

Langage C

Les fonctions

François Cuvelier

Institut Galilée
Université Paris XIII.

19 octobre 2019

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- Prototype d'une fonction
- Corps d'une fonction
- Exemple de définition
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Généralités

Les fonctions permettent

- d'automatiser certaines tâche répétitives au sein d'un même programme,
- d'ajouter à la clarté d'un programme,
- l'utilisation de portion de code dans un autre programme,
- de faciliter le debuggage et la validation,
- ...

Plan

1 Fonctions

- Généralités
- **Syntaxe**
- Définition de fonctions
- Prototype d'une fonction
- Corps d'une fonction
- Exemple de définition
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Syntaxe

Une fonction est composée de son **prototype** (ou en-tête) et de son **corps**

Le prototype permet au compilateur de vérifier la validité des appels à cette fonction

```
1 | int f(int a, int b)
2 | {
3 |     int x;
4 |     x = a+b;
5 |     return x;
6 | }
```

Plan

1 Fonctions

- Généralités
- Syntaxe
- **Définition de fonctions**
- Prototype d'une fonction
- Corps d'une fonction
- Exemple de définition
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Définition de fonctions

Syntaxe

```
<type> <NomFonction> (<arg1>, ..., <argN>) {  
    <declarations eventuelles>  
    <instructions>  
}
```

On appelle **prototype** ou **en-tête** d'une fonction la partie :

```
<type> <NomFonction> (<arg1>, ..., <argN>)
```

La partie entre accolades est appelée **corps** de la fonction : c'est une instruction composée.

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- **Prototype d'une fonction**
- Corps d'une fonction
- Exemple de définition
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Prototype d'une fonction

Syntaxe

```
<type> <NomFonction> (<arg1>, ..., <argN>)
```

- `<NomFonction>` : identifiant de la fonction,
- `<type>` : type retourné par la fonction. Ne peut être un tableau ou une fonction. Si la fonction ne retourne aucune valeur, la déclarer du type `void`.
- `<arg1>, ..., <argN>` : arguments de la fonction. Doivent être fournis avec leur type. Si une fonction n'a pas d'argument, utiliser le type `void`.

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- Prototype d'une fonction
- **Corps d'une fonction**
- Exemple de définition
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Corps d'une fonction

Le corps d'une fonction est une instruction composée :

Syntaxe

```
{  
  <declarations eventuelles>  
  <instructions>  
}
```

A l'intérieur d'une instruction composée, il est impossible de définir une fonction.

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- Prototype d'une fonction
- Corps d'une fonction
- **Exemple de définition**
- Déclaration d'une fonction
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Exemple

```
1 | int f(int a, int b)
2 | {
3 |     int x;
4 |     x = a+b;
5 |     return x;
6 | }
```

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- Prototype d'une fonction
- Corps d'une fonction
- Exemple de définition
- **Déclaration d'une fonction**
- Exemple

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Déclaration d'une fonction

A pour but d'avertir le compilateur qu'elle va être utilisée et de l'informer du type de résultat, ainsi que du type des éventuels arguments.

On déclare un prototype de fonction dans deux cas :

- Utilisation d'une fonction définie dans un autre fichier.
- Utilisation d'une fonction définie plus loin dans le même fichier.

Syntaxe

```
<type> <NomFonction> (<arg1>, ..., <argN>);
```

Plan

1 Fonctions

- Généralités
- Syntaxe
- Définition de fonctions
- Prototype d'une fonction
- Corps d'une fonction
- Exemple de définition
- Déclaration d'une fonction
- **Exemple**

2 Exemples avec représentation mémoire

- Exemple 1
- Exemple 2

3 Pour aller plus loin...

Exemple complet

```
1 #include <stdio.h>
2
3 int f(int a, int b);
4
5 int main(void){
6     int x, s=1;
7     x=f(s, 10);
8     printf("x=%d\n", x);
9     return 0;
10 }
11
12 int f(int a, int b)
13 {
14     int x;
15     x = a+b;
16     return x;
17 }
```

Plan

- 1 Fonctions
 - Généralités
 - Syntaxe
 - Définition de fonctions
 - Prototype d'une fonction
 - Corps d'une fonction
 - Exemple de définition
 - Déclaration d'une fonction
 - Exemple
- 2 Exemples avec représentation mémoire
 - Exemple 1
 - Exemple 2
- 3 Pour aller plus loin...

