Programmation en C et structures de données

guillaume.revy@univ-perp.fr

CT, Jeudi 6/01/2022 de 14h à 16h

Aucun document n'est autorisé.
Les calculatrices, ordinateurs, et téléphones portables sont interdits.

Exercice 1. Importance de l'indentation, malgré tout

$0.75 + 1.25 = 2$ *pts*

Contrairement à certains autres langages, l'indendation d'un programme C n'a pas d'impact sur le résultat de son exécution. Mais elle a une importance pour sa lisibilité.

► **1.** Que fait le programme ci-dessous ?

```
int main(void){float ct,cc,note;scanf("%f",&ct);scanf("%f",&cc);note=.5f*cc+.5f*ct;printf
    ("Resultat=%f\n",note);if(note<10){printf("... non validee\n");}else{printf("...
    validee\n");}return 0;}
```

► **2.** Écrire un programme qui lit une valeur entière au clavier, puis qui indique si elle est positive, négative, ou nulle (en distinguant les 3 cas).

Exercice 2. Deux boucles à un point

$1 + 1 = 2$ *pts*

► **1.** Écrire un programme qui lit une valeur entière $n \geq 1$, puis qui affiche la suite de caractères suivante, en utilisant une boucle :

$$[1234321], \quad \text{pour } n = 4 \text{ par exemple.}$$

► **2.** Modifier le programme pour afficher la suite de caractères suivante, en utilisant un ou plusieurs boucles :

$$[1][121][12321][1234321], \quad \text{pour } n = 4 \text{ par exemple.}$$

Exercice 3. Produit scalaire

$1 + 0.5 + 0.5 = 2$ *pts*

Soit deux tableaux U et V représentant des vecteurs de réels de taille n. Le produit scalaire $U \cdot V$ est défini de la manière suivante :

$$U \cdot V = \sum_{i=0}^{n-1} U_i \times V_i.$$

► **1.** Écrire une fonction qui prend en paramètre deux vecteurs U et V de taille $n = 20$, puis qui calcule et renvoie la valeur du produit scalaire $U \cdot V$.

► **2.** Illustrer l'utilisation de cette fonction dans un programme principal.

► **3.** Donner le prototype de la fonction qui permet de faire le même traitement sur un tableau de taille n.

Exercice 4. À l'aide !

$1 + 1 = 2$ *pts*

Un étudiant vous envoie le mail suivant :
"*Je ne comprends pas, mon programme C compile sans erreur avec gcc-9.3.0, mais il plante (Segmentation fault : 11). Il est censé créer un nombre complexe, et l'afficher. Mais je vous assure, il n'y a pas d'erreur !*".

Pour l'aider, vous lui demandez de vous envoyer son programme (ci-dessous).

```c
struct complexe
{
  float I, R;
};

void afficher(struct complexe);

struct complexe*
creer(float I, float R)
{
  struct complexe C = {I, R};
  return &C;
}

int
main(void)
{
  struct complexe *C = NULL;
  C = creer(1, 2);
  afficher(*C);         // /!\ la fonction 'afficher' est bien definie quelque part /!\

  return 0;
}
```

► **1.** Proposer une explication, en vous aidant éventuellement d'un dessin.

► **2.** Corriger le programme, sans modifier le prototype des fonctions.

Exercice 5. Au secours ! $1 + 1 = 2$ *pts*

Vous venez de recevoir un autre mail :
"*Je ne comprends toujours pas, mon programme C compile encore sans erreur avec gcc-9.3.0, mais il plante (Segmentation fault : 11). Il est censé allouer un tableau, le remplir de* 0*, et l'afficher. Mais je vous assure, il n'y a pas d'erreur !*".

Pour l'aider, vous lui demandez de vous envoyer son programme (ci-dessous).

```c
void
allouer(int *T, int n)
{
  int i;
  T = (int *)malloc(n*sizeof(int));
  for(i = 0; i < n; i++)
    T[i] = 0;
}

int
main(void)
{
  int *T = NULL, i;
  allouer(T, 17);
  for(i = 0; i < 17; i++)
    printf("%d ", T[i]);
  printf("\n");
  free(T);

  return 0;
}
```

► **1.** Proposer une explication, en vous aidant éventuellement d'un dessin.

