Exercices :

Ex1 :

```
#include<stdio.h>
typedef struct
{
        int Jour;
        int Mois;
        int Annee;
}Date;
typedef struct
{
        char Ville[15];
        char Province[25];
        char Pays[20];
}Lieu;
typedef struct
{
        char Avenue[20];
        char Ville[20];
        char Pays[20];
        int Numero,code_postale,telephone;
}Domicile;
typedef struct
{
        char Etab_1[25];
        char Etab_2[25];
        char Etab_3[25];
        char Etab_4[25];
}Entreprise;
typedef struct {
        char Nom[40],Prenom[40],Nationalite[40],Diplome;
```

```
        Date Date_naissance;
        Lieu Lieu_naissance;
        char Etat_civil;
        int stage;
        Domicile Adresse;
        Entreprise Etablissement;
}Identite;
Identite employe;
int main()
{
printf("donner le prenom\n");
scanf("%s",employe.Prenom);
printf("donner le Nom\n");
scanf("%s",employe.Nom);
printf("donner le jour de la naissance\n");
scanf("%d",&employe.Date_naissance.Jour);
printf("donner le mois de naissance\n");
scanf("%d",&employe.Date_naissance.Mois);
printf("donner l annee de naissance\n");
scanf("%d",&employe.Date_naissance.Annee);
printf("donner le diplome\n");
scanf("%s",&employe.Diplome);

return 0;
}
```

Ex2 :

```
#include <stdio.h>
typedef struct {
float x; /* partie reelle */
float y; /* partie imaginaire */
} complexe;
int main()
{ int i ;
complexe tab[5];
for (i=0 ;i<5 ;i++)
{
printf("entrez la partie reelle du %d ieme element :\n ",i+1) ;
scanf("%f",&(tab[i].x));
printf("entrez la partie imaginaire du %d ieme element :\n ",i+1) ;
```

```
scanf("%f",&(tab[i].y));
}
printf("affichage du tableau\n");
for (i=0;i<5;i++)
{ printf("%f + %f*i\n",tab[i].x,tab[i].y);
}
return 0;
}
```

Ex3 :

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
/*-------------------------------------*/
typedef struct
{
float largeur, longueur, epaisseur;
char type;
}Panneaux;
/*-------------------------------------*/
/*-------------------------------------*/
Panneaux Saisie()
    {
        Panneaux p;
     printf("Entrez la largeur, la longueur et l paisseur :\n ");
     scanf("%f%f%f",&p.largeur,&p.longueur,&p.epaisseur);
     printf("Entrez lessence de bois\n");
     scanf("%c",&p.type);
     return p;
}
/*-------------------------------------*/
void Affichage(Panneaux p)
{
  printf("Panneau en ");
switch (p.type)
  {
   case 'a':
   printf("pin\n");
   break;
   case 'b':
```

```
    printf("chene\n");
    break;
    case 'c':
    printf("hetre\n");
    break;
    default:
    printf("inconnue\n");
 }
  printf("largeur = %f ; longueur = %f ; epaisseur = %f\n",p.largeu
}
/*-------------------------------------------------------------------
float Volume(Panneaux p)
 {
   return((p.largeur* p.longueur*p.epaisseur)/1000);
}
/*-------------------------------------------------*/
int main(){
        Panneaux P;
        float V;
        P=Saisie();
        Affichage(P);
        V=Volume(P);
        printf("%f",V);
        return 0;
}
```

Ex4 :

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

typedef struct
{
    float x;
    float y;
} point ;
point Saisie(void)
{  point P;
    printf("donnez un point \n");
     printf("donnez  x  :  ");scanf("%f",&P.x);
     printf("donnez  y :   "); scanf("%f",&P.y);
```

```
     return(P);
}
void Affichage(point *P)
{
     printf("\n point= (%lG, %lG)\n", P->x, P->y );
}
float distance(point *P, point *Q)
{
    return(sqrt(pow(P->x - Q->x, 2) + pow(P->y - Q->y,2)));
}
point *Projection(point *P)
{
    P->y=0;
    return(P);
}

int main ( )
  {
    point a, b ;
    a = Saisie();
    b = Saisie();
    printf("\ndistance entre\n");
    Affichage(&a) ;
    printf("\net\n");
    Affichage(&b) ;
    printf("\nest egale a %lG\n", distance(&a, &b) );
    printf("\nProjection\n");
    Affichage(Projection(&a));
    Affichage(Projection(&b));
    return 0;
}
```

Tp4 :

Ex0 :