Listing 4 – Exemple.c

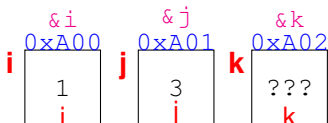
```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10  ▶ int i=1,j=3,k;
11     k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```

```
int i=1,j=3,k;
```

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10  ▶ int i=1,j=3,k;
11     k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```



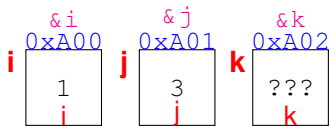
```
int i=1,j=3,k;
```

i initialisé à 1, j initialisé à 3, k non initialisé

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     ▶ k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```



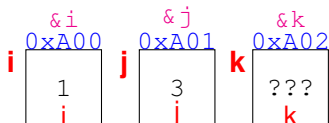
```
k=f(i,j);
```

Listing 4 – Exemple.c

```

1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     ▶ k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }

```



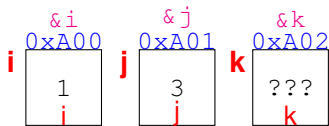
```
k=f(i,j);
```

appel de la fonction `f` avec la valeur de `i` en premier paramètre et la valeur de `j` en second

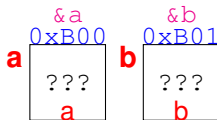
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```

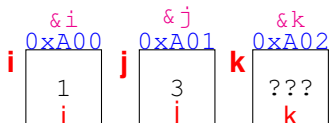


Mémoire : f



```
int f(int a, int b){
```

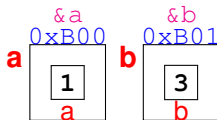
Mémoire : main



Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```

Mémoire : f



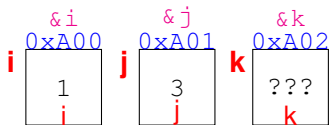
```
int f(int a, int b){
```

`a` est initialisé à 1 (valeur de `i`) et `b` est initialisé à 3 (valeur de `j`)

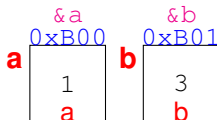
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



Mémoire : f

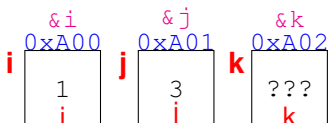


```
int x;
```

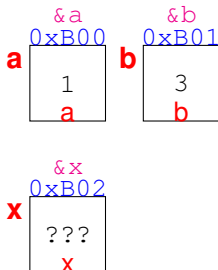
Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```

Mémoire : main



Mémoire : f



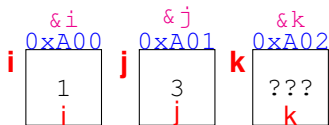
```
int x;
```

Déclaration de `x` (non initialisé)

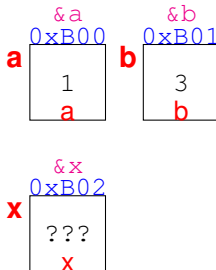
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     ▶ x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    ■ k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



Mémoire : f

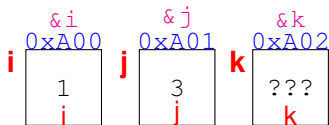


`x=a+2*b;`

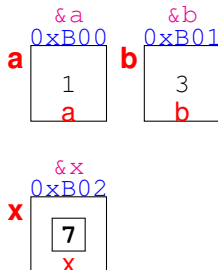
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     ▶ x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    ■ k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



Mémoire : f



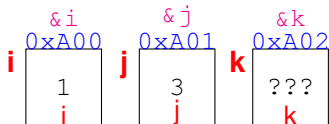
`x=a+2*b;`

`x ← a+2*b=7`

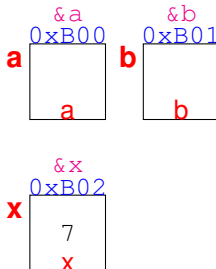
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



Mémoire : f

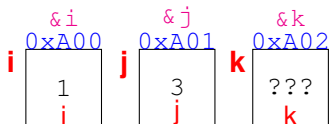


`return x;`

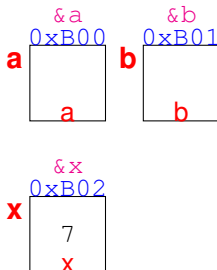
Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



Mémoire : f



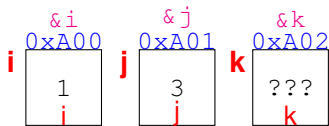
`return x;`

La valeur de `x` est retournée : 7

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     ▶ k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```

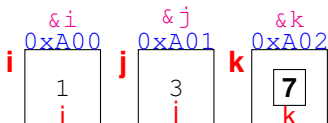


