Tableaux et boucle `For`

Pierre-Alain FOUQUE
Département d'Informatique
École normale supérieure

Plan

1 – Tableaux

2 – Boucles `for`

3 – Paramètres sur la ligne de commande

Limite des types de base

- Définir autant de variables que de cases mémoires nécessaires
- Accéder à chaque variable, une par une, et par son nom « en dur » dans le programme

⇒ pas d'indexation possible

Les tableaux

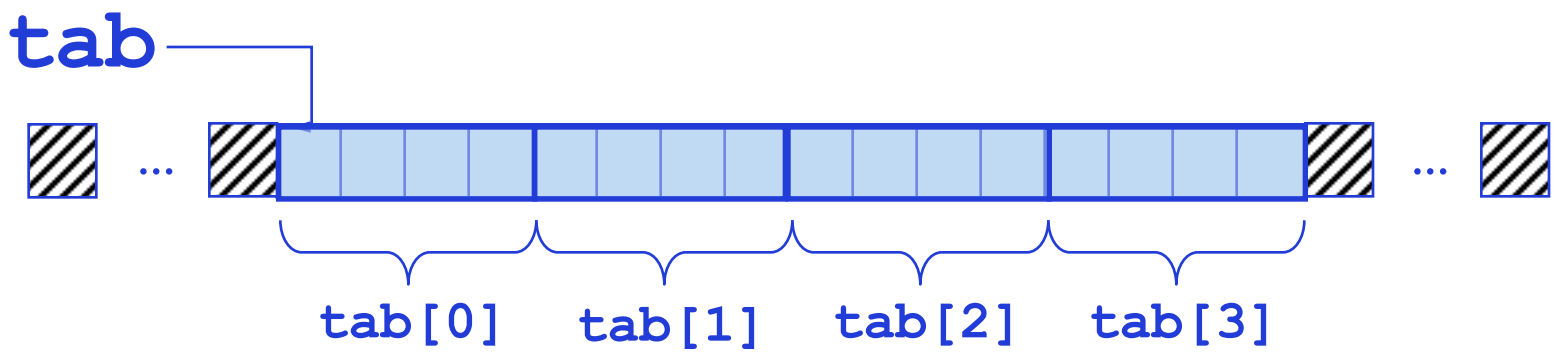
- Définir sous un nom unique
  un groupe de cases mémoires
  de même type

- Accéder à chaque case
  par sa position dans le tableau

La mémoire

```
int tab[4];
```

définit un tableau de 4 entiers consécutifs



Toutes sortes de tableaux

```
float tabF[10];
```
tabF, tableau de 10 **floats**

```
double tabD[100];
```
tabD, tableau de 100 **doubles**

```
char chaine[256];
```
chaine, chaîne de 256 caractères

```
int tabI[N];
```
tabI, tableau de N entiers

Taille constante

La longueur du tableau
doit être une **constante**

Taille constante

La longueur du tableau
doit être une **constante**

- soit une constante « en dur »
  dans le programme (10, 100, …)

Taille constante

La longueur du tableau
doit être une **constante**

```
int tabI[20];

float tabF[10];
```

soit une constante « en dur »
dans le programme (10, 100, …)

Taille constante

La longueur du tableau
doit être une **constante**

```
int tabI[20];

float tabF[10];
```

- soit une constante « en dur »
dans le programme (10, 100, …)

- soit (de façon équivalente !)
par le pré-processeur
`#define N 20`

Taille constante

La longueur du tableau
doit être une **constante**

```
int tabI[20];

float tabF[10];
```

soit une constante « en dur »
dans le programme (10, 100, …)

```
#define N 10

int tabI[N];
```

soit (de façon équivalente !)
par le pré-processeur
`#define N 20`

Taille constante

La longueur du tableau
doit être une **constante**

```
int tabI[20];

float tabF[10];
```

- soit une constante « en dur »
dans le programme (10, 100, …)

```
#define N 10

int tabI[N];
```

- soit (de façon équivalente !)
par le pré-processeur
`#define N 20`

cette dernière méthode est à préconiser :
il est alors facile de réviser la taille

Initialisation d'un tableau

Lors de sa création un tableau contient
n'importe quoi dans ses cases :
en effet, la déclaration réserve seulement
la mémoire, mais n'y touche pas !
⇒ il faut initialiser les cases, une à une.

