TP: premiers exercices en C

Compléments

Fonctions et procédures

Dans ce premier TP, nous utilisons parfois des *procédures* de zéro ou un argument pour *factoriser* le code. Voici comment introduire de telles procédures.

```
void procedure1 () //une procedure sans argument
  {
    // corps de procedure1
  }

void procedure2 (float f) //une procedure avec argument de type float
  {
    // corps de procedure2
  }
```

Nous pouvons aussi utiliser des fonctions à un paramètre comme celle-ci :

```
int doubler(int n)
  {// renvoie le double de n
   return 2*n;
}
```

Le mot clé `return` est le même qu'en PYTHON.

Tableaux

Pour déclarer par extension un tableau de 3 éléments d'un type donné, nous utilisons la syntaxe

`type nomTab[longueurTab] = {valeur1, valeur2,...}`

Par exemple, nous déclarons ainsi un tableau `t` de 3 flottants : `float t[3] = {1.2, 2.56,-1.89 }`. Les éléments du tableau `t` sont alors accessibles comme en PYTHON par `t[0]` vaut 1.2, `t[1]` vaut 2.56 et `t[2]` est égal à -1.89. En revanche, il n'y a pas de fonction clé en main donnant la taille d'un tableau.

Spécifieurs

On trouve une documentation sur les spécifieurs de format pour `prinf` ici et à cet endroit pour `scanf`.

Exercices

Afficher, saisir

Dans cette section, tout le code est dans la fonction `main`.

Exercice 1. Écrire un programme qui demande d'entrer un entier puis affiche sa valeur et son adresse.

On ne cherche pas à prévoir les problèmes de débordement.

```
$ ./a.out
entrer un entier : 35
le nombre choisi est 35, son adresse est 0x7fffc5aa0e44
```

Exercice 2. Demander à l'utilisateur deux nombres, les mémoriser dans deux variables, multiplier leurs valeurs en affectant le résultat à une troisième variable, puis l'afficher.

Exemple de runtime :

```
Produit de deux entiers
Entrez un nombre entier : 12
Entrez un deuxieme nombre entier : 3

12 x 3 = 36
```

Exercice 3. Demander à l'utilisateur combien font 2 fois 2 et répéter cette question aussi longtemps que la réponse est fausse.

Ajouter le message "Faux, recommencez" à chaque fausse réponse, et "Bravo !" pour la bonne réponse.

Exemple d'exécution :

```
Combien font 2 x 2 ? 5

Faux, recommencez
Combien font 2 x 2 ? 4

Bravo !
```

Exercice 4. Demander à l'utilisateur de saisir des notes (flottants entre 0 et 20) et lui expliquer qu'une valeur hors de cet intervalle arrêtera la saisie.

Compter les notes saisies. Une fois la saisie terminée, afficher le nombre de notes saisies et la somme de toutes les notes saisies.

Un message d'erreur s'affiche si aucune note dans l'intervalle n'a été saisie.

Exemple avec saisie de notes valides :

```
Entrez des notes (entre 0 et 20)
Pour stopper, saisir une note hors de cet intervalle.
12
vous avez saisi 12.000000. Saisissez une autre note
13
vous avez saisi 13.000000. Saisissez une autre note
-1
vous avez saisi -1.000000.
saisie de 1 note.s valides.
somme des notes valides saisies 25.000000
```

Exemple sans saisie de notes valides :

```
Entrez des notes (entre 0 et 20)
Pour stopper, saisir une note hors de cet intervalle.
-1
vous avez saisi -1.000000. ERREUR : Vous n'avez pas saisi de note valide.
```

Remarque. La fonction `scanf` permet de saisir plusieurs données en une fois. Par exemple, supposant les variables flottantes `x,y,z` bien définies :

- Pour récupérer les valeurs de 3 flottants en les séparant par des espaces : `scanf("%f%f%f",&x,&y,&z)`
  L'utilisateur saisit par exemple `13.2 12.5 -23.0`
- En séparant les flottants par un symbole comme « ; » (point-virgule) : `scanf("%f;%f;%f",&x,&y,&z);`
  L'utilisateur saisit par exemple `13.2;12.5;-23.0`

Exercice 5.

1. Ecrire une procédure qui demande d'entrer deux flottants et affiche ensuite leur produit avec 3 décimales seulement.

```
$ ./a.out
entrer deux réels : 1.365 2.896231
1.365000*2.896231=3.953
```

2. Ecrire un programme qui demande d'entrer une fraction (donc deux entiers séparés par la barre `/`) et retourne la valeur décimale de cette fraction avec 4 chiffres après la virgule.

```
entrer une fraction au format a/b : 2/5
2/5=0.400
```

Remarque. Pour utiliser les fonctions de la bibliothèque `math.h` il faut ajouter

```
1 #include<math.h>
```

au début du code et compiler avec l'option `-lm` qui permets d'ajouter les bons liens.

