Liste Lineaire

```
/*représentation physique chainée*/

struct element
{
    int cle;
    struct element*suivant;
};


struct liste
{
    struct element* premier;
    struct element* dernier;
};

/*services à exporter*/

void creer_liste(struct liste*);  /*service de création*/
unsigned liste_vide(struct liste); /*service de consultation*/
void ajout_entete(int, struct liste*); /*service de modification*/
void ajout_enqueue(int, struct liste*);/*service de modification*/
void supprimer_entete(struct liste *);/*service de modification*/
void supprimer_enqueue(struct liste *);/*service de modification*/
void supprimer_ele_ref(struct liste*,struct element*);/*service de modification*/
void effacer(struct liste *);/*service de modification*/
struct element * chercher(int info, struct element*p);/*service de consultation*/
void visiter(struct element*p,void(*oper)(struct element*));/*service de consulta
tion*/
unsigned liste_longueur(struct element *p);/*service de consultation*/
void trier(struct element *p);/*service de modification*/
```

```c
#include<stdlib.h>
#include<assert.h>
#include "liste.h"

void creer_liste(struct liste* ll)
{
    assert(ll!=NULL);
    ll->premier=NULL;
    ll->dernier=NULL;
}

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

unsigned liste_vide(struct liste ll)
{
    return ll.premier==NULL && ll.dernier==NULL;
}

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

void ajout_entete(int info, struct liste* ll)
{
    struct element *q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    q->cle=info;
    q->suivant=ll->premier;
    ll->premier=q;
    if(ll->dernier==NULL)/*if (liste_vide(*ll))*/
        ll->dernier=q;
}

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

void ajout_enqueue(int info, struct liste *ll)
{
    struct element*q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    q->cle=info;
    q->suivant=NULL;
    if(liste_vide(*ll))
    {
        ll->premier=q;
    }
```

```
    else
    {
        ll->dernier->suivant=q;
    }
    ll->dernier=q;
}

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

void supprimer_entete(struct liste *ll)
{
    struct element *p,*q;
    assert(ll!=NULL);
    p=ll->premier;
    if(p->suivant== NULL)
    {
        free(p);
        return creer_liste(ll);
    }
    else
    {
        ll->premier=p->suivant;
        free(p);
    }
    if(ll->premier==NULL) ll->dernier==NULL;
}

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

void supprimer_enqueue(struct liste *ll)
{
    struct element *p;
    if(ll->premier==ll->dernier) supprimer_entete(ll);
    else
    {
        p=ll->premier;
        while (p->suivant!=ll->dernier)
        {
            p=p->suivant;
        }
        p->suivant=NULL;
        free(ll->dernier);
        ll->dernier=p;
    }
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/
```

```c
void supprimer_ele_ref(struct liste *ll,struct element *ref)
{
    assert(ref!=NULL);
    struct element *p;
    p=ll->premier;
    while (p->suivant!=ref)
    {
        p=p->suivant;
    }
    p->suivant=ref->suivant;
    free(ref);

}

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

void effacer(struct liste *ll)
{
    if(liste_vide(li))
        return creer_liste(ll);

    while(ll->premier != NULL)
        supprimer_enqueue(ll);
}

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

struct element * chercher(int info, struct element*l)
{
    while(l&&(l->cle!=info))
    l=l->suivant ;
    //échec : !l => l== NULL
    //succès :l->cle==info*/
    return(l) ;
}

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

void visiter(struct element *l,void(*oper)(struct element*))
{
    while(l)
    {
    /*appliquer à l'élément porté par l le traitement est fourni par oper*/
        (*oper)(l) ;
    /*passer à l'élément suivant */
        l=l->suivant ;
```

```
    }
    /*    Exemple:
    struct element * point_de_depart ;
    void afficher(struct element * q)
    {
        printf (''%u d\n '', q->cle) ;
    }
    visiter(point_de_depart, afficher) ;
    void incrementer (struct element *q)
    {
        q->cle++ ;
    }
    visiter(point_de_depart, incrementer) ;
    */
}

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

unsigned liste_longueur(struct element *l)
{
    int length = 0;

    if(liste_vide(p))
        return length;

    while(l != NULL)
    {
        length++;
        l = l->suivant;
    }

    return length;
}

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

void trier(struct element *l)
{
    Struct element *tmp;
    while(l<l->suivant)
    {
        *tmp=*l;
        *l=*l->suivant;
        *l->suivant=*tmp;
    }
}
```

