Programmation avancée
Examen du 17 mai 2016

Exercice 1 : stockage des entiers

Expliquer comment est stocké un entier dans un ordinateur. Illustrer en donnant le stockage de -13.

Exercice 2 : types et fichiers

On considère le fragment de code suivant :

```
struct point {
   int abs;
   int ord;
};

struct point hexagone[6];
```

```
struct enreg {
   int num;
   char *nom;
   char *description;
   char id[7];
   float taille;
};
```

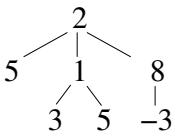
On fera attention à la gestion des erreurs, et on arrêtera le programme (par `err()` de préférence) à la première erreur détectée. Les fonctions listées à la fin de l'exercice suffisent pour répondre aux questions (mais vous pouvez en utiliser d'autres).

1. Écrire le fragment de code permettant de sauvegarder `hexagone` dans un fichier **binaire** nommé `hexa.bin`.
2. Écrire un type pour coder un polygone sous forme d'une liste chaînée de points.
3. Écrire une fonction `ecrirePoly()` écrivant dans un fichier **binaire** un polygone. Le nom du fichier et le polygone sont passés en paramètre.
4. Écrire une fonction `lirePoly()` qui fait l'inverse. Elle prend en paramètre un nom de fichier et renvoie un polygone (c'est-à-dire la tête de la liste chaînée). Les points doivent être dans le même ordre dans la liste chaînée que dans le fichier.
5. Écrire une fonction `void ecrireEnreg(char filename, int n, struct enreg *T)` qui permet de sauvegarder dans un fichier nommé `filename`, dans un format binaire, un tableau dynamique `T` de `n struct enreg`. Si vous utilisez (correctement) un format texte pour cette question et la suivante vous aurez la moitié des points.
6. Écrire une fonction `struct enreg *lireEnreg(FILE *f)` lit qui lit dans un fichier `f` (préalablement ouvert) un enregistrement écrit par la fonction précédente, et le renvoie.

```
void err(int codeRetour, const char *format, ...);
int fclose(FILE *f);
int feof(FILE *f);
char *fgets(char *s, int size, FILE *f);
FILE *fopen(const char *path, const char *mode);
int fputs(const char *s, FILE *f);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *f);
void free(void *ptr);
int fseek(FILE *f, long offset, int whence); whence = SEEK_SET SEEK_CUR SEEK_END
long ftell(FILE *f);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *f);
void *malloc(size_t size);
size_t strlen(const char *s);
```

Exercice 3 : Arbres

1. Donner un type `struct noeud` utilisable pour faire un arbre dont chaque nœud contient un entier et peut avoir un nombre quelconque de fils
2. Écrire une fonction `void freeAll(struct noeud *N)` qui appelle `free()` pour désallouer tous les noeuds du sous-arbre de racine `N`. On suppose que tous les noeuds avaient été alloués par `malloc()`.
3. Le *produit d'une arête* de l'arbre entre un père P et un fils F est la multiplication de la valeur stockée dans P par celle stockée dans F. Écrire une fonction `int produitAretes(struct noeud *N)` donnant la somme des produits de toutes les arêtes du sous-arbre de racine `N`. Par exemple, pour l'arbre suivant, appelée sur la racine, cette fonction renvoie 12 car $2 \times 5 + 2 \times 1 + 2 \times 8 + 1 \times 3 + 1 \times 5 + 8 \times -3 = 12$.



Exercice 4 : récursivité et mémoire

On considère la fonction suivante :

```
int f(int x) {
   static int y = 5;
   int *z = (int *) malloc(2*sizeof(int));
   int r=0;
   if(r>0) {
      z[0] = f(x-1);
      z[1] = f(x-2);
      r = z[0] + z[1] + y++;
   }
   free(z);
   return r;
}
```

Décrire le calcul de `f(3)` en donnant l'état de la mémoire (pile, tas, zone statique) à chaque étape.