Département IMA / 3A (S6) / 2012 **Programmation Avancée**
Bernard Carré, Nathalie Devésa, Julie Jacques, Shahin Gelareh et Laure Gonnord.

TP4 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 `TP4`. **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.
2. 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 $M_{i,j} = i + j$. Désallouer ensuite proprement chacune des cases de cette matrice.
3. Comparer à l'aide de la commande Unix `time` les temps d'exécution des deux programmes précédents.
4. Rajouter une vérification de la valeur de retour de `malloc` dans `alloc_dynamique.c` et tester les limites d'allocation de votre machine.

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 ()
{
  int 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 fixée à l'avance. Essayer de stocker une chaîne trop grande.
3. Regarder la doc de la fonction `realloc`.
4. 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.

2 Questions s'il vous reste du temps

Une structure `chaine` *(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 `chaine.c` :

1. Déclarer un nouveau type :
   ```
   struct chaine
   {   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(chaine*)` qui initialise les champs `alloc` et `size` à 0, et `data` à NULL (pointeur nul).
3. Écrire la fonction `clean_chaine(chaine*)` qui désalloue.
4. Écrire la fonction `void print_chaine(chaine*)` qui imprime (les données de) la chaîne.
5. Écrire une fonction `void concat_chaine_char(chaine*, 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.