Université Paul Sabatier TOULOUSE III

S3

Programmation impérative en langage C

feuille de TP n°2 : Tableaux dynamiques

Objectifs du TP :

- *Mise en œuvre des tableaux dynamiques*
- *Spécification*
- *Révision : fichiers texte*

L'allocation dynamique permet :

- de dimensionner un tableau avec la valeur d'une expression entière : malloc, realloc
- d'accéder à une zone de mémoire appelée tas (heap)

1 Vecteurs dynamiques

Les fonctions malloc,realloc,free sont dans la librairie stdlib.h, penser à ajouter en début de fichier : `include <stdlib.h>`
Exemple :

```
1  //création d'un vecteur dynamique de 100 réels dans le tas:
2  float* vec = malloc(100*sizeof(float));
3  //on vérifie ensuite que l'allocation s'est bien déroulée
4  //sinon on arrête le programme
5  if (vec ==NULL)
6          { printf("Pb d'allocation."); exit(1);   }
7  //on peut modifier la taille du tableau:
8  vec=realloc(vec,200*sizeof(float));
9  //on vérifie toujours que l'allocation s'est bien déroulée:
10 if (vec ==NULL)
11         { printf("Pb d'allocation."); exit(1);  }
12 //en fin de traitement, on libère la mémoire:
13 free(vec);
```

◇ Exercice **1** : On dispose d'un tableau `int T[5]`, écrire un programme qui recopie $N \geq 1$ fois les 5 cases de T dans un nouveau tableau de taille $5N$.

◇ Exercice **2** : Dans cet exercice, on utilise le type structuré :

```
struct Tab {
    int* tab; //tableau dynamique
    int n; //nombre de valeurs
};
```

1. Générer une variable de type `struct Tab` dont le nombre de valeurs est saisi au clavier. Les valeurs du tableau seront des entiers aléatoires compris dans l'intervalle $[|0, 100|]$.

2. On veut supprimer tous les multiples de 7 du tableau précédemment généré et les placer dans un autre tableau . A la fin, on récupère 2 objets de type struct Tab : l'objet initial débarrassé de tous les multiples de 7 et un nouvel objet contenant exclusivement tous les multiples de 7.

   On ne conservera pas nécessairement l'ordre initial. Cette tâche peut être effectuée en parcourant une seule fois le tableau de départ.

   Une fois le traitement terminé et les nombres de valeurs de chaque tableau connus, on pourra réallouer avec plus de précision.

3. Tester en affichant le tableau initial puis les 2 tableaux en sortie.

   Penser à libérer la mémoire en fin de programme.

2 Matrices dynamiques

Cas d'une matrice à N lignes et P colonnes :
Deux étapes sont ici nécessaires à l'allocation dynamique :
Exemple :

```
int N=2;
int P=3;
//allocation dans le tas d'un tableau dynamique de pointeurs sur int
int** m=malloc(N*sizeof(int*));
//on vérifie que l'allocation est ok
if (m==NULL)
{       printf("Pb allocation"); exit(1); }
int i,j;
for (i=0;i<N;++i)
{
    //allocation dans le tas d'un tableau dynamique m[i]
        m[i]=malloc(P*sizeof(int));
        //on vérifie que chaque allocation est ok
        if (m[i]==NULL)
        { printf("Pb allocation"); exit(1); }
}
```

◇ Exercice **3** : Une matrice avec n lignes et p colonnes est représentée par un tableau de dimension 2.
Par définition, le produit m1*m2 de deux matrices m1 et m2 s'effectue ligne par colonne. Ce produit n'est défini que si le nombre de colonnes de m1 est égal au nombre de lignes de m2.
par exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1\times 1+2\times 0+3\times(-1)+4\times 1=2 & 4 & 9 & 13 \\ 1 & 1 & 0 & 2 \\ -2 & -2 & 1 & -1 \\ 2 & 2 & 3 & 5 \end{pmatrix}$$

ou encore :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} -1 & 2 \\ 1 & 0 \\ 1 & 2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 12 \\ 1 & 1 \\ 3 & 3 \end{pmatrix}$$

Écrire un programme qui fait le produit matriciel de deux matrices saisies par l'utilisateur. Ces deux matrices sont de dimensions quelconques ; si le produit n'est pas possible on affiche un message sinon on affiche la matrice produit. Une fois l'affichage effectué, libérer la mémoire dynamique avant la fin de l'exécution.

3 Spécifications

On spécifie un programme par :

- un prédicat d'entrée PE réalisé par les données du programme (hormis le type des données)
- un prédicat de sortie PS réalisé par les résultats du programme
- des prédicats intermédiaires si nécessaire

Dans le programme, ces prédicats prendront la forme de commentaires en français ou bien de formules de la logique des prédicats. On utilisera les symboles habituels vus en cours :

- Quantificateurs :
  universel $\forall I :\ P_1 \to P_2$
  existentiel $\exists I :\ P_1 \wedge P_2$
  numérique $\nu I : (I \in E) \wedge P(I)$ quantifie le nombre d'éléments I de E vérifiant la propriété P
- Connecteurs :
  $\neg$ négation
  $\to$ implication
  $\wedge$ conjonction
  $\vee$ disjonction
- prédicats de la théorie des ensembles :
  $\in$ appartenance
  $\subset$ inclusion

avec le parenthèsage nécessaire.

Pour alléger certaines écritures on pourra utiliser :

- $(T(I..J), <)$ ou $(T(I..J), <=)$ qui exprime la croissance de T entre les indices I et J inclus
- $t = A$ pour exprimer que 2 vecteurs t et A ont la même taille et contiennent des valeurs identiques dans chacune de leurs cases
- $t = permut(A)$ pour exprimer que le vecteur t contient toutes les valeurs de A mais peut-être dans le désordre (multiensembles)

◇ Exercice **4** : Écrire un programme qui satisfait à la spécification suivante :

> *Objet du programme ?* *à compléter en langage naturel*
> *Données : (int N, int T[10])* , $0 \leq N \leq 10$,
> *Résultat : (int t[10])* , $\forall i : 0 \leq i < N \to t[i] = T[N-1-i]$

Dans cette spécification t représente le tableau T modifié.
La solution ne doit pas faire intervenir de tableau auxiliaire.

4 Exercice supplémentaire

◇ Exercice **5** : On dispose d'un fichier texte f.txt contenant un nombre inconnu de valeurs entières séparées par des caractères de contrôle. Écrire un programme qui place les entiers du fichier f.txt dans un tableau dynamique T.