Listes, Piles, Files
Implémentations en C

Dynamique (liste chaînée)
et
Statique (en tableau)

Renaud VÉRIN

Structures chaînées

On désire implémenter des listes, files et piles dynamiques d'entiers à l'aide de pointeurs.

1) Définir *Liste*, le type de la structure de données de base de gestion de ces structures dynamiques et *PtrListe*, le type « pointeur sur Liste ».
   Déclarer les variables globales *liste, pile et tete et queue* (pour la file).

2) Soit la définition suivante :
```
#define MALLOC(x)((x * ) malloc(sizeof(x)))
```
   Écrire les fonctions utiles de prototypes suivants :
```
PtrListe creerNoeud(int x, PtrListe suiv);
void AffListe(PtrListe p);
int longueur(PtrListe p);
```

3) Écrire les fonctions pour empiler et dépiler :
```
PtrListe Empile(PtrListe pile, int x);
PtrListe Depile(PtrListe pile, int *x );
```

4) Écrire les fonctions pour enfiler et défiler :
```
void Enfile(PtrListe *tete, PtrListe *queue, int x);
int Defile(PtrListe *tete, PtrListe *queue );
```

5) Écrire les fonctions pour insérer en ordre, supprimer et rechercher un élément x dans la liste chaînée triée :
```
PtrListe AjoutListe(PtrListe liste, int x);
PtrListe SuppListe(PtrListe liste, int x);
int RechListe(PtrListe liste, int x);
```

6) Écrire une fonction principale `main` pour tester toutes ces fonctions.

Structure dynamique

1) Structures et variables globales :

```c
#include <stdio.h>
#include <stdlib.h>

#define MALLOC(x)((x * ) malloc(sizeof(x)))

/*
 * Definition du type liste
 */
typedef struct lst {
    int val;
    struct lst * suiv;
} Liste;
typedef Liste * PtrListe;

/*
 * Variables globales des structures chainees
*/
PtrListe pile = NULL, lst = NULL,   // lst= liste chainee triee
         tete = NULL, queue = NULL; // pour la file
```

Fonctions utiles

2) Créer un nœud, afficher la liste, longueur de la liste.

```
/* Creation d'un noeud de valeur x et de suivant suiv, retourne le noeud cree
PtrListe creerNoeud(int x, PtrListe suiv) {
    PtrListe ptr;
    if ((ptr = MALLOC(Liste)) == NULL) {
        fprintf(stderr, "ERREUR ALLOCATION MEMOIRE");
        exit(1);
    }
    ptr->val = x;
    ptr->suiv = suiv;
    return ptr;
}

// Affichage d'une liste quelconque
void AffListe(PtrListe liste) {
    if (!liste)
        puts("Liste vide");
    else {
        while (liste) {
            printf("%d,", liste->val);
            liste = liste->suiv;
        }
        puts("");
    }
}

// Longueur d'une liste quelconque
int longueur(PtrListe liste) {
    int i = 0;
    while (liste) i++;
    return i;
}
```

Pile dynamique

3) Empiler et dépiler.

```
// Empilement : retourne la pile
PtrListe Empile(PtrListe pile, int x) {
    PtrListe ptr = creerNoeud(x, pile);
    return ptr;
}

// Depilement : retourne la pile et la valeur dans x passee par adresse
PtrListe Depile(PtrListe pile, int *x) {
    PtrListe ptr;
    if (!pile) return NULL;
    ptr = pile;
    *x = ptr->val;
    pile = pile->suiv;  // la tete de pile passe au suivant
    free(ptr);          // liberation de la tete de pile
    return pile;
}
```

Test de pile dynamique

6) Fonction main.

```
int main() {

  int x;

  pile = Empile(pile, 2);  AffListe(pile);
  pile = Empile(pile, 1);  AffListe(pile);
  pile = Empile(pile, 3);  AffListe(pile);

  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);

}
```

```
Résultat:
2,
1,2,
3,1,2,
x =  3 Pile = 1,2,
x =  1 Pile = 2,
x =  2 Pile = Liste vide
x =  2 Pile = Liste vide
```

