Université de Nice-Sophia Antipolis — POLYTECH
ELSE3-FISE — 2023–2024

Examen de Info C

Durée : 1h
Aucun document autorisé
Mobiles interdits

Notez que les affirmations (antécédents, conséquents, rôles, et invariants) dans vos codes C entreront pour partie dans la note finale.

▶ 1. Un liste linéaire, comme vue en TD, est définie dans `liste.h` comme suit :

```
typedef ... T;
typedef struct noeud {
  T elt;
  struct noeud *suivant;
} *Liste;

extern Liste initListe(void);
extern int longueur(const Liste);
extern T ième(const Liste, const int);
extern void insérer(Liste *, const int, const T);
extern void supprimer(Liste *, const int);
```

Écrivez les procédures `ajouterEnTête` et `supprimerEnQueue` qui, respectivement, ajoute un élément en tête d'une liste, et supprime le dernier élément d'une liste.

```
/*
 * Rôle : ajoute en tête de liste l'élément x
 */
void ajouterEnTête(Liste *l, const T x) {
  insérer(l, 1, x);
}

/*
 * Rôle : supprime le dernier élément de la liste
 */
void supprimerEnQueue(Liste *l) {
  supprimer(l, longueur(*l));
}
```

.....

▶ 2. Avec **typedef**, déclarez le type structuré `Étudiant` formé des champs `nom` (un pointeur sur **char**) et `score` (un **double** $\in [0.0\,;100.0]$).

```
typedef struct {
  char *nom;
  double score; // [ 0.0 ; 100.0 ]
} Étudiant;
```


.....

▶ 3. Un fichier binaire (pas de texte!), contient une suite de *trames*. Chaque trame représente un `Étudiant` et contient successivement, un entier n (de type **int** > 0), suivi de n caractères alphabétiques (pas de `'\0'`) pour le `nom` de l'étudiant, suivis d'un réel **double** qui représente son `score`. Écrivez la procédure `lireTrames` qui, à partir d'un fichier de trames, construit une liste (type `Liste`) d'`Étudiant` (`T`=`Étudiant`). Cette procédure possède deux paramètres, le nom du fichier et la liste à construire. Le fichier est correctement formé.

```
/*
 * Antécédent : f fichier de trames
 * Rôle :  construit une liste d'Étudiant
 *
 */
void lireTrames(const char *f, Liste *l) {
  FILE *fd;
  if ((fd=fopen(f, "r"))==NULL) {
    perror(f);
    exit(errno);
  }
  // le fichier de trames est ouvert en lecture
  *l = initListe();
  int n;
  while (fread(&n, sizeof(int), 1, fd)) {
    assert(n>0);
    // lire n caractères
    char nom[n+1] = {};
    fread(nom, 1, n, fd);
    // lire la note
    double score;
    fread(&score, sizeof(double), 1, fd);
    // construire l'étudiant et l'insérer
    Étudiant e = { strdup(nom), score };
    ajouterEnTête(l, e);
  }
  //
  fclose(fd);
}
```

.....

▶ 4. Écrivez la procédure `écrireTrames` qui prend en paramètre un nom de fichier, une liste d'`Étudiant` et un réel **double** n. À partir de la liste d'`Étudiant`, cette procédure construit un fichier formé des trames qui correspondent aux étudiants dont le score est supérieur ou égal à n.

```
/*
 * Antécédent : n score minimum
 * Rôle : écrit dans le fichier f les étudiants dont le score
 *        est supérieur ou égal à n
 *
 */
void écrireTrames(const char *f, const Liste l, const double n) {
  FILE *fd;
  if ((fd=fopen(f, "w"))==NULL) {
```

```
    perror(f);
    exit(errno);
  }
  //
  for (int i=1; i<=longueur(l); i++) {
    Étudiant e = ième(l, i);
    if (e.score>=n) {
      int lg = strlen(e.nom);
      fwrite(&lg, sizeof(int), 1, fd);
      fwrite(e.nom, 1, lg, fd);
      fwrite(&(e.score), sizeof(double), 1, fd);
    }
  }
  //
  fclose(fd);
}
```

.....

► 5. Écrivez le programme qui prend comme paramètres programmes : deux noms de fichier de trames, et un score minimal. À partir du premier fichier, le programme construit le second fichier formé des trames dont les scores sont supérieurs ou égaux au score minimal. Vous ferez les vérifications nécessaires.

```
int main(int argc, char *argv[]) {
  if (argc != 4) {
    fprintf(stderr, "Usage : trames fin fout n\n");
    exit(EXIT_FAILURE);
  }
  // le nombre de paramètres programme est valide
  Liste l;

  lireTrames(argv[1], &l);
  écrireTrames(argv[2], l, atof(argv[3]));

  return EXIT_SUCCESS;
}
```

.....

► 6. Dans une application munie d'une interface graphique programmée avec *libsx*, expliquez comment s'assurer de l'indépendance entre le *modèle* et la (ou les) *vue(s)*.

Pour s'assurer de l'indépendance, entre le *modèle* et la (ou les) *vue(s)*, il faut, d'une part, qu'il n'y ait aucune utilisation de *libsx* dans le *modèle* (et donc pas de **#include** `<libsx>`), et d'autre part, qu'il n'y ait aucun accès **direct** à la structure de données partagée depuis les *callbacks* (l'accès doit se faire uniquement par l'intermédiaire des fonctions fournies par le *modèle*).

.....