \n", a);
```

???
100 120 120

Instructions imbriquées

Instructions imbriquées. La grammaire simplifiée

Ici, le symbole “|” signifie “soit”.

```
BLOC = {
         INSTRUCTION
         INSTRUCTION
         ...
       }

INSTRUCTION =
   BLOC | AFFECTATION; | DECLARATION; | EXPRESSION; |
   if (EXPRESSION) INSTRUCTION |
   if (EXPRESSION) INSTRUCTION else INSTRUCTION |
   WHILE (EXPRESSION) INSTRUCTION |
   do INSTRUCTION while (EXPRESSION); |
   switch (EXPRESSION) {CASE (VALEUR ENTIER) : INSTRUCTION
                        CASE (VALEUR ENTIER) : INSTRUCTION ... }
```

Instructions imbriquées. Exemple

```
int j = 0;
while (j < 100) {
  j = j + 1;
  if (j % 2 == 0) {
    continue;
  }
  if (j < 30) {
    printf ("%d=", j);
  } else {
    printf ("%d\n", j);
    break;
  }
}
```

Fonctions

Fonctions. La grammaire simplifiée

```
TYPE_DE_RESULTAT  NOM_DE_FONCTION  (TYPE_DE_PARAMETRE_1 NOM_DE_PARAMETRE_1,
                                     TYPE_DE_PARAMETRE_2 NOM_DE_PARAMETRE_2,
                                     ...
                                     )
                                   {
                                     DÉCLARATIONS DE VARIABLES LOCALES
                                     LISTE D'INSTRUCTIONS
                                   }

TYPE_DE_RESULTAT = void | int | double | ...
```

`void` est un type spéciale qui n'ai pas de valeur.
Le symbole "|" signifie "soit".

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

1. Le système commence l'exécution par la fonction `main`.

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

1. Le système commence l'exécution par la fonction `main`.
2. La fonction `main` appelle la fonction `puissance_de_2` en lui donnant, comme paramètre, la valeur de la variable `k`.

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

1. Le système commence l'exécution par la fonction `main`.
2. La fonction `main` appelle la fonction `puissance_de_2` en lui donnant, comme paramètre, la valeur de la variable `k`.
3. La fonction `puissance_de_2` apelle la fonction `puissance_de_2`

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

1. Le système commence l'exécution par la fonction `main`.
2. La fonction `main` appelle la fonction `puissance_de_2` en lui donnant, comme paramètre, la valeur de la variable `k`.
3. La fonction `puissance_de_2` apelle la fonction `puissance_de_2`
4. ...

Fonctions. Exemple. Recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   if (n == 0) return 1;
   return 2 * puissance_de_2 (n - 1);
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

1. Le système commence l'exécution par la fonction `main`.
2. La fonction `main` appelle la fonction `puissance_de_2` en lui donnant, comme paramètre, la valeur de la variable `k`.
3. La fonction `puissance_de_2` apelle la fonction `puissance_de_2`
4. ...
5. La fonction `main` appelle la fonction `printf` en replaçant `puissance_de_2 (k)` par 256.

Fonctions. Exemple. Sans recursion

```
#include <stdio.h>
int puissance_de_2 (int n) {
   int i;
   int p = 1;
   for (i = 0; i < n; i++) {
     p = p * 2;
   }
   return p;
}
int main () {
  int k = 8;
  printf ( "2**n = %d \n", puissance_de_2 (k));
  return 0;
}
```

Tableaux, chaînes de caractères et pointeurs

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

```
int quplet[4];
```

Valeur				
Indice	0	1	2	3

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2			
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2	6		
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
quplet[1] = 6;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2	6		
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
quplet[1] = 6;
int q = 8;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2	6	7	
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
quplet[1] = 6;
int q = 8;
quplet[q/4] = 7;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2	6	7	0
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
quplet[1] = 6;
int q = 8;
quplet[q/4] = 7;
quplet[3] = -1 + quplet[2] - 6;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Tableaux

Un tableau est une séquence fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Valeur	2	6	7	0
Indice	0	1	2	3

