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
```

▽ **Correction**

Algorithme principal :

ligne	@1 nb	@2 ptr	remarques et affichages	mémoire
debut	?	?	allocation statique des variables	{@1 : ?, @2 : ?}
1	5	?		{@1 : 5, @2 : ?}
2	5	@1		{@1 : 5, @2 : @1}
3	8	@1		{@1 : 8, @2 : @1}
4	8	@1	aff : "nb (@1) = 8	{@1 : 8, @2 : @1}
5	8	@1	aff : "ptr (@2) = @1 -> 8	{@1 : 8, @2 : @1}
fin	8	@1	désallocation statique des variables	{}

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
```

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ; Le contenu de toutes les cases étant indéterminé, le plus intéressant est ici de signaler quelle variable occupe quelle case.

@1 i	@2 j	@3 t, tabc[1]	@4 tabc[2]	@5 tabc[3]
@6 tabc[4]	@7 tabc[5]	@8 c	@9 truc, truc.b1	@10 truc.b2
@11 truc.b3	@12 table, table[1], table[1].b1	@13 table[1].b2	@14 table[1].b3	@15 table[2], table[2].b1
@16 table[2].b2	@17 table[2].b3	@18 table[3], table[3].b1	@19 table[3].b2	@20 table[3].b3

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
```

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ; Algorithme principal :

ligne	@1 nb	@2 i	@3 p	remarques et affichages	mémoire
debut	?	?	?	allocation statique des variables	{@1 : ?, @2 : ?, @3 : ?}
1	?	?	?	Aff : combien d'entiers ?	{@1 : 2, @2 : ?, @3 : ?}
2	2	?	?		{@1 : 2, @2 : ?, @3 : ?}
3	2	?	@8		{@1 : 2, @2 : ?, @3 : @8, @8 : ?, @9 : ?}
4	2	2	@8	$i \geq 1 = 2 \geq 1$ = vrai	{@1 : 2, @2 : 2, @3 : @8, @8 : ?, @9 : ?}
5	2	2	@8	Aff : ?	{@1 : 2, @2 : 2, @3 : @8, @8 : ?, @9 : ?}
6->4	2	1	@8	$i \geq 1 = 1 \geq 1$ = vrai	{@1 : 2, @2 : 1, @3 : @8, @8 : ?, @9 : ?}
5	2	1	@8	Aff : ?	{@1 : 2, @2 : 1, @3 : @8, @8 : ?, @9 : ?}
6->4	2	0	@8	$i \geq 1 = 0 \geq 1$ = faux	{@1 : 2, @2 : 0, @3 : @8, @8 : ?, @9 : ?}
7	2	0	@8	désallocation de p	{@1 : 2, @2 : 0, @3 : @8}
fin	2	0	@8	désallocations statiques	{}

saisie_tableau(2) :

ligne	@4 n	@5 i	@6 x	@7 ptab	remarques et affichages	mémoire
debut	2	?	?	?	allocations statiques	{@4 : 2, @5 : ?, @6 : ?, @7 : ?}
1	2	?	?	@8	allocation dynamique	{@4 : 2, @5 : ?, @6 : ?, @7 : @8, @8 : ?, @9 : ?}
2	2	1	?	@8	(i $\leq$ n) = (1 $\leq$ 2) = vrai	{@4 : 2, @5 : 1, @6 : ?, @7 : @8, @8 : ?, @9 : ?}
3	2	1	?	@8	Aff : veuillez saisir le 1eme entier :	{@4 : 2, @5 : 1, @6 : ?, @7 : @8, @8 : ?, @9 : ?}
4	2	1	7	@8		{@4 : 2, @5 : 1, @6 : 7, @7 : @8, @8 : ?, @9 : ?}
5	2	1	7	@8		{@4 : 2, @5 : 1, @6 : 7, @7 : @8, @8 : 7, @9 : ?}
6->2	2	2	7	@8	(i $\leq$ n) = (2 $\leq$ 2) = vrai	{@4 : 2, @5 : 2, @6 : 7, @7 : @8, @8 : 7, @9 : ?}
3	2	2	7	@8	Aff : veuillez saisir le 2eme entier :	{@4 : 2, @5 : 2, @6 : ?, @7 : @8, @8 : ?, @9 : ?}
4	2	2	9	@8		{@4 : 2, @5 : 2, @6 : 9, @7 : @8, @8 : ?, @9 : ?}
5	2	2	9	@8		{@4 : 2, @5 : 2, @6 : 9, @7 : @8, @8 : 7, @9 : 9}
6->2	2	3	9	@8	(i $\leq$ n) = (3 $\leq$ 2) = faux	{@4 : 2, @5 : 2, @6 : 7, @7 : @8, @8 : 7, @9 : 9}
7	2	3	9	@8		{@4 : 2, @5 : 2, @6 : 7, @7 : @8, @8 : 7, @9 : 9}
fin	2	3	9	@8	désallocations statiques	{@8 : 7, @9 : 9}

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.

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ;

