Langage C/C++

TD 4 : Création dynamique d'objets

Hubert Godfroy

4 décembre 2014

Le but du problème est de modéliser la gestion mémoire d'un ordinateur.

— On utilise pour cela un tableau de taille (en octets) `TAILLEMEMOIRE`.
— Une `zone` est la donnée d'une adresse d'où part la zone et de la taille de la zone.
— Pour savoir quelles zones sont allouée, on utilise une liste chaînée des structure donnant la taille de la zone allouée, et sa position dans la mémoire.
— Lors de la destruction d'objets en mémoire, la zone mémoire allouée est libérée. On utilise une deuxième liste donnant l'ensemble des zones libérées de la mémoire.

Lors de l'allocation d'un objet de taille l en mémoire, la localisation en mémoire est choisie selon cet algorithme :

1. On cherche dans la liste `zones_liberees` une zone de taille au moins l.
2. Si aucune zone n'est disponible, le programme crash.
3. Sinon, on remplace la zone choisie par une zone amputée de la taille de l'allocation et on ajoute à `zones_allouees` la zone fraichement allouée.

De même, lors de la libération de la mémoire :

1. La zone libérée est supprimée de la liste `zones_allouees`.
2. On ajoute la zone fraichement libérée à la liste `zones_liberees`. Ce faisant, on fera vérifiera que lors de l'ajout d'une zone libérée z à la liste des autres zones libérées, s'il existe une zone libre z' attenante à z, on veux pas ajouter bêtement z à la liste, mais on veux que les zones soient fusionnées.

On propose la structure suivante pour la mémoire :

```
1 typedef struct _memoire {
2   char tas[TAILLEMEMOIRE];
3   liste* zones_allouees;
4   liste* zones_liberees;
5 } memoire;
```

1 Stratégie d'allocation

Question 1 :
Proposer une définition du type `zone`.

Question 2 :
Proposer une définition du type `liste`.

Question 3 :
Écrire une fonction prenant une zone et une taille `l` inférieure à la taille de la zone, et renvoyant la zone obtenue en tronquant les `l` premier octets de la zone de départ.

Question 4 :
Écrire une fonction prenant une liste de zones et une taille `l` et renvoyant l'adresse de la première zone de taille supérieure ou égale à `l`.

Question 5 :
Écrire une fonction ajoutant à la liste de zones triées par adresse une nouvelle zone. On fera attention à conserver la liste triée, mais on ne s'occupera pas dans cette question de la fusion de zones attenantes.

Question 6 :
En utilisant les questions 3, 4 et 5, écrire une fonction d'allocation prenant une taille `l` en paramètre et cherchant en mémoire un endroit où allouer un espace de taille `l`.

Question 7 :
Écrire une fonction prenant en paramètre une liste supposée triée par adresses et fusionnant les zones attenantes [1].

Question 8 :
En utilisant les questions 7 et 5, écrire une fonction de désallocation d'une zone mémoire allouées (donnée par son adresse)

2 Gestion des pointeurs

On ajoute à la mémoire une pile sous-forme d'une tableau d'entiers et de taille `TAILLEPILE` et d'un *marqueur de pile* indiquant la fin de la partie allouée de la pile.

2.1 Fonctions de base

Question 9 :
Modifier le type mémoire en conséquence de ces changements.

Question 10 :
Écrire une fonction `empiler` ajoutant une valeur sur la pile, c'est à dire ajoutant la valeur à la position du marqueur, puis incrémentant le marqueur de 1.

1. Attention, la fusion peut concerner plus de deux zones à la fois : si par exemple la liste contient les zones z_1, z_2 et z_3 et que z_i est attenante à z_{i+1}, il faut fusionner les trois zones en une.

Question 11 :

Écrire une fonction `dépiler` supprimant la valeur au sommet de la pile et diminuant de 1 le marqueur de pile.

2.2 Comportement de la mémoire

Question 12 :

Écrire une fonction `get` prenant un entier n et une mémoire et renvoyant le n-ème élément de la pile.

Question 13 :

Écrire une fonction `set` prenant un entier n, une mémoire et un entier v et écrivant la valeur v à la place n dans la pile

Question 14 :

Écrire une fonction `mem_alloc` prenant en paramètre une mémoire m et un entier l, allouant dans la mémoire m un espace de taille l et ajoutant en sommet de pile l'adresse où a été créée l'allocation.

Question 15 :

Écrire une fonction `access` prenant en paramètre une mémoire et un entier n et donnant la valeur stockée dans la partie du tas allouée à l'adresse en position n dans la pile.

Question 16 :

Écrire une fonction `mem_free` prenant en paramètre une mémoire et un entier n désallouant la partie mémoire dont l'adresse se trouve en position n dans la pile de la mémoire.