```
int quplet[4];
quplet[0] = 2;
quplet[1] = 6;
int q = 8;
quplet[q/4] = 7;
quplet[3] = -1 + quplet[2] - 6;
```

Déclaration d'un tableau :
`TYPE NOM [TAILLE];`
où taille est un entier.

Déclaration-initialisation : `int quplet[4] = {2, 6, 7, 0};`

Chaînes de caractères

Une chaîne de caractères (string) est un tableau de caractères. Dans l'ordinateur chaque caractère est représenté par un nombre correspondant (code ASCII).

```
char str[10] = "ESIREM";
```

Chaînes de caractères

Une chaîne de caractères (string) est un tableau de caractères. Dans l'ordinateur chaque caractère est représenté par un nombre correspondant (code ASCII).

```
char str[10] = "ESIREM";
```

Valeur	'E'	'S'	'I'	'R'	'E'	'M'	'\0'			
Indice	0	1	2	3	4	5	6	7	8	9

Chaînes de caractères

Une chaîne de caractères (string) est un tableau de caractères. Dans l'ordinateur chaque caractère est représenté par un nombre correspondant (code ASCII).

```
char str[10] = "ESIREM";
```

Valeur	69	83	73	82	69	77	0			
Indice	0	1	2	3	4	5	6	7	8	9

Chaînes de caractères

Une chaîne de caractères (string) est un tableau de caractères. Dans l'ordinateur chaque caractère est représenté par un nombre correspondant (code ASCII).

```
char str[10] = "ESIREM";
```

Valeur	69	83	73	82	69	77	0			
Indice	0	1	2	3	4	5	6	7	8	9

Expérimentez !

```
#include <stdio.h>
int main () {
  char str[10] = "ESIREM";
  for (int i = 0; i < 10; i++) {
    printf ("%d ", str[i]);
  }
}
```

Zéro à la fin d'une chaîne

Le caractère special `'\0'` (code ASCII 0) est utilisé pour marquer la fin d'une chaîne de caractères. Par conséquent, une chaîne de longueur n caractères occupe n + 1 octets en mémoire.

Zéro à la fin d'une chaîne

Le caractère special '\0' (code ASCII 0) est utilisé pour marquer la fin d'une chaîne de caractères. Par conséquent, une chaîne de longueur n caractères occupe n + 1 octets en mémoire.

```
#include <stdio.h>
int main () {
  char str[10] = "ESIREM";
  printf ("%s \n", str);
  str[3] = '\0';
  printf ("%s \n", str);
  return 0;
}
```

????

Zéro à la fin d'une chaîne

Le caractère special '\0' (code ASCII 0) est utilisé pour marquer la fin d'une chaîne de caractères. Par conséquent, une chaîne de longueur n caractères occupe n + 1 octets en mémoire.

```
#include <stdio.h>
int main () {
  char str[10] = "ESIREM";
  printf ("%s \n", str);
  str[3] = '\0';
  printf ("%s \n", str);
  return 0;
}
```

```
????
ESIREM
ESI
```

Astuces

- Veillez à ne pas dépasser les limites de tableau. C vous permet de faire cela, mais le résultat n'est pas défini.

Astuces

- Veillez à ne pas dépasser les limites de tableau. C vous permet de faire cela, mais le résultat n'est pas défini.
- Un tableau à deux dimensions est un tableau dont chaque élément est un autre tableau. Exemple :