Liste Bidirectionnelle

```c
/*représentation physique*/
struct element
{
    int cle;
    struct element *precedent;
    struct element *suivant;
};

struct listeBi
{   struct element* premier;
    struct element*dernier;
};

/*services à exporter*/

void creer_liste(struct listeBi*);  /*service de céreation*/
unsigned liste_vide(struct listeBi); /*service de consultation*/

void ajouter_entete(int,struct listeBi*); /*service de modification*/
void ajouter_enqueue(int,struct listeBi*); /*service de modification*/
void ajouter_apres_ele_ref(int,struct element*,struct listeBi*);/*service de modi
fication*/
void ajouter_avant_ele_ref(int info,struct element* ref,struct listeBi* l);/*serv
ice de modification*/
void supprimer_ele_ref(struct element*,struct listeBi*);/*service de modification
*/
void tri_insertion(struct listeBi*);/*service de modification*/

void afficher_premier_vers_dernier(struct listeBi); /*service de consultation*/
void afficher_dernier_vers_premier(struct listeBi); /*service de consultation*/
struct element* rechercher(int, struct listeBi); /*service de consultation*/
```

```c
#include<stdlib.h>
#include<assert.h>
#include<stdio.h>
#include"liste.h"


void creer_liste(struct listeBi* l)
{
    assert(l!=NULL);
    l->premier=NULL;
    l->dernier=NULL;
}

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

unsigned liste_vide(struct listeBi l)
{
    return l.premier==NULL && l.dernier==NULL;
}

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

void ajouter_entete(int info,struct listeBi* l)
{
    struct element *e;
    assert(l!=NULL);
    struct e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
    e->cle=info;
    e->suivant=l->premier;
    e->precedent=NULL;
    if(l->premier!=NULL)/*la liste n'est pas vide*/
        l->premier->precedent=e;
    else /* il s'agit du premier élèment à ajouter dans une liste vide*/
        l->dernier=e;

    l->premier=e;
}

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

void ajouter_enqueue(int info,struct listeBi* l)
{
    struct element* e;
    assert(l!=NULL);
    e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
```

```
    e->cle=info;
    e->suivant=NULL;
    e->precedent=l->dernier;
    if(l->dernier!=NULL)/*la liste n'est pas vide*/
        l->dernier->suivant=e;
    else /*on va ajouter le premier élèment dans une liste vide*/
        l->premier=e;
    l->dernier=e;
}

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

void ajouter_apres_ele_ref(int info,struct element* ref,struct listeBi* l)
{
    struct element*e;
    assert(l!=NULL);
    assert(ref!=NULL);
    assert(!liste_vide(*l));
    e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
    e->cle=info;
    e->suivant=ref->suivant;
    e->precedent=ref;
    if(ref!=l->dernier)
        ref->suivant->precedent=e;
    else
        l->dernier=e;
    ref->suivant=e;
}

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

void ajouter_avant_ele_ref(int info,struct element* ref,struct listeBi* l)
{
    struct element*e;
    assert(l!=NULL);
    assert(ref!=NULL);
    assert(!liste_vide(*l));
    e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
    e->cle=info;
    e->precedent=ref->precedent;
    e->suivant=ref;
    if(ref!=l->dernier)
        ref->precedent->suivant=e;
    else
        l->dernier=e;
    ref->precedent=e;
```

```
}

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

void supprimer_ele_ref(struct element* ref,struct listeBi* l)
{
    assert(ref!=NULL);
    assert(l!=NULL);
    if(ref->precedent!=NULL)
        ref->precedent->suivant=ref->suivant;
    else
    {
        l->premier=ref->suivant;
    }
    if(ref->suivant!=NULL)
        ref->suivant->precedent=ref->precedent;
    else
    {
        l->dernier=ref->precedent;
    }
    free(ref);
}

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

void tri_insertion(struct listeBi* l)
{
    int v;
    struct element *p,*q;
    assert(l!=NULL);
    assert(!liste_vide(*l));
    p=l->premier->suivant;
    while(p) /* p!=NULL*/
    {
        v=p->cle;
        q=p;
        while(q->precedent!=NULL && q->precedent->cle>v)
        {
            q->cle=q->precedent->cle;
            q=q->precedent;
        }
        q->cle=v;
        p=p->suivant;
    }
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/
```

```c
void afficher_premier_vers_dernier(struct listeBi l)
{
    struct element*e;
    e=l.premier;
    while(e)
    {
        printf("%d\n",e->cle);
        e=e->suivant;
    }
}
////////////////////////////////////////////////////
void afficher_dernier_vers_premier(struct listeBi l)
{
    struct element*e;
    e=l.dernier;
    while(e)
    {
        printf("%d\n",e->cle);
        e=e->precedent;
    }
}

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

struct element* rechercher(int info, struct listeBi l)
{
    struct element *e;
    e=l.premier;
    while(e && e->cle!=info)
        e=e->suivant;
    return e;
}
```

