**TP4**

**Objectifs:**

** Pratiquer la programmation modulaire**

** Représenter des structures de données sous forme TDA**

** Utiliser les listes chainées pour implémenter des structures plus complexes**

** Réutiliser des services implémentés**

** Adapter des services implémentés**

**Exercice 1**

**Concevoir et réaliser un module permettant de définir le TDA liste linéaire bidirectionnelle à**

**travers une représentation chainée, contenant des entiers en exportant les services suivants :**

** creer\_liste : créer une liste linéaire vide.**

** liste\_vide : tester si la liste linéaire est vide ou non**

** opérations d’adjonction d'un élément à une liste linéaire :**

**--- en tête : avant le premier / en queue : après le dernier/ après un élément référencé /avant un élément référencé**

** opérations de suppression d'un élément de la liste linéaire :**

**--- premier élément / dernier élément / un élément référencé**

** recherche : vérifier l'existence d'un élément dans une liste linéaire. Le point de**

**départ peut être fourni comme paramètre.**

** visiter : visiter tous les éléments de la SD LL en effectuant pour chaque élément**

**visité une action donnée (paramètre).**

** trier\_liste : trier les éléments de la liste linéaire dans l'ordre croissant en**

**inspirant du tri insertion.**

**Ajouter un programme de test permettant de créer deux listes linéaires, les remplir respectivement**

**par n et m éléments, les trier et les fusionner dans une troisième liste linéaire triée dans l'ordre**

**croissant.**

**ListB.h**

#ifndef \_\_LISTB\_\_H\_\_

#define \_\_LISTB\_\_H\_\_

/\*représentation physique\*/

struct noeud

{

    int cle ;

    struct noeud\* suivant ;

    struct noeud\* precedent ;

}

struct liste

{

    struct noeud \*premier ;

    struct noeud  \*dernier ;

}

/\*Prototypes\*/

struct liste  \*creer\_liste(void);

unsigned liste\_vide(struct liste \*ll);

/\*insertion après dernier\*/

void inserer\_apres\_der(int info, struct liste \*ll);

/\*insertion avant premier\*/

void inserer\_avant\_pre(int info, struct liste \*ll);

/\*insertion après élément référencé\*/

void inserer\_apres(int info, struct noeud \*p);

/\*insertion avant un élément référencé\*/

void inserer\_avant(int info, struct noeud \*p);

/\*suppression un élément référencé\*/

void supprimer(struct noeud \*p);

/\*suppression du premier\*/

void supprimer\_pre(struct liste \*ll);

/\*suppression du dernier\*/

void supprimer\_der(struct liste \*ll);

struct noeud \* chercher(int info, struct noeud\*p);

void visiter(struct noeud\*p,void(\*oper)(struct noeud\*));

int liste\_longueur(struct noeud \*p);

void trier(struct noeud \*p);

#endif

**ListB.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "ListB.h"

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct liste  \*creer\_liste(void)

{

    struct liste \*ll ;

    ll=(struct liste\*) malloc (size of(struct liste)) ;

    ll->premier=NULL;

    ll->dernier=NULL;

    return ll;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

unsigned liste\_vide(struct liste \*ll)

{

    return((ll->premier==NULL) && (ll->dernier==NULL));

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*insertion après dernier\*/

void inserer\_apres\_der(int info, struct liste \*ll)

{

    struct noeud \*q;

    assert(ll!=NULL);

    q=(struct noeud\*)malloc(sizeof(struct noeud));

    assert(q!=NULL);

    q->cle=info;

    q->precedent=ll->dernier;

    q->suivant=ll->premier;

    ll->dernier=q;;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*insertion avant premier\*/

void inserer\_avant\_pre(int info, struct liste \*ll)

{

    struct noeud \*q;

    assert(ll!=NULL);

    q=(struct noeud\*)malloc(sizeof(struct noeud));

    assert(q!=NULL);

    q->cle=info;

    q->suivant=ll->premier;

    q->precedent=ll->dernier;

    ll->premier=q;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*insertion après élément référencé\*/

void inserer\_apres(int info, struct noeud \*p)

