Enseignants : A. Bonhomme, L. Bougé
Email : {Alice.Bonhomme,Luc.Bouge}@ens-lyon.fr

le 13/11/00

TD Programmation 6 : Listes en C

L'objectif de ce TD est de proposer une gestion de listes chaînées, similaire à celle vue en CAML. Pour ceci, nous allons définir notre propre module d'allocation dynamique, c'est-à-dire de gestion de la mémoire.

1 Module naïf de gestion de la mémoire

La mémoire est représentée par un tableau alloué statiquement au début du programme. On définit par une constante `NB_CELL` le nombre maximum de cellules que peut contenir la mémoire. On définit une liste et ses cellules de la manière suivante : une liste sera représentée par l'indice dans la mémoire de la première cellule, ou `EMPTY` pour une liste vide.

```
#define NB_CELL 50
#define EMPTY -999

typedef int ilist;

typedef struct{
    int item;
    ilist next;
}cell;

static cell memory[NB_CELL];
```

- Écrire une fonction d'initialisation de la mémoire `void init_memory()`.
- Écrire une fonction `print_memory()` qui permette de visualiser l'état de la mémoire à un instant donné.
- La fonction `ilist get_new_list_cell()` renvoie l'indice d'une cellule de la mémoire disponible. Pour ceci, on utilise un indice global qui permet de savoir quelles sont les cellules allouées du tableau. Avant cet indice, les cellules sont allouées, après elles ne le sont pas. Écrire cette fonction.

2 Manipulation des listes

- Implémenter les fonctions usuelles sur les listes `empty`, `cons`, `car`, `cdr`, `is_empty`.
- Écrire une fonction `generate(int n, int k)` qui génère une liste de `n` éléments de valeur aléatoire tirée entre `0` et `k` (utiliser la fonction `lrand48`, pour plus d'information taper `man lrand48`).
- Implémenter quelques fonctions classiques sur des listes : `length` et `print_list` de manière récursive puis itérative.
- Proposer un programme de test permettant de tester vos fonctions de manipulation des listes en utilisant plusieurs listes.

3 Module avancé de gestion de la mémoire

Dans la version qu'on vous propose, la gestion de la mémoire est faite de manière brutale : on va très vite remplir la mémoire car on ne libère pas les cellules. On propose de modifier la structure des cellules pour pouvoir distinguer les cases mémoire vides et pleines. Par exemple, chaque case pourra contenir une information en plus, de type `status` : `typedef enum {FREE, BUSY} status` ;

Ceci permet de pouvoir libérer des cellules et de les réutiliser par la suite.

- Modifier la structure des cellules mémoires.
- Modifier la fonction d'initialisation de la mémoire `init_memory`, ainsi que la fonction de visualisation de le l'état de la mémoire `print_memory`.
- Modifier en conséquence la fonction `ilist get_new_list_cell()`.
- Écrire une fonction `free_list(ilist list)` qui libère chaque cellule de la liste.
- Reprendre votre programme de test avec cette nouvelle gestion de la mémoire, et rajouter des tests de libération de listes (les fonctions de manipulation des listes ne changent pas).
- Reprendre les fonctions de manipulation des listes `car`, `cons`, `cdr` et `is_empty`, en incluant une vérification de l'état de la liste manipulée. Par exemple, la fonction `car` appliquée à une liste non vide mais dont le statut est `FREE`, doit retourner une erreur.

4 Module optimisé de la gestion de la mémoire

La gestion de la mémoire proposée plus haut a l'inconvénient de devoir parcourir régulièrement le tableau afin de trouver une cellule disponible. Pour éviter ceci, on peut chaîner les cellules disponibles en remplaçant le `status` par l'indice de la prochaine cellule disponible. Ce fonctionnement est illustré dans la figure 1. Pour ce fonctionnement, il est nécessaire de garder l'indice de la première case disponible. Ainsi, lorsque l'on veut prendre une nouvelle cellule, on prend celle indiquée par cet indice. Lorsqu'on libère une cellule, on modifie cet indice.

- Modifier la structure définissant la mémoire afin d'intégrer le chaînage des cellules.

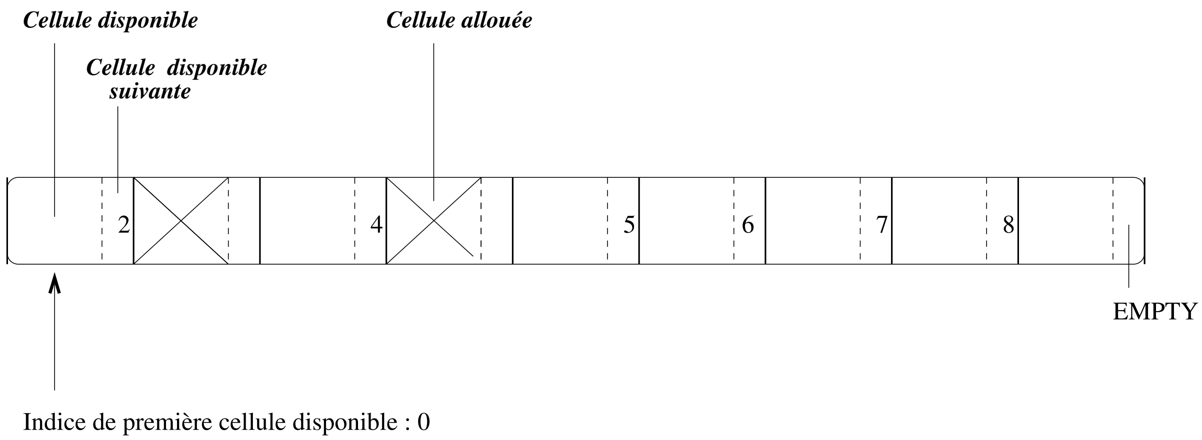

Fig. 1: Exemple de gestion de la mémoire en chaînant les cellules disponibles

- Modifier la fonction d'initialisation de la mémoire pour initialiser ce chaînage ainsi que l'indice de la première case disponible.
- Modifier en conséquence les fonctions `get_new_list_cell` et `free_list` et tester à nouveau avec le petit programme de test défini précédemment.

5 Module paramétrable de la gestion de la mémoire (pour ceux qui ont fini)

On veut pouvoir maintenant utiliser le même espace mémoire pour gérer différents types de liste (notamment des listes dont le contenu a une taille différente).

Les solutions proposées précédemment ne sont plus adaptées car une cellule ne correspondra pas forcément à une seule case du tableau représentant la mémoire, mais peut-être à plusieurs. On parlera donc maintenant non plus de cellule mais d'espace mémoire.

On va donc associer à un espace mémoire, une taille, ainsi qu'un chaînage vers les autres espaces mémoires disponibles.

Proposer un module de gestion de la mémoire implémentant cette solution.