Université A. Mira Béjaia
Faculté de Sciences Exactes
Département Mathématique
L1 Math (tronc commun)
Module : ASD1

Support de TP 2: Sous-programme en langage C

Un **sous-programme** est une part de code que vous pouvez définir une fois et appeler plusieurs fois dans le programme principal ou dans d'autres parties du programme. **Les sous-programmes en C** sont créés en utilisant des **fonctions**. Cela permet de rendre le code plus modulaire, réutilisable et facile à lire.

Structure générale d'un sous-programme en C

Une fonction en C est définie avec la syntaxe suivante :

```
type_retour nom_fonction(liste_paramètres) {
    // Corps de la fonction
    return valeur;  // (Optionnel selon type_retour)
}
```

- **type_retour** : Type de la valeur retournée par la fonction (par exemple, int, float, void, etc.).
- **nom_fonction** : Nom de la fonction, utilisé pour l'appeler.
- **liste_paramètres** : Liste des paramètres d'entrée (avec leur type et nom), qui servent à transmettre des données à la fonction.
- **return valeur** : (Facultatif pour **void**) Retourne une valeur au programme appelant.

En langage C, il n'existe pas de mot-clé spécifique pour les **procédures**, comme dans d'autres langages (par exemple, Pascal). Cependant, on utilise des fonctions pour représenter les **procédures** et les **fonctions** en C.

- Une **procédure** en C est une fonction qui ne retourne **aucune valeur**. Pour cela, on utilise le type **void** comme type de retour.
- Une **fonction**, en revanche, retourne une valeur (par exemple, int, float, etc.).

Exemple: un sous-programme pour calculer la somme de deux entiers :

```c
#include <stdio.h>

// Définition du sous-programme (fonction)
int somme(int a, int b) {
    return a + b;
}

int main() {
    int x, y, resultat;

    // Demande des valeurs à l'utilisateur
    printf("Entrez deux nombres : ");
    scanf("%d %d", &x, &y);

    // Appel du sous-programme
    resultat = somme(x, y);

    // Affichage du résultat
    printf("La somme de %d et %d est : %d\n", x, y, resultat);

    return 0;
}
```

Points importants :

1. **Déclaration et définition** : Les fonctions doivent être définies avant d'être utilisées ou au moins déclarées en haut du programme (prototypes de fonctions).
2. **Prototypes de fonctions** : Si une fonction est définie après main, son prototype doit être déclaré avant main :

```c
int somme(int a, int b);  // Prototype
```

Types de sous-programmes en C

1. **Sans retour et sans paramètres** : Ces fonctions ne prennent aucun paramètre et ne retournent aucune valeur. Elles effectuent une tâche spécifique.

```c
void afficher_message() {
    printf("Bonjour, ceci est un message.\n");
}

int main() {
    afficher_message();  // Appel de la fonction
    return 0;
}
```

2. **Avec retour mais sans paramètres** : Ces fonctions retournent une valeur mais ne prennent aucun paramètre.

```c
int obtenir_valeur() {
    return 42;
}

int main() {
    int val = obtenir_valeur();
    printf("La valeur obtenue est : %d\n", val);
    return 0;
}
```

3. **Avec paramètres mais sans retour** : Ces fonctions prennent des paramètres mais ne retournent pas de valeur.

```c
void afficher_somme(int a, int b) {
    printf("La somme est : %d\n", a + b);
}

int main() {
    afficher_somme(10, 20);
    return 0;
}
```

4. **Avec paramètres et avec retour** : Ces fonctions prennent des paramètres et retournent une valeur.

```c
float moyenne(float a, float b) {
    return (a + b) / 2.0;
}

int main() {
    float resultat = moyenne(4.5, 3.5);
    printf("La moyenne est : %.2f\n", resultat);
    return 0;
}
```

Exemple complet avec plusieurs sous-programmes

```c
#include <stdio.h>

// Déclarations des fonctions
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
float division(int a, int b);

int main() {
    int x, y, choix;
    printf("Entrez deux entiers : ");
    scanf("%d %d", &x, &y);

    printf("Choisissez une opération :\n");
    printf("1. Addition\n2. Soustraction\n3. Multiplication\n4. Division\n");
    scanf("%d", &choix);
```

```c
    switch (choix) {
        case 1:
            printf("Résultat : %d\n", addition(x, y));
            break;
        case 2:
            printf("Résultat : %d\n", soustraction(x, y));
            break;
        case 3:
            printf("Résultat : %d\n", multiplication(x, y));
            break;
        case 4:
            if (y != 0)
                printf("Résultat : %.2f\n", division(x, y));
            else
                printf("Erreur : Division par zéro !\n");
            break;
        default:
            printf("Choix invalide.\n");
    }

    return 0;
}
```

```c
// Définition des fonctions
int addition(int a, int b) {
    return a + b;
}

int soustraction(int a, int b) {
    return a - b;
}

int multiplication(int a, int b) {
    return a * b;
}

float division(int a, int b) {
    return (float)a / b;
}
```

Passage par valeur et passage par adresse en langage C

En langage C, le passage des paramètres à une fonction peut se faire de deux manières principales : **par valeur** et **par adresse**. Voici une explication détaillée avec des exemples pour illustrer ces concepts.

1. Passage par valeur

Dans le passage par valeur, une copie de la variable est transmise à la fonction. Toute modification apportée au paramètre dans la fonction n'affecte pas la variable originale dans le programme appelant.

Exemple :

```c
#include <stdio.h>

void incrementer_par_valeur(int x) {
    x = x + 1;  // Modifie uniquement la copie locale de x
    printf("Dans la fonction : x = %d\n", x);
}

int main() {
    int a = 5;

    printf("Avant la fonction : a = %d\n", a);
    incrementer_par_valeur(a);  // Passe une copie de `a` à la fonction
    printf("Après la fonction : a = %d\n", a);  // La valeur de `a` reste inchangée

    return 0;
}
```

Sortie :

```
Avant la fonction : a = 5
Dans la fonction : x = 6
Après la fonction : a = 5
```

Explication :

- La fonction **incrementer_par_valeur** reçoit une copie de **a**.
- La modification de **x** dans **la fonction** ne change pas **a** dans le **programme principal.**

2. Passage par adresse

Dans le passage par adresse, l'**adresse** de la variable est transmise à la fonction. Cela permet à la fonction de modifier directement la valeur de la **variable originale,** car elle accède à la mémoire où cette variable est stockée.

Exemple :

```c
#include <stdio.h>

void incrementer_par_adresse(int *x) {
    *x = *x + 1;  // Modifie directement la valeur à l'adresse pointée par x
    printf("Dans la fonction : *x = %d\n", *x);
}

int main() {
    int a = 5;

    printf("Avant la fonction : a = %d\n", a);
    incrementer_par_adresse(&a);  // Passe l'adresse de `a` à la fonction
    printf("Après la fonction : a = %d\n", a);  // La valeur de `a` est modifiée

    return 0;
}
```

Sortie :

```
Avant la fonction : a = 5
Dans la fonction : *x = 6
Après la fonction : a = 6
```

Explication :

- La fonction **incrementer_par_adresse** reçoit **l'adresse de a** (indiquée par **&a**).
- Elle utilise un pointeur (**int *x**) pour accéder et modifier la **valeur originale de a.**