UNIVERSITÉ CÔTE D'AZUR

Université Côte d'Azur — PO2 SITE
Programmation C

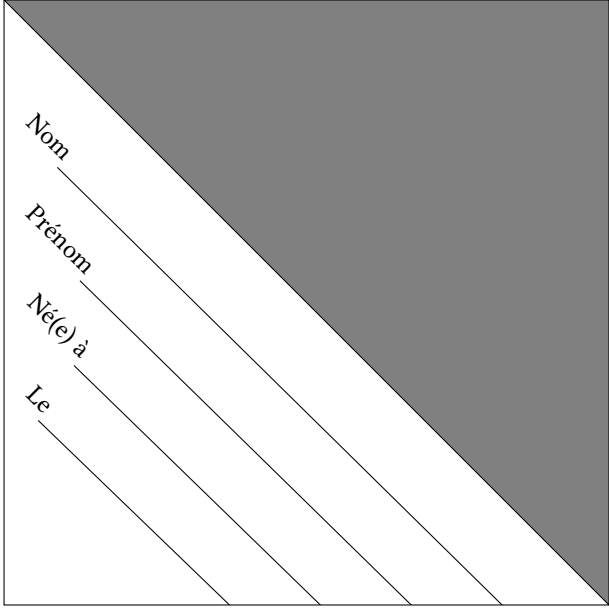
# Examen : Programmation C

## 11 janvier 2023

**Durée :** 1 heure
Tous documents autorisés. Il est interdit d'accéder à internet.

| Note |
| --- |
| |

Toutes les questions sont indépendantes.
Tous les codes devront être écrits en **Langage C**.
La notation est donnée à titre indicatif.
Nombre de pages : 6




Il est de votre responsabilité de rabattre le triangle grisé et de le cacheter au moyen de colle, agrafes ou papier adhésif. Si ne vous le faites pas, vous acceptez implicitement que votre copie ne soit pas anonyme.

## Exercices divers (7 points)

1. Écrire une fonction `nombre_chiffres` qui prend en argument une chaîne de caractères et renvoie le nombre de caractères correspondant à des chiffres. Par exemple `nombre_chiffres("abc 45+")` renverra 2.

```c
int nombre_chiffres(char * chaine) {
  int i = 0;
  int n = 0;
  for (i=0; chaine[i]!='\0'; i++) {
    if ('0' <= chaine[i] && chaine[i] <= '9') {
      n++;
    }
  }
  return n;
}
```

2. Écrire une procèdure `fusion` qui prend en argument un entier `n` et trois tableaux, `t1`, `t2` et `t3` de taille `n` et qui initialise `t3` en affectant chaque `t3[i]` (avec `i<n`) à la plus grande des valeurs entre `t1[i]` et `t2[i]`. En prenant, par exemple, `n=3`, `t1` = | 12 | 5 | 13 | et `t2` = | 20 | 2 | 13 |, alors `t3` devra être initialisé à | 20 | 5 | 13 |.

```c
void fusion(int n, int t1[], int t2[], int t3[]) {
  int i;
  for (i=0; i<n; i++) {
    if (t1[i]>t2[i]) {
      t3[i] = t1[i];
  } else {
    t3[i] = t2[i];
  }
}
```

3. Que fait le code suivant ? On ne vous demande pas de paraphraser. Vous décrirez sa fonction et vous l'illustrerez par un exemple.

```
int mystere(char * c) {
  char * oo = c ;
  while ( *(c++) != '\0') ;
  return c - oo - 1 ;
}
```

```
/* Il calcul la longueur de la chaîne c
   cc est l'adresse initiale
   c termine sur '\0' à l'adresse finale
   mystère("olivier") renvoie l'entier 7 */
```

4. Écrire une fonction `echange`, qui prend en argument deux pointeurs vers des flottants et qui échange leurs valeurs. Par exemple, si `a` pointe vers la valeur `3.5` et `b` pointe vers la valeur `2.0`, l'appel de `echange(a,b)` fera pointer `a` vers `2.0` et `b` vers `3.5`.

```
void echange(float * a, float * b) {
  float tmp = *a;
  *a = *b;
  *b = tmp;
}
```

5. Utiliser la fonction `echange` pour compléter le code suivant. À la fin, les valeurs de `x` et `y` doivent être permutées. Votre code ne doit pas dépasser une ligne.

```
float x = 2.0;
float y = 3.0;
/* ligne  à compléter pour que x vaille 3.0 et y 2.0 */
```

```
echange(&x, &y);
```

# Problème (13 points)

On cherche dans cet exercice à implémenter une file à partir de deux tableaux. Ce n'est pas forcément optimal, mais c'est un prétexte pour faire du C. En pratique, la file est séparée en deux tableaux : un premier auquel on ajoute les éléments et un second, duquel on les retire. Lorsque le second tableau est vide, on transfère le contenu du premier vers celui-ci.

Exemple. Prenons la file abstraite suivante : | → | 3 | 5 | 7 | 2 | 1 | → | ; elle sera impléméntée en C par les deux tableaux ci-dessous.

- `entree` : | 7 | 5 | 3 | | | | | | | | ← (notez comme l'ordre est inversée par rapport à la file)
- `sortie` : | 2 | 1 | | | | | | | | | →

Les cases vides, représentent des valeurs indéfinies. Pour connaître à chaque instant le nombre de cases effectivement remplies on ajoute deux variables correspondant à des indices.