- Déclaration + initialisation
- Initialisation systématique : boucle **`for`**
- Initialisation case par case

Déclaration + initialisation

Comme pour les variables de types simples, il est possible de combiner la déclaration et l'initialisation :

```
int tab[4] = { 2, 3, -1, 5 };
```

crée le tableau tab de 4 entiers avec

```
tab[0] = 2;
tab[1] = 3;
tab[2] = -1;
tab[3] = 5;
```

Initialisation partielle

On peut n'initialiser que certaines cases :

```
int tab[10] = { 2, , -1, 5 };
```

crée le tableau **tab** de 10 entiers avec

- **tab[0] = 2**
- **tab[2] = -1**
- **tab[3] = 5**
- les autres cases ne sont pas initialisées

Tableaux de caractères

Un tableau de caractères est un tableau particulier : chaîne de caractères

```
char mot[10] = "toto";
```

crée le tableau mot de 10 caractères avec

```
mot[0] = 't'        mot[1] = 'o'
mot[2] = 't'        mot[3] = 'o'
mot[4] = '\0' (fin de chaîne)
```

les autres cases ne sont pas initialisées

Initialisation systématique : Boucle `for`

Il est possible d'initialiser chaque case en fonction de sa position **`i`**

```
tabI[i] = f(i);
```

où **`f`** est une fonction qui dépend de **`i`**

La commande **`for`** permet de boucler en incrémentant le compteur **`i`** à chaque tour

Boucle `for`

La boucle **`for`** répète une instruction plusieurs fois, avec un compteur qui s'incrémente à chaque tour :

```
for (<init>; <test>; <incrémentation>)
  <instruction>
```

- `<init>` : **initialisation** du compteur
- `<test>` : test de **continuation**
- `<incrémentation>` : **incrémentation** du compteur

`for` = `while`

La boucle **`for`** n'est en fait qu'une reformulation du **`while`** :

for = while

La boucle **for** n'est en fait qu'une reformulation du **while** :

```
for (<init>; <test>; <incrémentation>)
                <instruction>
```

for = while

La boucle **for** n'est en fait qu'une reformulation du **while** :

for
```
for (<init>; <test>; <incrémentation>)
                    <instruction>
```

while
```
<init>
while (<test>)
{
            <instruction>
            <incrémentation>
}
```

