Université de Nice-Sophia Antipolis — POLYTECH
ELEC3 — 2020–2021

Examen de Langage C

Durée : 1h30
Aucun document autorisé
Mobiles interdits *Toutes les fonctions que vous écrirez doivent être clairement commentées avec des affirmations significatives (antécédents, conséquents, invariants). Vous prendrez soin de définir les bons paramètres et les bons types des données manipulées. Pensez à définir des fonctions auxiliaires si cela est nécessaire.*

Exercice 1

▶ 1. Donnez la déclaration du type `liste` pour représenter une liste simplement chaînée d'élément d'un type `T` quelconque, comme vue en TD.

```
typedef int T; // par exemple

typedef struct noeud {
  T elt;
  struct noeud *suivant;
} * liste;
```

▶ 2. À l'aide des fonctions de manipulation de la liste vues en TD (`longueur`, `ieme`, `inserer` et `supprimer`), écrivez la fonction `inverser` qui renvoie l'inversion d'une liste passée en paramètre. Par exemple, si la liste à inverser est `<-9 0 5 19>`, la fonction renvoie son inversion : `<19 5 0 -9>`. L'en-tête de cette fonction est le suivant :

```
liste inverser(liste l)
```

```
/*
 * Rôle : renvoie l'inversion de la liste l
 */
liste inverser(liste l) {
  int lg = longueur(l);
  for (int i=1; i<=lg; i++) {
    // insérer le dernier élément au rang i
    inserer(&l, i, ieme(l,lg));
    // puis le supprimer
    supprimer(&l, lg+1);
  }
  return l;
}
```

▶ 3. Que pensez-vous de l'efficacité votre fonction ? Donnez sa complexité en termes du nombre d'accès aux éléments d'une liste de taille n. Expliquez.

Cette version de la fonction `insérer` n'est pas très efficace dans la mesure où elle parcourt plusieurs fois la liste à inverser. Pour une liste de n éléments :

— le calcul de la longueur de la liste pour la variable `lg` fait n accès ;
— puis, la boucle principale fait n itérations. À chaque itération, les appels des fonctions `ième` et `supprimer` font respectivement n et $n+1$ accès. On a donc $n \times (n+n+1) = 2n^2 + n$ accès aux éléments de la liste ;
— enfin, dans cette même boucle, les appels de la fonction `inserer` produisent $\sum_{i=1}^{n} i = n^2 + 1/2n$ accès.

Donc, le nombre total d'accès aux éléments pour une liste de longueur n est :

$$n + 2n^2 + n + n^2 + 1/2n = 3n^2 + 5/2n$$

On dit que la complexité de la fonction `inverser` est $\mathcal{O}(n^2)$ (quadratique). Moralité, avant d'utiliser des fonctions toutes faites, il faut calculer l'efficacité de leur emploi !

▶ 4. Pour rendre plus efficace cette fonction d'inversion, on va directement manipuler les pointeurs qui lient les noeuds en eux. Donc, sans utiliser les fonctions `longueur`, `ieme`, `inserer` et `supprimer`, récrivez la fonction `inverser` pour effectuer l'inversion en un seul passage sur les éléments de la liste.

En agissant directement sur les liens entre les nœud, on peut inserver la liste en un seul passage. La compléxité de la fonction `inverser` donnée ci-dessous est donc $\mathcal{O}(n)$.

```
/*
 * Rôle : renvoie l'inversion de la liste l
 */
liste inverser(liste l) {
  struct noeud *p=NULL, *q;
  // parcourir la liste l
  while (l!=NULL) {
    q=l->suivant;
    l->suivant=p;
    p=l;
    l=q;
  }
  return p;
}
```

Exercice 2

▶ 5. Écrivez un programme qui prend en paramètre un nom de fichier. Ce nom correspond à un fichier d'entiers de type **int** (ce n'est donc pas un fichier de texte). Votre programme lit l'intégralité du fichier (éventuellement vide), et crée une liste (du type `liste` de l'exercice 1) qui devra contenir uniquement les entiers pairs contenus dans le fichier. Une fois la liste constituée, votre programme l'inverse à l'aide de la fonction `inverser` (de l'exercice 1) et l'affiche sur la sortie standard. Vous écrirez une procédure `afficherListe`. Votre programme devra faire toutes les vérifications nécessaires.

```
/*
 * Rôle : affiche la liste l sur la sortie standard
 */
void afficherListe(liste l) {
  while (l != listeVide) {
    printf("%d ", l->elt);
    l = l->suivant;
```

```
  }
  printf("\n");
}

int main(int argc, char *argv[])
{
  if (argc!=2) {
    fprintf(stderr, "Usage : %s file\n", argv[0]);
    exit(EXIT_FAILURE);
  }
  // le nombre de paramètres est correct
  // ouvrir le fichier d'entiers de nom argv[1]
  FILE *in;
  if ((in = fopen(argv[1], "r")) == NULL) {
    perror(argv[1]);
    exit(errno);
  }
  // le fichier d'entiers est ouvert en lecture
  liste l = listeVide;
  // créer la liste d'entiers pairs
  int x;
  while (fread(&x, sizeof(int), 1, in))
    if ((x&1)==0) inserer(&l, 1, x);
  // EOF => fermer le fichier
  fclose(in);
  // inverser la liste et l'afficher
  afficherListe(inverser(l));

  return EXIT_SUCCESS;
}
```