Initiation à la programmation C

Rachida Chakir

rachida.chakir@esiee.fr

3R-IN1A
2024-2025

ESIEE PARIS

Programme du cours

- Partie 1 : Les bases de la programmation en C
- Partie 2 : Pointeurs et fonctions
- Partie 3 : Tableaux et allocation mémoire
- Partie 4 : Structures et liste chaînées.

ESIEE PARIS

Pointeurs et Fonctions

- Les pointeurs
  - Définition et utilisation de base
  - Les erreurs et l'opérateur NULL
- Les fonctions : utilisation de base
  - Déclaration, définition, utilisation
  - Paramètres en entrée ou en sortie
- Les fonctions : utilisation avancée
  - Recursivité et itérations

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

- **Une variable a**
  - un nom, qu'on appelle **identifiant** et qui renvoie à une position dans la mémoire (**une adresse**)
  - un **type**, le type définit la façon dont il est représenté en mémoire
  - une **valeur**

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

identifiant	adresse	valeur
i	45325325	3
j	45325329	3

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Comment récupérer l'adresse d'une variable

Example

```
int age =10;
printf("La variable age vaut : %d ",age);
printf("L'adresse de la variable age est : %p", &age);
```

```
$ La variable age vaut : 10
$ L'adresse de la variable age : 0028FF1C
```

Donc à retenir

- **age** : désigne la valeur de la variable
- &**age** : désigne l'adresse de la variable

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

- Il existe deux façons d'accéder au contenu d'une variable

Adressage direct

Par le nom de la variable.

Adressage indirect

En passant par un pointeur qui contient l'adresse de la variable

Un pointeur c'est quoi ?

Un pointeur est une variable spéciale contenant l'adresse d'une autre variable d'un type donné
Si un pointeur *p* contient l'adresse d'une variable *a*, on dit que

"p pointe sur a"

Les pointeurs : Définition et utilisation de base

Les pointeurs ont-ils un type ?

- Il n'y a pas de type "pointeur" comme il y a un type *int* ou *float*. On n'écrit donc pas

$$\text{pointeur p}$$

- On utilise le symbole $*$, mais on continue à indiquer quel est le type de la variable dont le pointeur va l'adresse.

Example

```
int a = 10;
int* p = &a;
```

- Comme le pointeur *p* va contenir l'adresse de la variable *a* (qui est de type *int*), alors le pointeur doit être de type *int*$*$
- Si la variable *a* avait été de type *float*, alors on aurait du écrire *float* $*$ *p*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Lors du travail avec des pointeurs, nous avons besoin de l'opérateur

- & pour obtenir l'adresse d'une variable.
- * derrière le type pour déclarer un pointeur,
- * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans le pointeur

Exemple de déclaration d'un pointeur

```
int* p;
```

- *p est de type *int*
- p est un pointeur sur *int*
- p peut contenir l'adresse d'une variable de type *int*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

- Déclaration d'un pointeur sur un *int*

```
int * p;
```

- Convention d'initialisation avec *NULL*

```
p = NULL;
```

- Affectation avec une adresse de variable

```
int a;
p = & a;
```

- Déréférencement d'un pointeur

```
int b;
b = *p;
```

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exemple basique

```
int * p;
int a = 4;
```


```
p = &a;
int b;
b  = *p
```

p « pointe » sur ***a*** et ***&a*** est l'adresse de ***a***

****p*** : contenu de la variable « pointée » par ***p*** ou « déréférencement » de ***p***

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

```
float a = 0.5 ;
int *p = &a;
```

Erreur de compilation : un pointeur est un pointeur des variables d'un type donné

```
float a = 0.5 ;
float* p = &a ;
```

Une variable de type *truc** pointe sur un *truc*

Si un pointeur *p* pointe sur une variable *a*, alors **p* peut être utilisé partout où l'on peut écrire *a*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

