TP 4 : Lien entre pointeurs, tableaux et passage de paramètres en C

Notions : Fonctions, Pointeurs, Tableaux

1 Passage par adresse dans des fonctions

1.1 Echange des variables

Echange de 2 variables : écrire une fonction `swap2` qui échange le contenu de deux variables réelles. Vérifier son fonctionnement à l'aide du programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    double a = 10, b = 2;
    swap2Double( &a, &b );
    printf("Valeur de a=%d et b=%d apres l'echange\n",a,b);
    /* ici a doit valoir 2 et b 10 */
    return EXIT_SUCCESS;
}
```

Echange de 3 variables : écrire une fonction `swap3` qui effectue une permutation circulaire vers le droite de trois variables, **en utilisant la fonction swap2 précédente** pour les échanges effectués dans `swap3`.Vérifier son fonctionnement à l'aide du programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    double a = 10, b = 2, c = 3;
    swap3Double(&a,&b,&c);
    /* ici a == 3, b == 10, c == 2 */
    printf("Valeur de a=%d  b=%d c=%d apres l'echange\n",a,b,c);
    return EXIT_SUCCESS;
}
```

1.2 Somme et produit

Produit de 2 variables : écrire une fonction qui **retourne** le produit de ses 2 paramètres réels.

Somme de 2 variables : écrire une fonction qui calcule la somme de 2 réels **a** et **b**. Cette fonction prend 3 paramètres : les deux réels **a** et **b**, ainsi qu'une variable pour stocker la somme. **La fonction ne retourne aucune valeur**. Cette fonction ne fait aucun affichage à l'écran.

Somme et produit de 2 variables : écrire une fonction qui calcule la somme et le produit de ses 2 premiers paramètres. La somme est contenue dans le $3^{\text{ième}}$ paramètre, le produit dans le $4^{\text{ième}}$ paramètre. Cette fonction ne fait aucun affichage à l'écran et ne retourne aucune valeur. **Vous devez utiliser les deux fonctions précédentes pour faire cette fonction**.

Programme complet : écrire un programme qui lit 2 réels au clavier, calcule leur somme et produit en utilisant la fonction précédente, puis affiche cette somme et ce produit.

2 Parcours des tableaux avec des pointeurs

Pour ces questions, on utilisera des **indices pointeurs**. Dans ces fonctions, l'accès à un élément de tableau est noté `*p` ou `*p++` selon le cas. Aucun **indice entier** ne doit être utilisé pour manipuler le tableau de réels, c'est à dire aucune notation de type `t[i]` ou `*(t+i)`.

On donne les fonctions de lecture et d'affichage d'un tableau de n réels, disponibles sur le site dans le fichier `fonctions.c`. Si votre fichier source **C** se nomme `progtp2_1.c`, pour construire l'exécutable correspondant aux 2 fichiers `fonctions.c` et `progtp2_1.c`, vous devez compiler avec la ligne de commande `clang progtp2_1.c fonctions.c -o progrtp2_1.exe` ou `gcc progtp2_1.c fonctions.c -o progrtp2_1.exe`. Le programme s'appelle `progrtp2_1.exe`, si bien que la commande `./progrtp2_1.exe` permet de l'exécuter.

Vous pouvez **(devez)** écrire le code des fonctions des questions 2, 3 et 6 dans le fichier `fonctions.c` et les prototypes dans le fichier `fonctions.h`, en suivant l'exemple des fonctions `lectureTabDouble` et `afficheTabDouble`. Les intérêts essentiels sont de pouvoir ré-utiliser ces fonctions avec les autres programmes demandés dans ce tp ou les suivants, de structurer votre code pour le debugger et le maintenir plus facilement, le rendre plus lisible, etc...

```
void lectureTabDouble(double* t, int n) { int i ;
  for (i=0 ; i<n ; i++) scanf("%lf",t+i) ;
}
void afficheTabDouble(double* t, int n) { int i ;
  for (i=0 ; i<n ; i++) printf("%lf",t[i]) ;
  printf("\n") ;
}
```

2.1 Présence d'un nombre négatif dans un tableau

Adresse du premier element négatif : écrire une fonction `adressePremierNegTabDouble` qui retourne l'adresse du premier élément négatif d'un tableau de n réels. Si tous les éléments sont positifs ou nul, la fonction retourne `NULL`. Son prototype sera :

```
double* adressePremierNegTabDouble(double* t, int n);
```

Programme : écrire un programme qui trouve l'adresse du premier élément non nul d'un tableau.

Si cet élément existe, le programme met cet élément à 0.

Vérifier en utilisant la fonction d'affichage des éléments d'un tableau proposé ci dessous.

3 Parcours de tableaux avec plusieurs pointeurs

Pour ces questions, on utilisera des **indices pointeurs**. Dans ces fonctions, l'accès à un élément de tableau est noté `*p` ou `*p++` selon le cas. Aucun **indice entier** ne doit être utilisé pour manipuler le tableau de réels, c'est à dire aucune notation de type `t[i]` ou `*(t+i)`.

