Initiation à la programmation en C

Correction du TP n°10

Antoine Miné

10 mai 2007

Site du cours: `http://www.di.ens.fr/~mine/enseignement/prog2006/`

Exercice 1. Génération de listes.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

struct cell {
  int          data;
  struct cell* next;
};

struct cell* insere_tete(struct cell* head, int elem)
{
  struct cell* c = malloc(sizeof(struct cell));
  assert(c);
  c->data = elem;
  c->next = head;
  return c;
}

struct cell* liste_aleatoire(int nb)
{
  struct cell* head = NULL;
  for (; nb>0; nb--)
    head = insere_tete(head, lrand48() % 100);
  return head;
}

void affiche_liste(struct cell* head)
{
  for (; head; head=head->next )
    printf("%i ", head->data);
  printf("\n");
}
```

Exercice 2. Retournement de listes.

```
struct cell* retourne_liste_copie(struct cell* head)
{
  struct cell* h = NULL;
```

```
  while (head) {
    h = insere_tete(h, head->data);
    head = head->next;
  }
  return h;
}

struct cell* retourne_liste_en_place(struct cell* head)
{
  struct cell *prev, *cur, *next;
  prev = NULL;
  cur = head;
  while (cur) {
    next = cur->next;
    cur->next = prev;
    prev = cur;
    cur = next;
  }
  return prev;
}
```

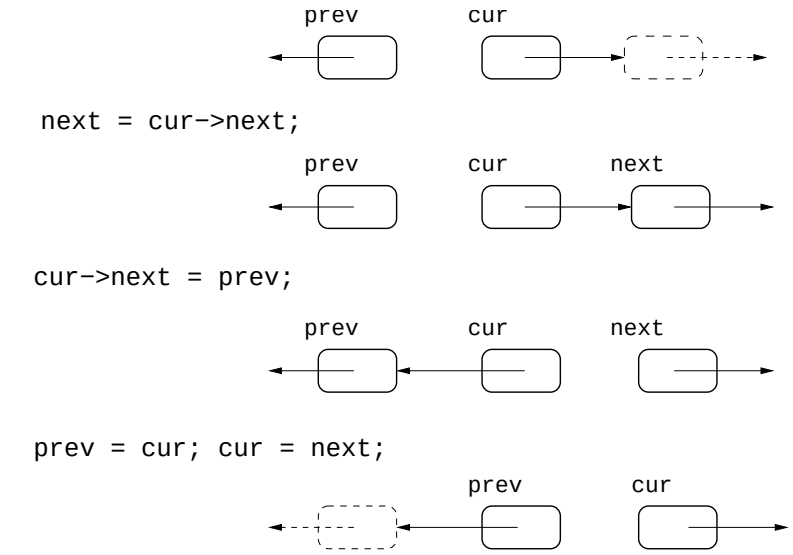

Illustration de `retourne_liste_en_place`.

Exercice 3. Tri par fusion.

```
struct cell* decoupe(struct cell* head)
{
  struct cell *c, *d;
  int i, nb;

  /* calcule la taille de la liste */
  for (nb=0, c=head; c; nb++, c=c->next);

  /* trouve la cellule de milieu de liste */
  for (i=0, c=head; i<nb/2-1; i++, c=c->next);

  d = c->next;    /* d = tête de la 2ème liste */
  c->next = NULL; /* c = queue de la 1ère liste */

  return d;
```

```
}

struct cell* fusion(struct cell* head1, struct cell* head2)
{
  struct cell *head, *tail;

  /* cas où une des listes est vide */
  if (!head1) return head2;
  if (!head2) return head1;

  /* choisit soit head1, soit head2 comme tête de liste */
  if (head1->data <= head2->data) {
    head = head1;
    head1 = head1->next;
  }
  else {
    head = head2;
    head2 = head2->next;
  }
  tail = head;

  /* tant qu'il reste des éléments dans une des listes */
  while (head1 && head2) {
    /* ajoute soit head1 soit head2 à tail */
    if (head1->data <= head2->data) {
      tail->next = head1;
      head1 = head1->next;
    }
    else {
      tail->next = head2;
      head2 = head2->next;
    }
    tail = tail->next;
  }

  /* ajoute les éléments restant à tail */
  tail->next = head1 ? head1 : head2;

  return head;
}

struct cell* tri(struct cell* head)
{
  struct cell* head2;
  if (!head || !head->next) return head; /* cas liste vide ou à un élément */
  head2 = decoupe(head);
  head  = tri(head);
  head2 = tri(head2);
  return fusion(head, head2);
}
```

Exercice 4. Tri générique.

```
struct cell {
  char*        data;
  struct cell* next;
};
```

La fonction `fusion` prend en argument supplémentaire le pointeur de fonction `compare`. Les deux comparaisons `head1->data <= head2->data` ont été changées en `compare(head1->data, head2->data) <= 0`.

```
struct cell* fusion(struct cell* head1, struct cell* head2, int (*compare)(char*,char*))
{
  struct cell *head, *tail;

  /* cas où une des listes est vide */
  if (!head1) return head2;
  if (!head2) return head1;

  /* choisit soit head1, soit head2 comme tête de liste */
  if (compare(head1->data,head2->data)<=0) {
    head = head1;
    head1 = head1->next;
  }
  else {
    head = head2;
    head2 = head2->next;
  }
  tail = head;

  /* tant qu'il reste des éléments dans une des listes */
  while (head1 && head2) {
    /* ajoute soit head1 soit head2 à tail */
    if (compare(head1->data,head2->data)<=0) {
      tail->next = head1;
      head1 = head1->next;
    }
    else {
      tail->next = head2;
      head2 = head2->next;
    }
    tail = tail->next;
  }

  /* ajoute les éléments restant à tail */
  tail->next = head1 ? head1 : head2;

  return head;
}
```

La fonction `tri` prend en argument supplémentaire le pointeur de fonction `compare` à passer à `fusion` (et, récursivement, à `tri`).

```
struct cell* tri(struct cell* head, int (*compare)(char*,char*))
{
  struct cell* head2;

  /* cas liste vide ou à un seul élément */
  if (!head || !head->next) return head;

  head2 = decoupe(head);
  head  = tri(head, compare);
  head2 = tri(head2, compare);
  return fusion(head, head2, compare);
}
```

La fonction `decoupe` n'est pas modifiée.