

Travail pratique #2

IFT-2035

November 4, 2009

⌘ Dû le 25 novembre à 8h30 !!

1 Survol

Ce TP a pour but de vous familiariser avec le langage C, les pointeurs, la gestion de mémoire explicite. Comme pour le TP précédent, les étapes sont les suivantes:

1. Parfaire sa connaissance de C.
2. Lire et comprendre cette donnée.
3. Écrire le code.
4. Écrire un rapport. Il doit décrire **votre** expérience pendant les points précédents: problèmes rencontrés, surprises, choix que vous avez dû faire, options que vous avez sciemment rejetées, etc... Le rapport ne doit pas excéder 5 pages plus 2 d'annexe (voir plus loin).

Ce travail est à faire en groupes de 2 étudiants. Le rapport (au format PDF) et le code sont à remettre par remise électronique avant la date indiquée. Aucun retard ne sera accepté. Indiquez clairement votre nom au début de chaque fichier.

Si un étudiant préfère travailler seul, libre à lui, mais l'évaluation de son travail n'en tiendra pas compte. Si un étudiant ne trouve pas de partenaire, il doit me contacter au plus tard mercredi 11 novembre. Des groupes de 3 ou plus sont **exclus**.

2 Une librairie de gestion mémoire

Vous devez simplement écrire une librairie de gestion mémoire pour des chaînes de caractères qui fournisse les fonctions suivantes:

```
typedef struct String String;
String *str_alloc (int size);
int str_size (String *str);
char *str_data (String *str);
void str_free (String *str);
void str_compact (void);
```

Libre à vous de définir `struct String` comme bon vous semble. La fonction `str_alloc` crée une nouvelle chaîne de taille `size`. La fonction `str_size` renvoie la taille d'une chaîne et la fonction `str_data` renvoie le tableau de bytes de la chaîne. Finalement `str_free` permet de libérer l'espace occupé par une chaîne et `str_compact` permet de compacter l'espace mémoire occupé par toutes les chaînes de manière à réduire la fragmentation.

Pour que la compaction puisse se faire, les programmes qui utilisent cette librairie doivent s'assurer de ne pas utiliser de *str_data* pendant la compaction et de plus il doivent faire attention à ne plus utiliser les *str_data* obtenu avant la compaction vu qu'ils peuvent ne plus être valides.

Vous devez écrire le code dans un fichier `stralloc.c`. Pour obtenir de la mémoire, cette librairie ne doit pas utiliser `malloc`. À la place, il faudra utiliser des fonctionnalités de plus bas niveau, plus précisément `mmap` et `munmap`. Ces fonctions sont spécifiques au système d'exploitation utilisé, et votre code sera testé dans un système GNU/Linux raisonnablement récent.

`mmap` ne peut allouer que des multiples de pages (i.e. des multiples de 4KB par exemple). Il faudra donc souvent les partager entre plusieurs chaînes de caractères.

Une partie importante du TP est non seulement l'écriture du code, mais aussi l'analyse de ses performances (en termes théoriques et non pas sur la base d'expérimentation):

- Coût en mémoire de la librairie dans les différents cas de figure (différentes tailles de régions, différents nombres et tailles d'objets dans les région, fragmentation, ...).
- Complexité algorithmique de chaque opération.

Si vous voulez ajouter des mesures expérimentales pour confirmer ou nuancer ces résultats théoriques, ne vous gênez pas.

3 Ce que vous devez faire

- Écrire le code dans `stralloc.c`.
- Analyser ses performances (maximum 2 pages en annexe du rapport).

Le code fourni dans `stralloc.h` ne peut pas être changé, à moins d'une très bonne raison. Remarquez que ce TP2 peut se décomposer en 3 parties:

1. Implanter seulement `str_alloc`, `str_size` et `str_data`; utiliser pour `str_free` et `str_compact` une implantation triviale qui ne fait rien.
2. Modifier le code pour implanter un vrai `str_free`.
3. Modifier le code pour implanter un vrai `str_compact`.

4 Notes

Vous devez remettre deux fichiers: `stralloc.c` et `rapport.pdf`. La commande pour remettre ces fichier est la suivante:

```
% remise ift2035 tp2 stralloc.c rapport.pdf
```

- Un bon point de départ pour vous aider à décider comment gérer vos pages de mémoire est à <http://www.memorymanagement.org/articles/alloc.html>. Voir aussi <http://www.memorymanagement.org/articles/begin.html>.
- Tout usage de matériel (code ou texte) emprunté à quelqu'un d'autre (ou trouvé sur le web) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat.
- Chaque ligne de code doit faire moins de 80 caractères. Tout dépassement sera considéré comme une erreur.
- Votre code doit compiler avec `gcc -Wall` sans générer plus d'avertissements que le code fourni.
- Vérifiez la page web du cours, pour d'éventuels errata, et d'autres indications supplémentaires.