Ecrivez les fonctions dans le fichier `fonctions.h` et le programme principal dans le fichier `progtp3_1.c`. Pour compiler, utilisez alors la commande `clang progtp3_1.c fonctions.c -o progtp3_1.exe` et executez avec la commande `./progtp3_1.exe`

3.1 Symétrie d'un tableau

Symétrie d'un tableau Tester si les éléments du tableau `t1` de `n` réels sont symétriques par rapport à son centre

Il s'agit de tester si les éléments du tableau sont symétriques : le premier élément est identique au dernier, le deuxième élément est identique à l'avant dernier, etc....

On parcourt le tableau `t1` avec 2 pointeurs : `p1` sur le début, `p2` sur le dernier. On regarde si l'élément pointé par `p1` et celui pointé par `p2` sont identiques. S'il ne sont pas identiques, la réponse

est donc connue (pas de symétrie) et on quitte la fonction avec la valeur 0. Sinon, on incrémente le pointeur `p1` pour qu'il pointe sur l'élément suivant, on décrémente `p2` pour qu'il pointe sur l'élément précédent et on itère tant que `p1` est inférieur à `p2`. Si tous les éléments ont été parcourus, alors le tableau est symétrique et la fonction retourne 1. La fonction retourne 1 si le tableau est symétrique, 0 sinon.

Ecrire dans le fichier `fonctions.c` la fonction `estSymetriqueTabDouble` dont le prototype est :

`int estSymetriqueTabDouble(double t[], int n);`

Programme Faire un programme pour tester cette fonction. La structure du programme, téléchargeable sur le site des tp, ressemblera à :

```
#include <stdio.h>
#include <stdlib.h>
#include "fonctions.h"
#define DIM 10

int main(void) {
                 /* Le tableau  */
    double t1[DIM];
                /* La dimension reellement utile */
    int dim1;

    /* Combien de nombres dans le premier tableau ? */
    do {
        printf("Combien d'elements dans le premier tableau (doit etre inferieur a %d)\n",DIM);
        scanf("%d",&dim1);
    } while  (dim1<=0 || dim1>=DIM);
                /* Lecture clavier des nombres du premier tableau puis affichage des nombres lus */
    printf("Entrer les elements du tableau\n");
    lectureTabDouble(t1,dim1);
    afficheTabDouble(t1,dim1);

                /* A Completer : appel de la fonction */
    if (estSymetriqueTabDouble (/* A Completer ....*/)
      printf("Le tableau est symetrique\n");
    else
      printf("Le tableau n'est pas symetrique\n");

    return EXIT_SUCCESS;
}
```

4 Chaînes de caractères

Ecrire les fonctions suivantes en utilisant des **indices entiers**. **Vous n'utiliserez bien sûr pas les fonctions de la bibliothèque C commençant par `(strxxx)`.**

Ecrire ces fonctions dans un fichier `chaine.c` et leurs prototypes dans un fichier `chaine.h`. Si le fichier contenant le programme principal est `progtp4_1.c`, il sera compilé en utilisant la ligne de commande `clang progtp4_1.c chaine.c -o progtp4_1.exe` et vous pourrez l'exécuter avec la commande `./progtp4_1.exe`

Rappels :

1. une chaîne de caractères en C est simplement un tableau d'octet, donc défini par une instruction de type `char tmp[512]; // une chaine de 512 caracteres`.
2. Cependant, toutes les fonctions travaillant avec la notion de chaînes de caractères comme `printf("%s", tmp);` ou toutes les fonctions de la bibliothèque `string` comme (`strlen, strcat, strcpy, strxxx` en général) sont basées sur la présence de la marque de fin de chaîne (le caractère `'\0'` dont la valeur est 0) dans le tableau `tmp`. Tout ce qui suit cette marque dans le tableau n'est pas utilisé par ces fonctions, seule la partie précédent cette marque est considérée comme utile.

3. C'est pourquoi les fonctions qui *remplissent* une chaîne de caractères comme `scanf("%s",tmp);` ajoutent cette marque de fin de chaîne automatiquement. Quand vous tapez la chaîne `"Phelma"`, seuls les 6 premiers octets de `tmp` seront utilisés, le 7^{ieme} sera la marque de fin de chaîne et les 512-7 autres seront inutilisés.
4. Lorsque vous créez ou remplissez vous-même une chaîne, par exemple en copiant une chaîne dans une autre, n'oubliez pas de bien positionner vous-même cette marque de fin de chaîne.

Pour une lire une chaîne de caractères définie par la variable `tmp` (`char tmp[512];`), vous avez 2 possibilités, qui ont un comportement légèrement différent :

— l'instruction : `scanf("%s",tmp);`

— les 2 instructions : `fgets(tmp,511,stdin); if (tmp[strlen(tmp)-1]=='\n') tmp[strlen(tmp)-1]='\0';`

4.1 Longueur d'une chaîne de caractères

Longueur d'une chaîne : Ecrire une fonction `longueurChaineInt` qui retourne le nombre de caractères d'une chaîne.

Le prototype de cette fonction est : `int longueurChaineInt(char* s);`

Programme : Ecrire un programme qui lit une chaîne de caractère au clavier puis affiche la longueur de cette chaîne.

