Examen d'Algorithmique - (2013/2014 - session 1)

Durée : 1h30 - Aucuns documents autorisés

Formation Ingénieurs IT2I - CNAM 1A

Ce sujet comporte 3 pages. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit en aucun cas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix faits pour l'implantation. Il est de plus indispensable de respecter les notations données dans l'énoncé.

1 Echauffement

On rappelle la structure de données définie pour représenter les listes d'entiers ainsi que les opérations de base `vide` et `cons`. Afin de pouvoir tester notre implantation, on programme également une fonction d'affichage `print` (version itérative).

```
#include<stdio.h>

typedef struct liste
{
  int x;
  struct liste *suivant;
} Sliste, *Liste;

Sliste *vide()
{ return (NULL); }

Sliste *cons(int v, Sliste *l)
{
  Sliste *m=(Sliste *)malloc(sizeof(Sliste));
  m->x=v;
  m->suivant=l;
  return m;
}

void print(Sliste *l)
{
  while (l!=NULL)
    {
      printf("%d ",l->x);
      l=l->suivant;
    }
  printf("\n");
}
```

Question 1 Ecrire une fonction `print2` qui affiche exactement la même chose que `print`, mais en l'implantant de manière *récursive*.

```
void print2(Sliste *l)
```

Question 2 Programmer une fonction `opp_test` qui vérifie que tous les nombres de la liste sont positifs. On procèdera de manière *itérative* (en utilisant une boucle `while`). Le type de retour de la fonction demandée est `int` et représente un booléen (`true` ou `false`).

```
int opp_test (Sliste *l)
```

Question 2bis Programmer de manière *récursive* une fonction `opp` qui calcule l'opposé de tous les nombres de la liste. Les transformations se font dans la liste elle-même (le type de retour de la fonction `opp` est `void`).

```
void opp (Sliste *l)
```

Question 3 Ecrire une fonction récursive `produit` qui calcule le produit de tous les éléments de la liste. On arrêtera le calcul dès que l'on rencontre un 0 au cours du parcours de la liste.

```
int produit(Sliste *l)
```

2 Représentation des entiers en précision infinie

2.1 Principe général et exemples

On va représenter des grands entiers sous la forme d'une liste. Pour des raisons de simplicité pour l'implantation de l'addition, on considère que les nombres sont représentés avec le mot de poids faible en premier (c'est le contraire de l'écriture habituelle des nombres). A des fins de débuggage, on suppose que chaque élément de la liste ne peut contenir que des éléments strictement inférieurs à 100.

Considérons par exemple le nombre 57 230 887 (cinquante sept millons deux cents trente mille huit cents quatre-vingt sept). Ce nombre sera représenté par la liste suivante : $87 \rightarrow 8 \rightarrow 23 \rightarrow 57 \rightarrow NULL$. En effet, on découpe le mot en morceaux dont la taille est inférieure à 100 et on place les 2 chiffres de poids faible dans la première position de la liste, puis les 2 chiffres suivants dans la deuxième position et ainsi de suite jusqu'à avoir représenté le nombre en entier.

Question 4 Expliquer comment le nombre suivant 4 138 679 (quatre millions cent trente huit mille ...) peut être représenté par une telle liste.

Question 4bis Si l'on rencontre la liste suivante $90 \rightarrow 1 \rightarrow 23 \rightarrow 7 \rightarrow NULL$, quel nombre représente-t-elle ? L'écrire à la fois en chiffres et en lettres.

2.2 Exemple pour l'addition

On va maintenant étudier comment additionner 2 nombres représentés par des listes. Il faudra additionner en commençant par les chiffres de poids faible vers les chiffres de poids fort en transportant bien sûr l'éventuelle retenue.

Question 5 On considère tout d'abord un exemple. Supposons que l'on veuille additionner $a = 59414$ et $b = 11556$. Il s'agira de trouver comme résultat 70970.

Décrire pas-à-pas les étapes du calcul. Quelles sont les représentations sous forme de listes de nombres a et b ? Comment les éléments des listes représentant a et b sont-ils additionnés entre eux ? Préciser l'évolution de la variable contenant la retenue à chaque étape. Le résultat obtenu est-il correct ?

2.3 Tous les cas possibles pour l'addition

Question 6 L'exemple précédent correspond au cas général de l'addition de 2 entiers représentés sous forme de listes. Quelles autres cas particuliers faut-il prévoir ? Présenter au moins 3 cas de figure différents de l'exemple précédent (taille des listes, débordement au niveau du mot de poids fort,etc.).

Question 7 En déduire une implantation de l'opération `addition_avec_retenue`. On considère toujours que les nombres sont représentés avec le poids faible d'abord (le contraire de la notation habituelle).

```
Sliste *addition_avec_retenue(Sliste *a, Sliste *b, int r)

Sliste *addition (Sliste *a, Sliste*b)
{ return (addition_avec_retenue(a,b,0)); }
```

Question 7bis Cette fonction d'addition vous permet-elle d'implanter l'opération `++` d'incrémentation de 1 sur les entiers en précision infinie ?

2.4 Autres opérations

Question 8 Ecrire une fonction `valeur` qui calcule la valeur d'un grand nombre (dans le cas où les mots de poids faible sont en premiers dans la liste). Attention : il y a bien sûr un risque probable de débordement de capacité. Cette fonction permettra uniquement de tester nos fonctions sur ce petit exemple.

```
int valeur(Sliste *l)
```

Question 9 Programmer une fonction qui inverse l'ordre des éléments dans une liste. On pourra utiliser une fonction intermédiaire `ajout_fin` qui ajoute un élément à la fin d'une liste.

```
Sliste* ajout_fin(Sliste *l, int u)

Sliste* reverse (Sliste *l)
```

Question 10 Supposons que nos listes représentent maintenant les nombres en plaçant les mots de poids fort en premier comme c'est le cas dans les mathématiques usuelles. Ainsi 45678 serait représenté par la liste $4 \rightarrow 56 \rightarrow 78 \rightarrow NULL$. Comment pourrait-on implanter directement l'opération d'addition entre 2 listes représentant des entiers ? Quelles précautions faudrait-il prendre ?