Rattrapage de Programmation C

ING1 – GM

16 février 2022

— *Durée : 2 heures*

— *Vous devez rédiger votre copie à l'aide d'un **stylo à encre** exclusivement.*

— *Toutes vos affaires (sacs, vestes, trousse, etc.) doivent être placées à l'avant de la salle.*

— *Aucun document n'est autorisé.*

— *Aucune machine électronique ne doit se trouver sur vous ou à proximité, même éteinte.*

— *Aucun déplacement n'est autorisé.*

— ***Aucune question au professeur n'est autorisée**. Si vous pensez avoir détecté une erreur, continuez en expliquant les hypothèses que vous faites.*

— *Aucun échange, de quelque nature que ce soit, n'est possible.*

— *Le barème est donné à titre indicatif.*

Exercice 1 (4.5pts)

1. Donner le résultat d'exécution du programme C suivant :

```c
#include <stdio.h>
int main(int argc, char** argv){
    int a;
    int b;
    int c;
    int *p1;
    int *p2;
    a = 10;
    b = 20;
    c = 30;
    p1 = &a;
    p2 = &c;
    *p1 = (*p2) + 10;
    printf("\n a = %d c = %d", a, c);
    p2 = &b;
    c = (*p2) - 10;
    printf("\n b = %d c = %d", b, c);
    b = 10;
```

```
19      a = 40;
20      *p1 = (*p1) / (*p2);
21      *p2 = *p1;
22      printf("\n *p1 = %d *p2 = %d", *p1, *p2);
23      return (0);
24  }
```

2. Soit *p* un pointeur qui *pointe* sur un élément d'un tableau `tab` d'entiers :

```
1  int tab[DIM] = {7, 5, 8, 0, 6, 2, 1, 9, 4, 3};
2  int *p;
3  p = tab + 4;
```

Quelles valeurs ou adresses fournissent les expressions suivantes :

(a) *p+2
(b) *(p+2)
(c) (p+4)-tab
(d) &tab[5] - p
(e) *(p+*(p+5)-tab[5]);
(f) &(tab[tab[tab[3]]]);

3. Le programme ci-dessous comporte des erreurs. Indiquer les erreurs et justifier pourquoi?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define DIM 10
4  void main(void) {
5      int i;
6      int n;
7      int *p;
8      int t1[DIM];
9      int t2[DIM];
10     int t3[DIM];
11     int t4;
12     n = DIM;
13     t4 = t2;
14     for(i = 0; i < n; i++){
15         t1 + i = i * 2;
16         *(p + i) = i;
17     }
18     t3 = t2;
19 }
```

Exercice 2 (2.75 pts)

Donner le résultat d'exécution des programmes suivant :

```c
#include <stdio.h>
int tutu(int i, int *j){
    i += 5;
    *j += 6;
    return (i + (*j));
}
void tata(int i, int *j ){
    i = tutu(i, j);
    *j = tutu(i, j);
}
void toto(int *i , int j){
    tata(*i, &j);
    *i = j;
}
void main(void){
    int i;
    int j;
    int k;
    i = 4;
    j = 2;
    k = tutu(i, &j);
    printf("\n i = %d j = %d k = %d",
            i , j, k);
    i = 4;
    j = 2;
    toto(&i , j) ;
    printf("\n i = %d j = %d", i , j);
}
```

```c
#include <stdio.h>


int tata(int n) {
  if (n == 0) {
    return 1;
  } else {
    return (n * tata(n - 1));
  }
}


int toto(int n) {
  if (n == 0) {
    return 0;
  } else {
    return tata(n) + toto(n - 1);
  }
}


void main(void) {
  int n;
  n = 4;
  printf("tata(%d) = %d\n", n, tata(n));
  printf("toto(%d) = %d\n", n, toto(n));
}
```

Exercice 3 (2 pts)

```c
#include <stdio.h>
#define DIM 10
typedef int TElement;
```

```c
int estPalain(int n, TElement *tab){
    TElement *p1;
    TElement *p2;
    p1 = tab;
    p2 = tab + n - 1;
    while((p1 < p2) && (*p1 == *p2)){
        p1++;
        p2--;
    }
    return (p1 >= p2);
}
void main(void){
    TElement tab[DIM]={5,2,4,0,4,2,5};
    printf("\n est Palindrome =%d ", estPalain(7, tab));
}
```

1. **Illustrer à chaque itération**, l'exécution de la fonction `estPalain`.
2. Réécrire en C, la fonction `estPalain` ci-dessus, en remplaçant la boucle while par une boucle for équivalente.
3. Réécrire en C, la fonction `estPalain`, en utilisant le formalisme tableau (on utilise des indices entiers pour parcourir le tableau au lieu des pointeurs).

Exercice 4 (2.5 pts)

