INPT
LA GRANDE ÉCOLE
DES TÉLÉCOMMUNICATIONS
ET DES TECHNOLOGIES
DE L'INFORMATION

Filière Smart-ICT

Algorithmique et Programmation C

Mr N.EL FADDOULI
elfaddouli@emi.ac.ma
nfaddouli@gmail.com

Année Universitaire:2024/2025

1

Les fonctions

125

Le langage C: Les fonctions

- Pour écrire un programme qui traite un problème complexe, on le décompose en plusieurs sous-problèmes simples.
- Chaque **sous-problème** sera résolu par un **sous-programme** appelé **fonction** qui prend en entrée un ensemble de données ou **paramètres** et fournit **0** ou **une** valeur de sortie ou **valeur de retour**.
- Avantages:
  - Facilité de maintenance *(détection facile des erreurs, …).*
  - Réutilisation:
    - ♦ Absence de répétition de bloc d'instructions.
    - ♦ Le partage et la réutilisation.

Le langage C: Les fonctions – Déclaration (1/2)

- Une fonction est généralement déclarée avant le programme principal ou dans un fichier **.h** à inclure.
- La déclaration consiste à préciser le type de retour, le nom et les types des paramètres de la fonction:

  type_de_retour *nom_fonction* ($type\_param_1$, $type\_param_2$, ... , $type\_param_n$) ;

  ⚠ *Les noms des paramètres sont optionnels.*
- Si la fonction n'a pas de valeur de retour (*procédure*) on utilise le type **void** pour celui de retour.
- Exemple:

```
float  div_reelle(int ,int);  /* division réelle de 2 entiers */
void   menu(void);            /* affichage d'un menu de choix */
void   table(int);            /* table de multiplication d'un entier*/
```

Le langage C: Les fonctions – Déclaration (2/2)

- Une fonction doit être **définie ou déclarée avant son appel.**
- ⚠ Le programme principal est une fonction particulière notée **main()** qui représente l'entrée du programme.
- Compilateur se base sur la déclaration des fonctions pour détecter des erreurs lors de leur appel: *nombre de paramètres incorrect*, *incompatibilité des types de paramètres* ou *celui de la valeur de retour*.

Le langage C: Les fonctions – Définition et appel

- La définition d'une fonction consiste à donner des **noms à ses paramètres** et préciser son **corps** .

type_de_retour nom_fonction (*type_param*$_1$ *param*$_1$, ... , *type_param*$_n$ *param*$_n$)

{ ***<Déclaration de variables locales>***
<Bloc d'instructions>
}

- Exemple:

```
void afficher(void);
int  triple(int);
void main()
   {int a, b;
    afficher();
    printf("Entier:"); scanf("%d",&a);
    b= triple(a);  printf("Triple de %d=%d\n",a,b);
    printf("Triple de %d=%d\n", b, triple(b) );   }
```

```
void afficher(void)  /* aucun paramètre ni résultat */
   {printf("bonjour");  }
int triple (int n)        /* triple d'un entier */
    {  int r;
       r = n*3;
      return r ;   /* résultat retourné par la fonction */
    }
```

Le langage C: Les fonctions - Exercices

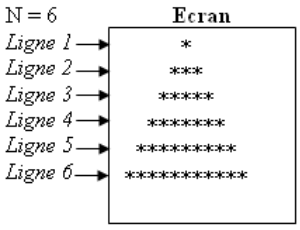
☞ Définir et appeler les fonctions suivantes

1) Calculer et retourner X^Y.

2) Calculer et retourner N!

3) Afficher le triangle de Pascal de hauteur N.

Exemples: N = 6



Le langage C: Les fonctions – Porté des variables (1/2)

☞ Dans un programme C, une variable peut être **globale**, **locale** ou **statique**.

☞ Une variable **globale** est déclarée en dehors de toute fonction. Elle est accessible dans toutes les fonctions du programme **définies** après sa déclaration.

☞ Une variable **locale** est déclarée dans une fonction ou un bloc d'instructions où elle peut être accessible. C'est une variable **locale automatique**.

☞ Exemple:

```
void f();
int N=3;
void g();
void  main(){
  int N=9 ; float k;
  printf("%d \n",N);
  f(); g();
}
void f(){ printf("%d\n", N );
          printf("%d\n", A);
        }
int A=10;
void g(){printf("%d %d\n",N, A);}
```

C'est la variable locale N dans main()

ERREUR

```
int N=3;
void  main(){
  int N=9 ; float k;

  printf("%d \n",N);
  {
   extern int N;
   printf("%d\n", N);
  }

}
```

*C'est la variable **globale** N qui sera utilisée*

Le langage C: Les fonctions – Porté des variables (1/2)

- Par défaut, une variable **locale est automatique**: son espace mémoire est alloué à l'entrée de la fonction (ou bloc) et libéré à la sortie.
- Une variable **statique** une variable **locale** qui garde sa valeur d'un appel à un autre de la fonction où elle déclarée.
- Exemple:

