Université Paul Sabatier TOULOUSE III

S3

Programmation impérative en langage C

feuille de TP n°4 : fonctions

Objectif du TP : Ecrire des sous-programmes avec leur spécification
L'analyse du problème à résoudre conduit à une formalisation appelée **spécification**. *Il s'agit d'un triplet :*

$$/*Q*/ \quad , prog\_sol([E],[es],[s]) \quad , /*R*/$$

où :

- $/*Q*/$ *: est un prédicat d'entrée contenant les informations disponibles pour résoudre le problème, hormis le type des données*
- $prog\_sol([E],[es],[s])$ *: est l'abstraction du programme exprimée par une action paramétrée par 3 listes : d'entrée E, d'entrée/sortie es ou de sortie s.*
  - *. E liste des valeurs en entrée*
  - *. s,es listes des variables contenant le résultat identifiées avant le développement et à partir du problème*
    - *s : zones mémoires contenant le résultat*
    - *es : zones mémoires contenant des valeurs en entrée et modifiées dans le programme*
- $/*R*/$ *: est un prédicat de sortie qui spécifie les résultats attendus.*

Conventions :

- *On note en majuscule : les constantes et les variables mathématiques (qui sont utilisées dans les prédicats Q et R)*
- *On note en minuscule : les variables du programme*

Rappels :

- Chaque fonction doit être testée et mise au point avant d'être utilisée dans l'écriture d'autres fonctions.
- Il n'y a **ni saisie ni affichage** dans une fonction à moins que cela ne soit sa vocation.

1 Généralités

Pour chaque exercice, on créera un main pour tester les fonctions demandées.

◇ Exercice **1** :

1. Ecrire une fonction f1 qui pour un caractère donné :
   - si ce caractère est une lettre minuscule, renvoit la majuscule correspondante
   - si ce caractère est une lettre majuscule, renvoit la minuscule correspondante
   - sinon renvoit le caractère non modifié

```
//Données : char C
//Résultats : char d
//PE : ⊤
char f1 (char c);
//PS : ( C in alphabet min -> d=c+ 'A'-'a') and ( C in alphabet maj -> d=c+ 'a'-'A') and
( not(C in alphabet) -> d=C )
```

Exemple :

```
f1('a')='A'
f1('A')='a'
f1(';')=';'
```

2. Utiliser cette fonction pour modifier toutes les lettres d'une phrase entrée au clavier se terminant par le caractère '.'

```
Taper une phrase: 3:Le Docteur JIVAGO.
3:lE DOCTEUR jivago.
```

Remarque : pour répondre à cette question, aucun tableau n'est nécessaire, on traite les caractères au fur et à mesure de leur saisie.

◇ Exercice **2** : Ecrire une fonction qui calcule la différence et le quotient d'un entier et d'un réel.

```
//Données : int e, float x
//Résultats : float dif, float q
//PE : ⊤
void diff_quot (int e, float x, float* dif,float* q);
//PS : ( *dif=e-x) and (*q= e/q)
```

Exemple :

```
Donner un entier: 5
Donner un réel: 2.32
La difference entre 5 et 2.32 est 2.68 et le quotient 2.1552
```

◇ Exercice **3** : Ecrire une fonction qui prend en entrée 2 paramètres et les modifie en permutant leurs valeurs.

```
//Données : char C1, char C2
//Résultats : char c1, char c2
//PE : ⊤
void permute( char* c1, char* c2);
//PS : ( *c1=C2) and (*c2= C1)
```

Exemple :

```
Avant permutation x='a' et y='k'
Après permutation: x='k' et y='a'
```

◇ Exercice **4** :

```
int bits[10]=0;
int i=5;

void quoi ( int* x)
{
i=i+1;
*x=*x+1;
}

int main (void)
{
quoi(&(bits[i]));
return 0;
}
```

Quelle case du tableau est modifiée ?

Dans les exercices suivants vous devez écrire une spécification pour chaque fonction.

