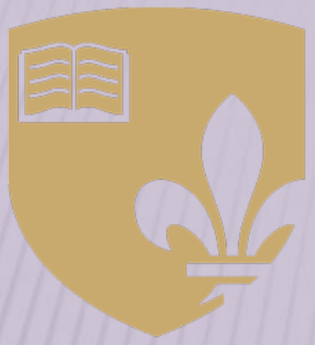

COLLEGE DE PARIS SUPÉRIEUR

DAKARTECH

PROGRAMMATIONS C

Dr Pape Abdoulaye **BARRO**
Enseignant – Chercheur
Spécialiste en Télémétrie & Systèmes Intelligents

SOUS PROGRAMME

SOUS PROGRAMME
DÉFINITION

Lorsque le programme à écrire devient de plus en plus important (volumineux), des difficultés d'aperçu global sur son fonctionnement se posent. Il devient très difficile de coder et de devoir traquer les erreurs en même temps.

- Il est donc utile de découper le problème en de sous problème;
- de chercher à résoudre les sous problèmes (sous-programmes);
- puis de faire un regroupement de ces sous-programmes pour reconstituer une solution au problème initial.

Un sous-programme est une partie d'un programme. Il est d'habitude déclaré dans la partie entête et est réutiliser dans le corps du programme.

- Un sous-programme est un programme. Il possède donc les même caractéristiques d'un programme.

SOUS PROGRAMMES

- Un sous-programme peut utiliser les variables déclarés dans le programme. Dans ce cas, ces **variables** sont dites **globales**. Il peut également utiliser ses propres variables. Dans ce cas, les variables sont dites **locales**. Ces dernières ne pourront alors être utilisable qu'à l'intérieur du sous-programme et nulle part ailleurs (**notion de visibilité**). Ce qui signifie que leur allocation en mémoire sera libérer à la fin de l'exécution du sous-programme.
- Un sous-programme peut être utilisable plusieurs fois avec éventuellement des paramètres différents.
- **En C, nous ne connaissons qu'un seul sous-programme: les fonctions. Dans d'autres, nous avons les fonctions et les procédures.**

SOUS PROGRAMMES

- **Syntaxe d'une fonction**

```
typeDeRetour nomFonction([arguments]){
        //instructions;
}
```

- Exemple:

```
#include <stdio.h>

int abs(int nombre) {
        if (nombre<0)
                nombre=-nombre;
        return nombre; //Valeur renvoyée
}

int main(){
        int val, valAbs;
        printf("Entrez un nombre : ");
        scanf("%d", &val);
                //Appel de la fonction et affectation
        valAbs = abs(val);
        printf("La valeur absolue de %d est %d" , val ,valAbs);
        return 0;
}
```

SOUS PROGRAMMES

- L'instruction *return* permet à la fois de fournir une valeur de retour et à mettre fin à l'exécution de la fonction.
- Dans la déclaration d'une fonction, il est possible de prévoir pour un ou plusieurs arguments (obligatoirement les derniers de la liste) des *valeurs par défaut* ;
  - + elles sont indiquées par le signe **=**, à la suite du type de l'argument.
    - × **Exemple**: **float op(**char c, float x=1.0, float y=1.0**);**

SOUS PROGRAMMES

- Une fonction peut ne pas renvoyer de valeur. Dans ce cas, le type de la fonction est **void**.
  - Exemple:

```
void abs(int nombre)
{
    if (nombre<0)
        nombre=-nombre;
}
```

- Lorsqu'une fonction s'appelle elle-même, on dit qu'elle est « ***récursive*** » (on y reviendra).

SOUS PROGRAMMES

- **Mode de passages de paramètres**

On distingue deux types de passage de paramètres: par valeur et par variable (dite aussi par référence ou encore par adresse).

- Passage par valeur

Le mode de passage par valeur qui est le mode par défaut, consiste à copié la valeur des paramètres effectifs dans les variables locales issues des paramètres formels de ma fonction appelée.

- **Dans ce mode, nous travaillons pas directement avec la variable, mais avec une copie. Ce qui veut dire que le contenu des paramètres effectifs n'est pas modifié. À la fin de l'exécution de la fonction, la variable conservera sa valeur initial.**

- **Rappel:**
  - **Paramètre formel**

    Un paramètre d'entrée de la fonction utilisé à l'intérieur de la fonction appelée.
  - **Paramètre effectif**

    Un paramètre d'appel de la fonction utilisé à l'extérieur de la fonction appelée.

