SCIENCES SORBONNE UNIVERSITÉ

Votre numéro d'anonymat : | | | |

Éléments de programmation 2– 1I002

Examen du 16 mai 2019
2h
Aucun document n'est autorisé.

Les calculatrices, baladeurs et autres appareils électroniques sont interdits. Les téléphones mobiles doivent être éteints et rangés dans les sacs.

Le barème sur 66 points (13 questions) n'a qu'une valeur indicative. Les questions sont indépendantes.

Tableaux ou comment bien mélanger

Question 1 (3 points)

Définir la fonction `int *creer_tab(int len)` qui renvoie un tableau de longueur `len` contenant les valeurs de la suite `0 ...len-1` (l'élément en position `i` doit avoir la valeur i).

Solution: Le zone mémoire allouée pour stocker le tableau renvoyé par la fonction doit être accessible une fois la fonction terminée, le tableau doit donc être alloué dynamiquement.

```
int *creer_tab(int len) {
  int *r= malloc(len*sizeof(int));
  int i;
  for(i=0; i<len; i++) r[i] = i;
  return r;
}
```

Question 2 (3 points)

Définir la fonction `void deplacer(int i, int t[])` qui place la valeur de `t[0]` en position `i`. Toutes les valeurs se trouvant entre les indices `1` et `i` sont décalées vers la gauche. On fait l'hypothèse que `i` est positif et strictement inférieur à la longueur du tableau.

Exemple : si le tableau `t` contient la suite `0 1 2 3 4 5`, alors, après appel de `deplace(3,t)`, le tableau `t` contient la suite `1 2 3 0 4 5`.

Solution:

```
void deplacer(int i, int t[]) {
  /* Hypothese: 0 <= i < taille du tableau t */
  int x = t[0];
  int j;
  for(j=0; j<i; j++) t[j] = t[j+1];
  t[i] = x;
```

```
}
```

Question 3 (6 points)

Définir la fonction `void coupe(int c, int t[], int tt[], int len)` qui range dans `tt` les valeurs de `t` de la manière suivante : les valeurs de `t` entre les indices `0` et `c` (exclus) sont en fin de `tt` et celles entre `c` (inclus) et la fin du tableau `t` sont au début de `tt`.

Exemple : si le tableau `t` contient la suite `0 1 2 3 4 5`, alors, après `coupe(3,t,tt,6)`, le tableau `tt` contiendra la suite
`3 4 5 0 1 2`.

Solution:

```
void coupe(int c, int t[], int tt[], int len) {
  /* Hypothese: 0 <= c < len */
  int i, ii;
  ii = 0;
  for(i=c; i<len; i++) { tt[ii] = t[i]; ii++; }
  for(i=0; i<c; i++) { tt[ii] = t[i]; ii++; }
}
```

Question 4 (6 points)

Définir la fonction `void intercale(int c, int t[], int tt[], int len)` qui range dans le tableau `tt` les valeurs du tableau `t` intercalées de la manière suivante : `t[c] t[0] t[c+1] t[1]`, etc. Autrement dit, on dispose de deux intervalles d'indices `[0, c-1]` et `[c, len-1]` ; on remplit le tableau `tt` en prenant alternativement une valeur de `t` dans chaque intervalle d'indices, en commençant par le deuxième. Lorsque l'on a épuisé un des intervalles, si l'autre n'a pas été parcouru entièrement, les valeurs de `t` correspondantes sont ajoutées à la fin de `tt`.

Exemples : si `t` contient la suite `1 2 3 4 5 6` et
si `c` vaut `3` alors `tt` contiendra la suite `4 1 5 2 6 3`
si `c` vaut `2` alors `tt` contiendra la suite `3 1 4 2 5 6`
si `c` vaut `4` alors `tt` contiendra la suite `5 1 6 2 3 4`

Solution:

```
void intercale(int c, int t[], int tt[], int len) {
  /* Hypothese: 0 <= c < len */
  int i1, i2;
  int ii;
  i1 = 0;
  i2 = c;
  ii = 0;
  while(ii<len) {
    if(i2 < len) { tt[ii] = t[i2]; i2++; ii++; }
    if(i1 < c) { tt[ii] = t[i1]; i1++; ii++; }
  }
}
```

Question 5 (3 points)

Définir la fonction `int randb(int i1, int i2)` qui renvoie un entier choisi aléatoirement en `i1` et `i2` (inlcus).

On fait l'hypothèse que `i1 < i2`.

Solution:

```
int randb(int i1, int i2) {
  /* Hypothese i1 < i2 */
  return i1+rand()%(i2-i1+1);
}
```

Question 6 (9 points)

Nous allons maintenant utiliser les fonctions précédentes pour calculer une permutation d'un tableau. L'algorithme s'inspire de la manière dont on bat un jeu de cartes. On suppose que le nombre de cartes est donné par `len`.

