IMA 3ème année

Programmation Avancé

TP Optionnel
Allocation statique vs dynamique – Réallocation

1 Objectifs

— Savoir utiliser les listes contiguës en C.
— Savoir allouer un vecteur et une matrice dynamiquement
— Savoir utiliser la fonction realloc

Contexte et préparation : Ce TP n'a pas de pré-requis en terme de code existant. Vous travaillerez dans un nouveau répertoire nommé TP3.

IMPORTANT : Ce TP est assez court, assurez-vous que vous comprenez bien pourquoi vos programmes fonctionnent (ou pas !).

Pour minimiser les risques de faire planter votre PC, lancez votre programme en utilisant la plus faible priorité possible avec les commandes `ionice` et `nice`.
E.g., `ionice -c3 nice -n19 ./programme`

2 Questions du TP (à faire impérativement)

Toutes les questions sont à tester au fur et à mesure (appels dans le main), comme d'habitude !

2.1 Allocation statique vs dynamique

1. Dans un fichier `alloc_statique.c` déclarer une fonction avec une matrice d'entiers `M` de taille `SIZE*SIZE` avec `SIZE` constante (par exemple, égale à 400 pour commencer), et initialiser la matrice tel que `M[i][j]=i+j`. Compiler et tester. Augmenter la constante `SIZE` et observer les limites de l'allocation sur la pile. Noter à quelle taille votre code cesse de fonctionner, et à combien de bytes/kilobytes cela correspond.
2. Dans le même fichier, déclarer maintenant la matrice `M` comme variable globale, et tester les limites d'allocation dans la zone de données. Donner les limites.
3. Dans un fichier `alloc_dynamique.c`, déclarer `(int *v)`, allouer et initialiser un vecteur de `SIZE` entiers.
4. Dans le fichier `alloc_dynamique.c`, déclarer `(int **mat)`, allouer et initialiser une matrice de taille `SIZE*SIZE` : la matrice est un *vecteur de vecteurs*. Pour faciliter ces allocations, vous pouvez utiliser un boucle qui alloue un vecteur à chaque itération et l'assigne à la matrice.
5. Tester les limites d'allocation sur le tas en affichant le total de la taille mémoire allouée la première fois que `malloc` retourne `NULL`. Pour trouver les limites, changer la taille des vecteurs (par exemple, allouez 1 mégaoctet par itération, 10 mégaoctets par itération, 100 mégaoctets par itération, 1 gigaoctet par itération, ...).
6. Désallouer proprement la matrice. Vérifier qu'il n'y a pas de fuite mémoire avec l'utilitaire *valgrind* [1].

1. Regardez la documentation pour apprendre à utiliser valgrind `http://valgrind.org/docs/manual/quick-start.html#quick-start.intro`

2.2 Realloc *(Un exercice de F. Boulier pour GIS)*

Le programme suivant lit une chaîne de caractères au clavier et l'imprime sur la sortie standard.

```c
#include <stdio.h>
#include <ctype.h>

int main()
{
    char c;
    c = getchar();

    while (! isspace(c))
    {
        putchar(c);
        c = getchar();
    }
    putchar('\n');

    return 0;
}
```

Questions

1. Que fait la fonction `isspace` (man 3 isspace) ?
2. Modifier ce programme de façon à stocker les caractères dans un vecteur alloué dynamiquement, au fur et à mesure de la lecture. Attention de veiller à ne pas dépasser les limites du vecteur alloué (ajouter une condition d'arrêt dans le `TQ`). Attention également à désallouer correctement le vecteur en fin d'exécution.
3. Consulter la documentation de la fonction `realloc` (man realloc).
4. Utiliser la fonction `realloc` afin de redimensionner la taille du vecteur lorsque les limites sont atteintes (par exemple, lui rajouter 8 caractères), afin de permettre la fin de saisie uniquement sur lecture d'un caractère blanc d'espacement (' ', '\n', '\t', ...). Veiller à désallouer correctement le vecteur en fin d'exécution. Vérifier avec *valgrind* qu'il n'y a pas de fuite mémoire.

3 Questions s'il vous reste du temps

Une structure chaine *(D'après un exercice de F.Boulier pour GIS).*
Pour faciliter l'écriture d'algorithmes de chaînes, et ne plus se préoccuper d'allocation dynamique, nous allons encapsuler les chaînes de caractère dans une structure, et écrire les fonctions de base pour cette structure. Dans un nouveau fichier `chaine.c` :

1. Déclarer un nouveau type :

   ```c
   typedef struct {
       char * data;
       int alloc;
       int size;
   } chaine ;
   ```

   Le sens de cette structure est le suivant :
   - Le champs `data` contient l'adresse d'une zone de mémoire allouée dynamiquement qui contient une chaîne de caractères (terminée par `'\0'`).
   - Le champs `alloc` contient le nombre de char alloués à `data`.
   - Le champs `size` contient la longueur de la chaîne (le `'\0'` n'est pas compté)
   - On a toujours `size+1 <= alloc`.
2. Écrire la fonction `init_chaine(chaine *)` qui initialise les champs `alloc` et `size` à `0`, et `data` à `NULL`.
3. Écrire la fonction `clean_chaine(chaine *)` qui désalloue la mémoire allouée dynamiquement pour la chaîne.
4. Écrire la fonction `void print_chaine(chaine *)` qui imprime (les données de) la chaîne.
5. Écrire une fonction `concat_chaine_char(chaine * , char)` qui ajoute un caractère à la fin de la chaîne, en redimensionnant le champs `data` si nécessaire (avec `realloc`).
6. Tester ces fonctions de la même façon que précédemment : initialisation, copie des caractères demandés à l'utilisateur, puis impression et enfin désallocation.
7. Écrire une fonction `concat_chaine_chaine(chaine * , chaine *)` qui ajoute tous les caractères de la deuxième chaîne à la fin du premier, en redimensionnant le champs `data` si nécessaire (avec `realloc`). Testez cette nouvelle fonctionnalité.