```
int matrix[10][10];
int i = 4;
int j = 3;
matrix[i][j] = 10;
```

Pointeurs

Les cellules de mémoire ont des adresses.

Pointeurs

Les cellules de mémoire ont des adresses.

Un pointeur est une variable dont la valeur est l'adress de quelque chose dans la mémoire de l'ordinateur.

Pointeurs

Les cellules de mémoire ont des adresses.

Un pointeur est une variable dont la valeur est l'adress de quelque chose dans la mémoire de l'ordinateur.

Déclaration d'un pointeur vers une valeur de type TYPE :
`TYPE* NOM_DE_POINTEUR;`

Pointeurs

Les cellules de mémoire ont des adresses.

Un pointeur est une variable dont la valeur est l'adress de quelque chose dans la mémoire de l'ordinateur.

Déclaration d'un pointeur vers une valeur de type TYPE :
`TYPE* NOM_DE_POINTEUR;`

Opérations

- `&X` donne l'adresse de valeur de variable `X`
- `*P` permet d'accéder directement à la valeur de l'objet pointé par le pointeur `P` (déréférencement)

Pointeurs

Les cellules de mémoire ont des adresses.

Un pointeur est une variable dont la valeur est l'adress de quelque chose dans la mémoire de l'ordinateur.

Déclaration d'un pointeur vers une valeur de type TYPE :
`TYPE* NOM_DE_POINTEUR;`

Opérations

- `&X` donne l'adresse de valeur de variable `X`
- `*P` permet d'accéder directement à la valeur de l'objet pointé par le pointeur `P` (déréférencement)

```
int i;
int* p;
p = &i;
*p = 10;
printf("%d \n", i);
```

Exemple

```
// to compile : gcc -Wall test.c -o test
// to run : ./test
#include <stdio.h>

void hop (int* p1, int* p2) {
  printf ("First address is %p \n", p1);
  printf ("Second address is %p \n", p2);
  printf ("First value is %d \n", *p1);
  printf ("Second value is %d \n", *p2);
  printf ("Swap them !!! \n");
  int tmp;
  tmp = *p1;
  *p1 = *p2;
  *p2 = tmp;
}

int main () {
   int A = 10;
   int B = 20;
   printf("Before swap A = %d  and B = %d\n", A, B);
   hop (&A, &B);
   printf("After swap A = %d  and B = %d\n", A, B);
}
```

Exemple

```
// to compile : gcc -Wall test.c -o test
// to run : ./test
#include <stdio.h>

void hop (int* p1, int* p2) {
  printf ("First address is %p \n", p1);
  printf ("Second address is %p \n", p2);
  printf ("First value is %d \n", *p1);
  printf ("Second value is %d \n", *p2);
  printf ("Swap them !!! \n");
  int tmp;
  tmp = *p1;
  *p1 = *p2;
  *p2 = tmp;
}

int main () {
   int A = 10;
   int B = 20;
   printf("Before swap A = %d  and B = %d\n", A, B);
   hop (&A, &B);
   printf("After swap A = %d  and B = %d\n", A, B);
}
```

```
Before swap A = 10 and B = 20
First address is 0x5a567bec8f00
Second address is 0x5a567bec8f04
First value is 10
Second value is 20
Swap them ! ! !
After swap A = 20 and B = 10
```

Astuces

Il y a des pointeurs non seulement vers des valeurs, mais aussi vers d'autres objets, par exemple vers des corps des fonctions !

Arithmétique des pointeurs

Exemple :

```
// to compile : gcc -Wall test.c -o test
// to run : ./test
#include <stdio.h>

int main () {
  double a[3] = {3, 3.14, 3.15};
  double* p;
  p = a;
  printf (" %p --> %f \n", p, *p);
  printf (" %p --> %f \n", (p+1), *(p+1));
  double* q;
  q = p;
  q = q + 2;
  printf (" %p --> %f \n", q, *q);
}
```

Arithmétique des pointeurs

Exemple :

```
// to compile : gcc -Wall test.c -o test
// to run : ./test
#include <stdio.h>

int main () {
  double a[3] = {3, 3.14, 3.15};
  double* p;
  p = a;
  printf (" %p --> %f \n", p, *p);
  printf (" %p --> %f \n", (p+1), *(p+1));
  double* q;
  q = p;
  q = q + 2;
  printf (" %p --> %f \n", q, *q);
}
```

