Programmation Avancée

Examen Terminal

(parfois appelé « partiel »)

lundi 3 mai 2021 — 9h–12h
Durée : 3h

Tout document papier est autorisé. L'usage d'outils informatiques ou électroniques n'est pas autorisé, sauf cas spécifiques (dans ces derniers cas, cet usage doit être restreint à la rédaction d'un rendu). Sauf indication contraire, toute question correctement traitée rapporte 1 point, pour un total de 40 points. Le barème sera ajusté *a posteriori*, de sorte qu'il ne sera probablement pas nécessaire d'avoir au moins 20 points pour valider.

Exercice 1 : QCM (8 points)

Pas besoin de justifier. Une bonne réponse rapporte 1 point, une mauvaise coûte 1/3 de point.

1. On a une variable `str` définie par `char *str = "Bonjour !"`. Quelle instruction permet d'afficher correctement le contenu de `str` ?

a) `printf("%d",str);` ?
b) `printf("%d",&str);` ?
c) `printf("%s",str);` ?
d) `printf("%s",&str);` ?

2. On a une variable `int i` dans laquelle on souhaite mettre un entier lu depuis le terminal. Quelle instruction permet de faire cela correctement ?

a) `scanf("%d",i);` ?
b) `scanf("%d",&i);` ?
c) `scanf("%s",i);` ?
d) `scanf("%s",&i);` ?

3. Étant donnés un type `typedef struct { int age; char *nom; } personne;` et une variable `personne p` remplie par des valeurs, comment afficher correctement l'âge associé à `p` ?

a) `printf("%d",p->age);` ?
b) `printf("%d",p.age);` ?
c) `printf("%s",p->age);` ?
d) `printf("%s",&p.age);` ?

4. Toujours avec le type `personne` ci-dessus, étant donné un pointeur `personne *ptr` correctement alloué, et tel que l'attribut `nom` associé à `ptr` pointe vers un bloc alloué (que l'on pourra supposer de taille suffisante), comment récupérer correctement un nom lu dans le terminal dans cet attribut `nom` ?

a) `scanf("%d",p->nom);` ?
b) `scanf("%d",&p.nom);` ?
c) `scanf("%s",p->nom);` ?
d) `scanf("%s",&p.nom);` ?

5. Par quelle valeur minimale faut-il remplacer `/*...*/` dans `char *str = malloc(/*...*/);` afin de pouvoir exécuter l'instruction `strcpy(str,"pomme");` sans problèmes de mémoire (rappel : `strcpy` copie la `C` -chaîne donnée par le deuxième argument dans le bloc pointé par le premier argument) ?

a) `4*sizeof(char)` ?
b) `5*sizeof(char)` ?
c) `6*sizeof(char)` ?
d) `7*sizeof(char)` ?

6. Par quelle valeur minimale faut-il remplacer `/*...*/` dans `int *tab = malloc(/*...*/);` afin que l'instruction `tab[5] = 3;` n'induise pas de problème de mémoire ?

a) `6*sizeof(int)` ?
b) `5*sizeof(char)` ?
c) `5*sizeof(int)` ?
d) `4*sizeof(char)` ?

7. À quoi est égal `0b00110100 << 2` ?

a) `0b00110111` ? b) `0b00011010` ?

c) `0b00001101` ? d) `0b11010000` ?

8. À quoi est égal `10^10`, toujours quand on l'écrit dans un programme `C` (`10` s'écrit aussi `0b1010` en `C`) ?

a) `10` ? b) `10000000000` ?

c) `0` ? d) `1410065408` ?

Exercice 2 : Horloges (8 points)

On définit une structure `horloge` par

```
typedef struct {
  int heures, minutes, secondes;
} horloge;
```

et qui contiendra une heure entre 00h 00min 00sec et 23h 59min 59sec. On demande ici d'écrire plusieurs fonctions pour manipuler cette structure.

1. Écrire une fonction `horloge horloge_creer(int heures,int minutes,int secondes)` qui crée une `horloge` à partir des valeurs données en argument. On pourra supposer que ces arguments décrivent une heure correcte.

2. Écrire une fonction `void horloge_afficher(horloge h)` qui affiche le contenu d'une `horloge` donnée en argument.

3. Écrire une fonction

`void horloge_regler(horloge *ph,int heures,int minutes,int secondes)`

qui change le contenu de l'horloge pointée par `ph` selon les autres arguments (que l'on pourra supposer comme décrivant une heure correcte).

4. (2 points) Écrire une fonction `void horloge_tic(horloge* ph)` qui ajoute une seconde à l'`horloge` donnée en argument. Il faudra s'assurer que l'heure donnée par `ph` reste correcte, en modifiant éventuellement les minutes et les heures de `ph`.

5. (3 points) Écrire une fonction

`void horloge_allouer_regler(horloge **pph,int heures,int minutes,int secondes)`

qui alloue une horloge à une adresse qui sera mise dans le pointeur pointé par `pph` et qui la réglera selon les valeurs des autres arguments. Par exemple,

```
horloge *ph = NULL;
horloge_allouer_regler(&ph,15,23,56);
```

allouera une horloge à l'adresse pointée par `ph` associée à l'heure 15h 23min 56sec.

Exercice 3 : Caractères et `C` -chaînes (8 points)

Dans cet exercice, on pourra supposer que tous les caractères manipulés sont des caractères de la table ASCII.