A retenir

Example

```
int a = 5;
int* p = & a;
printf("a= %d et *p = %d", a, *p)
```

- Le nom d'une variable reste toujours lié à la même adresse.
  - *a* signifie : "je veux la valeur de la variable *a*"
  - *&a* signifie : "je veux l'adresse à laquelle se trouve la variable *a*"
- Un pointeur est une variable qui peut pointer sur différentes adresses (on peut modifier la variable vers laquelle le pointeur pointe)
  - *p* signifie " je veux la valeur de *p* ", cette valeur étant une adresse
  - $*p$ signifie " je veux la valeur de la variable qui se trouve à l'adresse contenue dans le pointeur *p*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

A retenir

Ne pas confondre les différentes significations de l'opératoire étoile !

- Lorsque on **déclare** un pointeur, l'étoile sert à juste à indiquer qu'on veut créer un pointeur

$$\text{int* p}$$

- En revanche, lorsqu'on **utilise** le pointeur en écrivant

$$\text{printf(“\%”, *p)};$$

cela ne signifie pas qu'on "veut créer un pointeur" mais que l'on "veut la valeur de la variable sur laquelle pointe le pointeur *p*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=1,*p=2*

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=1,*p=2*

```
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=1,*p=2*

```
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3,*p=2*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=1,*p=2*

```
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3,*p=2*

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

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=1, b=1,*p=1*

```
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=1,*p=2*

```
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3,*p=2*

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

*a=4, b=3,*p=3*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = ; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
p = &b; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3,*p=3*

```
a = 4; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = ; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
p = &b; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3, *p=3*

```
a = 4; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=4, b=3, *p=3*

Les pointeurs : Définition et utilisation de base

Exo 1 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = ; int* p = &a; int b = *p;
printf("a = %d, b = %d, *p = %d \n", a, b, *p);
a = 2; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
b = 3; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
p = &b; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=2, b=3,*p=3*

```
a = 4; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=4, b=3,*p=3*

```
b = 5; printf("a = %d, b = %d, *p = %d \n", a, b, *p);
```

*a=4, b=5,*p=5*

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 1, b = 2, *p = 1, *q=1*

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 1, b = 2, *p = 1, *q=1*

```
a = 3;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 1, b = 2, *p = 1, *q=1*

```
a = 3;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 3, b = 2, *p = 3, *q=3*

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 1, b = 2, *p = 1, *q=1*

```
a = 3;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 3, b = 2, *p = 3, *q=3*

```
b = *q + 10;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

ESIEE PARIS

Les pointeurs : Définition et utilisation de base

Exo 2 : Après chaque ligne, donner la sortie du bloc suivant.

```
int a = 1; int b = 2; int* p = &a; int* q = p;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 1, b = 2, *p = 1, *q=1*

```
a = 3;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 3, b = 2, *p = 3, *q=3*

```
b = *q + 10;
printf("a=%d, b=%d, *p=%d, *q=%d \n", a, b, *p, *q);
```

*a = 3, b = 13, *p = 3, *q=3*

ESIEE PARIS

Le pointeurs : les erreurs et l'opérateur NULL

Un pointeur pointe sur

- La variable adéquate et on peut s'en servir
- Une variable inadéquate et le programme est incorrect

Le pointeurs : les erreurs et l'opérateur NULL

Les erreurs

- Un pointeur non initialisé sur une adresse de variable ne peut être déréférencé.

```
int* p;
int b;
b = *p;
```

```
int* p = NULL;
int b;
b = *p;
```

$\Rightarrow$ ERREUR D'EXÉCUTION

Le pointeurs : les erreurs et l'opérateur NULL

Utilisation de *NULL*

- On affecte *NULL* quand un pointeur ne sert plus ou pas encore
- Avant d'utiliser un pointeur on teste sa valeur

```
if (p != NULL)
{... b = *p; ... }
else
{...}
```

