Structures de données : les bases

Un premier programme C sous LINUX

Dans un terminal LINUX, créez un répertoire de travail (`3i_in9`, par exemple) et placez-y vous.

```
acme1:~> mkdir 3i_in9 ↵
acme1:~> cd 3i_in9 ↵
acme1:~/3i_in9>
```

Avec un éditeur de texte (par exemple, gedit),

```
acme1:~/3i_in9> gedit & ↵
```

saisissez le programme source C suivant :

```
/*
pgm01.c Un premier exemple de programme C
*/
#include<stdio.h>
int main()  {
  int n;
  printf("Entrez un nombre entier : ");
  scanf("%d", &n);
  printf("%d x %d = %d\n", n, n, n*n);
  return 0;
}
```

Sauvegardez-le sous le nom `pgm01.c`, puis compilez-le :

```
acme1:~/3i_in9> gcc pgm01.c ↵
```

Si la compilation se passe bien, un fichier exécutable de nom `a.out` est créé et vous pouvez l'exécuter en lançant la commande :

```
acme1:~/3i_in9> ./a.out ↵
```

Un message apparaît, correspondant à la ligne 7 du listing.

```
Entrez un nombre entier :
```

Le programme s'interrompt alors, attendant l'intervention de l'utilisateur pour la saisie du nombre (ligne 8 du listing).

Si on entre 12, par exemple, le programme affiche le résultat (ligne 9 du listing), puis se termine :

```
Entrez un nombre entier : 12 ↵
12 x 12 = 144
```

Exemple d'écriture d'une fonction

Écrivez deux fonctions calculant la factorielle de n, la première à l'aide d'une boucle `for` et la deuxième en utilisant la récursivité.

Exemple de solution :

```
/* int factorielle_iterative(int n)
 * calcule la factorielle de n
 * Principe : accumulation des produits des n premiers nombres
 * arguments : n entier positif
 * retour : factorielle de n si n>=0
                    1 si n est negatif
 * JCG ecrit le 22/03/2001 modif le 14/04/2013
 */
int factorielle_iterative(int n) {
  int i, p = 1;
  for (i = 1; i <= n; ++i) p *= i;
  return p;
}
/*****************************************************************/
/* int factorielle_recursive(int n)
 * calcule la factorielle de n
 * Principe : F(n)=1 si n=0, F(n)=n*F(n-1) si n>0
 * arguments : n entier positif
 * retour : factorielle de n si n>=0
                    1 si n est negatif
 * JCG ecrit le 22/03/2001 modif le 14/04/2013
*/
int factorielle_recursive(int n) {
  if (n <= 0)
    return 1;
  else
    return n * factorielle_recursive(n-1);
}
/*****************************************************************/
#include <stdio.h>
int main(void) {
  int n;
  do {
    printf("Entrez un nombre entier (0 pour terminer) : ");
    scanf("%d", &n);
    printf("factorielle_iterative(%2d) = %15d\n", n, factorielle_iterative(n));
    printf("factorielle_recursive(%2d) = %15d\n", n, factorielle_recursive(n));
  } while (n != 0);
  return 0;
}
```

Compilez et exécutez ce programme. Entrez les valeurs 1, 2, 5, 10, 13. Comment peut-on voir que le résultat pour 13 est faux ? Pourquoi est-il faux ?

Comment modifier les fonctions pour que le nombre retourné soit un nombre impossible (-1 par exemple) lorsque le résultat ne peut pas être calculé (Indice : https://www.tutorialspoint.com/c_standard_library/limits_h.htm) ?

1 Les types du langage C

1.1 Types de base

Exercice 1. Détermination de type

Choisissez parmi les types de base (`char`, `int`, `long`, `double`, `char[]`) le type vous semblant le plus adapté pour représenter les données suivantes :

- une année de naissance ;
- un numéro de téléphone ;
- un numéro de département français ;
- un taux de TVA ;
- une taille de fichier ;
- un numéro de chambre d'hôtel.

Exercice 2. Proximité de deux `double`

Avec les nombres flottants, il est fortement déconseillé de tester un nombre flottant avec une égalité stricte (par exemple, `if (a == 0.0)`). En effet, la precision des nombres flottants étant fini, les résultats d'opérations sur les nombres flottants sont approximés. Il est donc préférable d'utiliser une fonction pour tester l'égalité approximative.

Avec la définition suivante,

```
#define REL_TOL 1e-9
#define ABS_TOL 0.0
```

et en vous inspirant de la documentation python suivante : https://www.python.org/dev/peps/pep-0485/, écrivez la fonction

```
int proche(double a, double b);
```

qui renvoie un entier non nul si `a` et `b` sont proches l'un de l'autre et 0 sinon.

Exercice 3. Limite d'un `float`

Dans cette exercice, nous allons chercher à atteindre la limite des nombres flottants. Pour cela nous allons simplement générer N nombres aléatoires uniformément distribué entre 0 et 20 et calculé la moyenne de ces nombres.