File dynamique

4) Enfiler et défiler.

```
// Enfilement : on insere l'element,
// tete et queue sont modifiees donc passees par adresse (PrtListe *)
void Enfile(PtrListe *tete, PtrListe *queue, int x) {
    PtrListe ptr = creerNoeud(x, NULL);
    if (*queue) (*queue)->suiv = ptr;   // queue pointe sur le nouvel element
    else *tete = ptr;                   // si file vide = nouvel element devient la tete
    *queue = ptr;                       // nouvel element = nouvelle queue
}

// Defilement : tete et queue sont modifiees donc passees par adresse (PrtListe *)
int Defile(PtrListe *tete, PtrListe *queue) {
    PtrListe ptr = *tete;
    int x;
    if (!ptr) return -1;              // File vide
    *tete = ptr->suiv;                // tete passe au suivant
    if (*queue == ptr) *queue = NULL; // File devient NULL tete = queue
    x = ptr->val;                     // on recupere la valeur
    free(ptr);                        // on libere l'element
    return x;
}
```

Test de file dynamique

6) Fonction main.

```
int main() {

  Enfile( &tete, &queue, 2);     AffListe(tete);
  Enfile( &tete, &queue, 1);     AffListe(tete);
  Enfile( &tete, &queue, 3);     AffListe(tete);

  printf("Defile : %d File = ",  Defile( &tete, &queue));  AffListe(tete);
  printf("Defile : %d File = ",  Defile( &tete, &queue));  AffListe(tete);
  printf("Defile : %d File = ",  Defile( &tete, &queue));  AffListe(tete);
  printf("Defile : %d File = ",  Defile( &tete, &queue));  AffListe(tete);

}
```

```
Résultat :
2,
2,1,
2,1,3,
Defile : 2 File = 1,3,
Defile : 1 File = 3,
Defile : 3 File = Liste vide
Defile : -1 File = Liste vide
```

Ajout Liste dynamique

5) Ajouter dans la liste.

```
// Ajout trie dans la liste : renvoi de liste
PtrListe AjoutListe(PtrListe liste, int x) {
    PtrListe ptr = creerNoeud(x, NULL);
    PtrListe p = liste;
    if (!p) return ptr;     // Liste vide ==> ptr devient la liste
    if (p->val > x) {       // x plus petit => insere en tete
        ptr->suiv = p;
        return ptr;
    }
    while (p->suiv && p->suiv->val < x) p = p->suiv; // Recherche de la place de x
    ptr->suiv = p->suiv;   // chainage entre p et son suivant
    p->suiv = ptr;
    return liste;
}
```

Suppression Liste dynamique

5) Supprimer et rechercher dans la liste.

```
PtrListe SuppListe(PtrListe liste, int x) {
    PtrListe ptr;
    PtrListe p = liste;
    if (!p) return NULL;    // liste vide
    if (p->val == x) {      // suppession de la tete
        ptr = p->suiv;
        free(p);
        return ptr;
    }
    while (p->suiv && p->suiv->val != x) p = p->suiv;
    if (p->suiv) {    // Suppression du chainon de valeur x
        ptr = p->suiv;
        p->suiv = ptr->suiv;
        free(ptr);
    }
    return liste;
}

// Recherche de x dans la liste : renvoi de la valeur ou -1
int RechListe(PtrListe liste, int x) {
  PtrListe p = liste;
  while (p && p->val != x) p = p->suiv;
  if (p)
    return p->val;
  return -1;
}
```

Test de Liste dynamique

6) Fonction main.

```
int main() {

  liste = AjoutListe(liste, 6); AffListe(liste);
  liste = AjoutListe(liste, 4); AffListe(liste);
  liste = AjoutListe(liste, 5); AffListe(liste);
  liste = AjoutListe(liste, 9); AffListe(liste);

  printf("Recherche de %d Retour = %d\n", 4, RechListe( liste, 4));
  printf("Recherche de %d Retour = %d\n", 6, RechListe( liste, 6));
  printf("Recherche de %d Retour = %d\n", 9, RechListe( liste, 9));
  printf("Recherche de %d Retour = %d\n", 2, RechListe( liste, 2));

  liste = SuppListe(liste, 4); AffListe(liste);
  liste = SuppListe(liste, 6); AffListe(liste);
  liste = SuppListe(liste, 9); AffListe(liste);
  liste = SuppListe(liste, 5); AffListe(liste);
  liste = SuppListe(liste, 5); AffListe(liste);

}
```

```
Résultat :
6,
4,6,
4,5,6,
4,5,6,9,
Recherche de 4 Retour = 4
Recherche de 6 Retour = 6
Recherche de 9 Retour = 9
Recherche de 2 Retour = -1
5,6,9,
5,9,
5,
Liste vide
Liste vide
```

Structures statiques (1)

On désire implémenter des listes, files et piles statiques d'entiers à l'aide de tableaux.