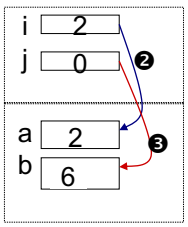
```
int f();
void  main(){  int x;
x=f();  printf("Résultat: %d\n", x);
x=f();  printf("Résultat: %d\n", x);
x=f();  printf("Résultat: %d\n", x);
}
void f(){ static int  a=0;
          a++; return a;   }
```

```
Résultat: 1
Résultat: 2
Résultat: 3
```


Le langage C: Les fonctions – Passage des arguments par valeur

- Lors de l'appel d'une fonction, on lui passe des **arguments** (*paramètres effectifs*) dont les valeurs seront utilisées par la fonction. ⇨ **Les arguments gardent leurs valeurs initiales après l'appel de la fonction**.
- Exemple:

```
void  triple (int, int);
main( )
 {    int i = 2, j = 0;
  ❶ triple ( i, j );
  ❺ printf ("Après j=%d", j);
 }
void triple (int  a, int  b)
 ❹{
      b = 3*a;
      printf (" b=%d",b);
 }
```




```
b = 6
après j = 0
```

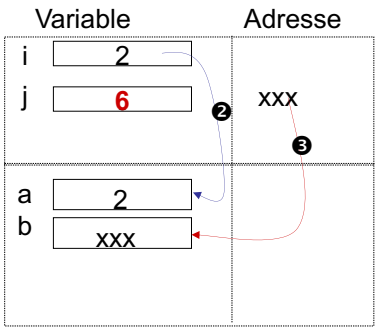
Le langage C: Les fonctions – Passage des arguments par adresse

- Lors de l'appel d'une fonction, les **arguments** sont des **adresses** (*pointeurs*)

  ⇨ **Le contenu pointé peut être modifié par la fonction**.

- Exemple:

```
void  triple (int, int *);
main( )
{     int i = 2, j = 0;
    ❶ triple ( i, &j );
    ❺ printf ("Après j=%d", j);
 }
void triple (int  a, int  *b)
  {❹
       *b = 3 * a;
       printf (" b=%d",*b);
  }
```




```
b = 6
après j = 6
```


134

Le langage C: Les fonctions – Plusieurs valeurs fournies

- Une fonction retourne au maximum une valeur indiquée par l'instruction **return**.
- Comment une fonction peut fournir plus qu'une valeur ?

  ⇨ **On utilise des paramètres passés par adresse**

- Exemple: Une fonction qui fournit la somme et le produit de deux entiers

```
int  Calcul (int, int, int *    );
main( )
{ int x, y, s, p;
  scanf("%d%d", &x,&y);
  p = Calcul (x, y, &s  );
  printf ("s= %d, p=%d\n", s, p);
  }
int  Calcul (int  a, int  b,  int *c   )
  { *c  = a+b;
       return a*b;
  }
```

```
void  Calcul (int, int,  int  *, int *  );
main( )
{ int x, y, s, p;
  scanf("%d%d", &x, &y);
  Calcul (x, y,  &s, &p   );
  printf ("s= %d, p=%d\n", s, p);
  }
void  Calcul (int  a, int  b, int *c, int *d )
  {   *c= a+b;
      *d= a*b;
  }
```


135

Le langage C: Les fonctions – Paramètres de type tableau

- On peut passer à une fonction un **argument de type tableau** comme étant **un pointeur sur le premier élément**. ⇨ **Un tableau est passé automatiquement par adresse**
- Exemple: Une fonction qui inverse l'ordre d'un tableau

```
void  Inverser (int *T, int );       /* ou   int T[ ]  */
main( )
{ int M[ ] ={6, 7, 8, 9, 10}, i ;
  Inverser (M, 5 );
 for (i=0 ; i<5 ; i++)    printf("%d\n", M[ i ] );
 }
void  Inverser (int  *T, int  N )   /* ou   int T[ ]  */
 {   int k, i;
    for(i=0;i<N/2 ; i++){ k=T[ i ]; T[ i ]=T[N-i-1];
                          T[ N-i-1 ]= k; }
   }
```


Le langage C: Fonction retournant un pointeur (1/3)

- Une fonction peut retourner un pointeur d'un type donné:

  type_ptr_retour * *nom_fonction*(*type_param*$_1$, ..., *type_param*$_n$) ;

- Il faut que le **pointeur retournée** pointe sur une **donnée valide** qui est toujours disponible **après la fin de la fonction.**

  ⇨ elle ne doit pas être une variable **locale automatique** (**≠ statique**)

  ⚠ *Une variable locale automatique est détruite après la fin de la fonction*

- Exemple: **Fonction invalide**

```
int *  test (void) ;
int main(void) {
        int *p = test ();
        *p = 10;
        printf("%d\n", *p);
        return 0;
        }
```

⚠ *La fonction **main** utilise, via p, une **zone invalide***

```
int * test (void) {
        int n=4;
        return &n;
        }
```