Voici comment l'on procède :

```
créer un tableau de longeur len contenant la suite 0... len-1
choisir un nombre d (entre 1 et len-1) (cf. randb)
glisser la première carte à la position d (cf. deplace)
puis répéter 42 fois:
  couper le jeu en (len/2)-1 (cf. coupe)
  intercaler toujours avec (len/2)-1 (cf. intercale)
```

Définir la fonction `int *battre(int len)` qui renvoie le tableau résultant de l'algorithme ci-dessus.

REMARQUE : Pour appliquer les fonctions `coupe` et `intercale`, penser à utiliser un tableau temporaire, sans générer de fuite mémoire.

Solution:

```
int *battre(int len) {
  int *t1 = creer_tab(len);
  int t2[len];
  int d = randb(1,len-1);
  deplacer(d,t1);
  d = (len/2)-1;
  int i;
  for(i=0; i<42; i++) {
    coupe(d, t1, t2, len);
    intercale(d, t2, t1, len);
  }
  return t1;
}
```

Il est possible d'utiliser une allocation dynamique pour `t2`, mais dans ce cas, il faut libérer la mémoire utilisée avant la fin de la fonction.

Structures et pointeurs

On se donne

```
struct s_carte {
  char *figure;
  char *couleur;
```

```
  struct s_carte *suivant;
};

typedef struct s_carte carte;
```

Avec une telle structure, on peut construire et manipuler des listes de cartes à jouer.

On supposera que l'on dispose de la constante et des fonctions suivantes :

- `NBCARTES` qui donne le nombre total de cartes du jeu. On suppose que `NBCARTES` est un nombre pair et que les cartes sont numérotées de 0 à `NBCARTES` $-$ 1
- `int hauteur_carte(carte c)` qui renvoie le rang de la carte `c`, selon la valeur de sa figure. Ainsi, cette fonction retournera 1 pour la plus petite des figures et le nombre de figures possibles pour la plus grande. Cette fonction permettra de comparer deux cartes entre elles.

Dans cet exercice nous supposons que les appels à `malloc` s'exécutent sans problème. Il n'est donc pas nécessaire de tester la valeur de retour.

Question 7 (3 points)

Définir la fonction `int carte_sup(carte c1, carte c2)` qui renvoie `1` si les cartes `c1` et `c2` sont de même couleur et si la hauteur de `c1` est strictement supérieure à celle de `c2` ; le résultat est `0` sinon.

Rappel : la fonction `strcmp` permet de comparer deux chaînes de caractères (type `char *`). Elle renvoie `0` si les deux chaînes sont égales.

Solution:

```
int carte_sup(carte c1, carte c2) {
  if (strcmp(c1.couleur,c2.couleur)== 0 && hauteur_carte(c1) >
     hauteur_carte(c2)) {
    return 1;
  }
  return 0;
}
```

Question 8 (6 points)

Définir la fonction `carte carte_min(carte *cs)` qui renvoie *une carte* de hauteur minimale parmi les cartes de la liste `cs`. On suppose que `cs` n'est pas une liste vide.

Solution: Si deux cartes de la liste sont de même hauteur et que cette hauteur est minimale, la fonction suivante renvoie la première carte rencontrée dans la liste. Pour qu'elle renvoie la dernière carte rencontrée, dans la condition du `if` il suffit de remplacer le < par <=.

```
carte carte_min(carte *cs) {
  /* Hypothese: cs != NULL */
  carte r = *cs;
  cs = cs->suivant;
  while(cs != NULL) {
    if (hauteur_carte(*cs) < hauteur_carte(r)) r = *cs;
    cs = cs-> suivant;
  }
  return r;
}
```

Question 9 (3 points)

Définir la fonction `carte *ajouter_carte(char *fig, char *coul, carte *cs)` qui crée et

ajoute en tête de la liste `cs` la carte dont le champ `figure` vaut `fig` et le champ `couleur` vaut `coul`. Il n'est pas nécessaire de réallouer ou dupliquer les chaines de caractères `fig` et `coul`.

Solution:

```
carte *ajouter_carte(char *fig, char *coul, carte *cs) {
  carte *r = malloc(sizeof(carte));
  r->figure = fig;
  r->couleur = coul;
  r->suivant = cs;
  return r;
}
```

Question 10 (6 points)

Définir la fonction `carte *liste_cartes_sup(carte c, carte *cs)` qui renvoie la listes des cartes contenues dans `cs` qui sont supérieures (au sens de la fonction `carte_sup`) à la carte `c`. Les cartes retournées seront dupliquées.

