Feuille de travaux dirigés n° 5
Pointeurs

Partie TD (5 séances)

Exercice 5.1 (Simulation)

Effectuer la simulation de l'algorithme suivant en donnant une représentation explicite de la mémoire adressée.

```
variables
   entier nb
   pointeur vers entier ptr
debut
1    nb ← 5
2    ptr ← adresse(nb)
3    memoire(ptr) ← 8
4    ecrire("nb (", adresse(nb), ") = ", nb)
5    ecrire("ptr (", adresse(ptr), ") = ", ptr, "->", memoire(ptr))
fin
```

Exercice 5.2 (Allocation statique)

Représenter le contenu de la mémoire adressée à la fin du traitement du lexique suivant :

```
type
   t_bool = enregistrement
        reel b1, b2, b3
   fin enregistrement

variables
   entiers i, j
   tableau de 5 caracteres tabc
   chaine c
   t_bool truc
   tableau de 3 t_bool table
```

Exercice 5.3 (Simulation)

Effectuer la simulation de l'algorithme suivant (déjà vu en cours) en donnant une représentation explicite de la mémoire adressée. L'utilisateur saisit la valeur 2.

```
fonction saisie_tableau(d n : entier) : pointeur vers tableau d'entiers
variables
   i, x : entiers
   ptab : pointeur vers tableau d'entiers
debut
1 ptab ← allocation (tableau de n entiers)
2 pour i de 1 a n faire
3    ecrire("veuillez saisir le " , i , "eme entier : ")
4    lire(x)
5    memoire(ptab)[i] <- x
6 fin pour
7 retourner ptab
```

```
fin
//-----------------------
// algorithme principal
variables
   nb, i : entier
   p : pointeur vers tableau d'entiers
debut
1 ecrire("combien d'entiers ?")
2 lire(nb)
3 p ← saisie_tableau(nb)
4 pour i de nb a 1 par pas de -1 faire
5    ecrire(memoire(p)[i])
6 fin pour
7 desallouer(p)
fin
```

Exercice 5.4 (Pointeurs et sous-algorithmes)

1. Écrire une procédure qui échange le contenu de deux entiers sans que ces deux paramètres soient passés en modification.
2. Écrire un algorithme qui utilise cette procédure pour échanger les valeurs de deux entiers donnés par l'utilisateur.
3. Simuler cet algorithme en représentant explicitement la mémoire adressée.

Exercice 5.5 (Allocation dynamique)

Effectuer la simulation de l'algorithme suivant en donnant une représentation explicite de la mémoire adressée.

```
variables
   booleen test
   pointeur vers booleen pb
debut
1  test ← vrai
2  pb ← allocation(booleen)
3  memoire(pb) ← faux
4  ecrire("test (", adresse(test), ") = ", test)
5  ecrire("pb (", adresse(pb), ") = ", pb, " -> ",  memoire(pb))
6  test ← memoire(pb)
7  desallouer(pb)
8  ecrire("test (", adresse(test), ") = ", test)
9  ecrire("pb (", adresse(pb), ") = ", pb, " -> ",  memoire(pb))

fin
```

Exercice 5.6 (Renverser)

Il s'agit d'écrire un algorithme qui demande à l'utilisateur le nom d'un fichier contenant des entiers puis les écrit, en ordre inverse dans un second fichier dont le nom est également saisi par l'utilisateur.
Effectuer l'analyse de ce problème puis énoncer un algorithme
Simuler ensuite son exécution sur le fichier "test.txt" = <12 ; 15 ; 82 ; 4>. Le nom du fichier inverse est : "test_inv.txt".

Exercice 5.7 (Vecteurs)

Il s'agit de manipuler un tableau de réels dont la taille peut être choisie par l'utilisateur en cours d'exécution. Ce tableau représente un vecteur.

1. Créer un type pour représenter un vecteur.
2. Écrire la fonction `taille_vecteur` prenant un vecteur en paramètre et retournant sa taille.
3. Écrire une fonction `saisie_vecteur` qui demande à l'utilisateur le nombre de réels qu'il veut saisir puis lui fait saisir ces réels puis et retourne le vecteurdans lequel ils sont mémorisés.

4. Écrire une procédure `affiche_vecteur` qui prend en paramètre un vecteur et l'affiche sur la sortie standard.
5. Écrire la procédure `detruire_vecteur` qui libère l'espace mémoire occupé par le vecteur passé en paramètre.
6. Écrire une fonction `copie_vecteur` qui prend en paramètres un entier $dimensiosn$ et un vecteur $vect$ et retourne un vecteur de taille $dimensions$ qui contient une copie des éléments de $vect$.
   Si $dimensions$ est supérieur à la taille de $vect$, le reste du vecteur copié sera initialisé à 0 ; sinon, seuls les $dimensions$ premiers éléments de $vect$ seront recopiés.
7. Écrire la fonction `chargement_vecteur` qui prend en paramètre le nom d'un fichier contenant des nombres réels et retourne le vecteur les contenant tous. Le vecteur retourné devra avoir juste la bonne taille.
8. Écrire la fonction `produit_scalaire` qui prend deux vecteurs en paramètres et retourne leur produit scalaire s'il peut être calculé.
   Rappel : le produit scalaire de $[u_1, \ldots, u_n]$ par $[v_1, \ldots, v_m]$ n'est défini que si $n = m$, $n \neq 0$ et $m \neq 0$. Il vaut alors $u_1 * v_1 + \ldots + u_n * v_n$.
9. Écrire un algorithme principal qui calcule le produit scalaire entre un vecteur saisi par l'utilisateur et une série de données contenues dans un fichier.