Dans le fichier `limite_float.c` vous trouverez le code suivant :

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

float rand_float(float min, float max){
        return rand()/((float) RAND_MAX)*(max-min)+min;
}

int main(){
        srand( time( NULL ) );
        size_t N = 10000;

        float x = 0.0f;
        for(size_t i = 0 ; i < N ; i++)
                x += rand_float(0, 100);
        x = x / N;

        printf("Moyenne = %f\n", x);

        return 0;
}
```

Compilez et testez ce code pour différente valeur de $N = 100000, 1000000, 10000000, 100000000$. Que constatez vous ? Pourquoi cela se produit-il ? Proposez une modification de ce code pour qu'il retourne -1 si le résultat est erroné.

Exercice 4. Visualisation de `double`

Nous voulons "visualiser" le codage hexadecimal d'un `double` (voir le site : http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml).
Sachant que un `double` est stocker en mémoire sous la forme de 8 octets, proposez une méthode pour convertir une variable de type `double` en un tableau de `unsigned char[8]`.
Astuces : pensez à la conversion (cast) de pointeur ou à l'utilisation du type `union` (voir : https://gcc.gnu.org/onlinedocs/gcc/Unnamed-Fields.html)
Écrivez la fonction

```
void print_hexa(double x);
```

qui affiche la valeur (avec la précision maximum) et le codage hexadécimal du double `x`.
Testez votre fonction, affichiez la valeur d'un double avant et après lui avoir ajouter une très petite valeur (par exemple :10^{-100}). Visualisez le codage utilisé pour les valeurs `inf` et `nan`.

1.2 Les pointeurs

Un pointeur est une variable destinée à contenir l'adresse d'une zone mémoire typée. Le rôle principal d'un pointeur est de permettre d'accéder à une zone mémoire de façon indirecte, pour pouvoir la modifier.

Exercice 5. Modification d'une variable par pointeur

Écrivez la fonction suivante

```
void eleve_au_carre(int* p) {
/* modifie en l'elevant au carre l'entier dont l'adresse est dans p */
// a completer
}
```

Testez-la avec :

```
int main() {
    int n = 5;
    eleve_au_carre(&n);
    printf("%d\n", n); /* => 25 */
    return 0;
}
```

1.3 Les tableaux

En C, l'accès à un élément de tableau n'est pas sécurisé. Pour éviter de *sortir* d'un tableau, on utilise principalement deux solutions :

- on fournit à la fonction travaillant sur le tableau la longueur exploitable du tableau : tout accès peut alors être testé ;
- on marque la *fin* du tableau par une sentinelle (un élément impossible) : utilisable principalement pour une exploitation séquentielle.

Exercice 6. Parcours de tableau

Complétez les fonctions manquantes du programme suivant. Vérifiez que vos fonctions passent les tests.

```c
#include <stdio.h>
/* la reponse a la question precedente */
#define REL_TOL 1e-9
#define ABS_TOL 0.0
/* inclure <math.h> et compiler avec gcc -lm */
int proche(double a, double b) {
    // Votre code de l'exercice 2
}

double moyenne(double t[], int n) {
/* calcule la moyenne des n premiers elements du tableau t */
/* --------------
 * a faire
 * --------------
*/
  return 0.0;
}

double moyenne_positifs(double t[]) {
/* calcule la moyenne des elements du tableau t jusqu'a rencontrer un
element negatif et -1.0 si le premier element est deja negatif */
/* --------------
 * a faire
 * --------------
*/
  return 0.0;
}

double test_moyenne() {
  double v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, -1.0};
  double d, attendu;
  attendu = 1;
/* test moyenne */
  if (! proche((d = moyenne(v, 1)), attendu)) {
    printf("Pb moyenne. Attendu : %f Obtenu :%f\n", attendu, d);
  }
  attendu = 2.0;
  if (! proche((d = moyenne(v, 3)), attendu)) {
    printf("Pb moyenne. Attendu : %f Obtenu :%f\n", attendu, d);
  }
  attendu = 3.5;
  if (! proche((d = moyenne(v, 6)), attendu)) {
    printf("Pb moyenne. Attendu : %f Obtenu :%f\n", attendu, d);
  }
/* test moyenne positifs */
  attendu = 3.5;
  if (! proche((d = moyenne_positifs(v)), attendu)) {
    printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
  }
  attendu = 5.0;
  if (! proche((d = moyenne_positifs(v + 3)), attendu)) {
    printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
  }
  attendu = -1.0;
  if (! proche(d = moyenne_positifs(v + 6), -1.0)) {
    printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
  }
}
int main() {
  test_moyenne();
  return 0;
}
```

Exercice 7. Parcours récursifs de tableau

Écrivez la fonction :

```c
int chaine_longueur_rec(char* s);
```

qui calcule **récursivement** la longueur d'une chaîne de caractères terminée par `'\0'`.

Écrivez la fonction **récursive** :

```
int chaine_debute_par(char* s1, char* s2);
```

qui retourne un entier non nul si `s1` commence par `s2`, et 0 sinon.

Exercice 8. Recherche de motif

Écrivez la fonction ;

```
int chaine_index(char* s1, char* s2);
```

qui retourne la position de `s2` dans `s1` si `s1` contient `s2`, -1 sinon.