► **2.** Corriger le programme, sans modifier le prototype des fonctions.

Exercice 6. Au secours, vraiment ! 2 *pts*

Vous venez encore de recevoir un autre mail :
"*Je ne comprends plus rien, mon programme C compile sans erreur avec gcc-9.3.0 (ça c'est normal), mais surtout il semble qu'il ne plante pas. Pourquoi ?*".

Pour l'aider, vous lui demandez de vous envoyer son programme (ci-dessous).

```c
struct tableau
{
  int *T, n;
};
```

```c
void
allouer(struct tableau *S, int n)
{
  int i;
  S->T = (int *)malloc(n*sizeof(int));
  S->n = n;
  for(i = 0; i < n; i++)
    S->T[i] = 0;
}

int
main(void)
{
  int i;
  struct tableau S;
  allouer(&S, 17);
  for(i = 0; i < S.n; i++)
    printf("%d ", S.T[i]);
  printf("\n");
  free(S.T);

  return 0;
}
```

► **1.** L'étudiant a-t-il raison ? Si oui, proposer une explication. Si non, corriger le programme.

Exercice 7. Tri à bulle sur une liste doublement chaînée $0.5 + 0.5 + 1 + 1 + 1 + 1 = 5$ *pts*

Soit L une liste doublement chainée d'entiers, composée d'un ensemble de maillons.

► **1.** Rappeler la structure utilisée pour manipuler une liste doublement chaînée d'entiers.

On souhaite trier cette liste de manière croissante, en utilisant le tri à bulle. Pour rappel, le tri à bulle fonctionne de la manière suivante sur un tableau T de taille n.

```
    pour i de 0 a n-2 faire
        pour j de 0 a n-2-i faire
            si T[j] > T[j+1] alors
               echanger T[j] et T[j+1]
            fsi
        fpour
    fpour
```

► **2.** Écrire une fonction `longueur` qui renvoie la longueur d'une liste L passée en paramètre.

► **3.** Écrire une fonction `element` qui renvoie un pointeur sur le i-ème élément d'une liste passée en paramètre, ou `NULL` si cet élément n'existe pas.

► **4.** Écrire une fonction `echange` qui prend en paramètre une liste et un indice i, qui échange le i-ème élément avec le suivant dans la liste (si cela est possible), et renvoie un entier indiquant si l'échange a eu lieu. *Cette fonction ne devra pas simplement échanger les valeurs des maillons, mais les maillons eux-mêmes.*

► **5.** En utilisant les fonctions `longueur`, `element`, et `echange`, écrire une fonction `tri_bulle_v1` qui trie une liste passée en paramètre, et renvoie la liste triée.

► **6.** Enfin sans utiliser les fonctions `longueur` et `element`, écrire une fonction `tri_bulle_v2` qui trie une liste passée en paramètre, et renvoie la liste triée.

Exercice 8. Plus grand sous-tableau commun $2 + 1 = 3$ *pts*

Soient A et B deux tableaux d'entiers $\in \{0, 1\}$ de taille n et p, respectivement (avec éventuellement $n \neq p$). On souhaite déterminer la longueur du plus grand sous-tableau commun. Pour cela, on note $\ell_{i,j}$ la longueur d'un plus grand sous-tableau commun aux tableaux $[A_0 \cdots A_{i-1}]$ et $[B_0 \cdots B_{j-1}]$, défini de la manière suivante :

$$\forall\ (i,j) \in [0,n] \times [0,p],\ \ \ell_{i,j} = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0, \text{ ou } A_{i-1} \neq B_{j-1}, \\ 1 + \ell_{i-1,j-1} & \text{sinon.} \end{cases}$$

La longueur du plus grand sous-tableau commun à A et B est alors le maximum des $\ell_{i,j}$.

Dans cet exercice, nous n'utiliserons que des tableaux alloués dynamiquement.

► **1.** Écrire une fonction qui prend en paramètre les tableaux A et B, puis qui détermine et retourne la longueur du plus grand sous-tableau commun à A et B.

► **2.** Illlustrer l'utilisation de cette fonction dans un programme principal.