Université Paris Dauphine
IUP Génie Mathématique et Informatique
IUP1 – Année 2001-2002
Algorithmique et structures de données

Devoir à rendre le 8 novembre 2001

Le but de ce devoir est de concevoir une bibliothèque de *tableaux de caractères extensibles*, qui grossissent au fur et à mesure des besoins. Le nombre de caractères que contiendra un tableau n'est pas connu à l'avance : il n'est limité que par la mémoire disponible dans l'ordinateur. Le tableau est donc *dynamique*. On essayera de trouver une méthode intelligente qui évite les recopies intempestives du contenu du tableau.

Voici le contenu du fichier `tab.h` décrivant l'*interface* de la bibliothèque :

```
typedef enum { NoErr=0, MemErr=1 } err_t;

struct tab_t;
typedef struct tab_t tab_t;

tab_t* tab_create  (void);
void   tab_destroy (tab_t* tab);
err_t  tab_add     (tab_t* tab, char c);
char*  tab_get     (tab_t* tab);
```

Ce fichier se contente de *déclarer* des fonctions sans préciser leur *définition*. Remarquez également que le type `tab_t` est déclaré comme étant une structure, mais son contenu n'est pas précisé.
C'est le rôle du ficher `tab.c` de définir l'*implémentation* du type `tab_t` et des fonctions déclarées dans `tab.h`.

Voici un exemple de programme client `client.c` utilisant la bibliothèque de tableaux extensibles :

```
#include <stdio.h>
#include <stdlib.h>
#include "tab.h"

int main()
{
  FILE* f = fopen ("client.c","r");
  tab_t* tab = tab_create ();
  char *s,*ss;
  if (!tab) { fprintf (stderr, "Impossible de créer le tableau!\n"); return 1; }
  if (!f) { fprintf (stderr, "Fichier introuvable!\n"); return 1; }
  for (;;) {
    int c = getc (f);
    if (c==EOF) break;
    if (tab_add (tab, (char)c)!= NoErr) { fprintf (stderr, "Erreur dans tab_add!\n"); return 1; }
  }
  fclose (f);
  s = tab_get(tab);
  tab_destroy (tab);
  printf ("Voici le contenu initial du fichier :\n%s\n", s);
  for (ss=s;*ss;ss++) if (*ss=='(') *ss='['; else if (*ss==')') *ss=']';
  printf ("Voici le contenu du fichier modifié :\n%s\n", s);
  return 0;
}
```

Remarquez que `client.c` *n'a à sa disposition que l'interface de la bibliothèque* `tab.h`*, pas son implémentation* `tab.c`. En particulier, le client ne peut pas déclarer de variable de type `tab_t`, ou accéder aux champs de la variable `tab` ; il ne peut que créer un *pointeur* sur un objet de type `tab_t`, et est obligé de manipuler les tableaux extensibles au travers des fonctions `tab_create`, `tab_destroy`, `tab_add` et `tab_get`. On dit que `tab_t` est un *type abstrait*.

Les fichiers `client.c` et `tab.c` étant indépendants, ils peuvent être *compilés séparément*. Pour obtenir un exécutable, il faut *lier ensemble* les fichier objets correspondant à la bibliothèque et au client.

Question 1 :

Sachant :
- que `tab_create()` retourne un *pointeur* vers un objet de type `tab_t` fraîchement alloué (ou `NULL` en cas d'erreur) et que ce tableau ne contient initialement aucun caractère ;
- que `tab_add(tab,c)` ajoute le caractère `c` *à la fin* du tableau `tab` et renvoie `NoErr` si tout s'est bien passé, `MemErr` si il n'y a plus assez de mémoire pour effectuer cette opération ;
- que `tab_destroy(tab)` libère *toute* la mémoire occupé par le tableau `tab` ; au retour de `tab_destroy`, le pointeur `tab` ne peut plus être utilisé par le client ;
- et que `tab_get(tab)` retourne une *copie* du contenu du tableau sous forme de *chaîne de caractères terminée par 0* ; le pointeur retourné est fraîchement alloué par `tab_get` et peut être libéré par le client en utilisant `free` ; modifier la chaîne de caractères retournée par `tab_get` ne modifie pas le contenu du tableau `tab` ;

décrire *succinctement* ce que fait le programme `client.c`.
Prévoir, en particulier, quelle sera la sortie sur l'écran.

Les fonctions `fopen`, `getc`, `fclose`, `printf` et `fprintf` ainsi que le type `FILE` et les symboles `EOF` et `stderr` font partie de la bibliothèque C standard.

Question 2 :

Dans cette question, on suppose que la structure `tab_t` possède deux champs :
- le nombre `nb` de caractères contenu dans le tableau extensible ;
- un pointeur vers un bloc de taille `nb` contenant les caractères du tableau.

Aucune taille maximale de tableau n'est définie : le client souhaite pouvoir agrandir le tableau avec `tab_add` tant qu'il reste assez de mémoire disponible.

Implémenter dans un fichier `tab.c` le type `tab_t` et les fonctions `tab_create`, `tab_add`, `tab_destroy`, `tab_get` qui ont le comportement décrit dans la question 1. Imprimer le fichier `tab.c`.
Compiler et exécuter l'ensemble client+bibliothèque et imprimer la sortie sur l'écran.

Quel est le coût au pire de `n` appels successifs à `tab_add`, en tenant compte du coût des copies éventuelles qui peuvent se produire quand on change la taille d'un bloc ?

Question 3 :

Pour éviter les recopies intempestives qui ont lieu lorsque le tableau extensible est agrandi par `tab_add`, on a l'idée d'allouer un bloc un peu plus grand que necessaire. La structure `tab_t` possède maintenant trois champs :
- le nombre de caractères `nb` dans le tableau ;
- la taille `max` du bloc contenant le tableau (`max`$\geq$`nb`) ;
- un pointeur vers un bloc de taille `max` dont les `nb` premiers caractères correspondent au contenu du tableau.

Tant que `max`$\geq$`nb`, `tab_add` se contente de mettre à jour le contenu du bloc et `nb`. Quand il n'y a plus assez de place, `tab_add` agrandit le tableau et met à jour `max`.

Supposons que, quand il n'y a plus de place, `tab_add` agrandit le bloc *d'une taille constante* `c`. Quel est le coût au pire de `n` appels successifs à `tab_add` ? Après ces `n` appels, quel est l'espace gâché (`max-nb`), au pire ?

Supposons que, quand il n'y a plus de place, `tab_add` *double* la taille du bloc. Quel est le coût au pire de `n` appels successifs à `tab_add` ? Après ces `n` appels, quel est l'espace gâché, au pire ?

Programmer cette dernière version, compiler et exécuter l'ensemble client+bibliothèque. Imprimer le fichier `tab.c` ainsi que la sortie sur l'écran.