```
0x3a8eb20b1e50 --> 3.000000
0x3a8eb20b1e58 --> 3.140000
0x3a8eb20b1e60 --> 3.150000
(La taille d'un double est 8 octets)
```

Allocation dynamique

Comment déclararer un tableau avec la taille `n` que nous ne connaissons pas à l'avance ? ? ?

Allocation dynamique

Comment déclararer un tableau avec la taille `n` que nous ne connaissons pas à l'avance ? ? ?

```
int* p;
p = (int*) malloc ( sizeof(int) * n);
...
...
...
free (p); // n'oubliez pas de libérer
          // la mémoire sinon elle fuit
```

Documentation : `calloc`, `realloc`, `memset`

Pointeurs et tableaux

Exemple 1 :

```
char str[10] = "ESIREM";
char* pointer_vers_string;
pointer_vers_string = str; // c'est tout à fait valide
                           // str peut être vue
                           // comme char*
```

Pointeurs et tableaux

Exemple 1 :

```
char str[10] = "ESIREM";
char* pointer_vers_string;
pointer_vers_string = str; // c'est tout à fait valide
                           // str peut être vue
                           // comme char*
```

Exemple 2 :

```
int matrix[10][10];
int*  ligne = matrix[3];
ligne[4] = 10;
printf ("%d \n", matrix[3][4]);
```

Arguments de la ligne de commande

```
// to compile : gcc -Wall test.c -o test
// to run :
// ./test
// ./test asd weiur 234
// ./test "asd weiur 234"
// ./test "asd weiur" 234
#include <stdio.h>

int main (int argc, char** argv) {
    printf ("Number of command line arguments is %d \n",
            argc);
    for (int i = 0; i < argc; i++) {
      printf ("Argument %d is %s \n", i, argv[i]);
    }
    return 0;
}
```

Hello world II

Hello world II

```
// to compile : gcc -Wall hw.c -o hw
// to run : ./hw or ./hw Name

#include <stdio.h>
#include <stdlib.h>
// stdlib c'est pour la fonction "atoi"

int main (int argc, char** argv) {
    char* who = "world";
    int i = 0;
    int m = 1;

    if (argc > 1) {
      who = argv[1];
    }

    if (argc > 2) {
      m = atoi (argv[2]);
    }

    for (i = 0; i < m; i++) {
      printf("Hello %s!\n", who);
    }
    return 0;
}
```

Pour la suite

- L'allocation de mémoire : malloc, calloc, free, realloc, memset

- L'allocation de mémoire : malloc, calloc, free, realloc, memset
- Le préprocesseur : #include, #define, #ifdef, #if

  `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre5.html`

- L'allocation de mémoire : malloc, calloc, free, realloc, memset
- Le préprocesseur : #include, #define, #ifdef, #if

  `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre5.html`
- Les types composés : les structures, les champs de bits, les unions, les énumérations, typedef. `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre2.html`

- L'allocation de mémoire : malloc, calloc, free, realloc, memset
- Le préprocesseur : #include, #define, #ifdef, #if

  https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre5.html
- Les types composés : les structures, les champs de bits, les unions, les énumérations, typedef. https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre2.html
- Le débogueur GDB. https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/gdb.html

- L'allocation de mémoire : malloc, calloc, free, realloc, memset
- Le préprocesseur : #include, #define, #ifdef, #if

  `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre5.html`
- Les types composés : les structures, les champs de bits, les unions, les énumérations, typedef. `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre2.html`
- Le débogueur GDB. `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/gdb.html`
- Un outil "Valgrind" pour pour déboguer, effectuer du profilage de code et mettre en évidence des fuites mémoires.

Questions ?

sergey.kirgizov@u-bourgogne.fr

https://kirgizov.link/teaching/esirem/unix-and-c