S. REGIS Chapitre 2

Les fonctions

I Introduction

1. Une image pour mieux comprendre(facultatif)

Pour mieux comprendre la notion de fonction, prenons une image tirée d'un monde que vous connaissez bien : les jouets. Quand vos parents voulaient vous acheter un château- fort, ils avaient deux possibilités : vous acheter un château-fort en plastique (qu'on ne peut pas démonter) ou bien vous acheter des legos. L'avantage des legos est évident : vous pouvez les utilisez pour construire votre château mais vous pourrez les réutilisez dans d'autres constructions (maison, avion, etc.) contrairement au château en plastique. Ces legos peuvent avoir des couleurs et des formes différentes : ils jouent donc des rôles différentes et ont des places différentes dans la structure des jouets.

2. Notion de fonction

Quand vous écrivez un programme vous avez deux possibilités :

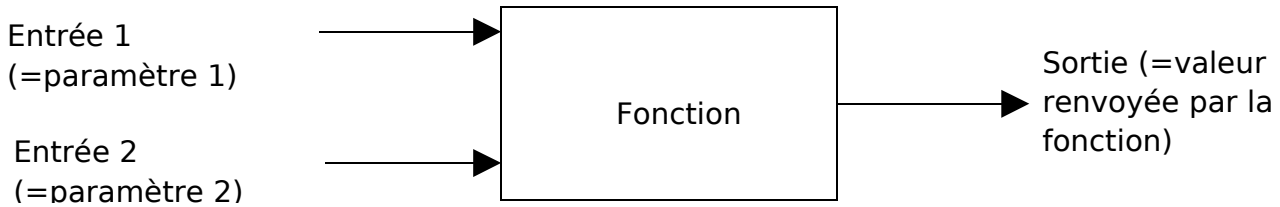
- soit tout écrire un seul bloc : il s'agit du programme principal (appelé « main » en langage C). C'est ce que vous avez fait jusqu'à présent en programmation. L'inconvénient c'est que vous ne pouvez pas vraiment réutiliser ce que vous avez écrit dans un autre programme (ou alors il faut tout recopier et modifier au fur et à mesure).
- soit utiliser plusieurs sous blocs (comme des legos) qui réalisent des tâches spécifiques et que vous pourrez réutiliser dans d'autres programmes : il s'agit de fonctions (qu'on appelle aussi sous-programme). Ces fonctions seront définies en dehors du programme principal et celui-ci pourra faire « appel » à ces fonctions autant de fois qu'il en a besoin.

Définition
Une fonction est un sous-programme qui calcule, à partir de paramètres éventuels, une valeur d'un certain type utilisable dans une expression du programme appelant.

Remarque :
Il existe aussi des procédures. Une procédure est un sous-programme qui effectue une ou *plusieurs* actions, et qui peut calculer et retourner *une ou plusieurs* valeurs, à partir de paramètres éventuels.
Néanmoins, en langage C, il n'y a pas de distinction entre la notion de fonction et la notion de procédure

La notion de fonction permet entre autre de décomposer un problème complexe en sous problèmes moins complexe : c'est l'approche descendante évoquée au 1ier semestre (ueo11).
L'utilisation des fonctions permet donc de réutiliser des sous-programmes déjà développés et surtout validés, et facilite d'éventuelles modifications ultérieures.

On utilise parfois le schéma suivant pour représenter une fonction :



Il faut aussi noter qu'en C il existe déjà des fonctions prédéfinies (qui sont déjà écrites) et que vous avez déjà utilisées sans savoir que c'était des fonctions (par exemple pour la lecture/écriture : scanf et printf). Les fonctions mathématiques utilisables en C (cosinus, exponentiel,…) sont également des fonctions prédéfinies

Dans la représentation algorithmique d'une fonction, les paramètres seront mis entre parenthèses après le nom de la fonction (tout comme les fonctions mathématiques), et si la fonction doit fournir une sortie alors :

- il faut préciser le type de cette sortie (qui sera en fait le type de la fonction)
- il faut que la fonction renvoie une valeur (cette valeur est la sortie)
-

On appelle aussi ces paramètres de la fonction, argument de la fonction.

Il faut noter qu'en dehors des paramètres, la fonction peut avoir ses propres variables (qui sont définies dans la structure de cette fonction) mais ces variables ne sont connues et ne sont utilisables qu'à l'intérieur de la fonction : on parle de variables locales à la fonction. Elles ne pourront pas être utilisées par le programme principal par exemple.

La structure algorithmique d'une fonction comprend trois parties:

Partie 1 : L'entête

Fonction *Nom_fonction(*type_paramètre 1 parametre1, type_parametre 2 parametre 2,…): **Type_fonction**

Partie 2 : Déclaration des variables locales

type_variable_i Nom_variable_i

Partie 3 : Corps de la fonction

Début

Instructions

Retour ***valeur retournée***

Fin

Notons également que certaines fonctions ne renvoient aucune sortie, elles n'ont donc pas de type spécifique.

