

TP3 Allocation Statique vs Dynamique / Réallocation

Objectifs

- Savoir utiliser les allocations dynamiques pour des tableaux.
- Savoir récupérer la valeur de retour de `malloc`.
- Savoir utiliser `realloc`.

Contexte et préparation Ce TP n'a pas de prérequis en terme de code existant. Vous travaillerez dans un répertoire tout neuf, nommé TP3. **Remarque :** ce TP est assez court, assurez-vous que vous comprenez bien pourquoi vos programmes fonctionnent (ou pas!).

1 Questions du TP (à faire impérativement)

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

Allocation Statique vs dynamique

1. Dans un fichier `alloc_statique.c`, déclarer et initialiser une matrice d'entiers M de taille `SIZE*SIZE` avec `SIZE` constante (égale à 400 par exemple), et $M_{i,j} = i + j$. Compiler et tester. Augmenter la taille, et observer les limites de l'allocation sur la pile.
2. Dans le même fichier, déclarer maintenant la matrice M comme variable globale, et tester les limites d'allocation dans la zone "data".
3. Dans un fichier `alloc_dynamique.c`, déclarer, *allouer* et initialiser une matrice comme étant un tableau de tableaux d'entiers (dessin!). `int **M`, toujours avec $M_{i,j} = i + j$. Tester les limites d'allocation sur le tas, en imprimant $i * j$ la première fois que `malloc` retourne NULL.
4. Désallouer ensuite proprement chacune des cases de cette matrice. Vérifier que vous n'avez pas de fuite mémoire avec l'utilitaire **Valgrind**.

realloc *Un exercice de F. Boulier pour GIS*

Le programme C suivant (à réécrire) lit une chaîne de caractères au clavier et l'imprime sur la sortie standard.

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

int main ()
{
    char c;
    c = getchar ();
    while (! isspace (c))
    {
        putchar (c);
        c = getchar ();
    }
    putchar ('\n');
```

```
return 0;
}
```

1. Que fait la fonction `isspace` (man 3 `isspace`) ?
2. Modifier ce programme de façon à stocker tous les caractères dans un tableau `t` de caractères de taille 5. *Pour la suite, on prendra soin de déclarer la variable entière `i` d'indice de stockage dans le tableau après le tableau de stockage*
3. Modifier de façon à imprimer `c` et le nombre de tours `i` à chaque tour de la boucle `while`. Essayer de stocker trop de caractères et observer ce qui se passe pour `i`.
4. Regarder la doc de la fonction `realloc`.
5. Modifier ce programme de telle sorte que tous les caractères lus soient stockés dans une variable `s`, de type `char*`, avant d'être imprimés. La variable `s` doit pointer vers une zone allouée dynamiquement, redimensionnée à l'aide de `realloc` dès qu'elle est pleine (par exemple, en lui rajoutant huit caractères). On veillera à bien désallouer proprement à la fin (et diagnostiquer avec Valgrind).

2 Questions s'il vous reste du temps

Une structure chaîne (*D'après un exercice de François 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 `chaîne.c` :

1. Déclarer un nouveau type :

```
struct chaîne
{
    char* data;
    int alloc;
    int size;
};
```

Le sens de cette structure est le suivant :

- On suppose que le champ `data` est toujours différent de zéro.
 - Le champ `data` pointe vers une zone allouée dynamiquement. Il contient une chaîne au sens du langage C (terminée par `'\0'`).
 - Le champ `alloc` contient le nombre de char alloués à `data`.
 - Le champ `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(chaîne*)` qui initialise les champs `alloc` et `size` à 0, et `data` à NULL (pointeur nul).
 3. Écrire la fonction `clean_chaine(chaîne*)` qui désalloue.
 4. Écrire la fonction `void print_chaine(chaîne*)` qui imprime (les données de) la chaîne.
 5. Écrire une fonction `void concat_chaine_char(chaîne*, char)` qui ajoute un caractère à la fin de la chaîne, en allouant de la mémoire fraîche si nécessaire (avec `realloc`).
 6. Tester ces fonctions de la même façon que précédemment : `init`, copie des caractères demandés à l'utilisateur, puis impression, et enfin désallocation.