Exercice 5.8 (Matrices triangulaires)

Il s'agit de représenter et manipuler des matrices triangulaires supérieures (resp. inférieures) de réels, c'est-à-dire dont seuls les éléments au-dessus (resp. en dessous) de la diagonale principale sont non nuls.

1. Définir un type `t_matrice_triangulaire` permettant de stocker de telles matrices en limitant l'espace mémoire occupé ; ce type devra permettre de distinguer les matrices triangulaires supérieures des inférieures.
2. Définir une fonction qui prend en paramètre un entier nb et un booléen sup et qui retourne une nouvelle matrice triangulaire de taille $nb \times nb$, supérieure si $sup = vrai$, inférieure sinon.
3. Définir une fonction qui prend en paramètre deux entiers i et j et une matrice triangulaire mat et qui retourne l'élément en ligne i colonne j dans mat ; tous les cas particuliers devront être pris en compte.
4. Écrire une procédure qui libère l'espace mémoire occupé par une matrice triangulaire allouée dynamiquement et passée en paramètre.
5. Écrire une procédure qui affiche une matrice triangulaire supérieure (resp. inférieure) passée en paramètre en faisant apparaîtres des 0 pour les éléments au-dessous (resp. en dessus) de la diagonale principale.
   Par exemple, pour une matrice triangulaire supérieure de taille 3 :

$$\begin{matrix} 8 & 2 & 2 \\ 0 & 6 & 5 \\ 0 & 0 & 1 \end{matrix}$$

6. Écrire une procédure qui prend une matrice triangulairede taille $nb \times nb$ en paramètre et initialise ses éléments aux entiers de 1 à $nb * (nb + 1)/2$ ligne par ligne. Par exemple, pour $nb = 3$ et une matrice triangulaire supérieure :

$$\begin{matrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{matrix}$$

Partie TP (2 séances)

1 Usage des pointeurs

Télécharger le programme `pointeurs.cpp` sur madoc.

Le programme C++ `pointeurs.cpp` définit une procédure `affiche` prenant en paramètres deux entiers, a et b, et un pointeur vers un entier p_entier, tous passés par référence (c'est-à-dire en modification) ; cette procédure affiche les valeurs et adresses de ces paramètres et la valeur de l'entier pointé par p_entier [1]. Cette procédure est utilisée dans la fonction principale afin de suivre les modifications des valeurs des variables $val1$, $val2$ et ptr déclarées de type respectifs entier, entier et pointeur vers entier.

En C++ les opérations algorithmiques `adresse` et `mémoire` se notent $\&$ et $*$. Ainsi, à la ligne 21, ptr reçoit l'adresse de $val1$, et à la ligne 24 la case mémoire pointée par ptr est incrémentée d'une unité.

L'allocation dynamique qui, en algorithmique, se fait par l'opération `allocation`, se fait en C++ au moyen de l'opérateur `new` : à la ligne 26, un emplacement mémoire permettant le stockage d'un entier (`int` sur 4 octets) est réservé et son adresse est récupérée dans la variable ptr. Toute mémoire allouée dynamiquement doit être libérée dès qu'elle n'est plus utile et, en tout état de cause, avant la fin du programme. L'opérateur algorithme `desallouer` est traduit en C++ par l'opérateur `delete` : à la ligne 30 l'emplacement mémoire pointé par ptr est libéré.

En C++, il n'y a pas de distinction entre un pointeur sur un entier, sur un tableau d'entiers, ou encore sur un tableau de tableaux de ... tableaux d'entiers. En effet la valeur d'un tableau en C++ est son adresse qui est aussi l'adresse de son premier élément. Ainsi à la ligne 31 la variable ptr se voit affectée l'adresse d'un tableau de 5 entiers alloué dynamiquement. Pour accéder au premier élément de ce tableau, il faut écrire $*ptr$ ou $ptr[0]$. Au moment de libérer la donnée pointée par ptr, il faut penser à indiquer au compilateur qu'il s'agit d'un tableau et non d'un seul entier : l'opérateur (utilisé à la ligne 35) est alors `delete[]`.

Essayez de deviner ce qui sera affiché lors de chaque appel à `affiche`, puis exécutez le programme afin de valider vos hypothèses.

2 Transcription

Transcrire en programmes C++ des algorithmes de cette feuille.

1. Le passage des paramètres par référence est essentiel pour l'affichage des adresses, sinon ce seraient les adresses des paramètres formels et non celles des paramètres effectifs qui seraient affichées.