{

    struct noeud \*q ;

    q = (struct noeud \*)malloc(size of(struct noeud)) ;

    q->cle = info;

    q->suivant = p->suivant ;

      /\*mise à jour du successeur de p\*/

    p->suivant = q ;

      /\*mise à jour du precedeur de p\*/

    q->precedent = p;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*insertion avant un élément référencé\*/

void inserer\_avant(int info, struct noeud \*p)

{

    struct noeud \*q ;

    q = (struct noeud \*)malloc(size of(struct noeud)) ;

    q->cle = info;

    q->precedent = p->precedent ;

      /\*mise à jour du precedeur de p\*/

    p->precedent = q ;

      /\*mise à jour du successeur de p\*/

    q->suivant = p ;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*suppression du premier\*/

void supprimer\_pre(struct liste \*ll)

{

    return (ll->premier==NULL);

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*suppression du dernier\*/

void supprimer\_der(struct liste \*ll)

{

    return(p->dernier==NULL);

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*suppression un élément référencé\*/

void supprimer(struct noeud \*p)

{

    if(p->suivant == NULL  && p->precedent == NULL)

    {

        free(p);

        p = NULL;

        return creer\_liste();

    }

    struct noeud \*temp = p;

    struct noeud \*before = p;

    while(temp->suivant != NULL)

    {

        before = temp;

        temp = temp->next;

    }

    before->next = NULL;

    free(temp);

    temp = NULL;

    return p;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*suppression un élément référencé\*/

void supprimer(struct noeud \*p)

{

    if(p->next == NULL)

    {

        free(p);

        p = NULL;

        return creer\_liste();

    }

    struct noeud \*temp = p;

    struct noeud \*before = p;

    while(temp->suivant != NULL)

    {

        before = temp;

        temp = temp->next;

    }

    before->next = NULL;

    free(temp);

    temp = NULL;

    return p;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct noeud \* chercher(int info, struct noeud\*p)

{

    while(p&&(p->cle!=info))

    p=p->suivant ;

    //échec : !p => p== NULL

    //succès :p->cle==info\*/

    return(p) ;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void visiter(struct noeud\*p,void(\*oper)(struct noeud\*))

{

    while(p)

    {

    /\*appliquer à l’élément porté par p le traitement est fourni par oper\*/

        (\*oper)(p) ;

    /\*passer à l’élément suivant \*/

        p=p->suivant ;

    }

    /\*    Exemple:

    struct noeud \* point\_de\_depart ;

    void afficher(struct nœud \* q)

    {

        printf (’’%u d\n ‘’, q->cle) ;

    }

    visiter(point\_de\_depart, afficher) ;

    void incrementer (struct nœud \*q)

    {

        q->cle++ ;

    }

    visiter(point\_de\_depart, incrementer) ;

    \*/

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

int liste\_longueur(struct noeud \*p)

{

    int l = 0;

    if(liste\_vide(p))

        return size;

    while(p != NULL)

    {

        l++;

        p = p->suivant;

    }

    return l;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void trier(struct noeud \*p)

{

    int tmp;

    while(p<p->suivant)

    {

        tmp=p;

        p=p->suivant;

        p->suivant=tmp;

    }

}

**Exercice 2**

**Concevoir et réaliser un module permettant de gérer des listes linéaires circulaires**

**unidirectionnelles à travers une représentation chainée, contenant des entiers, en exportant les**

**services suivants :**

**- Création d’une liste circulaire unidirectionnelle vide**

**- Adjonction après un élément référencé**

**- Suppression d’un élément référencé**

**- Recherche d’un élément à partir d’une clé donnée**

**- Visite de tous les éléments de la liste en effectuant un traitement donné comme paramètre.**

**ListC.h**

#ifndef \_\_LISTC\_\_H\_\_

#define \_\_LISTC\_\_H\_\_

/\*représentation physique\*/

struct noeud

{

    int cle ;

    struct noeud \*suivant ;

}

struct liste

{

    struct noeud \*premier ;