1) Définir les types structurés *TypePile, TypeFile et TypeListe*, contenant le tableau de données et les champs utiles à leur gestion.
   Déclarer les variables globales *liste, pile et tete et queue* (pour la file).
2) Soient les définitions suivantes :

```
#define N 5          // Taille du tableau
#define TRUE  1
#define FALSE 0
```

3) Écrire les fonctions de gestion de pile :

```
void pileInit(TypePile *pile);
int pileVide(TypePile pile);
int pilePleine(TypePile pile);
int empile(TypePile *pile, int x);
int depile(TypePile *pile, int *x);
void pileAff(TypePile pile);
```

Structures statiques (2)

4) Écrire les fonctions de gestion de file :

```
void fileInit(TypeFile *file);
int fileVide(TypeFile file);
int filePleine(TypeFile file);
int enfile(TypeFile *file, int x);
int defile(TypeFile *file, int *x);
void fileAff(TypeFile file);
```

5) Écrire les fonctions de gestion de la liste chaînée triée :

```
void listeInit(TypeListeChainee *liste);
int listeVide(TypeListeChainee liste);
int listePleine(TypeListeChainee liste);
int listeInsere(TypeListeChainee *liste, int x);
int listeSupp(TypeListeChainee *liste, int x);
void listeAff(TypeListeChainee liste);
```

6) Écrire une fonction principale `main` pour tester toutes ces fonctions.

Définition des structures statiques

1) Structures et variables globales :

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

typedef struct {
    int tabPile[N];
    int tete;           // index de tete de pile
    } TypePile;

typedef struct {
    int tabFile[N];
    int tete, queue;    // Insertion en queue et defilement en tete
    int vide;           // Indicateur de file vide
    } TypeFile;

typedef struct {
    int tabListe[N];
    int indexListe[N];  // Tableau des index des elements tries
    int nbelt;          // = indice du prochain element libre
    } TypeListe;

TypePile pile;
TypeFile file;
TypeListe liste;
```

Pile statique (1)

3) Initiation, tests, empiler, dépiler et afficher.

```
void pileInit(TypePile *pile) {
    int i;
    for (i=0; i<N; i++) pile->tabPile[i] = -1;
    pile->tete = -1;
}

int pileVide(TypePile pile) {
     return pile.tete == -1;
}

int pilePleine(TypePile pile) {
     return pile.tete == N-1;
}

int empile(TypePile *pile, int x) {
    if (! pilePleine(*pile) ) {
        pile->tete++;
        pile->tabPile[pile->tete] = x;
        return TRUE;
    }
    return FALSE;
}
```

Pile statique (2)

3) Initiation, tests, empiler, dépiler et afficher.

```
int depile(TypePile *pile, int *x) {
    if (! pileVide(*pile) ) {
        *x = pile->tabPile[pile->tete];
        pile->tabPile[pile->tete] = -1;
        pile->tete--;
        return TRUE;
    }
    return FALSE;
}

void pileAff(TypePile pile) {
    int i;
    for (i=0; i<N; i++)
        if (pile.tabPile[i] != -1) printf("%3d", pile.tabPile[i]);
        else printf("  -");
    printf("\t\t: %d", pile.tete);
    puts("");
}
```

Test de pile statique

6) Fonction main.

```
int main() {

  int x;

  pile = Empile(pile, 2);  AffListe(pile);
  pile = Empile(pile, 1);  AffListe(pile);
  pile = Empile(pile, 3);  AffListe(pile);

  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
  pile = Depile(pile, &x); printf("x =  %d Pile = ", x);  AffListe(pile);
}
```

```
Résultat:
2,
1,2,
3,1,2,
x =  3 Pile = 1,2,
x =  1 Pile = 2,
x =  2 Pile = Liste vide
x =  2 Pile = Liste vide
```

File statique (1)

4) Initiation, tests, enfiler, défiler et afficher.