```
fonction echange(pointeur vers entier p1, pointeur vers entier p2)
variable
  entier tmp
debut
1 tmp ← memoire(p1)
2 memoire(p1) ← memoire(p2)
3 memoire(p2) ← tmp
fin

variables
  entier nb1, nb2
debut
1 ecrire("Donnez un entier : ")
2 lire(nb1)
3 ecrire("Donnez un autre entier : ")
4 lire(nb2)
5 echange(adresse(nb1),adresse(nb2))
6 ecrire("nb1 = , nb1, ", nb2 = ", nb2)
fin
```

ligne	@1 nb1	@2 nb2	Remarques	Mémoire
debut	?	?	Allocation des variables	{@1 : ?, @2 : ?}
1	?	?	Affichage : "Donnez un entier :"	{@1 : ?, @2 : ?}
2	2	?		Mémoire : {@1 :2, @2 : ?}
3	2	?	Affichage : "Donnez un autre entier :"	{@1 :2, @2 : ?}
4	2	5		{@1 :2, @2 :5}
5	5	2	echange(adresse(nb1),adresse(nb2)) $\Leftrightarrow$ echange(@1,@2)	{@1 :5, @2 :2}
6	5	2	Affichage : : nb1 = 5, nb2 = 2	{@1 :5, @2 :2}
fin	5	2	Désallocation des variables	

echange(adresse(nb1),adresse(nb2) :

ligne	@3 p1	@4 p2	@5 tmp	Remarques	Mémoire
debut	@1	@2	?	Allocations	{@1 :2, @2 :5, @3 :@1, @4 :@2, @5 : ?}
1	@1	@2	2		{@1 :2, @2 :5, @3 :@1, @4 :@2, @5 :2}
2	@1	@2	2		{@1 :5, @2 :5, @3 :@1, @4 :@2, @5 :2}
3	@1	@2	2		{@1 :5, @2 :2, @3 :@1, @4 :@2, @5 :2}
fin	@1	@2	2	Désallocations	

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
```

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ;

Ligne	@1 test	@2 pb	Remarques	Mémoire
debut	?	?	allocations statiques	{@1 : ?, @2 : ?}
1	vrai	?		{@1 : vrai ,@2 : ?}
2	vrai	@3	allocation dynamique	{@1 : vrai, @2 : @3, @3 : ?}
3	vrai	@3		{@1 : vrai, @2 : @3, @3 : faux}
4	vrai	@3	Aff : "test (@1) = vrai	{@1 : vrai, @2 : @3, @3 : faux}
5	vrai	@3	aff : "pb (@2) = @3 -> faux"	{@1 : vrai, @2 : @3, @3 : faux}
6	faux	@3		{@1 : faux, @2 : @3, @3 : faux}
7	faux	@3	désallocation dynamique	{@1 : faux, @2 : @3}
8	faux	@3	Aff : "test (@1) = vrai	{@1 : vrai, @2 : @3}
9	faux	@3	Aff : aff : "pb (@2) = @3 ->	{@1 : vrai, @2 : @3}

L'exécution de la ligne 9 provoque une violation de la mémoire puisque l'adresse @3 n'est plus allouée. L'exécution est donc interrompue.