```
k=f(i,j);
```

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio .h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     ▶ k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```



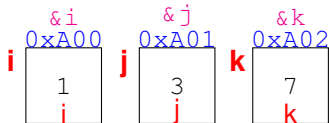
```
k=f(i,j);
```

`k` reçoit la valeur retournée par `f(i,j)` : 7

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio .h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10     int i=1,j=3,k;
11     k=f(i,j);
12     printf("k=%d\n",k);
13     return 0;
14 }
```

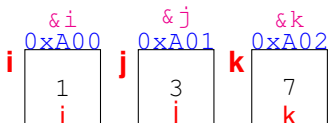


```
printf("k=%d\n",k);
```

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    ▶ printf("k=%d\n",k);
13    return 0;
14 }
```



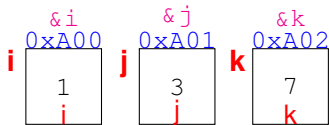
```
printf("k=%d\n",k);
```

Affichage de : k=7

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```

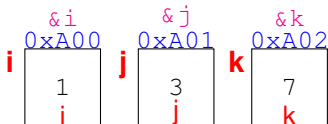


return 0;

Mémoire : main

Listing 4 – Exemple.c

```
1 #include <stdio.h>
2
3 int f(int a, int b){
4     int x;
5     x = a+2*b;
6     return x;
7 }
8
9 int main(){
10    int i=1,j=3,k;
11    k=f(i,j);
12    printf("k=%d\n",k);
13    return 0;
14 }
```



`return 0;`

La fonction main est terminée et retourne 0

Plan

- 1 Fonctions
 - Généralités
 - Syntaxe
 - Définition de fonctions
 - Prototype d'une fonction
 - Corps d'une fonction
 - Exemple de définition
 - Déclaration d'une fonction
 - Exemple

- 2 Exemples avec représentation mémoire
 - Exemple 1
 - Exemple 2

- 3 Pour aller plus loin...

Listing 4 – Exemple2.c

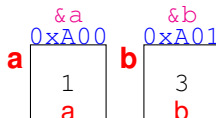
```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```

```
int a=1,b=3;
```

Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```



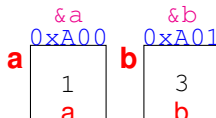
```
int a=1,b=3;
```

a initialisé à 1, b initialisé à 3.

Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    ▶ permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```

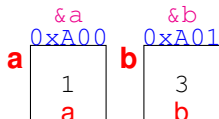


```
permut(&a,&b);
```


Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    ▶ permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```



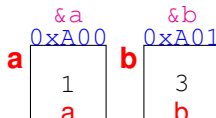
```
permut(&a,&b);
```

appel de la fonction `permut` avec `&a` \iff `0xA00` et `&b` \iff `0xA01`

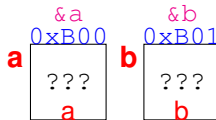
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut

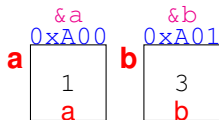


```
void permut(int *a, int *b){
```

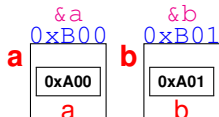
Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```



Mémoire : permut



```
void permut(int *a,int *b){
```

`a` est initialisé à `0xA00` (valeur de `&a` du `main`) et `b` est initialisé à `0xA01` (valeur de `&b` du `main`)

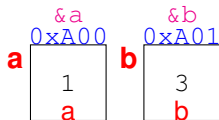
Listing 4 – Exemple2.c

```

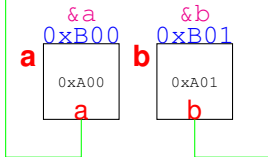
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }

```

Mémoire : main



Mémoire : permut



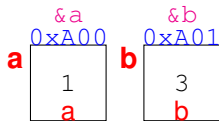
```
void permut(int *a,int *b){
```

*a (permut) \iff a (main) \iff 1 et *b (permut) \iff b (main) \iff 3

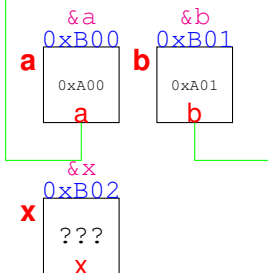
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10     permut(&a,&b);
11     printf("a=%d,b=%d\n",a;b);
12     return 0;
13 }
```

Mémoire : main



Mémoire : permut

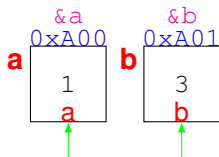


```
int x;
```

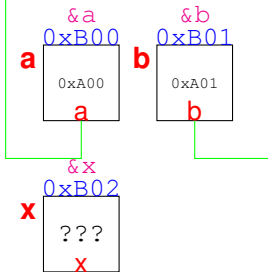
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut



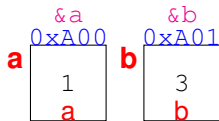
```
int x;
```

Déclaration de `x` (non initialisé)

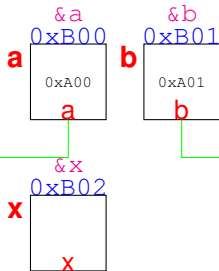
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1, b=3;
10    permut(&a, &b);
11    printf("a=%d, b=%d\n", a, b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut

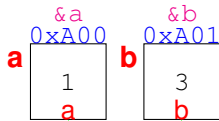


```
x=*a;
```

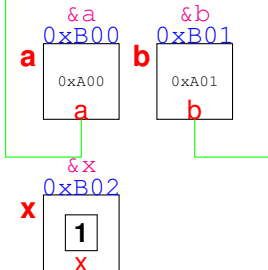
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1, b=3;
10    permut(&a, &b);
11    printf("a=%d, b=%d\n", a, b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut



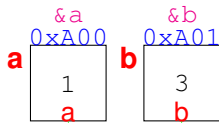
```
x=*a;
```

```
x ← *a=1
```

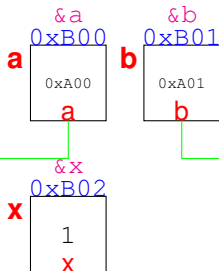

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut

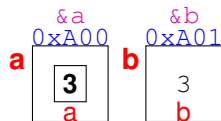