Pour la lecture de la chaîne, en supposant que la variable `tmp` soit définie comme un tableau de 512 éléments, quelle différence faites-vous entre les instructions `scanf("%s",tmp)` et `fgets(stdin,511,tmp)` ?

4.2 Nombre d'espaces dans une chaîne

Nombre d'espaces dans une chaîne : Ecrire une fonction `nbEspaceChaine` qui compte le nombre de caractère ESPACE présent dans la chaîne `s`. Le caractère ESPACE peut se noter de 2 manières : soit sous forme de caractère `' '`, soit avec son code ASCII c'est à dire la valeur numérique 32.

Le prototype de cette fonction est : `int ChaineInt(char* s);`

Programme : Ecrire un programme qui lit une chaîne de caractère, affiche la chaîne, affiche le nombre d'espaces dans la chaîne.

Utiliser la fonction `fgets` pour la lecture de la chaîne.

5 Chaînes de caractères et pointeurs

Pour ces questions, on utilisera des **indices pointeurs**. Dans ces fonctions, l'accès à un élément de tableau est noté `*p` ou `*p++` selon le cas. Aucun **indice entier** ne doit être utilisé pour manipuler le tableau de caractères, c'est à dire aucune notation de type `t[i]` ou `*(t+i)`.

Vous n'utiliserez bien sûr pas les fonctions de la bibliothèque C `(strxxx)`.

Ecrire ces fonctions dans un fichier `chaine.c` et leurs prototypes dans un fichier `chaine.h`. Si le fichier contenant le programme principal est `progtp5_1.c`, il sera compilé en utilisant la ligne de commande `clang progtp5_1.c chaine.c -o progtp5_1.exe` et vous pourrez l'exécuter avec la commande `./progtp5_1.exe`

5.1 Longueur d'une chaîne de caractères

Longueur d'une chaîne : Ecrire une fonction `longueurChainePtr` qui retourne le nombre de caractères d'une chaîne.

Le prototype de cette fonction est : `int longueurChainePtr(char* s);`

Programme : Ecrire un programme qui lit une chaîne de caractère au clavier puis affiche la longueur de cette chaîne.

Pour la lecture de la chaîne, en supposant que la variable `tmp` soit définie comme un tableau de 512 éléments, quelle différence faites-vous entre les instructions `scanf("%s",tmp)` et `fgets(stdin,511,tmp)` ?

5.2 Copie d'une chaîne de caractères

Copie d'une chaîne : Ecrire une fonction `copieChainePtr` qui copie une chaîne de caractère dans une autre.

On utilisera 2 pointeurs, l'un sur la premiere chaîne, l'autre sur la deuxième chaîne qui évolueront simultanément.

Le prototype de cette fonction est : `int copieChainePtr(char* dst, char* src);`

Programme : Ecrire un programme qui lit une chaîne de caractère au clavier puis copie cette chaîne dans une autre chaîne et affiche la nouvelle chaîne.

Pour la lecture de la chaîne, utiliser la fonction `fgets`.

6 Facultatif

6.1 Mélanger les éléments d'un tableau

Pour mélanger les N éléments d'un tableau de réels, on tire au hasard [1] un nombre entier `i1` compris entre 0 et N-1 [2] et on échange l'élément d'indice `i1` avec le dernier élément du tableau non encore mélangé (indice N-1). On recommencera avec un autre nombre `i1`, tiré entre 0 et **N-2**, qui sera échangé avec l'élément d'indice N-2, puis avec un autre nombre `i1`, tiré entre 0 et **N-3**, qui sera échangé avec l'élément d'indice N-3, etc.

— Combien de fois au minimum faut il faire cette action pour échanger tous les éléments du tableau ?

— Combien de fois au maximum faut il faire cette action pour échanger tous les éléments du tableau ?

— Ecrire dans le fichier `fonctions.c` une fonction qui réalise ce mélange et un programme pour tester cette fonction. Son prototype sera `void melangeTabDouble(double* t, int n);`

La structure du programme ressemblera à :

```
#include "fonctions.h"
#include <stdlib.h>
#define DIM 100

int main(void) {
                /* Le tableau utilise */
    double tab[DIM];
                /* La dimension reellement utilisee */
    int dimTab;

    /* Combien de nombres dans le premier tableau ? */
    do {
        printf("Combien d'elements dans le premier tableau (doit etre inferieur a %d)\n",DIM);
        scanf("%d",&dimTab);
    } while  (dimTab<=0 || dimTab>=DIM);
                /* Lecture clavier des nombres du premier tableau puis affichage des nombres lus */
    printf("Entrer les elements du tableau\n");
    lectureTabDouble(tab,dimTab);
    afficheTabDouble(tab,dimTab);

                /* A Completer : appel de la fonction */
    melangeTabDouble(/* A Completer ....*/);

                /* Affichage du resultat */
    afficheTabDouble(tab,dimTab);
    return EXIT_SUCCESS;
}
```

1. Les fonctions `int rand()` ou `int random()` retournent un entier aléatoire compris entre 0 et RAND_MAX
2. L'opérateur modulo `%` du C peut être utilisé ici