```c
#include <stdio.h>
#define DIM 20
typedef int TElement;
void itTranf(int *n, TElement elt, TElement *tab){
    int i;
    int j;
    j = 0;
    for(i = 0; i < (*n); i++) {
        if(tab[i] != elt) {
            tab[j] = tab[i];
            j++;
        }
    }
    *n = j;
}
```

```
16 void affTab(int n, TElement *tab){
17     int i;
18     printf("\nTab[%d] = [", n);
19     for(i=0 ; i<n ; i++) {
20         printf("%d ", tab[i]);
21     }
22     printf("]\n");
23 }
24 void main(void){
25     int n;
26     TElement elt;
27     TElement tab[DIM]={5, 3, 7, 2, 10, 7, 9, 7, 13, 7};
28     n = 10;
29     elt = 7;
30     itTranf(&n, elt, tab);
31     affTab(n, tab);
32 }
```

1. **Illustrer pas à pas, les changements** du tableau effectués par la procédure `itTranf`.
2. Réécrire en C, les procédures `itTranf` et `affTab` en utilisant le formalisme pointeur (**aucun indice entier** ne doit être utilisé pour manipuler le tableau).

Exercice 5 (3 pts)

1. Écrire en C, une fonction nommée **nb0ccurrenceF** qui détermine le nombre d'occurrences d'un élément donné dans un tableau d'entiers quelconque composé de n éléments.
2. Réécrire en C, la fonction précédente sous forme de procédure qu'on nomme **nb0ccurrenceP**.
3. Proposer un programme principal qui effectue les tâches suivantes :
   - Déclarer et initialiser un tableau statique.
   - Déclarer et initialiser un élément.
   - Faire appel à la fonction `nb0ccurrenceF` pour afficher le nombre d'occurrences de l'élément.
   - Faire appel à la procédure `nb0ccurrenceP` pour afficher le nombre d'occurrences de l'élément.

Exercice 6 (5.25 pts)

On souhaite manipuler les données relatives à des points du plan et à des lignes brisées (ouvertes) définies comme des suites de points du plan.

*Dans ce qui suit, on utilise le mot **algorithme** pour désigner soit une **fonction** soit une **procédure**.*
Les structures et les algorithmes ci-dessous doivent être écrites en C.
Un point P du plan est défini par un couple (x, y) où x et y sont des nombres réels représentant respectivement l'abscisse et l'ordonnée du point P.

1. Écrire une structure de donnée appelée **`Point`** permettant de grouper l'ensemble des informations relatives à un point (l'abscisse et l'ordonnée).
2. Écrire les primitives de bases qui sont liées à la structure `Point` :
   — `float abscisse (Point p);` // *retourne l'abscisse d'un point*
   — `float ordonnee (Point p);` // *retourne l'ordonnée d'un point*
3. Écrire une structure de donnée appelée **`Ligne`** permettant de grouper d'une manière contiguë l'ensemble des informations relatives à une ligne brisée du plan dont on suppose que le nombre de points ne dépasse pas 100. Pour simplifier les algorithmes, on inclura dans la structure le nombre de points qui compose la ligne.
4. Écrire les primitives de bases qui sont liées à la structure `Ligne` :
   — `int nbPoint(Ligne l);` // *retourne le nombre de points présent dans une ligne donnée*
   — `Point iEmePoint(int i, Ligne l);` // *retourne le iéme point présent dans une ligne non vide donnée et on suppose que i < nbPoint(l)*

On définit la distance entre deux points $P(x_1, y_1)$ et $Q(x_2, y_2)$ comme suit : $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

5. Écrire un algorithme qui détermine la distance entre deux points donnés P et Q (on utilise la fonction *sqrt* pour calculer la racine carrée d'un réel).

On considère maintenant deux lignes L_1 et L_2 données.

6. Écrire un algorithme qui construit un tableau D de valeurs réelles où chaque $D[i]$ représente la plus petite distance entre chaque point de la ligne L_1 et les autres points de la ligne L_2.
7. En déduire un algorithme qui détermine la distance minimale entre L_1 et L_2.