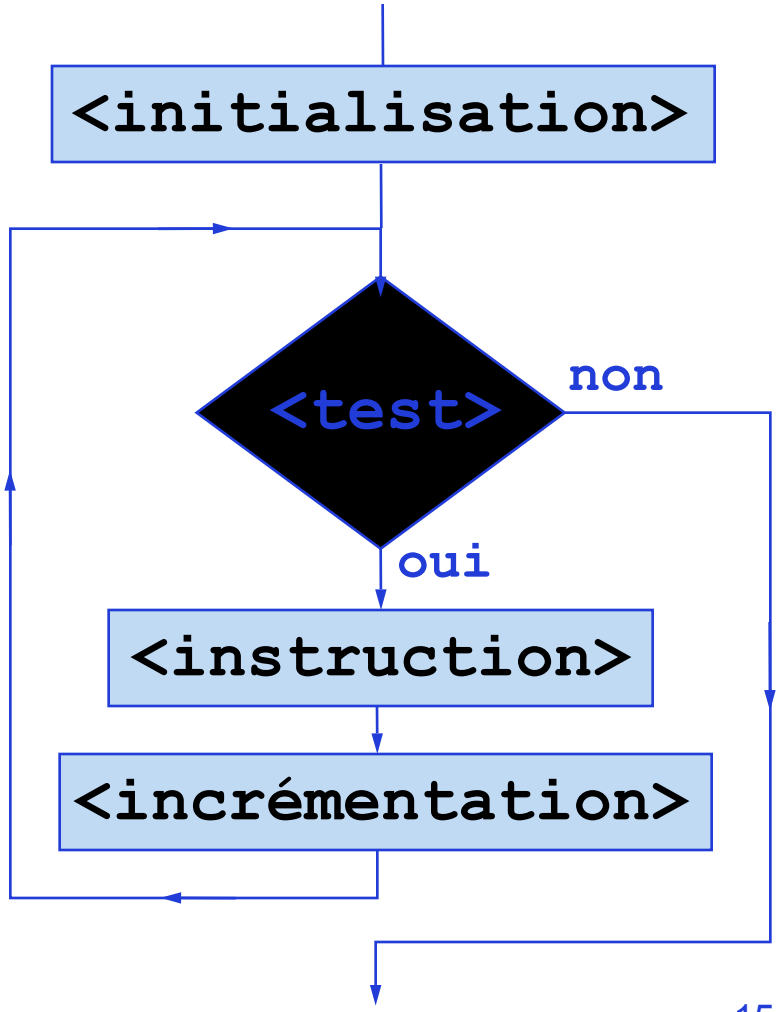
Boucle `for`

Boucle `for`

Une instruction
est exécutée,
plusieurs fois
avec un compteur:

Boucle for

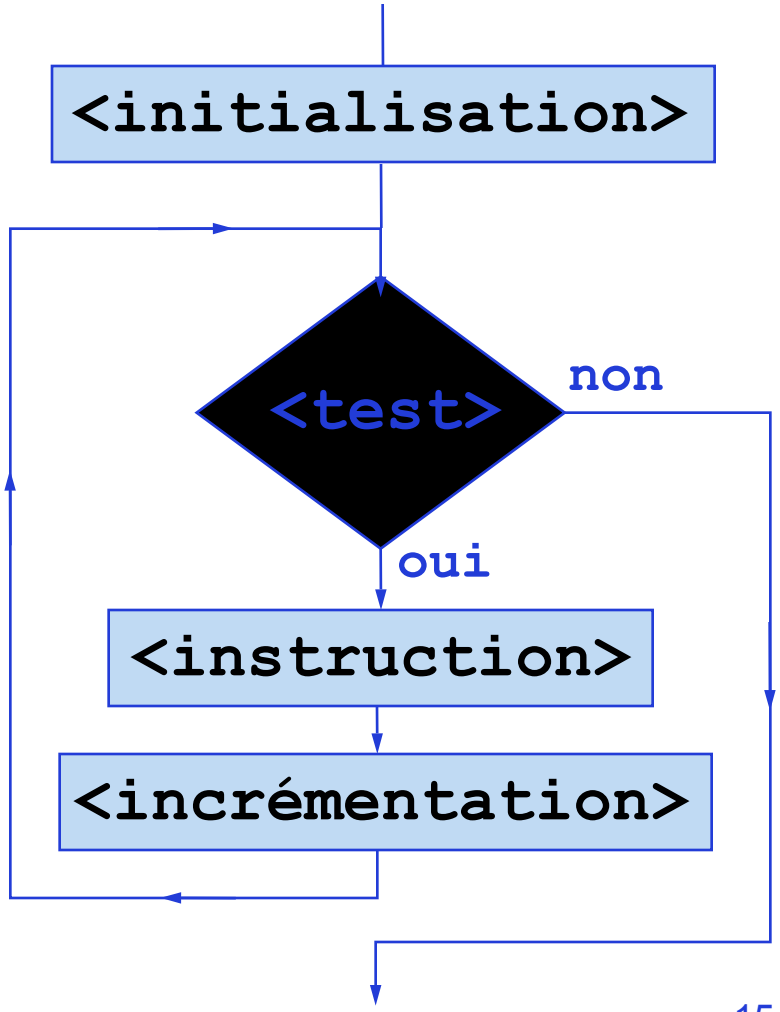
Une instruction est exécutée, plusieurs fois avec un compteur:



