Université Paul Sabatier
L2 - S4 - Printemps 2005

Programmation impérative et langage C

feuille de **TD n° 3 :** Sous-programmes.

---

1 Les sous-programmes en C

1.1 Fonctions ou procédures

Selon son utilisation, on définit un sous-programme comme une fonction ou comme une procédure :
– si son utilisation prend la forme d'une valeur, c'est une **fonction**
– si son utilisation prend la forme d'une instruction, c'est une **procédure**
En C, les sous-programmes sont tous des fonctions qui retournent un résultat de type précisé dans l'en-tête.

- Si on veut définir une procédure, il faut préciser void comme type de retour et ne pas retourner de valeur dans le corps de la fonction.
- Le type retourné ne peut être un tableau.

La déclaration générale d'une fonction est :

```
type_résultat  identificateur_fonction  (liste arguments typés)
    {
    corps_fonction
    }
```

On peut écrire des sous-programmes :
– d'affichage : destiné à afficher à l'écran les valeurs de diverses variables
– de lecture ou saisie : destiné à renseigner diverses variables
Mais, excepté ces cas trés particuliers, on ne fera apparaitre ni lecture ni écriture dans un sous-programme ; on utilisera plutôt les paramètres comme moyen d'Entrée\ Sortie.

- Avant d'utiliser une fonction, il faut qu'elle ait été définie auparavant ou que son **prototype** ait été déclaré.
- En C, les sous-programmes ne peuvent être imbriqués.

1.2 Cas des paramètres modifiables :

Un **pointeur** est une variable qui contient l'adresse en mémoire d'une autre variable.
L'opérateur & s'applique à une variable ; il retourne l'adresse de cette variable : aprés l'instruction p=& c, on dit que p pointe sur c.
L'opérateur * s'applique à un pointeur ; il retourne l'objet pointé.

- Déclaration : les arguments sont des pointeurs sur les quantités à modifier
- Appel : les arguments sont les adresses des quantités à modifier

Exemple :

```
void plus_un (int* p)
{
*p=*p+1
}
 ...

void main()
{
 int a=1;

plus_un(& a);
}
```

◇ Exercice **1** : On veut écrire un sous-programme qui renvoie la partie entière d'un réel positif : (on incrémente à partir de 0 un entier dans une boucle ...)
– Ecrire une fonction qui réalise cette tâche
– Ecrire une procédure qui réalise cette tâche
– Appeler ces sous-programmes dans le programme principal

◇ Exercice **2** : Le sous-programme suivant détermine le plus petit et le plus grand de deux entiers fournis en entrée.

```
void min_max (int A, int B, int* min, int* max)
{
if (A<B) {*min=A; *max=B; }
else min=B; *max=A; }
}

void main()
{
 int min,max;

min_max(2 ,5, &min, &max);
}
```

Quel est le problème si on remplace *min=A par min=&A dans le sous-programme min_max ?

◇ Exercice **3** : Le programme suivant est-il syntaxiquement correct ?
Quels sont les affichages lors de l'éxécution ?

```
void proc_1(int* F)
{
printf("Entrez f :");scanf("%d", F);
*F=*F + 2;
}
void proc_2(int* D)
{
printf("\ n D=%d", D);
*D=*D + 1;
}

void main()
{
int F=1;
int G=10;

proc_1(&F);
printf("\ n F=%d", F);
proc_2(&G);
printf("\ n G=%d", G);
}
```

1.3 Cas particulier des tableaux comme paramètres de fonctions :

- Une fonction en C ne peut retourner une valeur de type tableau ; on utilisera une procédure avec un paramètre de type tableau modifiable.
- En C, un tableau est un pointeur fixe. Ainsi, lors de la déclaration float a[80], a est l'adresse du début du tableau. a et &a[0] représentent la même adresse laquelle ne peut être modifiée. En conséquence :
  - Tout paramètre de type tableau est automatiquement modifiable.
  - Lors de la phase de déclaration, il n'est pas nécessaire de préciser la taille d'un paramètre **vecteur**.