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".

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ;

Il faut mémoriser, dans un tableau (nommé $stocke$), les entiers lus pour ensuite les écrire en ordre inverse dans un fichier.
Le nombre de données dans le fichier de données étant a priori inconnu, il sera nécessaire d'allouer dynamiquement le tableau.
Il est donc possible de définir le type T_tableau

```
type T_tableau = enregistrement
    entier taille
    pointeur vers tableau de entier ptab
fin enregistrement
```

L'analyse détermine plusieurs grandes étapes :

- 1 - choix du nom du fichier de données => variable nom (chaîne de caractères)
- 2 - détermination de la valeur de nb, le nombre de valeurs dans le fichier nom
- 3 - allocation d'une variable stockage de type T_Tableau de type nb
- 4 - lecture des valeurs contenues dans le fichier non et mémorisation de ces valeurs dans $vecteur$
- 5 - choix du nom du fichier inverse => variable nom_inv (chaîne de caractères)
- 6 - sauvegarde en ordre inverse des valeurs de $vecteur$ dans le fichier nom_inv

2 - déterminer nb, le nombre de valeurs dans le fichier nom

- ouverture du fichier de données en lecture => variable fic (flux_fichier)
- répétitive qui lit les valeurs (=> $donnee$ (entier)) et met à jour nb (entier, initialisé à 0)
- fermeture de fic

```
lexique
   chaine nom, nom_inv
   entier i, nb, num
   flux\_fichier fic
   T tableau stock
debut
   nb  ← 0
   ecrire("Donner le nom d'un fichier : ")
   lire(nom)

   ouvrir(fic, nom) en lecture
   si (fic) alors
      // compter le nombre de donnees dans le fichier
      lire(num) dans fic
      tant que non fini(fic) faire
         nb ← nb + 1
         lire(num) dans fic
      fin tant que
      fermer fic

      // allocation de stock
      stock.taille  ← nb
      stock.ptab  ← allocation (tableau de nb entier)

      // relecture du fichier pour memoriser chaque donnee
      ouvrir(fic, nom) en lecture
      i  ← 1
      lire(num) dans fic
      tant que non fini(fic) faire
         memoire(stock.ptab)[i] ← num
        i  ← i + 1
         lire(num) dans fic
```

```
        fin tant que
        fermer fic

        // ecriture en ordre inverse dans le second fichier
        ouvrir(fic, nom_inv) en ecriture
        pour i de nb a 1 par pas de -1 faire
            ecrire( memoire(stock.ptab)[i] , " ") dans fic
        fin pour
        fermer(fic)
    sinon
        ecrire("Le fichier ", nom, " n'existe pas.")
    fin si
fin
```

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.

▽ Correction

Exercice pouvant être soumis pour l'évaluation ; Limiter à une ou deux questions

1. Il faut donc créer un type qui mémorise à la fois la taille du tableau et le tableau. Mais, comme la taille du tableau n'est pas connue avant l'exécution du programme, il faut mémoriser un pointeur vers un tableau qui sera alloué dynamiquement en cours d'exécution

```
type T_vecteur = enregistrement
    entier taille
    pointeur vers tableau de reel ptab
fin enregistrement
```

2.
```
//-------------------------------------------------
fonction taille_vecteur(d T_vecteur vect) : entier
debut
    retourner(vect.taille)
fin
```

3. la fonction retourne un pointeur vers t_vecteur

```
//-------------------------------------------------
// Precondition : {nb > 0}
fonction saisie_vecteur() : pointeur vers T_vecteur
variables
   entier i
   entier nb
   T_vecteur vect
   reel val
debut
  ecrire("Combien de reels voulez-vous memoriser ?  ")
  lire(nb)
   vect.taille ← nb
   vect.ptab ← allocation(tableau de nb reels)
   pour i de 1 a nb faire
     ecrire("Reel ?  ")
     lire(val)
     memoire(vect.ptab)[i] ← val
   fin pour
   retourner vect
fin
```