}

/\*Prototypes\*/

struct liste  \*creer\_liste(void);

unsigned liste\_vide(struct liste \*ll);

/\*insertion un élément référencé\*/

void inserer(int info, struct noeud \*p);

/\*suppression un élément référencé\*/

void supprimer(struct noeud \*p);

struct noeud \* chercher(int info, struct noeud\*p);

void visiter(struct noeud\*p,void(\*oper)(struct noeud\*));

#endif

**List.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "ListC.h"

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct liste  \*creer\_liste(void)

{

    struct liste \*ll ;

    ll=(struct liste\*) malloc (size of(struct liste)) ;

    ll->premier=NULL;

    return ll;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

unsigned liste\_vide(struct liste \*ll)

{

    return((ll->premier==NULL);

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*insertion un élément référencé\*/

void inserer(int info, struct noeud \*p)

{

    struct noeud \*q ;

    q = (struct noeud \*)malloc(size of(struct noeud)) ;

    q->cle = info;

    q->suivant = p->suivant ;

      /\*mise à jour du successeur de p\*/

    p->suivant = q ;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

/\*suppression un élément référencé\*/

void supprimer(struct noeud \*p)

{

    if(p->next == NULL)

    {

        free(p);

        p = NULL;

        return creer\_liste();

    }

    struct noeud \*temp = p;

    struct noeud \*before = p;

    while(temp->suivant != NULL)

    {

        before = temp;

        temp = temp->next;

    }

    before->next = NULL;

    free(temp);

    temp = NULL;

    return p;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct noeud \* chercher(int info, struct noeud\*p)

{

    while(p&&(p->cle!=info))

    p=p->suivant ;

    //échec : !p => p== NULL

    //succès :p->cle==info\*/

    return(p) ;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void visiter(struct noeud\*p,void(\*oper)(struct noeud\*))

{

    while(p)

    {

    /\*appliquer à l’élément porté par p le traitement est fourni par oper\*/

        (\*oper)(p) ;

    /\*passer à l’élément suivant \*/

        p=p->suivant ;

    }

    /\*    Exemple:

    struct noeud \* point\_de\_depart ;

    void afficher(struct nœud \* q)

    {

        printf (’’%u d\n ‘’, q->cle) ;

    }

    visiter(point\_de\_depart, afficher) ;

    void incrementer (struct nœud \*q)

    {

        q->cle++ ;

    }

    visiter(point\_de\_depart, incrementer) ;

    \*/

}

**Exercice 3**

**Soit le type Personne défini par un nom, un prénom et une date de naissance.**

** Définir les TDAs Personne et DateNaissance avec les services nécessaires (création, getters**

**et setters).**

** Ajouter au TDA Personne :**

**o un service permettant de calculer et retourner l'âge d'une personne.**

**o un service pour définir la relation d'ordre suivant le nom**

**o un service pour définir la relation d'ordre suivant le prénom**

**o un service pour définir la relation d'ordre suivant l'âge.**

** Reprendre l'exercice 1 pour que le TDA liste linéaire bidirectionnelle soit capable de stocker**

**des personnes.**

** Adapter le service trier\_liste pour qu'il reçoit un paramètre fonctionnel définissant la**

**relation d'ordre à appliquer pour trier une liste linéaire de personnes.**

**Perso.h**

#ifndef \_\_PERSO\_\_H\_\_

#define \_\_PERSO\_\_H\_\_

/\*représentation physique\*/

struct dateNaissance

{

    unsigned j;

    unsigned m;

    unsigned a;

};

struct personne

{

    char \*nom;

    char \*prenom;

    struct dateNaissance dn;

};

struct liste

{

    struct personne \*first;

};

/\*Prototypes\*/

void initialiser(struct liste \*l);

void ajouter(struct liste \*l,struct personne \*P);

void remplir\_tab\_personnes(struct liste \*l,int n);

void affiche\_tab\_personnes(struct liste \*l);

void trier(struct liste \*l);

#endif

**Perso.c**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void initialiser(struct liste \*l)

{

    l->first=NULL;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void ajouter(struct liste \*l,struct personne \*P)