⚠ *La fonction **test** retourne une adresse d'une variable qui est détruite (son espace est considéré libre par le système d'exploitation)*

Le langage C: Fonction retournant un pointeur (2/3)

☞ Exemple: La fonction retourne un pointeur sur une variable **locale statique** *(elle ne sera pas détruite à la fin de la fonction)*

```
int *  test (void) ;

int main(void) {   int *p = test ();
                   printf("%d\n", *p);
                   *p = 10;
                   p = test ();
                   printf("%d\n", *p);
        return 0; }
```

⚠ *La fonction **main** a modifié la variable locale statique **a** de la fonction **test**.*

Résultat

```
1
11
```

```
int * test (void) {
        static int a=0;  a++;
        return  &a;
        }
```

⚠ *La variable **a** est statique ⇨ elle sera conservée d'un appel à un autre.*
Elle sera créée au premier appel.

138

Le langage C: Fonction retournant un pointeur (3/3)

☞ Exemple: La fonction retourne un **pointeur sur une zone mémoire allouée dynamiquement** dans la fonction *(elle ne sera pas détruite à la fin de la fonction)*

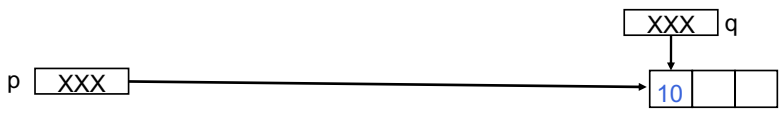
```
int *  test (void) ;

int main(void) {   int *p = test ();
                   *p = 10;
                   printf("%d\n", *p);
                    free(p) ;
        return 0; }
```

⚠ *La fonction **main** modifie la première case parmi les 3 créées par la fonction **test**.*

```
int * test (void) {
        int *q=(int *) malloc (3 * sizeof(int));
        return q;
        }
```

⚠ *La fonction **test** alloue une zone mémoire de 3 cases contiguës pour stocker 3 entiers et retourne l'adresse de la première case*

⚠ *La variable q sera détruite à la fin de l'appel mais l'espace réservé sera conservé*



139

Le langage C: Les fonctions – Exercices

☞ Ecrire les fonctions permettant de:

1) Calculer et retourner l'inverse d'un entier en inversant l'ordre de ses chiffres.

(**Exemple**: L'inverse de 4396 est 6934)

2) Déterminer l'indice de la première occurrence d'un élément X dans un tableau d'entiers T (non trié).

3) Identique à (2) mais pour un tableau T trié.

4) calculer la somme de deux entiers **sans valeur de retour**. La somme sera dans **l'un** des arguments de la fonction.

5) retourner la première position d'un entier dans un tableau trié.

6) déterminer le nombre d'occurrences et l'indice de la première occurrence d'un entier dans un tableau d'entiers triés dans l'ordre croissant.

7) retourner la somme de la moitié supérieure d'une matrice (nxn) par rapport à la diagonale principale.

8) détermine si une matrice (nxn) est symétrique ou non par rapport à la diagonale principale.

Le langage C: Les fonctions récursives (1/2)

☞ Exemple de calcul récursif: calcul de n!

3 ! = 1*2*3 = **2!** *3

4 ! = 1*2*3*4 = **3!** * 4

5 ! = 1*2*3*4*5 = **4!** * 5

n ! = 1*2*3*4*…*(n-2)*(n-1)*n = **(n-1)!** * n

et **1 !** = 1 et **0 !** = 1

Par Fonction Itératif:

```
int  Factoriel (int    n)
    { int  F= 1, i=1;
      while ( i <= n )
            {
               F = F * i ;
               i = i + 1;
             }
     return F;
    }
```

Factoriel(n) = n! = { 1 Si n=0 ; n * **Factoriel(n-1)** si n > 0

définition récursive.

⇨ Chaque cas est réduit à un cas plus simple.

Par Fonction récursive:

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

Le langage C: Les fonctions récursives (2/2)

- Une fonction récursive fait **appel à elle-même directement** ou **indirectement**
- La récursivité est **indirecte** ou **croisée** lorsque, par exemple, deux fonctions s'appellent mutuellement.
- Une fonction récursive est calculable en un temps fini grâce à une condition d'arrêt qui permet d'éviter la récursivité à l'infini.
- Lorsqu'elle est bien utilisée, la récursivité rend la programmation plus facile.
- La récursivité est utile pour définir tout type de fonction (*mathématique ou autre*)

Exemple: Inverser une chaîne

1- Inverse ('ABCD') = Inverse ('BCD') +'A'
2- Inverse ('BCD') = Inverse ('CD') + 'B'
3- Inverse ('CD') = Inverse ('D') + 'C'
4- Inverse ('D') = 'D'

En remontant on obtient :
3- 'D' + 'C'= 'DC'
2- 'D' + 'C' + 'B' = 'DCB'
1- 'D' + 'C' + 'B' + 'A' = 'DCBA'

