



## COURS n°5 STRUCTURER DANS UN PROGRAMME Fonctions

---

---

---

---

---

---

---

### ORGANISER Faire des FONCTIONS

- 1- Pourquoi faire SES FONCTIONS ?
- 2- Comment créer les FONCTIONS
- 3- Créer une FONCTION sans échange de donnée
- 4- Nécessité d'échanges de données entre FONCTIONS
- 5- Créer une FONCTION avec données en Entrée
- 6- Créer une FONCTION avec résultat en Sortie



---

---

---

---

---

---

---

### ORGANISER Faire des FONCTIONS

- 1- Pourquoi faire SES FONCTIONS ?



---

---

---

---

---

---

---

```

int main()
{
    int choixOp ;
    float a, b;

    // boucle de reprise générale du menu
    do {
        printf("n1 - addition n2 - soustraction\n");
        printf("n3 - multiplication n4 - division n5 - Sortir");
        scanf("%d", &choixOp);

        if (choixOp==1) // addition
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Addition : %.2f\n", a+b);
        }
        else if (choixOp==2) // soustraction
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Soustraction : %.2f\n", a-b);
        }
        else if (choixOp==3) // multiplication
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Multiplication : %.2f\n", a*b);
        }
        else if (choixOp==4) // division
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            if (b!=0) printf("n Division : %.2f\n", a/b);
            else printf("n Erreur");
        }
        else if (choixOp==5)
        {
            printf("n Erreur choix menu");
        }
    } while (choixOp != 5);
}
    
```

**FONCTION principale:**

- TROP longue
- TROP répétitive

---

---

---

---

---

---

---

---

---

---

```

int main()
{
    int choixOp ;
    float a, b;

    do {
        // boucle de reprise générale du menu
        printf("n1 - addition n2 - soustraction\n");
        printf("n3 - multiplication n4 - division n5 - Sortir");
        scanf("%d", &choixOp);

        if (choixOp==1) // addition
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Addition : %.2f\n", a+b);
        }
        else if (choixOp==2) // soustraction
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Soustraction : %.2f\n", a-b);
        }
        else if (choixOp==3) // multiplication
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            printf("n Multiplication : %.2f\n", a*b);
        }
        else if (choixOp==4) // division
        {
            // Saisie valide des opérandes
            printf("n Donnez les opérandes : ");
            scanf("%f%f", &a, &b);
            if (a==50 || b==50) printf("erreur de saisie");
            while (a==50 || b==50);

            if (b!=0) printf("n Division : %.2f\n", a/b);
            else printf("n Erreur");
        }
        else if (choixOp==5)
        {
            printf("n Erreur choix menu");
        }
    } while (choixOp != 5);
}
    
```

**2 FONCTIONS :**

- Longueur optimisée
- Plus lisible
- Mieux pour les tests

---

---

---

---

---

---

---

---

---

---

**Pourquoi faire des FONCTIONS ?**

**E** → Chaque fonction réalise une des tâches du programme et contient ses propres données et instructions → **S**

- ⚡ Code source plus court, plus lisible, plus facile à tester
- ⚡ Code source structuré

**Règle de programmation : modulariser le code**

- Organiser le code en fonctions, de façon logique.
- Traitements répétitifs écrits une seule fois dans une fonction, réutilisable par simple appel.

---

---

---

---

---

---

---

---

---

---

## ORGANISER Faire des FONCTIONS

2- Comment créer les FONCTIONS



---

---

---

---

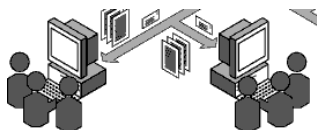
---

---

---

## DECOMPOSER le PROBLEME en Fonctions

### Simulateur de transmission numérique entre 2 ordinateurs



**Julien et Mathéo échangent des e-mails à partir de leur ordinateurs :**

Mais que se passe-t-il entre le moment où Julien clique sur "envoyer" et le moment où Mathéo voit s'afficher le message dans sa boîte mail ?

