Université de Corse - Pasquale PAOLI

Diplôme : Licence SPI 3ème année, parcours Informatique | 2024-2025

UE : Concepts des langages de programmation, Partie Impératif, langage C

Enseignants : Marie-Laure Nivet

Exercices C – partie 2

Pour tous les exercices il vous est demandé de respecter la structure de code proposée en cours via le fichier structureProgrammeC.c, de faire un fichier de déclaration *.h (cf transparents du cours n°200 à 207 ainsi que 214 à 217).
Vous veillerez également, à assurer la lisibilité de vos codes et pour cela, à les commenter, les indenter, à choisir les noms de vos variables et de vos fonctions, bref à respecter le manifeste que vous avez rédigé.
Vous réaliserez également pour chaque programme ou groupes d'exercices un MakeFile associé, permettant de faire le build de l'exécutable ainsi que le clean à posteriori (cf transparents du cours n°203 à 207).
Vous soignerez également vos tests.

A. Nombres aléatoires (cf cours pseudo-aléatoire, Tr 218 à 222)

1. Librairie

Écrire une série de fonctions, qui génèrent et retournent des nombres aléatoires selon les contraintes suivantes :

- un nombre entier aléatoire selon la plage maximum du générateur aléatoire.
- un nombre entier aléatoire compris entre 0 et une valeur "seuil haut" passée en paramètre.
- un nombre entier aléatoire entier compris entre deux bornes "seuil bas" et une valeur "seuil haut" passées en paramètres.
- un nombre réel aléatoire compris entre 0 et 1.
- un nombre réel aléatoire à deux décimales compris entre deux bornes "seuil bas" et une valeur "seuil haut".

Faire une librairie `util_rand.c` et `util_rand.h` avec ces fonctions.

2. Dés

Écrire un programme affichant un menu proposant de jouer avec un, deux, trois ou quatre dés.
Selon le choix fait, le programme lance les dés tous en même temps. Les dés identiques sont systématiquement relancés.
On calcule alors le cumul des points des différents dés.
L'utilisateur gagne si le total est supérieur aux deux tiers du maximum qu'on peut obtenir avec les dés lancés (avec deux dés cela fait 8 : (12/3)*2). Le programme indique combien il manque pour gagner ou combien il y a en plus.
On utilisera explicitement la librairie écrite au A.1 pour gérer les nombres aléatoires.
Vous manipulerez toutes les variables via des pointeurs…
Vous créerez les fonctions/procédures nécessaires.
Exemples d'interaction :

```
Avec combien de dés voulez vous jouer ?
Tapez 1, 2, 3 ou 4 ? 3
Vous avez choisi 3 dé(s)
Dé 0 : 3
Dé 1 : 4
Dé 2 : 4
Dés identiques relancés...
Dé 0 : 3
Dé 1 : 6
Dé 2 : 1
La somme des 3 dés lancés est de 10, le seuil était de 12
score inférieur de 2 au seuil
Désolé, vous avez perdu !
```

3. Test statistique de rand

Écrire une fonction qui tire aléatoirement 100000 fois une valeur comprise entre 0 et 5, compte les occurrences de chaque résultat et en affiche le pourcentage. Testez ensuite cette fonction depuis un main.
On utilisera explicitement la librairie écrite au A.1 pour gérer les nombres aléatoires.

B. Pointeurs

1. StackOverflow…

Écrivez un programme générant une erreur de type StackOverflow.

2. Somme des éléments d'un tableau.

Écrivez une fonction `sumArray` qui calcule la somme des éléments d'un tableau d'entiers remplis avec des nombres aléatoires. La fonction doit prendre en paramètre un pointeur vers le début du tableau, sa taille, et un paramètre `computed_sum` permettant de renvoyer la somme calculée. En cas d'erreur, par exemple si le pointeur sur le tableau est NULL, la fonction `sumArray` retournera le code d'erreur -1, si tout se passe bien elle renverra 0.

Testez 2 fois votre fonction depuis un `main`. La première fois en passant un tableau créé statiquement, la deuxième fois avec un tableaux alloué dynamiquement.
Vous utiliserez l'arithmétique des pointeurs pour en parcourir les cases, c'est-à-dire que vous n'avez pas le droit d'utiliser l'écriture utilisant les crochets pour accéder aux éléments du tableau, ni de variable d'indice.

3. Recherche d'un élément donné dans un tableau.

Écrivez une fonction `searchArray` qui recherche un élément donné dans un tableau d'entiers et permet de trouver, s'il est présent, l'emplacement de l'élément recherché dans le tableau. Si l'élément n'est pas présent ou si le tableau n'existe pas, la fonction retournera le code d'erreur -1, si tout se passe bien elle renverra 0.
A vous de définir quel doit être l'en-tête de la fonction et d'en écrire le code.