```
void fileInit(TypeFile *file) {
    int i;
    for (i=0; i<N; i++) file->tabFile[i] = -1;
    file->tete = 0;   // condition pour la file vide
    file->queue = -1; // pointe sur element de queue, passe a 0 sur la 1ere insertion
    file->vide = TRUE;
}

int fileVide(TypeFile file) { return file.vide; }

int filePleine(TypeFile file) {
     return !file.vide && (file.tete) % N == (file.queue + 1) % N;
}

// renvoie VRAI si l'operation a reussi et FAUX sinon
int enfile(TypeFile *file, int x) {
    if (filePleine(*file)) return FALSE;
    file->vide = FALSE;
    file->queue = (file->queue + 1) % N; // decalage de la queue modulo N
    file->tabFile[file->queue] = x;
    return TRUE;
}
```

File statique (2)

4) Initiation, tests, enfiler, défiler et afficher.

```c
// renvoie VRAI si l'operation a reussi et FAUX sinon
int defile(TypeFile *file, int *x) {
    if (fileVide(*file)) return FALSE;
    *x = file->tabFile[file->tete];  // depilement de x non vide
    file->tabFile[file->tete] = -1;  // vidage de la tete
    file->tete = (file->tete+1) % N; // decalage de la tete modulo la taille du tableau
    if ((file->tete % N) == ((file->queue + 1) % N)) file->vide = TRUE;
    return TRUE;
}

void fileAff(TypeFile file) {
    int i;
    for (i=0; i<N; i++)
        if (file.tabFile[i] != -1) printf("%3d", file.tabFile[i]);
        else printf("  -");
    printf("\t\t: %d : %d : %d", file.tete, file.queue, file.vide);
    puts("");
}
```

Test de file statique (1)

6) Fonction main.

```
int main() {
    int i;
    fileInit(&file);
    fileAff(file);
    for (i=0; i<6; i++) if (!enfile(&file,i)) puts("FILE PLEINE"); else fileAff(file);
    for (i=0; i<6; i++) if (!defile(&file,&i)) puts("FILE VIDE"); else fileAff(file);
    for (i=0; i<3; i++) if (!enfile(&file,i)) puts("FILE PLEINE"); else fileAff(file);
    for (i=0; i<4; i++) if (!defile(&file,&i)) puts("FILE VIDE"); else fileAff(file);
    for (i=0; i<6; i++) if (!enfile(&file,i)) puts("FILE PLEINE"); else fileAff(file);
    puts("");
}
```

```
Résultat :
-  -  -  -  -        : 0 : -1 : 1
0  -  -  -  -        : 0 : 0 : 0
0  1  -  -  -        : 0 : 1 : 0
0  1  2  -  -        : 0 : 2 : 0
0  1  2  3  -        : 0 : 3 : 0
0  1  2  3  4        : 0 : 4 : 0
FILE PLEINE
-  1  2  3  4        : 1 : 4 : 0
-  -  2  3  4        : 2 : 4 : 0
-  -  -  3  4        : 3 : 4 : 0
-  -  -  -  4        : 4 : 4 : 0
-  -  -  -  -        : 0 : 4 : 1
FILE VIDE
0  -  -  -  -        : 0 : 0 : 0
0  1  -  -  -        : 0 : 1 : 0
0  1  2  -  -        : 0 : 2 : 0
-  1  2  -  -        : 1 : 2 : 0
-  -  2  -  -        : 2 : 2 : 0
-  -  -  -  -        : 3 : 2 : 1
FILE VIDE
-  -  -  0  -        : 3 : 3 : 0
-  -  -  0  1        : 3 : 4 : 0
2  -  -  0  1        : 3 : 0 : 0
2  3  -  0  1        : 3 : 1 : 0
2  3  4  0  1        : 3 : 2 : 0
FILE PLEINE
```

Liste statique

5) Initialisation , tests et affichage de la liste.

```
void listeInit(TypeListe *liste) {
    int i;
    for (i=0; i<N; i++) {
        liste->tabListe[i] = -1;
        liste->indexListe[i] = -1;
    }
    liste->nbelt = 0;
}

int listeVide(TypeListe liste) { return (liste.nbelt == 0); }

int listePleine(TypeListe liste) { return (liste.nbelt == N); }

void listeAff(TypeListe liste) {
    int i;
    printf("Indices\t: ");
    for (i=0; i<N; i++) printf("%3d",i);
    printf("\nValeur\t: ");
    for (i=0; i<N; i++)
        if (liste.tabListe[i] != -1) printf("%3d", liste.tabListe[i]);
        else printf("  -");
    printf("\nIndex\t: ");
    for (i=0; i<N; i++)
        if (liste.indexListe[i] != -1) printf("%3d", liste.indexListe[i]);
        else printf("  -");
    puts("\n");
}
```

Insertion Liste statique

5) Insertion dans la liste.

