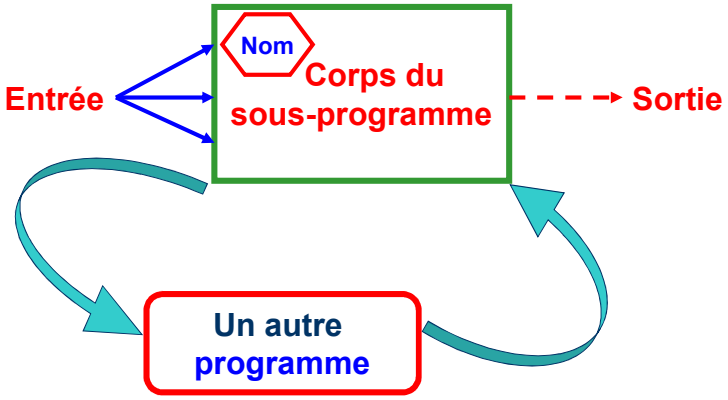
Filière IGE – 2017/2018
Algorithmique & Programmation
Leçon 9 : Sous-programmes

Prof. *A. DARGHAM*

Sous-programmes

- **Définition**
  - Un **sous-programme** est un **bloc d'instructions** (appelé **corps du sous-programme**) ayant les **propriétés suivantes** :
    - Il a sa **propre entrée** qui est **formée d'un ou plusieurs paramètres** (appelés aussi **arguments**).
    - Il peut **retourner une sortie** (**résultat**) ou non.
    - Il a possède un **nom**.
    - il **peut être appelé** en **invoquant son nom**.

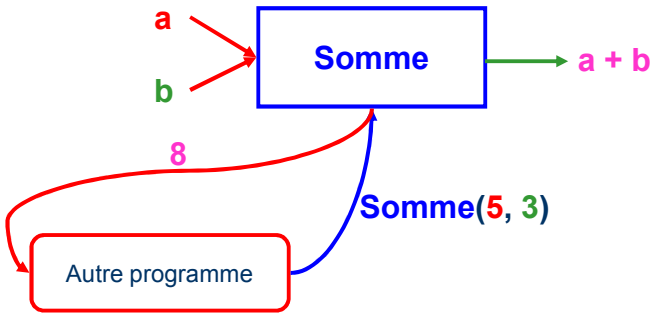
Sous-programmes



Sous-programmes

- **Exemple**
  - Un **sous-programme** qui calcule la somme de deux entiers est composé des éléments suivants :
    - **Deux arguments** de type **int**.
    - Un **résultat** de type **int**.
    - Un **nom**, par exemple « **somme** ».

Sous-programmes



Sous-programmes

- **Définition**
  - En **programmation** :
    - Une **fonction** est un **sous-programme** qui **retourne un résultat**.
    - Une **procédure** est un **sous-programme** qui ne **retourne aucun résultat**.

Sous-programmes

- **Déclaration d'un sous-programme en C :**
  - **Syntaxe** :

```
Type Nom(Type_1 Arg_1, …, Type_N Arg_N);
```

  - `Type` : est le **type de résultat retourné** par le **sous-programme** (**void** si le **sous-programme** est une **procédure**).
  - `Nom` : est le **nom du sous-programme**.
  - `Type_i` et `Arg_1` : le **type** et le **nom** du **$i^{ème}$ argument** du **sous-programme**.

Sous-programmes

- **Exemples** :

```
int somme(int a, int b);
int factoriel(int n);
float moyenne(float a, float b, float c);
int puissance(int x, int n);
void dire_bonjour(int n);
```

Sous-programmes

- **Définition d'un sous-programme en C :**
  - **Syntaxe** :

```
Type Nom(Type_1 Arg_1, …, Type_N Arg_N)
{
   /* instructions du sous-programme */
}
```

Sous-programmes

- **Exemples** :

```
int somme(int a, int b)
{
   return a + b;
}
int factoriel(int n)
{
   int i, f = 1;
   for(i = 2; i <= n; i++)
      f *= i;
   return f;
}
```