▽ Correction

Faire une simulation d'un appel de cette fonction pour que les étudiants comprennent bien ce qui se passe. Commencer à numéroter les adresses à partir de @10 (nombre arbitraire) pour montrer que des allocations mémoires ont déjà eu lieu avant cet appel.
saisie_vecteur(3) :

Ligne	@10 nb	@11 i	@12 vect.taille	@13 vect.ptab	@14 val	Remarques	Mémoire
debut	3	?	?	?	?	alloc. stat.	{@10 : ?, @11 : ?, @12 : ?, @13 : ?, @14 : ?}
1	?	?	?	?	?	affichage	{@10 : ?, @11 : ?, @12 : ?, @13 : ?, @14 : ?}
2	3	?	?	?	?		{@10 :3, @11 : ?, @12 : ?, @13 : ?, @14 : ?}
3	3	?	3	?	?		{@10 :3, @11 : ?, @12 :3, @13 : ?, , @14 : ?,
4	3	?	3	@15	?	alloc. dynamique	{@10 :3, @11 : ?, @12 :3 , @13 :@15, , @14 : ?, {@15 : ?, @16 : ?, @17 : ?}
5	3	1	3	@15	?	(i≤nb)=(1≤3)=V	{@10 :3, @11 :1, @12 : 3, @13 : @15, @14 : ?, {@15 : ?, @16 : ?, @17 : ?}
6	3	1	3	@15	?	aff : "Reels ?"	{@10 :3, @11 :1, @12 : 3, @13 : @15, @14 : ?, {@15 : ?, @16 : ?, @17 : ?}
7	3	1	3	@15	7		{@10 :3, @11 :1, @12 : 3, @13 : @15 :, @14 :7, {@15 : ?, @16 : ?, @17 : ?}
8	3	1	3	@15	7		{@10 :3, @11 :1, @12 : 3, @13 : @15 :, @14 :7, {@15 :7, @16 : ?, @17 : ?}
9->5	3	2	3	@14	7	(i≤nb)=(2≤3)=V	{@10 :3, @11 :2, @12 : 3, @13 : @15, @14 :7, {@15 :7, @16 : ?, @17 : ?}
6	3	2	3	@15	7	aff : "Reels ?"	{@10 :3, @11 :2, @12 : 3, @13 : @15, @14 :7, {@15 :7, @16 : ?, @17 : ?}
7	3	2	3	@15	4		{@10 :3, @11 :2, @12 : 3, @13 : @15, @14 : :4, {@15 :7, @16 : ?, @17 : ?}
8	3	2	3	@15	4		{@10 :3, @11 :2,@12 : 3, @13 : @15, @14 :4, {@15 :7, @16 :4, @17 : ?}
9->5	3	3	3	@14	4	(i≤nb)=(3≤3)=V	{@10 :3, @11 :3, @12 : 3, @13 : @15, @14 :4, {@15 :7, @16 :4, @17 : ?}
6	3	3	3	@15	4	aff : "Reels ?"	{@10 :3, @11 :3,@12 : 3, @13 : @15, @14 :4, {@15 :7, @16 : ?, @17 : ?}
7	3	3	3	@15	5		{@10 :3, @11 :3, @12 : 3, @13 : @15 :, @14 :5, {@15 :7, @16 : ?, @17 : ?}
8	3	3	3	@15	5		{@10 :3, @11 :3, @12 : 3, @13 : @15, @14 :5, {@15 :7, @16 :4, @17 :5}
9->5	3	4	3	@15	5	(i≤nb)=(4≤3)=F	{@10 :3, @11 :4,@12 : 3, @13 : @15, @13 :5, @14 :5, {@15 :7, @16 :4, @17 :5}
10	3	4	3	@15	5	désalloc. stat.	{ @15 :7, @16 :4, @17 :5}