Avec cette convention, un pointeur pointe sur

- La variable adéquate
- *NULL*

Le pointeurs : les erreurs et l'opérateur NULL

Le pointeur est mis à *NULL*

- Après utilisation

```
p = &a;
...
b = *p;
...
p = NULL;
```

- A la déclaration (si non simultanée de l'affectation)

```
int* p = NULL;
...
p = &a;
```

ESIEE PARIS

Le pointeurs

En résumé

- Chaque variable est stockée à une adresse précise en mémoire
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre, ils stokent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole & devant un **nom de variable**, on obtient son **adresse** au lieu de sa valeur
- Si on place un symbole $*$ devant un **nom de pointeur**, on obtient la **valeur** de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début.
  Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autre notions sont basées dessus.

ESIEE PARIS

Les fonctions : Utilisation de base

En C il faut distinguer,

- **La déclaration d'une fonction**
  qui est une instruction fournissant au compilateur un certain nombre d'information concernant une fonction, **qui déclare son existence**
- **La définition d'une fonction**
  qui revient à écrire le **corps** de la fonction, qui **définit** donc les traitements effectués dans le bloc { } de la fonction.
- **L'appel d'une fonction** qui est son utilisation

Les fonctions : Utilisation de base

Déclaration de fonction

Une fonction se caractérise par

- son **nom** (un identificateur)
- sa **liste de paramètre(s)** : le nombre et le type de paramètre(s) (la liste peut être vide)
- son **type de retour** (un seul résultat)

Les fonctions : Utilisation de base

Déclaration

```
float convertKmEnMiles (float x);
```

Définition

```
float convertKmEnMiles (float x)
{
   return x*0.621371;
}
```

Utilisation (appel)

```
float km = 5;
float mi = convertKmEnMiles(x);
```

ESIEE PARIS

Les fonctions : Utilisation de base

Définition d'une fonction

```
float km_to_miles (float x );
```

- Type de retour (sortie)
  ```
  float convertKmEnMiles (float );
  ```
- Type du paramètre (entrée)
  ```
  float convertKmEnMiles (float);
  ```
- Nom de la fonction
  ```
  float convertKmEnMiles(float);
  ```

ESIEE PARIS

Les fonctions : Utilisation de base

Valeur de retour

- Une fonction fournit **un résultat**. Pour cela, on lui déclare un **type de retour** et on renvoie une valeur (du type déclaré !) avec l'instruction **return**
- L'instruction **return** provoque immédiatement la sortie de la fonction appelée.
- La valeur de retour peut servir à renvoyer un *résultat* ou *un état sur l'exécution* de la fonction.

Pour une **procédure** son type de retour est **void** car elle ne retourne aucune valeur.

Les fonctions : Utilisation de base

Types de retour

- Les types pré-définis pour les variables :
  - *int*, *float*, *double*, *char*, ...
  - *int**, *float**, *double**, *char**, ...
- Le type vide :
  - *void*

Les fonctions : Utilisation de base

Exemple : Calcul du carré d'un nombre

```
# include < stdio.h >

int carre (int a) { /* Definition */
   return (a*a); /* retour */
}

int main ()
{
   int n, x = 0;
   printf("Saisir un entier \n");
   scanf("%d", &n);
   x = carre(n); /* appel */
   printf("carre = %d \n, x);
   return 0;
}
```

ESIEE PARIS

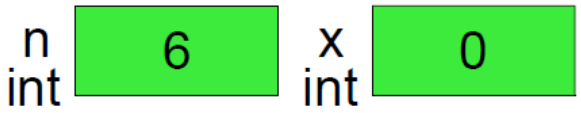
Les fonctions : Utilisation de base

Exécution

```
chakir@pmlv800838-ubuntu :~ $ ./test.exe
Saisir un entier
6
carre = 36
```

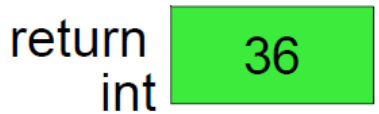
Comment `carre` marche ?

