Systèmes Informatiques
TP 4: Les Pointeurs et l'Allocation Dynamique en C

A rendre: au plus tard le 19.10.2010 à 13h15

Exercice. Dans cet exercice vous devez écrire un programme (**liste-contacts.c** et **liste-contacts.h**) qui implémente une *liste linéaire chaînée* contenant les prénoms et numéros de téléphone de vos camarades ou copains. Chaque nœud de la liste contiendra donc trois éléments: **prenom**, **numero** et **suivant**. L'élément prenom contiendra un pointeur vers une chaîne de caractères contenant un prénom, l'élément numero contiendra un entier long représentant le numéro de téléphone et l'élément suivant contiendra un pointeur vers le prochain nœud de la liste (sa valeur sera NULL si la liste ne contient plus de noeuds). Les nœuds de la liste devront être insérés automatiquement en respectant l'ordre alphabétique des prénoms. Par exemple, si la liste contient les nœuds suivants:

Aldo	0214886557
Ana	0784562544
Ernesto	0791313255
Udo	0765543585
Wendy	0224565878

et l'on veut insérer:

Boris	0227984355

la nouvelle liste devra être:

Aldo	0214886557
Ana	0784562544
Boris	0227984355
Ernesto	0791313255
Udo	0765543585
Wendy	0224565878

La longueur du prénom ne sera pas limitée, c'est-à-dire que le programme allouera *dynamiquement* suffisamment (mais sans gaspiller) d'espace mémoire pour le stocker.
Le programme aura les options de menu suivantes:

1. Insérer contact
2. Effacer contact
3. Rechercher contact
4. Imprimer liste de contacts
5. Sortir

Lorsque l'on choisit l'option 1, un prénom et un numéro de téléphone sont demandés. Si l'utilisateur tape un prénom déjà existant, le numéro de téléphone de celui-ci est tout simplement remplacé par le nouveau numéro. Pour effacer un contact (option 2) il suffit d'écrire le prénom que l'on veut effacer de sa liste. Lorsque l'on choisit l'option 3, la première lettre du prénom recherché est demandée et le programme affiche tous les prénoms (+ les numéros de téléphone associés) commençant par cette lettre. Lorsque l'on choisit l'option 4, toute la liste de contacts est affichée sur l'écran. Finalement, lorsque l'on choisit l'option 5, tout d'abord la liste est effacée (afin de libérer de l'espace mémoire) puis le programme se termine.

Afin d'homogénéiser les programmes rendus, la déclaration de structure suivante devra être utilisée (à taper en dehors de la fonction main()):

```
struct node {
   char *prenom; // Pointeur vers une chaîne de caractères.
   long numtel;  // Numéro de téléphone.
   struct node *suivant;
};

typedef struct node NOEUD;
```

Cette déclaration ainsi que d'autres définitions telles que des macros, des prototypes, etc. devront figurer dans le fichier d'entête (**liste-contacts.h**)

Tips:

1. N'oubliez pas que pour créer un nœud il faut réserver suffisamment d'espace mémoire à l'aide de la fonction **malloc()**. Voici un example:

```
NOEUD *p;  // Variable pointeur (vers un noeud).
p = (NOEUD *) malloc(sizeof(NOEUD));  // Allocation d'espace mémoire.
if (p == NULL) printf("Il n'y a plus assez de mémoire!\n");
```

2. Pour effacer un nœud il faut toujours libérer l'espace mémoire qu'il occupe à l'aide de la fonction **free()**. Attention, dans cet exercice, il faut libérer d'abord l'espace mémoire occupé par la chaîne de caractères contenant le prénom. Voici un example:

```
// Dans le code qui suit p pointe vers un noeud, c'est-à-dire, p != NULL.
free(p->prenom);
free(p);
p = NULL;  // Petite précaution.
```

Évaluation (total 6 points)

Ce TP est évalué dans la maniere suivant:

- Un programme qui suit à toutes les points ci-dessus arrive au moins à 4 points sur 6.
- La rapport qui explique clairement les fonctionnes et code (ou les commentaires suffisant et claire) et le formatage du code valent 1 point.
- Si le travail ne comprend ni des bugs (segfaults, mauvaise traitements des données, ni des oublis (fermeture des fichiers, etc.), et les deux points précedent sont remplis, il est possible d'arriver sur une note de 6.
- Si le travail va clairement plus loin, il est possible d'arriver sur les points bonus. Il faut bien signaler à ces faits dans la rapport. Les points bonus données sont toujours à la discretion des responsables des TPs.

Consignes générales

- Vous devez aussi rendre un makefile capable de compiler/créer *toutes* les sources et exécutables de ce TP en tapant simplement: **make** ou **make TP4**
- Respectez les noms (majuscules et miniscules) des fichiers, des fonctions, des structures de données , des étiquettes, etc.
- Commentez votre code. N'écrivez pas tout le code dans la fonction main() mais décomposez le problème et écrivez chaque partie dans une fonction différente.
- Selon le cas, faites toujours la vérification de l'existence, de la syntaxe et/ou du domaine des entrées de vos programmes.

- En utilisant la documentation des fonctions, faites toujours la vérification d'erreurs possibles.
- Documentez dans le rapport tout ce qui ne peut être expliqué à l'aide des simples commentaires dans le code source. Le rapport est un bon endroit pour donner un exemple concret comment votre algorithme marche.
- Envoyez le fichier compressé contenant tous vos codes sources à l'adresse systinfo@unige.ch.