{

    struct personne \*p;

    p=l->first;

    if(p==NULL)

        {

            p=P;

            l->first=p;

        }

    else

    {

        while (p->next != NULL)

        {

            p=p->next;

        }

        p->next=E;

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void remplir\_tab\_personnes(struct liste \*l,int n)

{

    char ch[100];

    struct personne \*P;

    int i;

    for(i=0;i<n;i++)

    {

        printf("\*\*\*\*\*\*\*\*\*Personne %d\n",i+1);

        P=(struct personne\*)malloc(sizeof(struct personne));

        printf("nom: ");

        fflush(stdin);

        gets(ch);

        P->nom=(char\*)malloc(sizeof(char)\*(strlen(ch)+1));

        strcpy(P->nom,ch);

        printf("prenom: ");

        gets(ch);

        P->prenom=(char\*)malloc(sizeof(char)\*(strlen(ch)+1));

        strcpy(P->prenom,ch);

        printf("donner la date de naissance de l'étudiant:ETD[i]= \t",i);

          do

          {

              printf("donner l'année de naissance \t");

              scanf("%d",&P->dn.a);

          } while (P->dn.a<=1980 && P->dn.a>=2003);

          do

          {

              printf("donner le mois de naissance \t");

              scanf("%d",&P->dn.m);

          } while (P->dn.m<0 && P->dn.m>=12);

          if (P->dn.m==2)

            {

                if(P->dn.a%4==0)

                  {

                      do

                           {

                               printf("donner le jour de naissance \t");

                               scanf("%d",&P->dn.j);

                           } while (P->dn.j<0 && P->dn.j>=29);

                  }

                else

                 {

                      do

                           {

                               printf("donner le jour de naissance \t");

                               scanf("%d",&P->dn.j);

                           } while (P->dn.j<0 && P->dn.j>=28);

                 }

            }

          if(P->dn.m==1 || P->dn.m==3 || P->dn.m==5 || P->dn.m==7 || P->dn.m==8 || P->dn.m==10 || P->dn.m==12)

            {

                do

                {

                    printf("donner le jour de naissance \t");

                    scanf("%d",&P->dn.j);

                } while (P->dn.j<0 && P->dn.j>=31);

            }

          else

           {

                 do

                {

                    printf("donner le jour de naissance \t");

                    scanf("%d",&P->dn.j);

                } while (P->dn.j<0 && P->dn.j>=30);

           }

    P->next=NULL;

    ajouter(l,P);

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void affiche\_tab\_personnes(struct liste \*l)

{

    struct personne \*p;

    p=l->first;

    int i;

    while(p!=NULL)

    {

        printf("\*\*\*\*\*\*\*\*\*Personne %d\n",i+1);

        printf("nom: %s\n",p->nom);

        printf("prenom: %s\n",p->prenom);

        printf("matricule: %d\n",p->matricule);

        printf("Date de naissance : %u/%u/%u\n",p->dn.j,p->dn.m,p->dn.a);

        p=p->next;

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void trier(struct liste \*l)

{

    int tmp;

    while(l<l->suivant)

    {

        tmp=l;

        l=l->suivant;

        l->suivant=tmp;

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

// void tri\_insertion\_personnes(personne T[],int n, int (\*oper)(personne,personne))

// {

//     int i,j;

//     personne v;

//     for(i=1;i<n;i++)

//     {

//         v=T[i];

//         j=i-1;

//         while(j>=0 && (\*oper)(T[j],v)>0)//strcmp(T[j].nom,v.nom)>0)

//         {

//             T[j+1]=T[j];

//             j--;

//         }

//         T[j+1]=v;

//     }

// }

// /\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

// int compare\_personne\_nom(personne p1,personne p2)

// {

//     return strcmp(p1.nom,p2.nom);

// }

// /\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

// int compare\_personne\_prenom(personne p1,personne p2)

// {

//     return strcmp(p1.prenom,p2.prenom);

// }

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void main()

{

    int n;

    struct liste \*L;

    do

    {

        printf("n=");

        scanf("%d",&n);