Boucle `for`

Une instruction est exécutée, plusieurs fois avec un compteur:

⇒ l'instruction peut ne jamais être exécutée



`for` classique

L'usage classique de la boucle **`for`** est le suivant :

```
int n=15;
int i;

for (i=0; i<n; i++)
     <instruction(i)>
```

`for` classique

L'usage classique de la boucle **`for`** est le suivant :

```
int n=15;
int i;

for (i=0; i<n; i++)
     <instruction(i)>
```

Initialisation d'un tableau

La boucle **`for`** permet d'initialiser aisément un tableau :

```
#define N 10
int tab[N];
int i;
for (i=0; i<N; i++)
        tab[i] = 0;
```

Les cases sont toutes initialisées à Zéro

Initialisation d'un tableau

La boucle **`for`** permet d'initialiser un tableau avec des valeurs qui dépendent de l'indice de la case :

```
#define N 10
int carre[N];
int i;
for (i=0; i<N; i++)
        carre[i] = i*i;
```

Affichage d'un tableau

La commande **printf** ne permet pas d'afficher un tableau tel quel, il faut afficher les cases une à une :

```
/* carre.c - Carrés */
#include <stdio.h>
#include <stdlib.h>
#define N 10
```

```
int main(int argc, char *argv[])
{
   int carre[N];
   int i;
   for (i=0; i<N; i++)
      carre[i] = i*i;

   for (i=0; i<N; i++)
      printf("%d ",carre[i]);
   printf("\n");
   return 0;
}
```

Arguments utilisateur

- Initialisations :

```
a = 3;
b = 5;
```

⇒ valeurs imposées à la compilation, pas modifiables au cours de l'exécution

⇒ exécutions toutes identiques

⇒ pas grand intérêt !

```
int main()
{
   int a,b,c;
   a = 3;
   b = 5;
   c = addition(a,b);
   printf("%d+%d=%d\n",a,b,c);
   return 0;
}
```

argc et argv

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("%s %s\n",argv[0],argv[1]);
   return 0;
}
```

Fonction **main** :

- Premier argument **int argc**

  nombre de cases du tableau
  = nombre de mots sur la ligne de commande

- Deuxième argument **char *argv[]**

  tableau qui contient les mots passés
  sur la ligne de commande

Exemple

```
#include <stdio.h>


int main(int argc, char *argv[])
{
   printf("%s %s\n",argv[1],argv[2]);
   return 0;
}
```

Exemple

```
#include <stdio.h>


int main(int argc, char *argv[])
{
    printf("%s %s\n",argv[1],argv[2]);
    return 0;
}
```

Saisir le programme

```
emacs affiche.c &
```

Compiler le programme

```
gcc -Wall affiche.c -o
affiche
```

Exécuter

```
>affiche toto tata
toto tata
```

Exemple

```
#include <stdio.h>


int main(int argc, char *argv[])
{
    printf("%s %s\n",argv[1],argv[2]);
    return 0;
}
```

Saisir le programme

```
emacs affiche.c &
```

Compiler le programme

```
gcc -Wall affiche.c -o
affiche
```

Exécuter

```
>affiche toto tata
toto tata
```

En effet, le tableau **`char *argv[];`**

contient

`affiche` dans **`argv[0]`**

`toto` dans **`argv[1]`**

`tata` dans **`argv[2]`**

et l'entier **`int argc;`**

est égal à 3

Conversion atoi

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int x;
   x = atoi(argv[1]);
   printf("%d -> %d \n",x,x+1);
   return 0;
}
```