Exercice 6. Ecrire un programme qui demande d'entrer les 3 coefficients réels d'un polynôme du second degré sous la forme `ax^2+bx+c` (sans espace) et affiche les racines (si elles sont réelles) avec une précision de trois décimales.

```
$ ./a.out
Entrer un polynôme du snd degré au format ax^2+bx+c: 2x^2+5x+1
Les racines sont : -0.219 , -2.281
$ ./a.out
Entrer un polynôme du snd degré au format ax^2+bx+c: 1x^2+1x+1
Racines complexes.
```

Exercice 7. Ecrire un programme qui demande d'entrer deux durées au format `HH:MM:SS` comme `01:23:18` et `00:59:33` et affiche 0,1 ou 2 selon que la première durée est plus petite que la seconde, plus grand ou égale. On suppose que le format donné par l'utilisateur est correct sans chercher à le vérifier.

```
entrer 1ere durée au format HH:MM:SS : 02:35:33
entrer 2nde durée au format HH:MM:SS : 02:36:18
0
```

Fonctions

Remarque. La fonction `scanf` a un comportement délicat à assimiler notamment avec les éléments de type `char`. Elle a en effet tendance à comprendre qu'il y a eu un retour chariot avant que l'utilisateur ait saisi le caractère voulu. Une façon d'éviter cela est d'introduire un espace avant le descripteur de format.

Il faut donc écrire `scanf(" %c",&caractere)` au lieu de `scanf("%c",&caractere)`.

Exercice 8. Écrire un programme qui demande de choisir un nombre 0, 1 ou 2. Si le choix est 0, le programme demande de saisir un entier puis l'affiche. Si le choix est 1, on fait de même avec un flottant. Le choix 2 correspond à un caractère.

La gestion de la saisie d'un entier est déléguée à une procédure `void entier()`, celle d'un flottant à `void flottant()` et celle d'un caractère à `void caractere()`

```
Que choisissez-vous ? (entier(0), flottant (1) ou caractère (2)) : 2
entrer un caractère : t
vous avez choisi un caractère : t
```

Exercice 9. On verra dans le cours de maths que la fonction exponentielle est la somme de la série entière $\sum_{n\in\mathbb{N}} \frac{x^n}{n!}$. Une conséquence de cela est que le polynôme

$$P_n(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots \frac{x^n}{n!}$$

est une approximation de e^x.

1. Écrire une fonction de prototype `int factorielle (int n)` qui calcule $n!$
2. Écrire une fonction de prototype `float power(float x, int n)` qui calcule x^n pour n entier.
3. Écrire une fonction de prototype `float expo(float x, int n)` qui calcule $P_n(x)$ pour un flottant x. Est-il pertinent d'utiliser la fonction factorielle ?

Des fonctions de tests doivent être écrites.

Exercice 10. Dans cet exercice on importe la bibliothèque des bouléens avec `#include <stdbool.h>`, ce qui permet d'utiliser `true, false`. On peut si on le désire utiliser un tableau de bouléens de longueur 2 `bool t[2] = {true, false}`.

Il n'y a pas de descripteur de format pour `printf` qui permette d'afficher `true, false` à l'écran en lisant un bouléen. On se débrouille donc autrement en se rappelant que `true` est en fait l'entier 1 et `false` vaut 0 :

```
1 bool b[2] = {true,false};// tableau de bool de lg 2
2 printf("%d,%d\n",b[0],b[1]);
```

donne la trace d'exécution :

```
1 true=1, false=0
```

1. Compléter les 3 procédures `et,ou,non` ci-dessous

```c
#include <stdio.h>

void et(){// afficher table du et
            }

void ou(){// afficher table du ou
            }

void non(){// afficher table du non
            }

int main(void)
{
  printf("table du ET\n");
  et();
  printf("table du OU\n");
  ou();
  printf("table du NON\n");
  non();  }

```

de façon à obtenir

```
$ ./a.out
table du ET
1 &&  1 : 1
1 &&  0 : 0
0 &&  1 : 0
0 &&  0 : 0
table du OU
1 ||  1 : 1
1 ||  0 : 1
0 ||  1 : 1
0 ||  0 : 0
table du NON
! 1 : 0
! 0 : 1
```

Le code des opérateurs binaires doit contenir des boucles imbriquées.