SOUS PROGRAMMES

- **Exemple de mode de passages de paramètres** par valeur

```
#include <stdio.h>

void permute(int a, int b)
{
        int tempon = a;
        a = b;
        b =tempon;
}

int main()
{
        int a=2, b=6;
        printf("a=%d et b=%d" ,a ,b); // avant
        permute(a, b);
        printf("a=%d et b=%d" ,a ,b); ; // après
        return 0;
}
```

Après exécution, on constate qu'on a pas le résultat attendu.

- Par défaut, le passage des arguments à une fonction se fait par valeur.
- Pour remédier à cela, il faut passer par adresse.

SOUS PROGRAMMES

- Passage par adresse

Dans le mode de passage par variable, il s'agit pas simplement d'utiliser la valeur de la variable, mais également son emplacement mémoire.

- **Le paramètre formel se substitue au paramètre effectif tout au long de l'exécution de la fonction et à la sortie, il lui transmet sa nouvelle valeur.**
- **Pour modifier le paramètre réel, on passe son adresse plutôt que sa valeur.**

SOUS PROGRAMMES

- **Exemple de mode de passages de paramètres** par adresse

```
#include <stdio.h>

void permute(int *a, int *b)
{
        int tempon = *a;
        *a = *b;
        *b =tempon;
}

int main()
{
         int a=2, b=6;
         printf("a=%d et b=%d" ,a ,b); // avant
         permute(&a, &b);
         printf("a=%d et b=%d" ,a ,b); ; // après
         return 0;
}
```

SOUS PROGRAMMES

- **On peut faire passer un tableau en paramètre. Nous avons dans ce cas, deux cas de figure: par pointeur ou par semi-référence.**

- **Exemple**: par pointeur

```
#include <iostream>

void affiche(int *tableau, int taille)
{
    for(int i=0; i<taille; i++)
        printf("%d",tableau[i]);
}

int main()
{
        int tab[5] = {1, 2, 3, 4, 5 };
        affiche(tab, 5);
        return 0;
}
```

- **Exemple**: par semi-référence

```
#include <stdio.h>

void affiche(int tableau[], int taille)
{
    for(int i=0; i<taille; i++)
        printf("%d",tableau[i]);
}

int main()
{
        int tab[5] = {1, 2, 3, 4, 5 };
        affiche(tab, 5);
        return 0;
}
```

SOUS PROGRAMMES

Récursivité: définitions

On appelle récursivité tout sous-programme qui s'appelle dans son traitement.

- Il est impératif de prévoir une condition d'arrêt puisque le sous-programme va s'appeler récursivement. sinon, il ne s'arrêtera jamais.
  - On teste la condition,
  - Si elle n'est pas vérifié, on lance à nouveau le sous-programme.
  - Exemple:

```
int factoriel(int n) {
        if(n<=1)
                return 1;
        else
                return(n*factoriel(n-1));
}
```

SOUS PROGRAMMES

Il également possible qu'un sous-programme appel un second qui a son tour appel le premier. On dit que la **récursivité** est indirecte, cachée, **croisée** ou mutuelle.

Exemples :

```
int pair (int n n) {
    if(n==0) {
        return 1 ;
    }else if(n==1) {
        return 0;
    }else{
        return impair(n-1);
    }
}
```

```
int impair (int n) {
    if(n==1) {
        return 1;
    }else if(n==0) {
        return 0;
    }else {
        return pair(n-1);
    }
}
```

SOUS PROGRAMMES
CAS PRATIQUES N°7

Exercice 32:

- Ecrire une fonction permettant de calculer la somme des n premiers nombres entiers, n étant saisie au clavier.
- Ecrire une fonction permettant de chercher la valeur absolue d'un nombre entier saisi au clavier.
- Ecrire une fonction permettant de faire la permutation de deux entiers a et b.

Exercice 33 :

- Ecrire un programme utilisant une fonction qui reçoit en argument 2 nombres flottants et un caractère (opération), et qui fournit le résultat du calcul demandé.
- Proposer le même programme mais cette fois ci, la fonction ne disposera plus que de 2 arguments en nombres flottants. L'opération est précisée, cette fois, à l'aide d'une variable globale.

Exercice 34 :

- Ecrire un programme faisant appel à une fonction qui ne renvoie aucune valeur et qui détermine la valeur maximale et la valeur minimale d'un tableau d'entiers.

Exercice 35 :

- Écrire un programme qui permet de saisir deux entiers n et p ($1 \leq p \leq n$) et de calculer puis afficher le nombre de combinaison de p éléments parmi n : CNP
  sachant que: $C_n^p = n! / (p! * (n-p)!)$

Exercice 36 :

- Un nombre est dit perParfait s'il est parfait (égale à la somme de ces diviseurs) et son successeur est premier. Exemple : 6 est perparfait car il est parfait (6=3+2+1) et son successeur 7 est premier.
- Déterminer tous les nombres Perparfaits <=100

Affaires à suivre