— `indice_entree` represente la dernière case remplie du tableau `entree` (ici, la troisième case d'indice 2).

— `indice_sortie` represente la première case vide du tableau `sortie` (ici aussi, la troisième case d'indice 2).

Ainsi, si j'ajoute l'élément 10 (à la fin de la file) et que je supprime l'élément du début de la file (ici, 1), j'obtiens comme nouvelle file | → | 10 | 3 | 5 | 7 | 2 | → | La file sera alors codée par :

- `entree` : | 7 | 5 | 3 | 10 | | | | | | | ← (et `indice_entree` vaut maintenant 3)
- `sortie` : | 2 | | | | | | | | | | → (et `indice_sortie` vaut maintenant 1)

On vous donne la structure suivante définissant le type `file`.

```c
typedef struct {
  int entree[N] ;
  int indice_entree ;
  int sortie[N] ;
  int indice_sortie ;
} file ;
```

1. Définir la valeur `N` à 10 à l'aide d'une macro (en utilisant le préprocesseur).

```c
#define N 10
```

2. Que faut-il écrire comme code pour créer une nouvelle file sur la pile avec les indices correctement initialisés.

```c
file f;
f.indice_entree = -1;
f.indice_sortie = 0;
```

3. Écrire une fonction `est_vide` qui prend une file en paramètre et renvoie un booléean (c'est-à-dire un entier) pour indiquer si la file est vide.

```c
int est_vide(file f) {
  return f.indice_entree == -1 && f.indice_sortie == 0 ;
}
```

4. Écrire une fonction `ajout` qui prend une file en paramètre (et non un pointeur) et un entier `x` et renvoie une nouvelle file à laquelle on a ajouté l'entier `x`. Si la tableau `entree` est déjà complet, la fonction renverra `NULL`.

```
file ajout(file f, int x) {
  if (f.indice_entree < N-1) {
    f.indice_entree++;
    f.entree[f.indice_entree] = x ;
    return f;
  }
  return NULL;
}
```

5. Écrire une fonction `retire_naif` qui prend en argument un pointeur vers une file (on supposera le tableau `sortie` non vide) et renvoie l'élément en tête de file. Cette procédure modifiera la file accessible via le pointeur.

```
int retrait_naif(file * f) {
  f->indice_sortie--;
  return f->sortie[f->indice_sortie];
}
```

6. Évidement un problème se pose si l'on veut lire une valeur et que le tableau `sortie` est vide. Écrire la fonction `transfert` qui copie les éléments du tableau `entree` vers le tableau `sortie`. La file sera passée en argument sous forme de pointeur. La fonction renverra `-1` en cas d'erreur (si le tableau `sortie` était non vide) et `1` sinon.

- | 12 | 15 | 13 | 10 | | | | | | | : `entree`
- | | | | | | | | | | | : `sortie`

En partant de l'état précédent et en appelant la procédure `transfert`, on obtient l'état ci-dessous.

- | | | | | | | | | | | : `entree`
- | 10 | 13 | 15 | 12 | | | | | | | : `sortie`

```
int transfert(file * f) {
  int i;
  if (f->indice_sortie != 0) {
    return -1;
  }
  for (i=0; i <= f->indice_entree ; i++) {
    f->sortie[i] = f->entree[f->indice_entree-i];
  }
  f->indice_sortie = f->indice_entree+1;
  f->indice_entree = -1;
  return 1;
}
```

7. En utilisant la question précédente, réécrivez la fonction `retrait`, mais en commençant par transférer les données si la liste `sortie` est vide. La fonction prendra en argument un pointeur vers une file et renverra `-1` si la file est vide.

```
int retrait(file * f) {
  if (est_vide(*f)) {
    return -1;
  }
  if (f->indice_sortie == 0) {
    transfert(f);
  }
  f->indice_sortie--;
  return f->sortie[f->indice_sortie];
}
```

On modifie maintenant la structure `file` afin de pouvoir changer dynamiquement la taille des tableaux.

```
typedef struct {
  int n; /* taille des tableaux entree et sortie */
  int * entree ;
  int indice_entree ;
  int * sortie ;
  int indice_sorties ;
} file ;
```

8. Écrire une fonction `initialisation` qui prend en argument un entier $n$ et qui renvoie un pointeur vers une file correctement allouée sur le tas.

```
file * initialisation(int n) {
  file * f = malloc(sizeof(file));
  f->n = n;
  f->entree = malloc(sizeof(int) * n);
  f->sortie = malloc(sizeof(int) * n);
  f->indice_entree = -1;
  f->indice_sortie = 0;
  return f;
}
```

9. Écrire un fonction `liberer` qui prend en argument un pointeur vers une file et qui désalloue proprement la mémoire correspondante.

```
void liberer(file * f) {
  free(f->entree);
  free(f->sortie);
  free(f);
}
```

10. Réecrire la fonction `ajout` pour doubler la valeur de `n` si le tableau `entree` est complet.

```
void doubler(file * f) {
  file * ff = initialiser(ff, 2*f->n);
  int i;
  for (i=0; i < f->n; i++) {
    ff->entree[i] = f->entree[i];
    ff->sortie[i] = f->sortie[i];
  }
  free(f->entree);
  free(f->sortie);
  f->entree = ff->entree;
  f->sortie = ff->sortie;

}

void ajout(file * f, int x) {
  if (f->indice_entree >= f->n-1) {
    doubler(f);
  }

  f.indice_entree++;
  f->entree[f->indice_entree] = x ;

}
```