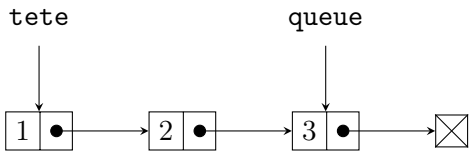
1. Écrire une fonction `int my_isupper(char c)` qui renvoie `1` si `c` est un caractère associé à une lettre majuscule de l'alphabet standard à 26 lettres, et `0` sinon.

2. Écrire une fonction `int compte_voyelles(char *str)` qui renvoie le nombre de voyelles (minuscules ou majuscules) dans la `C` -chaîne `str`. Rappel : en français, il y a 6 voyelles qui sont a, e, i, o, u et y.

3. (2 points) Écrire une fonction `int compte_ab(char *str)` qui renvoie le nombre de fois où la séquence `ab` apparaît dans la `C` -chaîne `str`. Par exemple, appeler la fonction `compte_ab` sur `"baababccaba"` renverra `3`.
4. (2 points) Écrire une fonction `void inverser(char *str)` qui inverse l'ordre d'apparition des lettres dans la `C` -chaîne `str`. La modification devra être faite dans l'argument lui-même. Par exemple, appeler `inverser` sur `str` défini par `char str[100] = "tortue"` modifiera `str` de sorte qu'il contienne la `C` -chaîne `"eutrot"`.
5. (2 points) Écrire une fonction `int est_palindrome(char *str)` qui renverra `1` dans le cas où la `C` -chaîne `str` est un palindrome (une chaîne de caractères qui est la même lue de gauche à droite ou de droite à gauche), et `0` sinon. Par exemple, appeler `est_palindrome` sur `"radar"` renverra `1`, tandis que l'appeler sur `"tortue"` renverra `0`.

Exercice 4 : Files (8 points)

Une file est une structure de données qui, comme les listes chaînées, permet de stocker des séquences d'éléments. Elle est de type « FIFO » (*First-In, First-Out*), c'est-à-dire que la première donnée à y être entrée sera la première à en sortir. Concrètement, on va implémenter une file contenant des entiers d'une façon similaire aux listes chaînées. Ce qui change, c'est que, en plus d'un pointeur vers la tête de la file, on aura un pointeur vers sa queue aussi. L'ajout d'un élément se fera toujours par la queue, et la suppression toujours par la tête. Par exemple, la file obtenue en ajoutant à une file vide les éléments 1, 2 et 3 (dans cet ordre) peut être représentée par le schéma :



On utilisera pour représenter une telle structure les types

```
typedef struct chainon {
  int valeur;
  struct chainon *suivant;
} chainon;
```

et

```
typedef struct {
  chainon *tete, *queue;
} file;
```

1. Écrire une fonction `void file_init(file *f)` qui initialise une file vide déjà allouée à l'adresse pointée par `f`.
2. Écrire une fonction `int file_devant(file *f,int v)` qui renvoie la valeur de l'élément en tête de la file `f` (on supposera que la file n'est pas vide).
3. (2 points) Écrire une fonction `void file_ajout(file *f,int v)` qui ajoute un élément à la file (pointée par) `f`, au niveau de la queue.
4. (2 points) Écrire une fonction `int file_suppr(file *f)` qui enlève un élément de la file `f`, au niveau de la tête (on supposera que la file n'est pas vide). On renverra la valeur de l'élément enlevé.
5. (0,5 points) Écrire une fonction `int file_pas_vide(file *f)` qui renvoie `1` si la file `f` n'est pas vide, et `0` sinon.
6. (1,5 points) Écrire une fonction `int file_taille(file *f)` qui renvoie le nombre d'éléments contenus dans la file `f`.

Exercice 5 : Tri fusion pour files (8 points)

On rappelle que le tri fusion est une méthode de tri consistant à diviser les données à trier en deux paquets, trier ces deux paquets récursivement, puis « fusionner » les deux paquets triés. Dans cet exercice, on implémente une telle méthode pour les files. La méthode devra produire une file triée où le plus petit élément sera positionné à la tête (par exemple, la liste [1,2,3] du précédent exercice est déjà triée). Il n'y aura pas besoin de manipuler directement les attributs `valeur`, `suivant`, `tete` et `queue` des structures vues précédemment pour faire cet exercice. À la place, il suffira d'appeler correctement les fonctions `file_*` codées précédemment.

1. (2 points) Écrire une fonction `file file_diviser(file *f)` qui divise une file `f` en deux. La première file sera alors `f`, de laquelle on aura enlevé la moitié (arrondi à l'entier inférieur si nécessaire) des éléments. La deuxième file sera constituée de ces éléments enlevés, et sera retournée par la fonction.
2. (3,5 points) Écrire une fonction `file file_fusion(file *l,file *r)` qui « fusionne » deux files triées en une seule, qui sera aussi triée et qui sera retournée par la fonction. Les files `l` et `r` seront vidées dans le processus.
3. (2,5 points) Écrire une fonction `file file_tri_fusion(file *f)` qui effectue l'algorithme récursif du tri fusion. Il ne faudra faire d'appel récursif que si la taille de `f` est ≥ 2. La file triée sera renvoyée par la fonction, et `f` sera vidée durant l'exécution de la fonction.