II Fonction en langage C

1. Déclaration de fonction

Nous allons maintenant voir comment ces fonctions s'écrivent en C.
Elles ont en générale la forme suivante :

```
type_de_donnee Nom_De_La_Fonction(type1 argument1, type2 argument2, ...) {

type variable_sortie ;
liste d'instructions
;

return variable_sortie ; /*return permet de renvoyer la valeur de sortie*/
}
```

La fonction prédéfinie return permet de renvoyer au programme appelant le résultat calculé par la fonction appelé.
Quand dans une fonction, on rencontre l'appel à return, il se produit :

- un retour au programme appelant (en général le programme principal « main »)
- la fourniture d'une valeur-résultat si return est suivie d'une expression :

```
return (expression) ;
```

Si la fonction ne renvoie rien, on utilisera comme type pour la fonction le mot « void » (que vous avez déjà vu…).

```
void  Nom_De_La_Fonction(type1 argument1, type2 argument2, ...) {

liste d'instructions
;

/*ici on n'a pas besoin d'utiliser la fonction return car la fonction n'a pas de sortie*/)
}
```

Pour utiliser cette fonction dans le programme principal (on parle d'appel de la fonction), on écrit le nom de cette fonction ainsi que les paramètres nécessaires dans le corps du programme.

```
int main (void)
{


        Nom_De_La_Fonction(argument1, argument2);
}
```

Prenons un exemple on veut écrire une fonction qui calcule et renvoie le double d'un entier. L'entrée (le paramètre) de la fonction sera un entier et le résultat (le double de l'entier) sera aussi un entier.
La structure algorithmique de cette fonction est la suivante (à faire éventuellement par les étudiants) :

```
Fonction DoubleEntier(entier x) : entier

variable locale
        entier y
Debut
        y=x*2
        retour y
Fin
```

Le programme C de cette fonction

```
int DoubleEntier (int x)
{
        int y ;

        y=2*x ;

        return y ;
}
```

Dans le programme principal, on utilise la fonction comme suit :

```
#include <stdio.h>

int DoubleEntier (int x)
{
        int y ;

        y=2*x ;

        return y ;
}

int main(void)
{
        int a,b ;

        printf(''Entrer un entier\n'') ;

        scanf(''%d'', &a) ;

        b=DoubleEntier(a) ;

        printf(''le double de cet entier est %d\n'',b) ;

}
```

Pour l'instant nous ne nous intéresserons qu'aux fonctions qui sont définies dans le même fichier que le programme principal. Ces fonctions sont donc considérées comme connues et utilisables à la fois par le programme principal (main) et par les autres fonctions présentes dans le fichier.

Notons que trois sortes de variables sont accessibles dans la fonction :

- Les variables dites globales
- Les variables dites locales
- Les paramètres formels

Les variables dites globales :

- Ce sont celles définies dans le programme appelant et considérées comme disponibles dans la fonction
- Elles peuvent donc être référencées partout dans le programme appelant et dans la fonction appelée

Leur portée est globale.

Les variables dites locales :

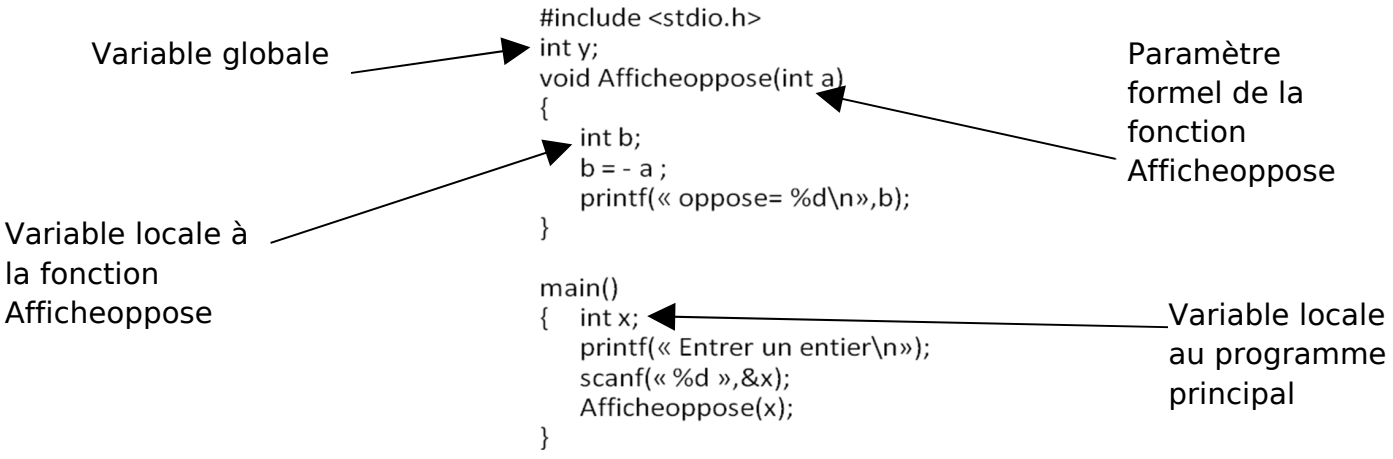
- Ce sont celles qui sont définies dans le corps de la fonction, utiles pour son exécution propre et accessibles seulement dans cette fonction
- elles sont donc invisibles pour le programme appelant