Q 4

```
//-----------------------------------------------
procedure affiche_vecteur(d T_vecteur vect)
variables
   entier i, nb
   reel val
debut
   nb ← vect.taille
   ecrire("(")
   pour i de 1 a nb-1 faire
      x ← memoire(vect.ptab)[i]
      ecrire(val, ", ")
   fin pour
   val ← memoire(vect.ptab)[nb]
   ecrire(val, ")")
fin
```

Q 5

Il faut désallouer le tableau.

```
//-----------------------------------------------
procedure desallocation_vecteur(d T_vecteur p_vect)
debut
   desallouer(p_vect.ptab)
fin
```

▽ **Correction**

Q 6

```
//-----------------------------------------------
fonction copie_vecteur(d entier dimension, d T_vecteur vect)
                 :  T_vecteur
variables
   entier  i, nb
   T_vecteur vect_res
debut
   si taille_vecteur(vect) < dimension alors
      nb ← taille_vecteur(v)
   sinon
      nb ← dimension
   fin si
   vect_res.taille  ← nb
   vect_res.ptab  ← allocation(tableau de nb reel)
   pour i de 1  nb faire
      memoire(vect_res.ptab)[i] ← memoire(vect.ptab)[i]
   fin pour
   pour i de nb+1   dimension faire
      memoire(vect_res.ptab)[i] ← 0
   fin pour
   retourner vect_res
fin
```

Q 7

```
//-----------------------------------------------
fonction chargement_vecteur(d chaine nom)
                 :  T_vecteur
variables
```

```
    entier nb
    reel val
    T_vecteur vect_res
    flux_fichier fic
debut
    // premiere lecture pour determiner la taille du vecteur,
    nb ← 0
    ouvrir(fic, nom) en lecture
    lire(val) dans fic
    tant que non fini(fic) faire
       nb ← nb +1
       lire(val) dans fic
    fin tant que
    fermer(fic)
    vect_res.taille  ← nb
    si (nb > 0)
    // allocation d'un tableau de la bonne taille
      vect_res.ptab  ← allocation (tableau de nb reel)
      // copie du contenu du fichier
      ouvrir(fic, nom) en lecture
      lire(val) dans fic
      tant que non fini(fic) faire
         memoire(vect_res.ptab)[i] ← val
         lire(val) dans fic
      fin tant que
      fermer(fic)
    fin si
    retourner vect_res
fin
```

Q 8

```
//-----------------------------------------------
// Precondition : la taille des vecteurs n'est pas nulle. Les vecteurs ont des tableaux de
fonction produit_scalaire(d T_vecteur vect1, d T_vecteur vect2)
                 :  reel
variables
    reel pdt
    entier i
debut
     pdt ← 0
     pour i de 1 a  taille_vecteur(vect1) faire
         pdt ← pdt + memoire(vect1.tab)[i] * memoire(vect2.tab)[i]
     fin pour
     retourner pdt
fin
```

Q 9

```
//-----------------------------------------------
variables
    T_vecteur vect1, vect2, vect2copie
    chaine nom
debut
    vect1 ← saisie_vecteur() // alloc. vecteur
    ecrire("Donnez le nom d'un fichier contenant des reels : ")
    lire(nom)
    vect2 ← chargement_vecteur(nom) // alloc. vecteur
    si (non(taille_vecteur(vect1) = taille_vecteur(vect2)) alors
    // tronquer ou completer vect2 (ou vect1, le sujet ne le specifie pas)
       vect2copie ← copie_vecteur(taille_vecteur(vect1), vect2) // alloc. vecteur
    fin si
```

```
    ecrire("Produit scalaire = ", produit_scalaire(vect1, vect2copie))
    desallocation_vecteur(pv1)
    desallocation_vecteur(pv2)
    desallocation_vecteur(pv2copie)
fin
```

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 :

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

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ; l'exercice étant long et difficile, n'en soumettre qu'une partie et seulement s'il a été traité en TD, voire en rappelant le type MatriceTriangulaire pour leur permettre d'écrire les routines demandées