→ **ANALYSE PAR APPROCHE DESCENDANTE...**



---

---

---

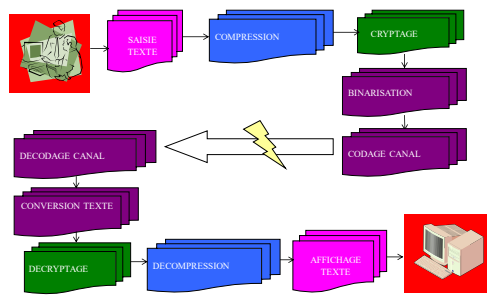
---

---

---

---

## Chaîne de transmission numérique



---

---

---

---

---

---

---

## ORGANISER Faire des FONCTIONS

3- Créer une FONCTION sans échange de donnée



---

---

---

---

---

---

---

## DEFINITION d'une FONCTION

**DEFINITION d'une Fonction :**  
ses INSTRUCTIONS et ses DONNEES locales

```
// FONCTION AfficherAide() : affiche l'aide du jeu
FONCTION AfficherAide()
    VARIABLES_FONCTION
    .....
    DEBUT_FONCTION
        AFFICHER .....
    FIN_FONCTION
```



---

---

---

---

---

---

---

## APPEL d'une FONCTION

**APPEL d'une Fonction :**  
demande d'exécution de la fonction

```
// Rôle : Jeu de Mastermind- Auteur : D. Garric - Date : 17/10/2019
VARIABLES
    .....// VARIABLES LOCALES
DEBUT_ALGORITHME
    APPELER_FONCTION AfficherAide()
FIN_ALGORITHME
```



---

---

---

---

---

---

---

## Exécution CPU d'un Programme avec Fonction

### ★ CPU commence au début de la fonction principale :

- Il exécute les instructions une par une, dans l'ordre où elles se présentent (séquentiellement), en suivant leur logique.

### ★ Si le CPU rencontre un appel de fonction :

- Il met en attente l'exécution de la fonction appelante.
- L'exécution est redirigée vers la fonction appelée ; cette fonction appelée est exécutée du début à la fin, séquentiellement.
- Lorsque l'exécution de la fonction est terminée, le CPU reprend l'exécution de la fonction appelante à l'endroit où elle était arrêtée.
- Arrivé à la fin de la fonction principale, le CPU sort du programme.



13

---

---

---

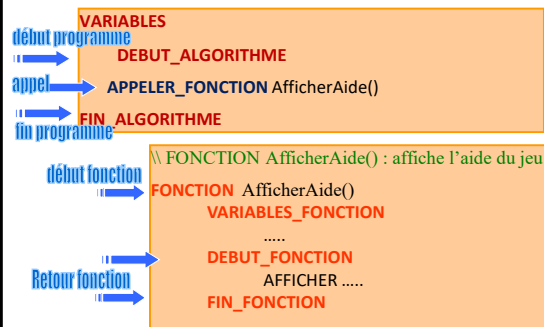
---

---

---

---

## Exécution séquentielle avec Redirection dans les Fonctions



14

---

---

---

---

---

---

---

## ORGANISER Faire des FONCTIONS

4- Nécessité d'échanger des données entre fonctions



---

---

---

---

---

---

---

VARIABLES LOCALES aux Fonctions	
<b>// FONCTION PRINCIPALE</b> <b>VARIABLES</b> p EST_DU_TYPE NOMBRE <b>DEBUT_ALGORITHME</b> ..... <b>FIN_ALGORITHME</b>	<b>// FONCTION calcule un produit</b> <b>FONCTION CalculerProd()</b> <b>VARIABLES_FONCTION</b> produit EST_DU_TYPE NOMBRE <b>DEBUT_FONCTION</b> ..... <b>FIN_FONCTION</b>
<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;"> <p><b>Les fonctions</b> <b>contiennent des données privées :</b></p> <ul style="list-style-type: none"> <li>➤ p : propriété exclusive de la fonction principale</li> <li>➤ produit : propriété exclusive de l'autre fonction</li> </ul> </div> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="margin-bottom: 10px;">p</div> <div style="margin-bottom: 10px;">produit</div> </div> </div>	
<b>Espace RAM</b> <i>Privé de la fonction principale</i>	<b>Espace RAM</b> <i>Privé de la fonction</i>