- Après le `scanf`:



- Appel de la fonction:

- Exécution du `return`



- Retour dans l'appelant

ESIEE PARIS

Les fonctions : Utilisation de base

Procédures - Déclaration

```
void ma_procedure (char);
```

- Pas de paramètre de retour
  ```
  void ma_procedure (char);
  ```

Procédure - Appel

```
char c;
ma_procedure (c); /* appel */
```

Les fonctions : Utilisation de base

Example

```
void afficherCaracteres(char caractere, int nb)
{   for (i = 0; i < nb; i++)
  {
    printf("%c", caractere);
  }
  printf("\ n");
}


/* Pour afficher 5 dièses */
afficherCaracteres('♯',5);
```

Les fonctions : Utilisation de base

Les variables et les paramètres

- les variables déclarées à l'intérieur d'une fonction ne sont accessible que dans cette fonction et pas à l'extérieur. Ces variables sont supprimées de la mémoire une fois que la fonction renvoie son résultat
- Les noms des arguments figurant dans l'en-tête de la fonction se nomment des *paramètres formels*. Leur rôles est de permettre, au sein du corps de la fonction, de décrire ce qu'elle doit faire
- Les arguments fournis lors de l'utilisation (l'appel) de la fonction se nomme les *les paramètres effectifs*. On peut utiliser n'importe quelle expression comme argument effectif.

ESIEE PARIS

Les fonctions : Utilisation de base

Passage des paramètres par valeur (copie) ou par adresse (référence)

- En langage C, il n'existe pas d'autre paramètre de sortie que le paramètre de retour *return*
- Passage de paramètre *par valeur*
  - On passe la valeur du paramètre (on dit aussi passage *par copie*)
  - Implique que le paramètre est en entrée

```
int carre1 (int i) {
   i = i*i;
   return i;
}
```

Les fonctions : Utilisation de base

Exemple : Passage de paramètre par valeur (copie)

```
# include <stdio.h>
int carre1 (int i) {
   i = i*i;
   return i;
}

int main (){
   int x = 5, y = 0;
   printf(" x = %d et y = %d\n", x,y);
   y = carre1(x);
   printf(" x = %d et y = %d\n", x,y);
   return 0;
}
```

- Que vaut x et y après l'appel de la fonction carre1 ?

Les fonctions : Utilisation de base

Fonctions avec plusieurs résultats à renvoyer

- On voudrait écrire une fonction à laquelle on donne une durée exprimée en minutes, et qui renvoie le nombre d'heure et minutes correspondantes.
- Par exemple, si on lui donne 45 la fonction renvoie 0 pour le nombre d'heure et 45 pour le nombre de minute

Problèmes

- On ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

ESIEE PARIS

Les fonctions : Utilisation de base

Passage des paramètres par adresse (référence)

- On passe l'adresse du paramètre
- Cela permet d'avoir un paramètre en sortie
- Comment ?

Les fonctions : Utilisation de base

Passage des paramètres par adresse (référence)

- On passe l'adresse du paramètre
- Cela permet d'avoir un paramètre en sortie
- Comment ?
  → En utilisant un pointeur

Le gros intérêt des pointeurs (mas ce n'est pas le seul) est qu'on peut les envoyer à des fonctions pour pour qu'elles modifient directement une variable en mémoire et non une copie comme on l'a déjà vu auparavant.

Les fonctions : Utilisation de base

Passage des paramètres par adresse (référence)

- On passe l'adresse du paramètre
- Cela permet d'avoir un paramètre en sortie
- Comment ?
  → En utilisant un pointeur

```
void carre2 (int i, int* j) {
   *j = i*i;
}
```

- `int*` : pointeur sur un *int*
- `*j` : déréférencement de `j`

  (ou contenu de la variable "*pointée*" par `j`)

ESIEE PARIS

Les fonctions : Utilisation de base

Exemple : Passage de paramètre par adresse

```
# include <stdio.h>
void carre2 (int i, int* j) {
   *j = i*i;
}

int main (){
   int x = 5, y = 0;
   printf(" x = %d et y = %d", x,y);
   carre2(x,&y);
   printf(" x = %d et y = %d\n", x,y);
   return 0;
}
```

- Que vaut x et y après l'appel de la procédure carre2 ?