Sous-programmes

- **Exemples** :

```
float moyenne(float a, float b, float c)
{
   return (a + b + c) / 3.0;
}
int puissance(int x, int n)
{
   int i, p = 1;
   for(i = 1; i <= n; i++)
      p *= x;
   return p;
}
```

Sous-programmes

- **Exemples** :

```
void dire_bonjour(int n)
{
   int i;
   for(i = 1; i <= n; i++)
      puts("bonjour");
}
```

Sous-programmes

- **Appel d'un sous-programme en C :**
  - **Cas d'une fonction** :

```
var = Nom_Fct(Val_Arg_1, …, Val_Arg_N);
```

  - **Cas d'une procédure** :

```
Nom_Proc(Val_Arg_1, …, Val_Arg_N);
```

Sous-programmes

```
#include <stdio.h>
int somme(int a, int b)
{
   return a + b;
}
int factoriel(int n)
{
   int i, f = 1;
   for(i = 2; i <= n; i++)
      f *= i;
   return f;
}
```

Sous-programmes

```
float moyenne(float a, float b, float c)
{
   return (a + b + c) / 3.0;
}

int puissance(int x, int n)
{
   int i, p = 1;
   for(i = 1; i <= n; i++)
      p *= x;
   return p;
}
```

Sous-programmes

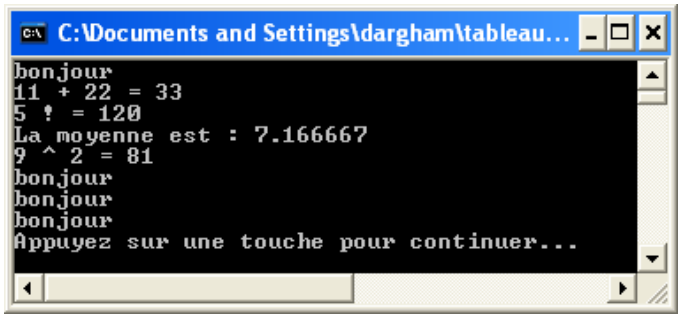
```
void dire_bonjour(int n)
{
   int i;
   for(i = 1; i <= n; i++)
      puts("bonjour");
}

main()
{
   int f, p;
   float x;
   dire_bonjour(1);
   printf("11 + 22 = %d\n", somme(11, 22));
```

Sous-programmes

```
    f = factoriel(5);
    printf("5 ! = %d\n", f);
    x = moyenne(2.5, 3.75, 15.25);
    printf("La moyenne est : %f\n", x);
    printf("9 ^ 2 = %d\n", puissance(9, 2));
    dire_bonjour(3);
    system("pause");
}
```

Sous-programmes



Sous-programmes

- **Instruction « return » :**
  - **Syntaxe** :

```
return expression;
```

  - C'est **l'instruction par laquelle une fonction retourne sa valeur** (**résultat**) au **sous-programme qui l'appelle**.
  - Cette instruction **provoque l'arrêt immédiat** de la fonction.
  - l **'expression retournée doit être du même type** que **celui de la fonction**.

Sous-programmes

- **Paramètres d'une fonction :**
  - **Paramètres formels** :
    - Ceux **utilisés dans la définition de la fonction** et **servent à écrire le code** de celle-ci.
    - Elles sont **considérés comme des variables locales** de la fonction.
  - **Paramètres effectifs** :
    - Ceux **utilisés lors de l'appel de la fonction**.
    - Sont des **expressions** de **même types** que les **paramètres formels correspondants**.

Sous-programmes

- **Exemples :**
  - Prenons la fonction « somme ».
  - Ses **paramètres formels** : a (int) et b (int).
  - Ses **paramètres effectifs** : **n'importe quelle expression** de **type int** :
    - `somme(1, 2)`
    - `somme(x, 17); /* x étant int */`
    - `somme(a * 2, somme(1, 7));`

Sous-programmes

- **Passage par valeur (exemple)** :