    }while(n<=0 || n>100);

    L=(struct liste\*)malloc(sizeof(struct liste));

    initialiser(L);

    remplir\_tab\_pointeurs\_personnes(L,n);

    affiche\_tab\_pointeurs\_personnes(L);

    trier(L);

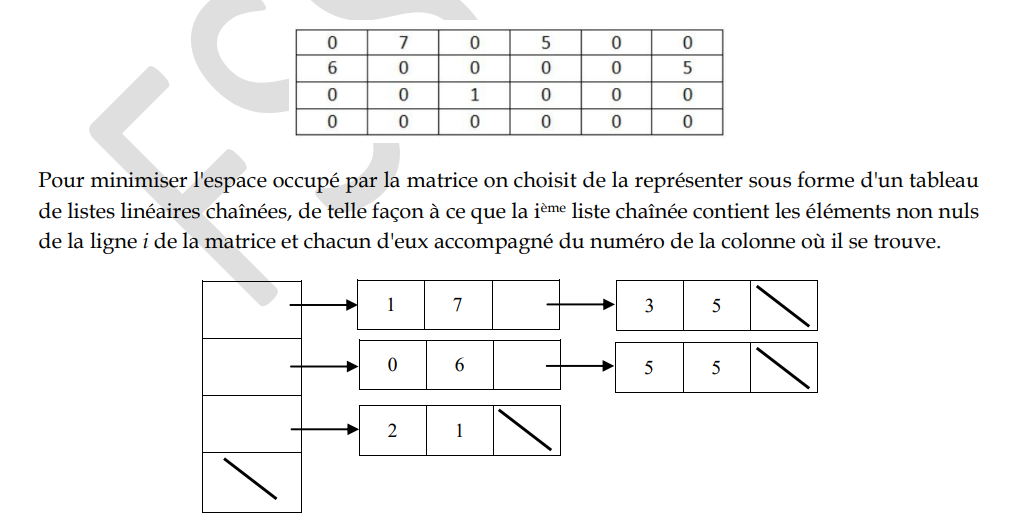
    affiche\_tab\_pointeurs\_personnes(L);

}

**Exercice 4**

**Matrice Creuse**

**On définit une matrice creuse comme étant une matrice dont plus que la moitié de ses éléments**

**sont nuls.**

**Concevoir et réaliser un module en C permettant de matérialiser la TDA MATRICE\_CREUSE. Un tel module doit exporter les opérations suivantes :**

** creer\_matrice : créer une matrice creuse ayant n lignes.**

** ajouter\_element : ajouter un élément non nul à une ligne i et une colonne j.**

** consulter\_element : retourner l’élément [i, j]**

** existe : vérifier l’existence d’un élément [i, j]**

** modifier\_element : modifier le contenu de l’élément [i, j]**

** afficher\_matrice : affiche le contenu d’une matrice**

**Matrice.h**

#ifndef \_\_MATRIX\_\_H\_\_

#define \_\_MATRIX\_\_H\_\_

/\*représentation physique\*/

struct Matrix

{

    int rows;      // number of rows

    int cols;      // number of columns

    double \*\*data; // a pointer to an array of n\_rows pointers to rows

};

struct liste

{

    struct ETD \*first;

};

/\*Prototypes\*/

void initialiser(struct liste \*l);

void ajouter(struct liste \*l,struct Matrix \*m);

struct Matrix \*creer\_matrice(void);

void  ajouter\_element(int,struct Matrix);

unsigned consulter\_element(struct Matrix);

struct Matrix existe(int,struct Matrix);

void modifier\_element(int struct Matrix);

struct Matrix afficher\_matrice(void);

#endif

**Matrice.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "MATRIX.h"

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void initialiser(struct liste \*l)

{

    l->first=NULL;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void ajouter(struct liste \*l,struct ETD \*E)

{

    struct ETD \*p;

    p=l->first;

    if(p==NULL)

        {

            p=E;

            l->first=p;

        }

    else

    {

        while (p->next != NULL)