Leur portée est locale

Les paramètres formels :

- ce sont les paramètres de la fonction qui servent à l'échange d'information entre les programmes appelant et appelé

Leur portée est également locale



Afficheoppose peut « voir » b, x et y. Le main peut voir x et y mais ne peut voir b.

Exercice 1

Ecrire un programme C qui permet de trouver le maximum de 4 entiers donnés par l'utilisateur.

Ecrire ensuite une fonction qui prend en paramètre deux entiers et retourne le maximum de deux entiers. On utilisera cette fonction dans un programme qui doit afficher le maximum de 4 entiers donnés par l'utilisateur.

Exercice 2

Ecrire une fonction qui prend un entier en paramètre et qui renvoie 1 si cet entier est divisible par 3 et 0 sinon. Ecrire un programme qui demande à l'utilisateur de rentrer un entier et qui lui affiche « ce nombre est divisible par 3 » (on utilisera la fonction que l'on vient d'écrire).

Exercice 3

Ecrire une fonction qui prend un entier en paramètre et qui renvoie 1 si cet entier est pair (divisible par 2) et 0 sinon. En utilisant cette fonction et celle de l'exercice 2, écrire un programme qui demande à l'utilisateur de rentrer un entier et qui lui indique s'il est pair, s'il est multiple de 3 ou s'il est multiple de 6.

2. Arguments (paramètres) d'une fonction

a) Passage par valeur

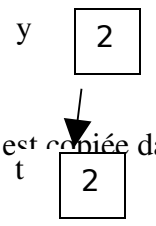
En langage C, tous les arguments des fonctions sont transmis par valeur. Cela signifie que la fonction n'utilise pas directement les variables choisies comme argument mais récupère uniquement les valeurs de ces variables. Plus précisément la fonction recoit la valeur de ses arguments mais pas leurs adresses.

Ces valeurs seront rangées dans des variables provisoires qu e la fonction pourra éventuellement modifier mais les variables du programme appelant ne pourront être modifiées

Exemple

int y ;

y=2 ;



P= DoubleEntier(y) ; la valeur numérique 2 est copiée dans une variable provisoire (que nous appellerons t)que la fonction

DoubleEntier va utiliser. La variable y ne peut être modifiée par DoubleEntier

En fait DoubleEntier reçoit une copie de la valeur de y mais ne peut pas toucher à la variable y. Même si on modifie t…cela ne modifiera pas y.

b) Passage par adresse – utilisation des tableaux

Pour que la fonction puisse modifier ses paramètres (c'est-à-dire ses arguments) il faut utiliser les adresses (dans la mémoire de l'ordinateur) des variables. Pour avoir l'adresse des variables on a deux possibilités :

- soit on utilise des pointeurs. Les pointeurs sont des variables particulières. Nous verrons cette notion de pointeur dans un prochain cours.
- Soit on utilise directement l'adresse des variables grâce au symbole & qui signifie « adresse de » (&a : adresse de a).Ex : dans scanf on met toujours & devant les variables pour modifier ces variables. scanf(« %d »,&a) ; si on écrit scanf(« %d »,a) sans le &, la fonction scanf ne pourra jamais modifier la variable a.

Les tableaux jouent un rôle particulier au niveau des fonctions car si un paramètre d'une fonction est un tableau et que l'on fournisse un tableau comme argument de la fonction, alors cette fonction peut modifier directement ce tableau…Ceci est dû au fait que le tableau est un pointeur…(nous reviendrons plus tard sur les pointeurs et les tableaux).

Ex :

```
void ModifTableau(int T[5])
{
        T[1]=T[3] ;
        T[0]=0 ;
}

int main (void)
{
        int A[5]={1,2,3,4,5} ;

        ModifTableau(A) ;
/* après l'appel de cette fonction, on a A[0]=0, A[1]=4,A[2]=3, A[3]=4, A[5]=5*/
}
```