Liste Circulaire

```
/*représentation physique chainée*/

struct element
{
    int cle;
    struct element*suivant;
};


struct listeC
{
    struct element* premier;
    struct element* dernier;
};

/*services à exporter*/

void creer_liste(struct listeC*);  /*service de création*/
unsigned liste_vide(struct listeC); /*service de consultation*/
void ajout_avant_ref(int info, struct listeC *ll,struct element *ref);/*service d
e modification*/
void ajout_apres_ref(int info, struct listeC *ll,struct element *ref);/*service d
e modification*/
void supprimer_entete(struct listeC *);/*service de modification*/
void supprimer_enqueue(struct listeC *);/*service de modification*/
void supprimer_ele_ref(struct listeC *,struct element *);/*service de modificatio
n*/
void effacer(struct listeC *);/*service de modification*/
struct element * chercher(int info, struct element*p);/*service de consultation*/
void visiter(struct element*p,void(*oper)(struct element*));/*service de consulta
tion*/
unsigned liste_longueur(struct element *p);/*service de consultation*/
void trier(struct element *p);/*service de modification*/
```

```c
#include<stdlib.h>
#include<assert.h>
#include "liste.h"

void creer_liste(struct listeC* ll)
{
    assert(ll!=NULL);
    ll->premier=NULL;
    ll->dernier=NULL;
}

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

unsigned liste_vide(struct listeC ll)
{
    return ll.premier==NULL && ll.dernier==NULL;
}

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

void ajout_apres_ref(int info, struct listeC* ll,struct element *ref)
{
    struct element *q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    q->cle=info;
    q->suivant=ref->suivant;
    ref->suivant=q;
}

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

void ajout_avant_ref(int info, struct listeC *ll,struct element *ref)
{
    struct element*q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    *q=*ref;
    /*en c l'affectation est définie sur les variables ayant un type struct :
    q->cle=ref->cle et q->suivant=ref->suivant*/
    /*mise à jour l'espace référencé par ref*/
    ref->cle=info;
    ref->suivant=q;
}
```

```c
/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++*/

void supprimer_entete(struct listeC *ll)
{
    struct element *p,*q;
    assert(ll!=NULL);
    p=ll->premier;
    if(p->suivant == p)
    {
        free(p);
        return creer_liste(ll);
    }
    else
    {
        ll->premier=p->suivant;
        free(p);
    }
    if(ll->premier==NULL) ll->dernier==NULL;
}

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

void supprimer_enqueue(struct listeC *ll)
{
    struct element *p;
    if(ll->premier==ll->dernier) supprimer_entete(ll);
    else
    {
        p=ll->premier;
        while (p->suivant!=ll->dernier)
        {
            p=p->suivant;
        }
        p->suivant=ll->premier;
        free(ll->dernier);
        ll->dernier=p;
    }
}

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

void supprimer_ele_ref(struct listeC *ll,struct element *ref)
{
    assert(ref!=NULL);
    struct element *p;
    p=ll->premier;
    while (p->suivant!=ref)
```

```
    {
        p=p->suivant;
    }
    p->suivant=ref->suivant;
    free(ref);

}

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

void effacer(struct listeC *ll)
{
    if(liste_vide(li))
        return creer_liste(ll);

    while(ll->premier != NULL)
        supprimer_enqueue(ll);
}

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

struct element * chercher(int info, struct element*l)
{
    while(l&&(l->cle!=info))
    l=l->suivant ;
    //échec : !l => l== NULL
    //succès :l->cle==info*/
    return(l) ;
}

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

void visiter(struct element *l,void(*oper)(struct element*))
{
    while(l)
    {
    /*appliquer à l'élément porté par l le traitement est fourni par oper*/
        (*oper)(l) ;
    /*passer à l'élément suivant */
        l=l->suivant ;
    }
    /*    Exemple:
    struct element * point_de_depart ;
    void afficher(struct element * q)
    {
        printf (''%u d\n '', q->cle) ;
    }
```

```
    visiter(point_de_depart, afficher) ;
    void incrementer (struct element *q)
    {
        q->cle++ ;
    }
    visiter(point_de_depart, incrementer) ;
    */
}

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

unsigned liste_longueur(struct element *l)
{
    int length = 0;

    if(liste_vide(p))
        return length;

    while(l != NULL)
    {
        length++;
        l = l->suivant;
    }

    return length;
}

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

void trier(struct element *l)
{
    struct element *tmp;
    while(l<l->suivant)
    {
        *tmp=*l;
        *l=*l->suivant;
        *l->suivant=*tmp;
    }
}
```