Reprenez le `main` de l'exercice précédent pour tester votre fonction. La première fois en passant un tableau créé statiquement, la deuxième fois avec un tableaux alloué dynamiquement.

4. Pointeur de fonction, compréhension de code (inspiré de Lambert)

(a) Étudiez le code suivant, en premier lieu sans l'exécuter et dîtes ce qu'il fait :

```
1. #include <stdio.h>
2. #include <stdlib.h>

3. int fois_deux(int i){
4.       return i*2;
5. }

6. void appliquerATableau(int f(int), int* t, int n){
```

```
7.       int i;
8.       for (i=0; i<n; i++)
9.             t[i]=f(t[i]);
10. }

11. int main(void){
12.      int tab[]={1,2,3,4};
13.      int (* f)(int) = &fois_deux;
14.      appliquerATableau(*f,tab,4);
15.      int i;
16.      for (i=0;i<4;i++) printf("%d ",tab[i]);
17.            printf("\n");
18.      return EXIT_SUCCESS;
19. }
```

(b) Écrivez une version de `appliquerATableau` que vous nommerez `appliquerATableauPurPointeurs` qui utilise seulement l'arithmétique des pointeurs pour parcourir le tableau.

(c) Écrivez une fonction entière f1 qui prend deux paramètres entier et qui retourne la somme des deux et utilisez la procédure `appliquerATableau` pour invoquer cette fonction f1 sur les éléments du tableau tab avec la valeur de votre choix en deuxième paramètre (Par exemple incrémentez tous les éléments du tableau de 10).

(d) Écrivez une série de fonction prenant deux paramètres entiers et qui retourne également un résultat entier, créez ensuite un tableau de pointeurs sur ces fonctions (type de retour entier et deux paramètre entiers), remplissez ce tableau avec des pointeurs sur les fonctions précédemment écrites et dans une boucle invoquez `appliquerATableau` en utilisant toutes les fonctions de votre tableau en affichant les états intermédiaires du tableau pour bien vérifier le bon déroulement des traitements.

5. Pointeur de fonction et tri de tableaux (inspiré de Lambert)

Soient les deux algorithmes suivants qui effectuent le tri des éléments d'un tableau d'entiers dans l'ordre croissant (pour le premier) ou décroissant (pour le deuxième). Ces deux algorithmes sont identiques, à l'exception de l'opérateur de comparaison. Afin d'éviter de dupliquer du code inutilement nous allons utiliser des pointeurs de fonction.

```
void tri_croissant(int* t, int n){
    int i,i_min,j,tmp;
    for (i=0;i<n-1;i++){
        i_min=i;
        for (j=i+1;j<n;j++){
            if (t[j]<t[i_min]) i_min=j;
        }
        if (i_min!=i){
                tmp=t[i];
                t[i]=t[i_min];
                t[i_min]=tmp;
        }
    }
}
void tri_decroissant(int* t, int n){
    int i,i_max,j,tmp;
    for (i=0;i<n-1;i++){
        i_max=i;
        for (j=i+1;j<n;j++){
            if (t[j]>t[i_max]) i_max=j;
        }
        if (i_max!=i){
                tmp=t[i];
                t[i]=t[i_max];
                t[i_max]=tmp;
        }
```

```
    }
}
```

(a) Écrire une fonction `superieur(int a, int b)` qui renvoie 1 si a est supérieur à b, 0 s'ils sont égaux et -1 sinon.
(b) Écrire une fonction `inferieur(int a, int b)` qui renvoie 1 si a est inférieur à b, 0 s'ils sont égaux et -1 sinon.
(c) Écrire une procédure tri qui trie un tableau t de taille n selon une fonction compare prise en paramètre.
(d) Écrire une procédure de main qui teste la procédure tri précédente dans les deux cas : tri croissant et tri décroissant.

C. Récursivité

1. Récursivité « classique » et terminale du calcul du produit des n premiers entiers

Écrivez, dans deux fonctions différentes, une en récursivité classique et une en récursivité terminale (c'est-à-dire utilisant un accumulateur) le calcul du produit des n premiers entiers, n étant passé en paramètre.
Testez ensuite ces deux fonctions dans un main.

2. Récursivité « classique » et terminale du calcul de la puissance d'un nombre n à la puissance p.

Écrivez, dans deux fonctions différentes, une en récursivité classique et une en récursivité terminale (c'est-à-dire utilisant un accumulateur) le calcul de la puissance d'un nombre n à la puissance p, ces deux valeurs étant données en paramètres.
Testez ensuite ces deux fonctions dans le même main que précédemment.