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

typedef struct Liste
```

```
{
        int n,b;
        char nom[12];
        struct Liste *next;
}liste;

liste *inti()
{
        return NULL;
}

liste *push_begin(liste *L,int a,int r,char er[12])
{
        liste *e,*p;
        e=(liste*)malloc(sizeof(liste));
        e->b=a;
        e->n=r;
        strcat(e->nom,er);
        e->next=L;
        return e;
}

liste *push_end(liste *L,int a,int r,char er[12])
{
        liste *e,*p;
        e=(liste*)malloc(sizeof(liste));
        e->b=a;
        e->n=r;
        strcat(e->nom,er);
        e->next=NULL;
        if(L==NULL)
        {
                L=e;
        }
        else
        {
                p=L;
                while(p->next!=NULL)
                {
                        p=p->next;
                }
```

```
                p->next=e;
        }
        return L;
}

liste *delet_begin(liste *l)
{
   if(l==NULL)
   {
        return NULL;
   }
   else
   {
        liste *aern=l->next;
        free(l);
        return aern;
   }
}

liste *delet_end(liste *l)
{
        if(l==NULL)
        {
                return NULL;
        }
        else if(l->next==NULL)
        {
                free(l);
        return NULL;
        }

    liste *t=l,*tp=l;
    while(t->next!=NULL)
    {
        tp=t;
        t=t->next;
        }
        tp->next=NULL;
        free(t);
        return l;
}
```

```
liste *delet_pos(liste *l,int a)
{
 if(l==NULL) return 0;
 if(l->b==a)
 {
        liste *t=l->next;
        free(l);
        t=delet_pos(t,a);
        return t;
 }
 else
 {
        l->next=delet_pos(l->next,a);
        return l;
 }

}

int taill(liste *l)
{
        int a=0;
        while(l!=NULL)
        {
                a++;
                l=l->next;
        }
        printf("la taille est %d \n",a);
        return a;
}

liste *push_pos(liste *l,int a,int r,char er[12],int pos,int t)
{
        int i;
        if(pos==0)
          return push_begin(l,a,r,er);
        else if(pos==0)
          return push_end(l,a,r,er);
          else
          {
                liste *no,*curr=l;
```

```
                no=(liste*)malloc(sizeof(liste));
                for(i=0;i<pos-1;i++)
                {
                        curr=curr->next;
                }
                no->b=a;
            no->n=r;
            strcat(no->nom,er);
            no->next=curr->next;
            curr->next=no;
          }
          return l;
}

void est_vide(liste *l)
{
        if(l==NULL)
        {
                printf("epmty list \n");
        }
        else
        {
                printf(" is not epmty list \n");
        }
}

void display(liste *l)
{
        liste *p;
        p=l;
        while(p!=NULL)
        {
                printf("(%d)->(%d)->(%s)<=>",p->b,p->n,p->nom);
                p=p->next;
        }
        printf("\n");
}
/*
void liberate(liste **l)
{
        pointeur *p;
```

```
        while(*l!=NULL)
        {
                p=*l;
                *l=(*l)->next;
                free(p);
        }
        *l=NULL;
    }
}
*/
```

Ex1 :

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

typedef struct Noeud
{
  int donnee;
  struct Noeud * suivant;
} Noeud;

Noeud * cree_liste_vide(){
        return NULL;
}

Noeud * ajout_debut(Noeud * liste, int e)
{
        Noeud *L;
        L=(Noeud*)malloc(sizeof(Noeud));
        L->donnee=e;
        L->suivant=liste;
        return L;
}

Noeud * supprime_debut(Noeud * liste){
        if(liste!=NULL)
        {Noeud *L=liste->suivant;
        free(liste);
        return(L);
        }
```

```
        else{
                return NULL;
        }
}
 int taille(Noeud * liste){
        int n=0;
        while(liste!=NULL)
        {n++;
        liste=liste->suivant;
         }
         return n;
 }

Noeud * ajout_jieme(Noeud * liste, int j, int e)
{
//      if(j==0) return ajout_debut(liste,e);
//      else{
        Noeud *cour=liste;
        Noeud *L=(Noeud*)malloc(sizeof(Noeud));
        int i;
        for(i=0;i<j-1;i++)
        {cour=cour->suivant;}
        L->donnee=e;
        L->suivant=cour->suivant;
        cour->suivant=L;

        return(liste);
}
Noeud * supprime_jieme(Noeud * liste, int j){
        if(liste==NULL) return 0;
        if(liste->donnee==j)
        {
        Noeud *tmp=liste->suivant;
        free(liste) ;
        tmp=supprime_jieme(tmp,j);
        return (tmp);
        }
        else{
                liste->suivant=supprime_jieme(liste->suivant,j);
                return(liste);
        }
```

```
}
void afficher(Noeud * liste){
        Noeud *p;
        p=liste;
        while(p!=NULL)
        {printf ("\n");
                printf("%d",p->donnee);
                p=p->suivant;
        }
        printf("\n");
}
int main()
{ int t;
        Noeud *L=NULL;
        L=ajout_debut(L, 3);
        L=ajout_debut(L, 4);
        L=ajout_debut(L, 5);
        L=ajout_debut(L, 6);
        L=ajout_jieme(L,2,7);
        afficher(L);
        printf("Suppression du premier elem\n" );
        L=supprime_debut(L);
        afficher(L);
        L=supprime_jieme(L,4);
        printf("Suppression de elemen pos 4\n" );
        afficher(L);
        t=taille(L);
    printf("taille(Liste) = %d\n",t );
        return 0;
}
```