        {

            p=p->next;

        }

        p->next=E;

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct Matrix \*creer\_matrice(void)

{

    struct Matrix \*m ;

    m=(struct Matrix\*) malloc (size of(struct Matrix)) ;

    m->cols=NULL;

    m->cols=NULL;

    return m;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void  ajouter\_element(int c,int r,struct liste l)

{

    struct Matrix m;

    m.cols=c;

    m.rows=r;

    for(int i=0;  i<m.rows ; i++)

    {

        for (int j = 0; j < m.cols; j++)

        {

            printf("donner l'entier m[i][j]",i,j);

            scanf("%f",m[i][j]);

        }

        consulter\_element(m);

        m->next=NULL;

        ajouter(l,m);

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

unsigned consulter\_element(struct Matrix m)

{

    if (m.data != 0)

    {

        return m.data;

    }

    return NULL;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct Matrix existe(int,struct Matrix m)

{

    int exist=0;

    if ( consulter\_element(m)!=NULL)

    {

        exist=1;

    }

    return exist;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void modifier\_element(int struct Matrix m)

{

    float tmp;

    printf("modifier l'element \n");

    scanf("%f",&tmp);

    m.data=tmp;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

struct Matrix afficher\_matrice(void)

{

    for (int i = 0; i < m->rows; i++)

    {

        for (int j = 0; j < m->cols; j++)

        {

            printf("%f", m->data[i][j]);

        }

    }

}

**Exercice 5**

**Graphe orienté**

**Travail demandé:**

**Ecrire une solution modulaire TDA pour la représentation d'un graphe orienté défini par un**

**ensemble de nœuds et un ensemble d'arcs.**

**Les services à exporter sont :**

** creer\_graphe : créer un graphe orienté initialement vide.**

** ajouter\_noeud : ajouter un nœud à un graphe donné.**

** supprimer\_noeud : supprimer un nœud d'un graphe donné.**

** ajouter\_arc : ajouter un arc entre deux nœuds appartenant à un graphe.**

** supprimer\_arc : supprimer un arc d'un graphe donné.**

** affiche\_graphe : afficher les informations d'un graphe.**

** degre\_noeud : retourner le degré d'un nœud dans un graphe, sachant que le degré d'un nœud est le**

**nombre d'arcs entrants et sortants d'un nœud.**

** matrice\_adjacence : retourner la matrice d'adjacence d'un nœud**

**Remarque : L'insertion dans la liste linéaire se fait en respectant l'ordre croissant des indices attribués aux**

**nœuds pour minimiser le cout de l'opération de recherche.**

**Questions :**

**1. Proposer une représentation physique**

**2. Implémenter les services cités**

**3. Implémenter un programme de test**

**Graph.h**

#ifndef \_\_GRAPH\_\_H\_\_

#define \_\_GRAPH\_\_H\_\_

/\* DÃ©finition d'un BoolÃ©en \*/

typedef enum

{

    false,

    true

} Bool;

/\* DÃ©finition d'une liste de Noeuds (sommet) \*/

typedef struct NodeListElement

{

    int value;

    struct NodeListElement \*next;

} NodeListElement, \*NodeList;

/\* DÃ©finition d'une liste d'ajdacence (tableau) \*/

typedef struct AdjencyListElement

{

    NodeListElement \*begin;

} AdjencyListElement, \*AdjencyList;

/\* DÃ©finition d'un Graphe \*/

typedef struct GraphElement

{

    Bool is\_oriented;

    int nb\_vertices;

    AdjencyList tab\_neighbours;

    FILE \*graph\_file;

} GraphElement, \*Graph;

/\* Prototypes \*/

Graph new\_graph(int vertices, Bool is\_oriented);

Bool is\_empty\_graph(Graph g);

NodeList add\_node(int x);

void add\_edge(Graph g, int src, int dest);

void print\_graph(Graph g);

void display\_graph(Graph g);

void erase\_graph(Graph g);

#endif

**Graph.c**

#include <stdio.h>

#include <stdlib.h>

#include "Ex5\_Graphe.h"