Solution: L'appel à la fonction `ajouter_carte` alloue la mémoire nécessaire pour chaque nouvelle carte et assure donc la duplication des cartes.

```
carte *liste_cartes_sup(carte c, carte *cs) {
  carte *r = NULL;
  while(cs != NULL) {
    if(carte_sup(*cs,c)) {
      r = ajouter_carte(cs->figure, cs->couleur,r);
    }
    cs = cs->suivant;
  }
  return r;
}
```

Battre et distribuer

On veut maintenant battre un jeu de cartes et le distribuer entre deux joueurs. L'idée est de

1. Obtenir une suite aléatoire des numéros de cartes avec la fonction `int *battre(int len)` du premier exercice
2. Créer une liste de `carte` à partir de cette suite
3. Répartir ces cartes entre deux joueurs

Pour créer une carte à partir d'un numéro, on suppose que l'on dispose des deux fonctions suivantes :

- `char *figure_of_num(int n)` qui, pour toute valeur de `n` comprise entre `0` et `NBCARTES-1`, renvoie une chaîne de caractères correspondant à un nom de la figure.
- `char *couleur_of_num(int n)` qui, pour toute valeur de `n` comprise entre `0` et `NBCARTES-1`, renvoie une chaîne de caractères correspondant à une couleur.

Ces fonctions garantissent que les `NBCARTES` couples (figure, couleur) obtenus sont différents.

Question 11 (6 points)

Définir la fonction `carte *battre_cartes(int len)` qui crée la liste des cartes d'un jeu de longueur `len` rangées de manière aléatoire. Autrement dit, qui crée la liste des cartes d'un jeu que l'on a «battu».

Indications : on sait créer une suite de numéros de cartes (nombres entre `0` et `len-1`) dans un ordre aléatoire avec la fonction `battre` de l'exercice 1. On sait associer à un numéro de carte sa figure et sa couleur avec les fonctions données `figures_of_num` et `couleur_of_num`. Avec la fonction `ajouter_carte` de l'exercice 2, on sait ajouter à une liste de cartes une carte créée avec ces données.

Solution: Le parcours du tableau `ns` peut très bien se faire en ordre croissant des indices. Les cartes seront alors dans l'ordre inverse dans la liste. Le tableau `ns` n'ayant d'utilité que pour cette fonction, il faut que la mémoire qu'il occupe soit libérée une fois la fonction terminée. Ceci n'est pas réalisé automatiquement puisque la mémoire a été allouée dynamiquement par la fonction `battre`. L'appel à la fonction `free` avant le `return` réalise cette libération de la mémoire.

```
carte *battre_cartes(int len) {
  carte * r = NULL;
  int *ns = battre(len);
  int i;
  for(i=len-1; i>-1; i--) {
    r = ajouter_carte(figure_of_num(ns[i]),couleur_of_num(ns[i]),r);
  }
  free(ns);
  return r;
```

Question 12 (3 points)

Définir une structure que l'on appellera `joueurs` et qui pemettra de représenter les cartes des deux joueurs d'une partie. Cette structure contient deux champs : le premier pour contenir la liste de cartes du premier joueur et le second pour contenir celle du second joueur.

Solution:

```
struct s_joueurs {
  carte *joueur1;
  carte *joueur2;
};

typedef struct s_joueurs joueurs;
```

Question 13 (9 points)

Définir la fonction `joueurs distribuer(int len)` qui renvoie une structure `joueurs` dont les champs contiennent les listes de cartes obtenues en «battant» le jeu de cartes puis en le distribuant selon le principe donné ci-dessous.

La distribution des cartes entre les deux joueurs se fera une par une alternativement : si la fonction `battre` renvoie la suite `n1 n2 n3 n4`, etc. alors le premier joueur aura les cartes de numéros `n1`, `n3`, etc. et le second, les cartes de numéros `n2`, `n4`, etc. On rappelle que l'on a supposé que `NBCARTES` a pour valeur un nombre pair. L'ordre des cartes dans la liste de chaque joueur n'a pas d'importance.

Une fois la distribution effectuée, les cartes du jeu ne sont présentes que dans la structure `joueurs` : il ne doit pas y avoir de fuite mémoire.

Solution:

```
joueurs distribuer(int len) {
  joueurs js;
  js.joueur1 = NULL;
  js.joueur2 = NULL;
```

```
  carte *cs = battre_cartes(len);
  carte *tmp;
  while(cs !=NULL) {
    js.joueur1 = ajouter_carte(cs->figure, cs->couleur, js.joueur1);
    tmp=cs; cs = cs->suivant; free(tmp);
    js.joueur2 = ajouter_carte(cs->figure, cs->couleur, js.joueur2);
    tmp=cs; cs = cs->suivant; free(tmp);
  }
  return js;
}
```

On peut aussi retirer au fur et à mesure les cartes du jeu (donc sans allocation / libération) :

```
   int i;
   for(i = 0; i < len; i++) {
      tmp = cs;
      cs = cs->suivant;
      if (i%2 == 0) {
         tmp->suivant = js.jeu1;
         js.jeu1 = tmp;
      }
      else {
         tmp->suivant = js.jeu2;
         js.jeu2 = tmp;
      }
   }
```