Ex2 :

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

typedef struct cell
{
        int data;
        struct cell *next;
```

```
        struct cell *prev;
}node;

typedef struct dlist
{
        int length;
        node *first;
        node *last;
}Dlist;
/************************/
Dlist * Allouer()
        {
                Dlist * dl = (Dlist*)malloc(sizeof (Dlist));
                if (dl != NULL)
                {
                        dl->length = 0;
                        dl->first = NULL;
                        dl->last = NULL;
                }
                return dl;
        }
/************************/
                Dlist * insertTete(Dlist *dl, int od)
        {
                node * nouv = (node*)malloc(sizeof(node));
                if (dl != NULL)
                {
                        if (nouv != NULL)
                        {
                                nouv->data = od;
                                nouv->prev = NULL;
                                if (dl->last == NULL)
                                {
                                        nouv->next = NULL;
                                        dl->first = nouv;
                                        dl->last = nouv;
                                }
                                else
                                {
                                        dl->first->prev = nouv;
                                        nouv->next = dl->first;
```

```
                                        dl->first = nouv;
                                }
                                dl->length++;
                        }
                }
                return dl;
        }
/************************/
        void display(Dlist *dl)
        {
                if (dl != NULL)
                {
                        node *temp = dl->first;
                        while (temp != NULL)
                        {
                                printf("%d  ",temp->data);
                                temp = temp->next;
                        }
                }
        }
/************************/
                Dlist *remove_pos(Dlist *dl, int pos)
        {
                if (dl != NULL)
                {
                        node * temp = dl->first;
                        int i = 1;
                        while (temp != NULL && i <= pos)
                        {
                                if (pos == i)
                                {
                                        if (temp->next == NULL)
                                        {
                                                dl->last = temp->pr
                                                dl->last->next = NU
                                        }
                                        else if (temp->prev == NULL
                                        {
                                                dl->first = temp->n
                                                dl->first->prev = N
                                        }
```

```
                                        else
                                        {
                                                temp->next->prev = temp->
                                                temp->prev->next = temp->
                                        }
                                        delete(temp);
                                        dl->length--;
                                }
                                else
                                {
                                        temp = temp->next;
                                }
                                i++;
                        }
                }
                return dl;
}
int length(Dlist *dl)
{
        int ret = 0;
        if (dl != NULL)
        {
                ret = dl->length;
        }
        return ret;
}
void liberer(Dlist **dl)
{
        if (*dl != NULL)
        {
                node *temp = (*dl)->first;
                while (temp != NULL)
                {
                        node *del = temp;
                        temp = temp->next;
                        free(del);
                }