---

---

---

---

---

---

---

---

DONNEES LOCALES des Blocs
<p>✦ Les données locales des fonctions sont privées :</p> <ul style="list-style-type: none"> <li>• Une fonction est un espace de travail privé : <b>ses variables sont LOCALES</b> et utilisables seulement dans cette fonction.</li> <li>• Les données locales ne sont pas utilisables en dehors de leur fonction. Toute autre fonction ne peut pas les utiliser.</li> <li>• Des variables locales de différentes fonctions ont un espace mémoire distinct, <u>même avec le même nom</u>.</li> <li>• Les fonctions doivent donc communiquer...</li> </ul>

---

---

---

---

---

---

---

---

Nécessité d'échanger des Données entre Fonctions
<div style="border: 1px solid black; border-radius: 10px; padding: 10px; margin-bottom: 10px;"> <p><b>Fonction : traitement privé</b></p> </div> <ul style="list-style-type: none"> <li><input type="checkbox"/> Peut posséder ses données locales visibles seulement par ses instructions</li> <li><input type="checkbox"/> Toute autre fonction ne peut pas les utiliser</li> <li><input checked="" type="checkbox"/> Les fonctions doivent donc s'échanger des données</li> </ul>

---

---

---

---

---

---

---

---

## DEFINITION d'une FONCTION avec E et S

### DEFINITION d'une Fonction : ses INSTRUCTIONS et ses DONNEES

```
// FONCTION COS() : calcule le cosinus d'un angle
// ENTREE : alpha (angle en radian)
// SORTIE : valeur de cosinus de alpha
FONCTION Cos(alpha)
  VARIABLES_FONCTION
    cos_de_alpha EST_DU_TYPE NOMBRE
  DEBUT_FONCTION
    cos_de_alpha PREND_LA_VALEUR ...alpha...+...
    RENVoyer cos_de_alpha
  FIN_FONCTION
```



19

---

---

---

---

---

---

---

---

## APPEL d'une FONCTION avec E et S

### APPEL d'une Fonction : demande d'exécution de la fonction

// Rôle : calcul trigonométrique - Auteur : D. Garric - Date : 17/10/2019

#### VARIABLES

```
// angle, cosinus variables LOCALES à la fonction principale
angle, cosi EST_DU_TYPE NOMBRE
```

#### DEBUT\_ALGORITHME

```
angle PREND_LA_VALEUR 1.5
cosi PREND_LA_VALEUR Cos(angle)
AFFICHER cosi
```

#### FIN\_ALGORITHME



20

---

---

---

---

---

---

---

---

## Fonction avec Entrée et Sortie

### PROTOTYPE de Fonction = MODE d'EMPLOI : fonction cos()

#### VARIABLES

```
// variables LOCALES à cette fonction
angle, cosi EST_DU_TYPE NOMBRE
```

#### DEBUT\_ALGORITHME

```
angle PREND_LA_VALEUR 1.5
cosi PREND_LA_VALEUR Cos(angle)
```

**AFFICHER** cosi

#### FIN\_ALGORITHME



**E** 1.5

#### FONCTION Cos(alpha)

```
VARIABLES_FONCTION
  cos_de_alpha EST_DU_TYPE NOMBRE
DEBUT_FONCTION
  cos_de_alpha PREND_LA_VALEUR .....
```

```
RENVoyer cos_de_alpha
FIN_FONCTION
```




---

---

---

---

---

---

---

---

## ORGANISER Faire des FONCTIONS

5- Créer une FONCTION avec données en Entrée




---

---

---

---

---

---

---

Comment transmettre une valeur à une Fonction ?

Fonction : possède ses propres données locales **privées**.  
☐ Comment transmettre la valeur de  $x$  à l'intérieur de la fonction ?

**FONCTION** AfficherX()  
**VARIABLES\_FONCTION**  
**DEBUT\_FONCTION**  
 AFFICHER  $x$   
**FIN\_FONCTION**

✚ Les fonctions doivent s'échanger des données :

- Les valeurs des paramètres en Entrée (passage par valeur) sont transmises à la fonction au moment de son appel.