ESIEE PARIS

Les fonctions : Utilisation de base

Exemple d'erreur

```
# include <stdio.h>
void carre3 (int i, int j) {
   j = i*i;
}

int main (){
   int x = 5, y = 0;
   printf(" x = %d et y = %d", x,y);
   carre3(x, y);
   printf(" x = %d et y = %d\n", x,y);
   return 0;
}
```

- Que vaut x et y après l'appel de la procédure carre3 ?
- Pourquoi la procédure carre3 n'est pas correcte

ESIEE PARIS

Les fonctions : Utilisation de base

Et si on revenait à notre exercice de départ

```
void decoupeMinute (int fduree, int* pHeure, int* pMinutes);

int main ()
{
  int duree = 0, heure = 0, minutes = 0;
  printf(" Donner le nombre de minutes : \ n ");
  scanf("%d", &duree);
  decoupeMinute(duree,&heure,&minutes);
  printf(" %d heures et %d minutes"," heures, minutes);
  return 0;

void decoupeMinute (int fduree, int* pHeure, int* pMinutes)
{
  *pHeure = fduree / 60;
  *pMinutes = fduree % 60;
}
```

ESIEE PARIS

Les fonctions : utilisation avancées

Récursivité

- C'est une technique qui permet à une fonction de **s'auto-appeler**

fonction factorielle récursive

```
int factorielle (int n) {
   if (n == 0)
     return 1;
   return (n*factorielle(n-1));
}
```

$$f(n-1) = 1 \times 2 \times \cdots \times n-2 \times n-1$$

$$f(n) = 1 \times 2 \times \cdots \times n-2 \times n-1 \times n$$

$$f(n) = n \times f(n-1)$$

- Comment ça fonctionne ?

Les fonctions : utilisation avancées

fonction factorielle itérative

```
int factorielle2 (int n) {
   int i, r = 1;
   for (i = 1; i <= n; i++)
     r*=i ;/* <=> r = r*i; */ ;
   return r;
}
```

$$f(n) = 1 \times 2 \times \cdots \times n-2 \times n-1 \times n$$

Les fonctions : Utilisation de base

Example

```
void afficherCaracteresRecurs(char caractere, int nb)
{
  if (nb > 0))
  {
    printf("%c", caractere);
    afficherCaracteresRecurs(caractere,nb-1));
  }
   else printf("\ n");
}


/* Pour afficher 5 dièses */
afficherCaracteres('#',5);
```

Les fonctions : utilisation avancées

MACRO

- Définition
  ```
  #define MA_MACRO (X,Y) X + X*Y
  ```
- Utilisation
  ```
  int i = MA_MACRO (4,3)
  ```
- Le compilateur (par exemple *gcc*) remplace `MA_MACRO(a,b)` par `a + a*b` partout où la macro apparaît dans le fichier source

En résumé

- Un pointeur est une variable dont le contenu est une adresse
- L'opérateur d'adressage & permet de récupérer l'adresse d'une variable
- Un pointeur d'un type peut uniquement contenir l'adresse de l'objet du même type.
- L'opérateur '*' permet d'accéder à l'objet référencé par un pointeur
- Une fonction peut retourner ou non une valeur et recevoir ou non des paramètres ;
- Une première utilisation des pointeurs est, pour des fonctions, le passage de paramètre par référence (adresse)

ESIEE PARIS

Questions de révision

Quelles sont les quatre partie d'une fonction ?

- Type de retour
- Le nom
- Liste des paramètres
- Le corps de la fonction en les accolades $\{...\}$

Comment s'appelle une fonction qui ne retourne aucun résultat ?

- Une procédure et son type de retour est void

ESIEE PARIS