                (*dl)->first=(*dl)->last = NULL;
        }
}
```

```
        int main(){
        Dlist *L;
        int T,S;
        L= Allouer();
        L=insertTete(L, 2);
        L=insertTete(L, 4);
        L=insertTete(L, 6);
         display(L);
         printf("\n");
         L=remove_pos(L, 2);
         display(L);
         printf("\n");
         T=length(L);
         printf("la taille est %d",T);
    liberer(&L);
        return 0;
}
```

Tp5 :

Ex1 :

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

typedef struct pile{
        int valeur;
        struct pile *suivant;
}Pile;
Pile *creer_pile(Pile* sommet){
        sommet=(Pile*)malloc(sizeof(Pile));
        sommet->suivant=NULL;
        return sommet;
}

int vide(Pile *sommet){
        return(sommet->suivant==NULL);
}

Pile *Empiler(Pile *sommet,int val){
        Pile *P;
```

```
        P=(Pile*)malloc(sizeof(Pile));
        P->valeur=val;
        P->suivant=sommet->suivant;
        sommet->suivant=P;
        return(sommet);
}

Pile *Depiler(Pile *sommet){
        Pile *P;
        if (vide(sommet)){
                printf("Pile vide\n");
                return NULL;
        }
        P=sommet->suivant;
        sommet->suivant=P->suivant;
        free(P);
        return sommet;
}
void afficher(Pile *sommet)
{
        Pile *P;
        if(vide(sommet))
        printf("pile est vide\n");
        else
        {
                P=sommet->suivant;
                printf("debut->");
                while(P!=NULL)
                {printf("%d ->",P->valeur);
                P=P->suivant;
                }
                printf("fin\n");
        }
}

        int main(){
                Pile *P;
                int p;
                P=creer_pile(P);
                printf("vide= %d\n", vide(P));
                afficher(P);
```

```
                P=Empiler(P,1);
                P=Empiler(P,2);
                P=Empiler(P,3);
                afficher(P);
        printf("la suppression du sommet\n");
                P=Depiler(P);
                afficher(P);
                printf("la suppression du sommet\n");
            P=Depiler(P);
                afficher(P);
                return 0;
        }
```

Ex2 :

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

typedef struct elem{
        int val;
        struct elem *suivant;
}Elem;
typedef struct file{
        Elem *prem;
        Elem *dern;
}File;
File * creation()
{
File *F;
F=(File*)malloc(sizeof(File));
F->prem=F->dern=NULL;
return F;
}
int vide(File *F)
{
return F->dern==NULL;
}

void  Enfiler(File *F, int val)
{
Elem *e;
```

```
e=(Elem*)malloc(sizeof(Elem));
e->val=val;
e->suivant=NULL;
if (vide(F))
 F->dern=F->prem=e;
else
{
F->dern->suivant=e;
F->dern=e;
}
}

File  *Defiler(File *F)
{
Elem *e=NULL;
if (!vide(F))
 {
 if (F->prem==F->dern)
{
e=F->prem;
F->prem=F->dern=NULL;
}
else
{
e=F->prem;
F->prem=F->prem->suivant;
}
return F;
}

}


void  afficher(File *F)
{
Elem *e;
e=F->prem;

if (vide(F))
printf("file vide");
printf("prem<-");
```

```
while(e!=NULL)
{printf("%d-> ",e->val);
  e=e->suivant;}
printf("dern\n");
}
int main(){
        File *F;
        F=creation();
        printf("vide= %d\n", vide(F));
        Enfiler(F,1);
        Enfiler(F,2);
        Enfiler(F,3);
        Enfiler(F,4);
        afficher(F);
        F=Defiler(F);
        afficher(F);
        F=Defiler(F);
        afficher(F);
        return 0;
}
```

Ex3 :

```
#include<stdio.h>
#include<stdlib.h>
typedef struct elem{
        int val;
        struct elem *suivant;
}Elem;
typedef struct file{
        Elem *prem;
        Elem *dern;
}File;
File * creation()
{
File *F;
F=(File*)malloc(sizeof(File));
F->prem=F->dern=NULL;
return F;
}
int vide(File *F)
```

```
{
return F->dern==NULL;
}
void  Enfiler(File *F, int val)
{
Elem *e;
e=(Elem*)malloc(sizeof(Elem));
e->val=val;
e->suivant=NULL;
if (vide(F))
 F->dern=F->prem=e;
else
{
F->dern->suivant=e;
F->dern=e;
}
}
File  *Defiler(File *F)
{
Elem *e=NULL;
if (!vide(F))
 {if (F->prem==F->dern)
{e=F->prem;
F->prem=F->dern=NULL;}
else
{
e=F->prem;
F->prem=F->prem->suivant;
}
return F;}}
void  afficher(File *F)
{
Elem *e;
e=F->prem;

if (vide(F))
printf("file vide");
printf("prem<-");
while(e!=NULL)
{printf("%d-> ",e->val);
  e=e->suivant;}
```

```
printf("dern\n");
}
int main(){
        File *F;
        F=creation();
        printf("vide= %d\n", vide(F));
        Enfiler(F,1);
        Enfiler(F,2);
        Enfiler(F,3);
        Enfiler(F,4);
        afficher(F);
        F=Defiler(F);
        afficher(F);
        F=Defiler(F);
        afficher(F);
        return 0;
}
```