Conversion `atoi`

Les **`argv[`*`i`*`]`** sont
des « chaînes de caractères »
⇒ à convertir en entiers
(quand elles codent des entiers)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int x;
   x = atoi(argv[1]);
   printf("%d -> %d \n",x,x+1);
   return 0;
}
```

Conversion `atoi`

Les **`argv[`*`i`*`]`** sont
des « chaînes de caractères »
⇒ à convertir en entiers
(quand elles codent des entiers)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int x;
   x = atoi(argv[1]);
   printf("%d -> %d \n",x,x+1);
   return 0;
}
```

```
int x;
x = atoi(argv[1]);
```

Suites

Exemple précédent : $u_i = f(i)$

Autres possibilités : $u_i = f(i, u_{i-k}, u_{i-k+1}, \ldots, u_{i-1})$

Ex : la factorielle

```
/* fact.c - Factorielle */
#include <stdio.h>
#include <stdlib.h>
#define N 30
int fact[N];
```

```
int main(int argc, char *argv[])
{
   int i;
   int n = atoi(argv[1]);
   fact[0] = 1;
   for (i=1; i<=n; i++)
      fact[i] = fact[i-1]*i;
   printf("Fact(%d) = %d \n",n,fact[n]);
   return 0;
}
```

Plusieurs dimensions

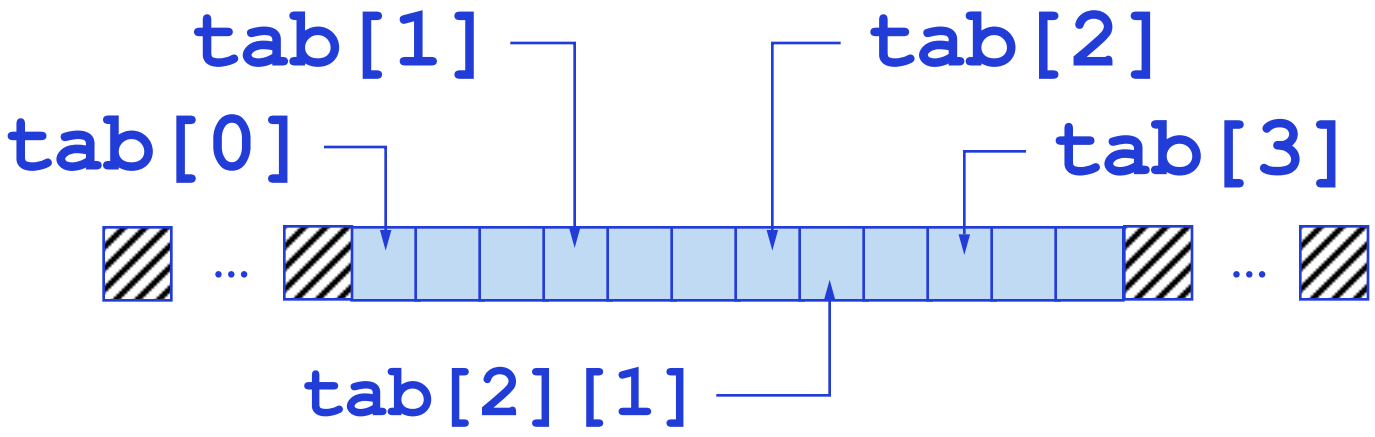
Un tableau à plusieurs dimensions est en fait un tableau de tableaux de tableaux de …

Un tableau à n dimensions est un tableau de tableaux à n-1 dimensions

La mémoire

`char tab[4][3];`

définit un tableau de 4 tableaux
de 3 caractères (**`char`**) consécutifs



Boucles `for` imbriquées

```
#define M 20
#define N 10

int tab[M][N];
int i,j;

for (i=0; i<M; i++)
      for (j=0; j<N; j++)
            tab[i][j] = 0;
```

Pointeurs

Les pointeurs permettront
de définir des tableaux
de taille non constante

(définie au cours de l'exécution
du programme)

Le C99 permet aussi de le faire ...

cf. les cours sur les pointeurs
et l'allocation dynamique