Tableaux statiques

Dans cette section, toutes les fonctions s'accompagnent de fonctions de tests. Par exemple, si on demande d'écrire une fonction `f`, il faut en plus coder une procédure de prototype `void test_f()` qui fait appel à `f` et produit un affichage en conséquence.

Exercice 11. Ecrire une fonction `int produit(int* tab, int size)` qui prend en paramètre un tableau d'entiers dont la taille est `size`.

La fonction renvoie le produit des éléments du tableau. Avec

```c
int main(){
  int tab[5] = {1,2,3,4,5};
  printf("%d\n",produit(tab,5));
  return 0;
}
```

On obtient

```
$ ./a.out
120
```

Exercice 12. Ecrire une procédure `void affiche(int t[], int n)` qui affiche le contenu du tableau `t` de taille `n`.

Par exemple si `t={10,20,30}` alors `affiche(t,3)` produit l'affichage :

```
{10,20,30}
```

Exercice 13. Ecrire une procédure `void inv(int t[], int nb)` qui prend en paramètres un tableau et sa taille et inverse l'ordre des éléments. La complexité spatiale doit être en $O(1)$.

Cette procédure fait bien entendu des *effets de bords*.

Par exemple si `t` est `{1,2,3}`, alors

```
1 inv(t,3);
2 affiche(t,3);
3
```

produit l'affichage

```
{3,2,1}
```

Exercice 14. Ecrire la fonction `bool is_sorted(int t[], int n)` qui indique par un bouléen si un tableau est trié dans l'ordre croissant (`true`) ou non (`false`).

Par exemple, la séquence suivante :

```
1 int t2[] = {-10,1,2,8};
2 printf("is_sorted(t2,4)=%d\n",is_sorted(t2,4));
3
```

produira l'affichage

```
is_sorted(t2,4)=1
```

Exercice 15. Écrire une procédure `void swap(int t[], int i, in j)` qui échange les éléments en position i et j du tableau t. On ne demande pas de tester que i, j sont bien des indices valides du tableau.

Tests :

```
1 void test_swap(void){
2   int t[] = {1,2,3};
3   printf("t="); affiche(t,3);
4   printf("\nappel de swap(t,0,2)\n");  swap(t,0,2);
5   printf("t="); affiche(t,3); printf("\n");
6 }
```

Après compilation et exécution :

```
$ ./a.out
t={1,2,3}
appel de swap(t,0,2)
t={3,2,1}
```

Exercice 16. Consulter le tuto sur le hasard en C. Nous utilisons la fonction `int my_rand(int n)`.

On veut mélanger les éléments d'un tableau aléatoirement. On utilise pour cela l'algorithme du *mélange de Knuth* :

On parcourt le tableau de gauche à droite et , pour chaque indice i, on échange le i-ème élément du tableau avec un élément choisi au hasard entre les positions 0 et i.

Écrire la procédure `void shuffle(int t[], int n)` qui réalise cet algorithme pour un tableau t de taille n.

Les instructions suivantes :

```
int t[] = {1,2,3,4,5};
printf("t=");
affiche(t,5);
knuth(t,5);
printf("\nAprès knuth(t,5), t=");
affiche(t,5);
```

produisent :

```
t=[1,2,3,4,5]
Après knuth(t,5), t=[4,1,5,3,2]
```

Exercice 17. Écrire une procédure `void tri_2_valeurs (t[], n)` qui prend en paramètres un tableau de deux valeurs 0 et 1 (on ne demande pas de vérifier ce fait).

La procédure met tous les 0 à gauche et tous les 1 à droite du tableau. La complexité doit être proportionnelle au nombre n d'éléments.

Avec les instructions :

```
int t[]={0,1,0,1};
printf("t=");
affiche(t,n);
tri_2_valeurs(t,n);
printf("\nAprès tri_2_valeurs(t,%d) t=",n);
affiche(t,n);
printf("\n");

```

on obtient :

```
t=[0,1,0,1]
Après tri_2_valeurs(t,4) t=[0,0,1,1]
```

Exercice 18. (figure au programme de colle) Écrire une fonction `int min(int t[], int n)` qui prend en paramètres un tableau et sa taille. La fonction retourne le minimum du tableau.

Étudier la complexité.

Tests :

```
void test_min(){
  int t[7] = {3,5,2,7,1,8,9};
  printf("mint(t)=%d\n",min(t,7));
}

```

Après compilation et exécution, on obtient :

```
$ ./a.out
mint(t)=1
```