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

Graph new\_graph(int vertices, Bool is\_oriented)

{

    int i;

    GraphElement \*element;

    element = malloc(sizeof(\*element));

    if(element == NULL)

    {

        fprintf(stderr, "Erreur : Probleme creation Graphe.\n");

        exit(EXIT\_FAILURE);

    }

    element->is\_oriented = is\_oriented;

    element->nb\_vertices = vertices;

    element->tab\_neighbours = malloc(vertices \* sizeof(AdjencyListElement));

    if(element->tab\_neighbours == NULL)

    {

        fprintf(stderr, "Erreur : Probleme creation Graphe.\n");

        exit(EXIT\_FAILURE);

    }

    for(i = 1 ; i < element->nb\_vertices + 1 ; i++)

        element->tab\_neighbours[i-1].begin = NULL;

    if(element->is\_oriented)

        element->graph\_file = fopen("digraph.out", "w");

    else

        element->graph\_file = fopen("graph.out", "w");

    if(element->graph\_file == NULL)

    {

        fprintf(stderr, "Erreur : Probleme creation du fichier.\n");

        exit(EXIT\_FAILURE);

    }

    if(element->is\_oriented)

        fprintf(element->graph\_file, "digraph my\_graph\n{\n");

    else

        fprintf(element->graph\_file, "graph my\_graph\n{\n");

    return element;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

Bool is\_empty\_graph(Graph g)

{

    if(g == NULL)

        return true;

    return false;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

NodeList add\_node(int x)

{

    NodeList n = malloc(sizeof(NodeListElement));

    if(n == NULL)

    {

        fprintf(stderr, "Erreur : Probleme creation Node.\n");

        exit(EXIT\_FAILURE);

    }

    n->value = x;

    n->next = NULL;

    return n;

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void add\_edge(Graph g, int src, int dest)

{

    NodeList n = add\_node(dest);

    n->next = g->tab\_neighbours[src-1].begin;

    g->tab\_neighbours[src-1].begin = n;

    if(!g->is\_oriented)

    {

        n = add\_node(src);

        n->next = g->tab\_neighbours[dest-1].begin;

        g->tab\_neighbours[dest-1].begin = n;

    }

    //Ajout d'un lien dans le fichier Graphviz

    if(g->is\_oriented)

        fprintf(g->graph\_file, "\t%d -> %d;\n", src, dest);

    else

        fprintf(g->graph\_file, "\t%d -- %d;\n", src, dest);

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void print\_graph(Graph g)

{

    int i;

    for(i = 1 ; i < g->nb\_vertices + 1 ; i++)

    {

        NodeList n = g->tab\_neighbours[i-1].begin;

        printf("(%d) : ", i);

        while(n != NULL)

        {

            printf("%d, ", n->value);

            n = n->next;

        }

        printf("NULL\n");

    }

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void display\_graph(Graph g)

{

    //Windows seulement !

    if(g->is\_oriented)

        system("%CD%/graphviz/bin/dotty.exe digraph.out");

    else

        system("%CD%/graphviz/bin/dotty.exe graph.out");

}

/\*++++++++++++++++++++++++++++++++++-----------+++++++++++++++++++++++++++++++++++++++\*/

void erase\_graph(Graph g)

{

    if(is\_empty\_graph(g))

    {

        printf("Rien a effacer, le Graphe n'existe pas.\n");

        return;

    }

    //Si il existe des sommets adjacents

    if(g->tab\_neighbours)

    {

        int i;

        for(i = 1 ; i < g->nb\_vertices + 1 ; i++)

        {

            NodeList n = g->tab\_neighbours[i-1].begin;

            while(n != NULL)

            {

                NodeList tmp = n;

                n = n->next;

                free(tmp);

            }

        }

        //LibÃ©ration de la liste d'adjacence

        free(g->tab\_neighbours);

    }

    //Fin et fermeture du fichier Graphviz

    fprintf(g->graph\_file, "}\n");

    fclose(g->graph\_file);

    //LibÃ©ration du Graphe de la mÃ©moire

    free(g);

}