Matrix Creuse

```c
/*représentation physique*/
struct element
{
    int ligne;      // numero de lignes
    int colone;     // numero de columns
    int valeur;     // la valeur du iéme case
    struct element *suivant;
};

struct liste
{
    struct element *premier;
    struct element *dernier;
};

/*Prototypes*/
struct liste *creer_liste(); /*service de création*/
unsigned vide(struct liste*); /*service de consultation*/
struct element *premier(struct liste *ll); /*service de consultation*/
struct element *case(struct element*,int,int); /*service de consultation*/
struct element *exist(int, struct element*); /*service de consultation*/
void ajouter(int ,int ,int ,struct liste *); /*service de modification*/
struct element *modifier(int l,int c,int v,struct liste *ll); /*service de modifi
cation*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "matrix.h"
#define k 100

struct liste *creer_liste()
{
    struct liste ll;
    ll=(struct liste*)malloc(sizeof(struct liste));
    ll->premier=NULL;
    ll->dernier=NULL;
    return(ll);
}

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

unsigned vide(struct liste*)
{
    return l.premier==NULL && l.dernier==NULL;
}

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

struct element *premier(struct liste *ll)
{
    assert(!vide(ll));
    return(ll->premier);
}

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

struct element *case(struct element* p,int l,int c)
{
    assert(!vide(ll));
    return(p->valeur);
}

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

struct element *exist(int info, struct element *p)
{
    while(p&&(p->valeur!=info))
    p=p->suivant ;
    //échec : !p => p== NULL
    //succès : p->cle==info*/
```

```
    return(p) ;
}

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

void ajouter(int l,int c,int v,struct liste *ll)
{
    struct element *p;
    p=(struct element*)malloc(sizeof(struct element));
    p->valeur=v;
    p->colone=c;
    p->ligne=l;
    p->suivant=NULL;
    if(vide(ll))
    {
        ll->premier=p;
    }
    else
    {
        ll->dernier->suivant=p;
    }
    ll->dernier=p;
}

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

struct element *modifier(int l,int c,int v,struct liste *ll)
{
    assert(!vide(ll));
    struct element *p;
    p=(struct element *)malloc(sizeof(struct element));
    p->colone=c;
    p->ligne=l;
    p->valeur=v;
    return(p);
}

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

void afficher_matrice(struct liste *ll)
{
    printf("M=[");
    while(!vide(ll))
    {
        printf("(%d,(%d,%d))",premier(ll)->valeur,premier(ll)->ligne,premier(ll)-
>colone);
        printf(";");
```

```
    }
    printf("]");
}

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

void saisie_matrix(int T[k][k],int n,int m)
{
    int i,j;
    for (i=0,j=0;i<n,j<m;i++,j++)
    {
        printf("T[%d][%d]",i,j);
        scanf("%d",&T[i][j]);
    }

}

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

void convertir_matrix(struct liste *ll, int M[k][k],int n,int m)
{
    int i,j;
    int i,j;
    for (i=0,j=0;i<n,j<m;i++,j++)
    {
        if (M[i][j]!=0)
        {
            ajouter(M[i][j],i+1,j+1,ll);
        }
        if (M[n-1][m-1]==0)
        {
            ajouter(0,n,m,ll);
        }
    }
}

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

void main()
{
    int m,n,M[k][k];
    struct liste *ll;
    ll=creer_liste();
    do
    {
        printf("donner le nbre du matrix \t");
        scanf("%d",&n);
```

```
    } while (n<=0 || n>k));
    saisie_matrix(M,n,m);
    convertir_matrix(ll,M,n,m);
    afficher_matrice(ll);
}
```