---

---

---

---

---

---

---

Paramètre en ENTREE : quel est le lien entre  $n$  et  $x$  ?

noms différents dans  
appel et dans définition

**VARIABLES**  
 // variables **LOCALES**  
 $n$  EST\_DU\_TYPE NOMBRE  
**DEBUT\_ALGORITHME**  
 $n$  PREND\_LA\_VALEUR 5  
 APPELER\_FONCTION AfficherX( $n$ )  
**FIN\_ALGORITHME**

**DEFINITION**  
**FONCTION** AfficherX( $x$ )  
**VARIABLES\_FONCTION**  
**DEBUT\_FONCTION**  
 AFFICHER  $x$   
**FIN\_FONCTION**

**APPEL**

$n: 5_{10}$

**$x = n = 5$**

$x: 5_{10}$




---

---

---

---

---

---

---



Exécution CPU du passage de paramètre en Entrée

```

VARIABLES
// variables LOCALES
n EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
n PREND_LA_VALEUR 5
→ APPELER_FONCTION AfficherX(n)
FIN_ALGORITHME ←

FONCTION AfficherX(x) ←
VARIABLES_FONCTION
DEBUT_FONCTION
  AFFICHER x ←
FIN_FONCTION ←
        
```

1. Exécution fonction principale
  - a. Initialisation de n : n reçoit 5
  - b. Appel fonction avec passage valeur 5 : AfficherX(5)
2. Exécution de la fonction appelée
  - a. Initialisation paramètre en E :  
Le CPU initialise x avec la valeur de n :  $x = n = 5$
  - b. Affichage dans la fonction
  - c. Retour à la fonction appelante
3. Fin exécution fonction principale

---

---

---

---

---

---


---

---

## ORGANISER

### Faire des FONCTIONS

6- Créer une FONCTION avec résultat en Sortie




---

---

---

---

---

---

---

---

Comment récupérer le résultat d'une Fonction ?

0 0

Fonction : possède ses propres données locales **privées**.

☐ Comment transmettre la valeur de produit, en retour, à la fonction appelante ?

```

FONCTION CalculerProd()
VARIABLES_FONCTION
  produit EST_DU_TYPE NOMBRE
DEBUT_FONCTION
  → produit PREND_LA_VALEUR 15
FIN_FONCTION
        
```

⚡ Les fonctions doivent s'échanger des données :

- La valeur d'un résultat en Sortie est renvoyée par la fonction, à la fin de son exécution, vers la fonction appelante.

---

---

---

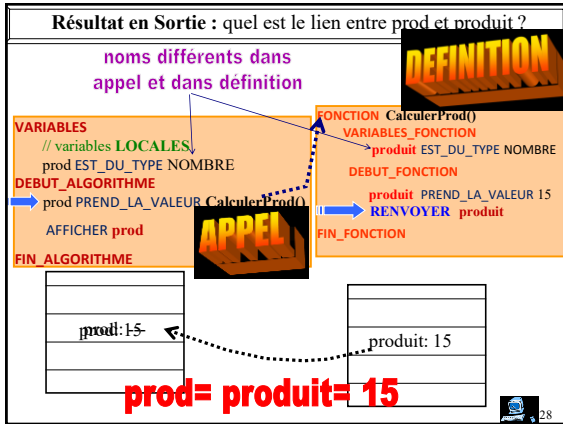
---

---

---

---

---




---

---

---

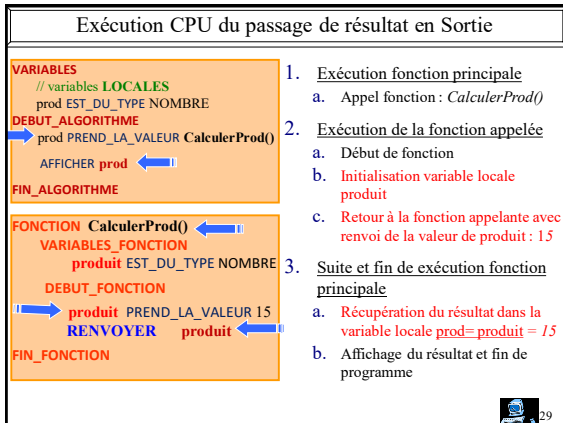
---

---

---

---

---




---

---

---

---

---

---

---

---