◇ Exercice **5** : Ecrire une fonction qui convertit une durée exprimée en secondes, en heures, minutes et secondes.(Bien sûr, l'affichage ne se fait pas dans la fonction)
Exemple :

```
Donner une durée (exprimée en secondes): 12548
12548s= 3h 29mn 8s
```

◇ Exercice **6** : Ecrire une fonction qui supprime d'un vecteur de n entiers toutes les valeurs paires sans changer l'ordre des valeurs. (On n'utilisera pas de tableau auxiliaire).
Exemple :

```
Nombre d'entiers du vecteur: 7
Donner les valeurs: 1 0  5  6  7  3  56
Aprés élimination des pairs, t= 1 5  7 3
```

2 Nombres parfaits, nombres amis

◇ Exercice **7** :

1. Ecrire une fonction de saisie d'un nombre entier strictement positif.
2. Ecrire une fonction qui détermine la somme des diviseurs d'un nombre sauf ce nombre. Exemple : $1 + 2 + 4 + 7 + 14$ est la somme des diviseurs de 28
3. Nombres parfaits :
   On rappelle qu'un nombre entier positif est dit parfait s'il est égal à la somme de ses diviseurs sauf lui-même.
   - Déduire de la fonction précédente une fonction qui détermine si un nombre est parfait.
   - Ecrire une fonction qui affiche tous les nombres parfaits compris entre 5 et 30 ? entre 100 et 400 ? entre 1000 et 5000 ? (On écrira une fonction qui fait le travail pour deux nombres passés en paramètre) .

- Quel est le nombre parfait le plus proche de 300 ? de 800 ? de 2000 ? (Ecrire une fonction qui permet de répondre...)
- Ecrire une fonction qui détermine (sans les afficher) tous les nombres parfaits compris entre deux nombres donnés.

4. Nombres amis :
   Deux nombres entiers sont amis si chacun est égal à la somme des diviseurs de l'autre nombre (sauf le nombre).
   par exemple :($220 = 1 + 2 + 4 + 71 + 142$ (diviseurs de 284)) et $284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110$ (diviseurs de 220) sont amis.
   On peut remarquer qu'un nombre parfait est ami avec lui-même.
   - Ecrire une fonction qui détermine si deux nombres sont amis.
   - Ecrire une fonction qui détermine tous les couples de nombres amis compris entre deux nombres donnés.
     Exemple : Les nombres amis compris entre 200 et 400 sont : 220 et 284 .

5. Menu :
   présenter le travail qui précède sous forme d'un menu :

```
Choisissez:
1: Un entier est-il parfait?
2: Chercher le parfait le plus proche.
3: Tous les parfaits d'un intervalle.
4: Deux entiers sont-ils amis?
5: Toutes les paires d'amis d'un intervalle.
Pour s'arreter taper 6.
```

3 Récursivité

◇ Exercice **8** : Ecrire une fonction qui calcule la factorielle d'un nombre. On écrira d'abord une version classique puis avec paramètre accumulateur dans la liste des paramètres.

◇ Exercice **9** : Les nombres de Fibonacci sont définis par la relation :

$$F_0 = 0,\ F_1 = 1,\ \text{et pour } n \geq 2,\ F_n = F_{n-1} + F_{n-2}$$

Ecrire une fonction récursive, Fibo, qui calcule le nième nombre de Fibonacci.
Afficher les 15 premiers nombres de Fibonacci.

◇ Exercice **10** : Critère de divisibilité par 3 :

1. Ecrire une fonction SommeChiffre prenant en argument un entier n et qui calcule récursivement la somme des chiffres qui le forment.
2. Un entier $c_n c_{n-1} \cdots c_1 c_0$ est divisible par 3 si la somme des chiffres qui le forment est elle-même divisible par 3. Autrement dit :

$$n\%3 = 0 \Leftrightarrow \Big[\sum_{i=0}^{n} c_i\Big]\%3 = 0$$

On remarquera que la somme des chiffres d'un entier supérieur à 10 est toujours strictement plus petite que cet entier. Utiliser cette remarque pour construire un algorithme récursif permettant de déterminer si un entier est divisible par 3.

Ecrire une fonction récursive div_rec_3 qui détermine si un entier est divisible par 3 en utilisant cet algorithme.

◇ Exercice **11** : Liste des parties d'un ensemble :
Disposant de la liste des parties d'un ensemble à $n-1$ éléments, comment en déduire la liste des parties d'un ensemble à n éléments ?
Ecrire un programme qui demande à l'auteur un entier n ($0 \leq n \leq 11$) puis qui affiche les parties de l'ensemble $E = \{1, \cdots, n\}$.
On pourra utiliser une matrice dans laquelle on stockera le contenu d'une partie sur chaque ligne.

```
Taper un entier n: 3
partie 1: (000)
partie 2: (001) {1}
partie 3: (010) {2}
partie 4: (011) {1,2}
partie 5: (100) {3}
partie 6: (101) {1,3}
partie 7: (110) {2,3}
partie 8: (111) {1,2,3}
```

Modifier le programme précédent pour qu'il permette de traiter tout ensemble de caractères.