Personne

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

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

struct dateNaissance
{
    unsigned j;
    unsigned m;
    unsigned a;
};

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

struct personne
{
    char *nom;
    char *prenom;
    struct dateNaissance dn;
};

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

struct liste
{
    struct personne *first;
};

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

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;
```

```c
    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;
//     }
// }

// /*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/
```

```c
// 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);
}
```

Etudiant

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

    /* DÃ©finition du type date de naissance */
    struct DN
    {
        int j;
        int m;
        int a;
    };
    /* DÃ©finition du type étudiant */
    struct ETD
    {
        char *prenom;
        char *nom;
        int matricule;
        struct DN DATE;
        struct ETD * next;
    };
    struct liste
    {
        struct ETD *first;
    };

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;
    }
}

void remplir_tab_pointeurs_personnes(struct liste *l, int n)
{
    int i;
    struct ETD *E;
    char ch[100];
    for(i=0;i<n;i++)
    {
        printf("*********ETUDIANT %d\n",i+1);
        E=(struct ETD*)malloc(sizeof(struct ETD));
        printf("nom: ");
        fflush(stdin);
        gets(ch);
        E->nom=(char*)malloc(sizeof(char)*(strlen(ch)+1));
        strcpy(E->nom,ch);
        printf("prenom: ");
        gets(ch);
        E->prenom=(char*)malloc(sizeof(char)*(strlen(ch)+1));
        strcpy(E->prenom,ch);
        printf("matricule: ");
        scanf("%d",&E->matricule);
        printf("Dat.e de naissance :\n");
        printf("jour: ");
        scanf("%u",&E->DATE.j);
        printf("mois: ");
        scanf("%u",&E->DATE.m);
        printf("annee: ");
        scanf("%u",&E->DATE.a);
        E->next=NULL;
        ajouter(l,E);
    }
}

void affiche_tab_pointeurs_personnes(struct liste *l)
{
    struct ETD *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->DATE.j,p->DATE.m,p->DATE.a);
        p=p->next;
```

```
    }
}

void sauvegarde(struct liste *l,int n,char path[])
{
    struct ETD *p;
    p=(struct ETD*)malloc(sizeof(struct ETD));
    FILE *file;
    int i;
    file=fopen(path,"w");
    while(p!=NULL)
    {
        fprintf(file,"%s\t%s\t%d\t%u/%u/%u\n",p->nom,p->prenom,p->matricule,p-
>DATE.j,p->DATE.m,p->DATE.a);
        p=p->next;
    }
    fclose(file);
}

void chargement_donnees_personnes_tab_dynamique( struct liste *l,char path[])
{
    struct ETD *p;
    FILE *file;
    char ch1[100],ch2[100];
    file=fopen(path,"r");
    while(!feof(file))
    {
        p=(struct ETD*)malloc(sizeof(struct ETD));
        fscanf(file,"%s\t%s\t%d\t%u/%u/%u\n",ch1,ch2,&p->matricule,&p->DATE.j,&p-
>DATE.m,&p->DATE.a);
        printf("*********ETUDIANT %d\n");
        printf("nom: ");
        fflush(stdin);
        gets(ch1);
        p->nom=(char*)malloc(sizeof(char)*(strlen(ch1)+1));
        strcpy(p->nom,ch1);
        printf("prenom: ");
        gets(ch2);
        p->prenom=(char*)malloc(sizeof(char)*(strlen(ch2)+1));
        strcpy(p->prenom,ch2);
        printf("matricule: ");
        scanf("%d",&p->matricule);
        printf("Dat.e de naissance :\n");
        printf("jour: ");
        scanf("%u",&p->DATE.j);
        printf("mois: ");
        scanf("%u",&p->DATE.m);
        printf("annee: ");
        scanf("%u",&p->DATE.a);
        p->next=NULL;
```

```
        ajouter(l,p);
    }
    fclose(file);
}


main()
{
    int n=2;
    struct liste *L,*L1;

    L1=(struct liste*)malloc(sizeof(struct liste));
    initialiser(L1);

    L=(struct liste*)malloc(sizeof(struct liste));
    initialiser(L);
    remplir_tab_pointeurs_personnes(L,n);
    sauvegarde(L,n,"C://monFichier.txt");
    affiche_tab_pointeurs_personnes(L);

    chargement_donnees_personnes_tab_dynamique(L1,"C://monFichier.txt");
    affiche_tab_pointeurs_personnes(L1);
}
```