1. L'idée est de ne stocker que les éléments significatifs, donc d'avoir des lignes de longueurs variables (décroissantes si matrice triangulaire supérieure, croissantes dans le cas contraire). Pour ce faire il faut un tableau de pointeurs sur des tableaux, chacun étant alloué dynamiquement d'une taille calculée selon le niveau de la ligne (et le type de matrice).

```
type
  p_vecteur = pointeur vers tableau de reels

type
   T_matrice_triangulaire =   enregistrement
                        booleen sup // vrai ssi triangulaire superieure
                        entier taille
                        pointeur vers tableau de p_vecteur p_table
                    fin enregistrement
```

2. Il faut définir une l'allocation mémoire nécessaire à une matrice triangulaire de taille nb

```
//-------------------------------------------------------------------------------------------
fonction allocation_matrice(d entier nb, d entier sup)
: T_matrice_triangulaire
variables
   T_matrice_triangulaire mat
debut
   mat.sup ← sup
   mat.taille ← nb
   // allocation des nb lignes
   mat.p_table ←
                  allocation(tableau de nb pointeurs vers tableau de p_vecteur)
   // allocation des nb colonnes
   si (sup) alors
   // matrice triangulaire superieure : la ligne i contient nb-i+1 colonnes
      pour i de 1  a nb faire
         mat.p_table[i] = allocation(tableau de nb-i+1 reels)
      fin pour
   sinon
   // matrice triangulaire inferieure : la ligne i contient i colonnes
      pour i de 1 a nb faire
         mat.p_table[i] = allocation(tableau de i reels)
      fin pour
   fin si
   retourner mat
fin
```

3.

```
//-------------------------------------------------------------------------------------------
// acces a l'element de coordonnees [i, j]
// precondition : (i ∈ [1, mat.taille]) et (j ∈ [1, mat.taille])
fonction acces_matrice(d entier i, d entier j, d T_matrice_triangulaire mat) : reel
variable
   reel res
debut
      si (((mat.sup = vrai) et (j < i)) ou ((mat.sup = faux) et (j > i))) alors
         // c'est l'un des elements nuls non stockes qui est demande
         res ← 0
      sinon
         si (mat.sup = faux) alors
            res ← memoire(memoire(mat.p_table)[i])[j]
         sinon
            res ← memoire(memoire(mat.p_table)[i])[j -
memoire(p_mat).taille + i]
         fin si
      fin si
   retourner res
fin
```

4.

```
//-------------------------------------------------------------------------------------------
procedure liberer_matrice(pointeur vers T_matrice_triangulaire p_mat)
variable
   entier i
debut
   // la liberation doit se faire en ordre inverse de l'allocation
   // d'abord les colonnes ...
   pour i de 0 a (memoire(p_mat).taille - 1) faire
      desallouer(memoire(memoire(p_mat).p_table)[i])
   fin pour
   // ... puis les lignes ...
   desallouer(memoire(p_mat).p_table)
   // ... et enfin la matrice
```

```
    desallouer(p_mat)
fin
```

5.
```
//-------------------------------------------------------------------------------
// affichage de la matrice
procedure affiche_matrice(d T_matrice_triangulaire mat)
variable
    entier i, j
debut
    pour i de 1 a mat.taille faire
        pour j de 1 a mat.taille faire
            res ← acces_matrice(i, j, mat)
            ecrire (res, "  ")
        fin pour
        ecrire (nl) // passage a la ligne
    fin pour
fin
```

6.
```
//-------------------------------------------------------------------------------
// initialisation de la matrice
procedure init_matrice(d T_matrice_triangulaire mat)
variable
    entier i, j
    reel valeur
debut
    valeur ← 1
    pour i de 1 a mat.taille faire
        pour j de 1 a mat.taille faire
            si (((mat.sup = vrai) et (j < i)) ou ((mat.sup = faux) et (j > i))) alors
                memoire(memoire(mat.p_table)[i])[j] ← 0
            sinon
                memoire(memoire(mat.p_table)[i])[j] ← valeur
            fin si
            valeur ← valeur + 1
        fin pour
    fin pour
fin
```

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.