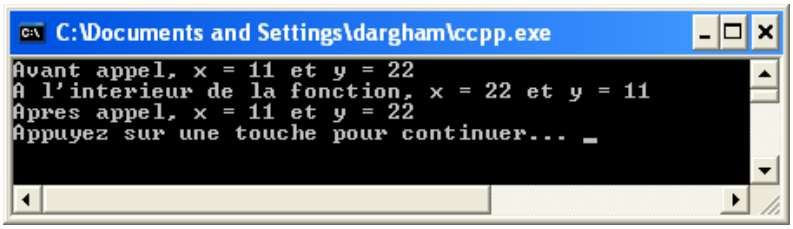
```
#include <iostream>
using namespace std;
void permuter(int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
   cout << "A l'interieur de la fonction,  x = " << x;
   cout << " et y = " << y << endl;
}
```

Sous-programmes

```
int main()
{
   int x = 11, y = 22;

   cout << "Avant appel, x = " << x << " et y = " << y;
   cout << endl;
   permuter(x, y);
   cout << "Apres appel, x = " << x << " et y = " << y;
   cout << endl;
   system("pause");
   return 0;
}
```

Sous-programmes



Sous-programmes

- **Avant l'appel**, les **paramètres effectifs** sont x = 11 et y = 22.
- **Lors de l'appel de la fonction** « permuter » :
  - Le **paramètre formel** x est **initialisé avec la valeur** 11.
  - Le **paramètre formel** y est **initialisé avec la valeur** 22.
- **À l'intérieur de la fonction** « permuter » :
  - Les **paramètres formels** x et y **ont été bien échangés**.
- **Après l'appel**, les **paramètres effectifs** sont x = 11 et y = 22 : ils **n'ont pas été modifiés**.

Sous-programmes

- Le **paramètre formel x** se **comporte comme** une **variable locale** de la fonction.
- Le **paramètre formel x** de la fonction **n'a rien avoir avec** le **paramètre effectif x**, sauf que **la valeur de celle-ci est utilisée pour l'initialiser**.
- Même chose pour y.
- C'est le **passage d'arguments par valeur**.
- **Règle** :

  Une **fonction ne peut pas modifier** un **argument** si ce dernier est **passé par valeur**.

Sous-programmes

- Passage par adresse (exemple) :

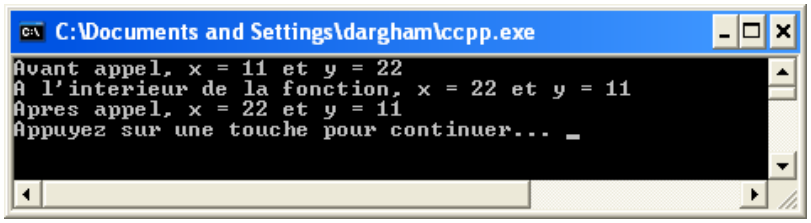
```
#include <iostream>
using namespace std;
void permuter(int * x, int * y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    cout << "A l'interieur de la fonction,  x = ";
    cout << *x << " et y = " << *y << endl;
}
```

Sous-programmes

```
int main()
{
    int x = 11, y = 22;

    cout << "Avant appel, x = " << x << " et y = " << y;
    cout << endl;
    permuter(&x, &y);
    cout << "Apres appel, x = " << x << " et y = " << y;
    cout << endl;
    system("pause");
    return 0;
}
```

Sous-programmes



Sous-programmes

- **Avant l'appel**, les **paramètres effectifs** sont x = 11 et y = 22.
- **Lors de l'appel de la fonction** « permuter » :
  - Le **paramètre formel** x est **initialisé avec l'adresse du paramètre effectif** x.
  - Le **paramètre formel** y est **initialisé avec l'adresse du paramètre effectif** y.
- **À l'intérieur de la fonction** « permuter » :
  - Les **paramètres formels** x et y **ont été bien échangés**.
- **Après l'appel**, les **paramètres effectifs** sont x = 22 et y = 11 : ils **ont été bien modifiés**.

Sous-programmes

- Comme le **paramètre formel x** de la fonction est un **pointeur**, sa **valeur sera toujours l'adresse du paramètre effectif** x (notez **&x**, lors de l'appel de la fonction).
- **Toute modification du paramètre formel affectera le paramètre effectif**.
- C'est le **passage d'arguments par adresse**.
- **Règle** :

  Une **fonction peut modifier** un **argument** si ce dernier est **passé par adresse**.