```
*a=*b;
```

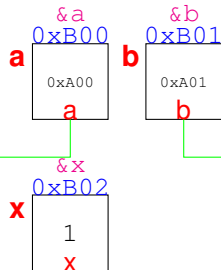
Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     ▶ *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    ■ permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```



Mémoire : permut



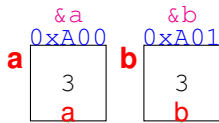
```
*a=*b;
```

La case mémoire pointée par **a** (i.e. **a** du main) reçoit la valeur contenue dans la case mémoire pointée par **b** (i.e. la valeur du **b** du main)

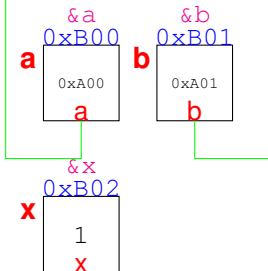
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut

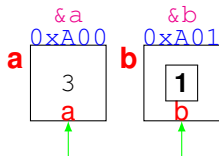


```
*b=x;
```

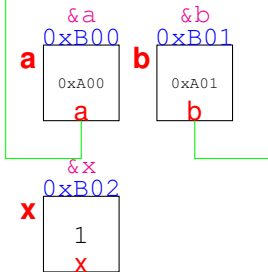
Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```

Mémoire : main



Mémoire : permut



```
*b=x;
```

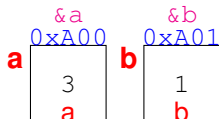
La case mémoire pointée par **b** (i.e. **b** du main) reçoit **x** (=1)

Listing 4 – Exemple2.c

```

1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    ▶ permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }

```



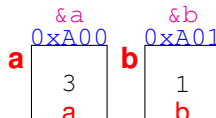
```
permut(&a,&b);
```

Fin de l'appel de la fonction `permut` : retour au `main`

Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```

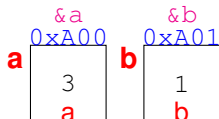


```
printf("a=%d,b=%d\n",a,b);
```

Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a,b);
12    return 0;
13 }
```



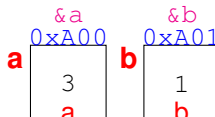
```
printf("a=%d,b=%d\n",a,b);
```

Affichage de : a=3,b=1

Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```

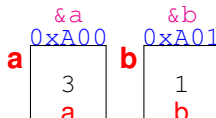


```
return 0;
```


Mémoire : main

Listing 4 – Exemple2.c

```
1 #include <stdio.h>
2 void permut(int *a, int *b){
3     int x;
4     x=*a;
5     *a=*b;
6     *b=x;
7 }
8 int main(){
9     int a=1,b=3;
10    permut(&a,&b);
11    printf("a=%d,b=%d\n",a;b);
12    return 0;
13 }
```



return 0;

La fonction main est terminée et retourne 0

Extern (1/2)

Une fonction ne peut-être définie à l'intérieur d'une autre fonction : une fonction est donc toujours externe.

```
1 | extern int max(int a, int b)
2 | {
3 |     return a>b ? a : b;
4 | }
5 |
6 | int max(int a, int b)
7 | {
8 |     return a>b ? a : b;
9 | }
```

Les deux définitions sont équivalentes.

Extern (2/2)

```
1 | extern int max(int a, int b);  
2 |  
3 | int max(int a, int b);
```

Les deux déclarations sont équivalentes.

Confidentialité des fonctions

Une fonction peut-être rendue confidentielle, c'est à dire inutilisable en dehors du fichier où elle a été définie en utilisant le mot clef `static`.

Passage d'arguments au **main**

```
1 #include <string.h>
2 int main(int argc, char *argv []){
3     void argument(char *s);
4     char **arg=argv;
5     arg++;
6     while (--argc)
7         argument(*arg++);
8 }
9 void argument(char *s){
10    printf("argument_=_%s, _longueur_=_%d\n", s, strlen(s))
```

- `argc` contient le nombre d'éléments du tableau pointé par `argv`.
- `argv` permet d'accéder aux différents arguments passés en paramètre au programme.

Pointeurs de fonctions

```
1  #include <string.h>
2
3  void f(int i){
4      printf("function_f:_%d\n", i);
5  }
6  void g(int i){
7      printf("function_g:_%d\n", i);
8  }
9
10 int main(void)
11 {
12     void (*p)(int);
13     p=f;
14     p(3);
15     p=g;
16     p(4);
17     return 0;
18 }
```