Exemple :

```
void debut_a_zero (int v[ ])
{
v[0]=0
}
 ...

void main()
{
 int a[5]={10, 12, 14, 16, 18} ;

debut_a_zero(a) ;
}
```

◇ Exercice **4** : Operations sur des vecteurs de $\mathbb{R}^3$ à composantes entières :
Les vecteurs seront du type : `typedef int Vec[3] ;`

Ecrire des sous-programmes de :

1. saisie des éléments d'un vecteur ;
2. affichage en ligne d'un vecteur ;
3. multiplication d'un vecteur par un entier λ ;
4. calcul de la somme des éléments d'un vecteur ;
5. addition de deux vecteurs ;
6. produit scalaire de deux vecteurs ;
7. norme euclidienne d'un vecteur ;
8. recherche de la position de la première occurence de l'élément minimum d'un vecteur ;
9. échange de 2 composantes d'un vecteur ;

puis écrire un programme principal dans lequel on appellera les divers sous-programmes.

2 Passages de paramètres

Les paramètres apparaissant dans la déclaration du sous-programme sont appelés **paramètres formels** alors que les paramètres fournis lors de l'appel du sous-programme sont appelés **paramètres effectifs**.

Lors de l'appel d'un sous-programme, la transmission des paramètres sefait **par valeur** : la valeur d'un paramètre effectif est recopiée dans un emplacement local au sous-programme qui travaille alors sur cette copie et non sur l'original. Ce mode de transmission peut-être coûteux et ne permet pas la modification du paramètre effectif, d'où la nécessité de transmettre les adresses (par usage de pointeurs).

```
int i,j ;

int f(int a, int b)
{
i=i+a ;
return a+b ;
}
void main()
{
int x ;
i=10 ;
j=40 ;
x=f(i,j) ;
}
```

Dans cet exemple, a et b sont des paramètres formels alors que i et j sont des paramètres effectifs de la fonction f.
i et j sont des **variables globales** ; x est une **variable locale** à la procédure main.
La variable i est modifiée dans le sous-programme, (à la fin de l'éxécution i=20) ; on dit qu'il y a **effet de bord** .
Pour une meilleure lisibilité des sous-programmes, on évite le plus souvent les effets de bord.

Règle de visibilité des identificateurs :
Un **bloc** est un ensemble de déclarations de données et d'instructions entourées d'accolades ouvrante et fermante.
Tout identificateur est visible dans le bloc où il est défini sauf dans tout bloc inclus ou il a été redéfini.

```
int i,j,x ;

int f(int a, int b)
{
int i=1 ;
i=i+a ;
return a+b ;
}

void main()
{
i=10 ;
j=40 ;
x=f(i,j) ;
}
```

La variable i utilisée dans la fonction est ici une variable locale (à cette fonction) et qui coexiste sans interactions avec la variable globale de même nom i : il n'y a pas effet de

bord.(à la fin de l'éxécution i=10)

◇ Exercice **5** : Soit le programme :

```
int f(int a, int b)
{
i=i+a;
return a+b;
}
void main()
{
int i,j,x;

i=10;
j=40;
x=f(i,j);
}
```

Que se passe t-il à la compilation ?

◇ Exercice **6** :

Soit le programme suivant :

```
int y;

int f(int x)
{
int t=x;

y=y+1;
t=t+1;
return t+y;
}

void main()
{
int i,z;

y=10;
i=1;
z=f(i) + y;
}
```

Etablir le tableau de situations correspondant.

◇ Exercice **7** :

Soit le programme suivant :

```
int i,j;

void pr(int* x, int* y, int* z)
{
*y=i + *y;
*z=*x + *y;
}

void main()
{
i=3;
j=7;
pr(&i,&i,&j);
i=3;
j=7;
pr(&j,&j,&i);
}
```

Etablir le tableau de situations correspondant.

◇ ◇ ◇◇ ◇ ◇◇