```
int listeInsere(TypeListe *liste, int x) {
    int i;
    if (listePleine(*liste)) return FALSE;
    liste->tabListe[liste->nbelt] = x;
    i = liste->nbelt - 1; // Recherche de la place de x et decalage des index
    while (i > -1 && x < liste->tabListe[liste->indexListe[i]]) {
        liste->indexListe[i + 1] = liste->indexListe[i];
        i--;
    }
    liste->indexListe[i + 1] = liste->nbelt;  // insertion de l'index de x
    liste->nbelt++;
    return TRUE;
}
```

Suppression Liste statique

5) Supprimer dans la liste.

```
int listeSupp(TypeListe *liste, int x) {
    int i, indicex;
    if (listeVide(*liste)) return FALSE;
    i = 0;
    while (i < liste->nbelt && liste->tabListe[i] != x) i++;   // Recherche de x
    if (i == liste->nbelt) return FALSE; // Non trouve
    indicex = i;
    for (; i < liste->nbelt; i++) liste->tabListe[i] = liste->tabListe[i + 1];
    liste->tabListe[liste->nbelt - 1] = -1;   // Supression du trou par decalage

    // Decalage des indexes apres indicex et mise a jour des indexes > indicex
    i = 0;
    while (i < liste->nbelt && liste->indexListe[i] != indicex) { // jusqu'a indicex : mise a jour
        if (liste->indexListe[i] > indicex) liste->indexListe[i]--;
        i++;
    }
    for (; i<liste->nbelt; i++) {   // apres indicex : decalage et mise a jour
        liste->indexListe[i] = liste->indexListe[i + 1];
        if (liste->indexListe[i] > indicex) liste->indexListe[i]--;
    }
    if (liste->nbelt < N) liste->indexListe[liste->nbelt - 1] = -1; // dernier index a -1
    liste->nbelt--;
    return TRUE;
}
```

Test de Liste statique (1)

6) Fonction main.

```
void insere(int i) {
    if (!listeInsere(&liste,i)) puts("INDEX PLEIN");
    else listeAff(liste);
}
void supprime(int i) {
    printf("Suppression de %d\n", i);
    if (!listeSupp(&liste,i)) printf("%d NON TROUVE\n", i);
    else listeAff(liste);
}

int main() {
    int i;
    listeInit(&liste);
    listeAff(liste);
    for (i=0; i<5; i+=2) insere(i);
    for (i=1; i<6; i+=2) insere(i);
    supprime(4);
    supprime(5);
    supprime(7);
    supprime(0);
    supprime(2);
    supprime(1);
    supprime(3);
}
```

Test de Liste statique (2)

```
Résultat insertions :

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   -  -  -  -  -  -  -  -  -  -
Index   :   -  -  -  -  -  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  -  -  -  -  -  -  -  -  -
Index   :   0  -  -  -  -  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  -  -  -  -  -  -  -  -
Index   :   0  1  -  -  -  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  4  -  -  -  -  -  -  -
Index   :   0  1  2  -  -  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  4  1  -  -  -  -  -  -
Index   :   0  3  1  2  -  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  4  1  3  -  -  -  -  -
Index   :   0  3  1  4  2  -  -  -  -  -

Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  4  1  3  5  -  -  -  -
Index   :   0  3  1  4  2  5  -  -  -  -
```

Test de Liste statique (3)

```
Résultat suppressions :

Suppression de 4
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  1  3  5  -  -  -  -  -
Index   :   0  2  1  3  4  -  -  -  -  -

Suppression de 5
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   0  2  1  3  -  -  -  -  -  -
Index   :   0  2  1  3  -  -  -  -  -  -

Suppression de 7
7 NON TROUVE
```

```
Suppression de 0
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   2  1  3  -  -  -  -  -  -  -
Index   :   1  0  2  -  -  -  -  -  -  -

Suppression de 2
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   1  3  -  -  -  -  -  -  -  -
Index   :   0  1  -  -  -  -  -  -  -  -

Suppression de 1
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   3  -  -  -  -  -  -  -  -  -
Index   :   0  -  -  -  -  -  -  -  -  -

Suppression de 3
Indices :   0  1  2  3  4  5  6  7  8  9
Valeur  :   -  -  -  -  -  -  -  -  -  -
Index   :   -  -  -  -  -  -  -  -  -  -:
```