```
//==========Fichier : liste.h====================
#ifndef LISTE_H_INCLUDED
#define LISTE_H_INCLUDED

struct cellule {
      int donnee;
      struct cellule * suivant;};

typedef struct cellule cellule;
typedef cellule *liste;

#endif // LISTE_H_INCLUDED

//=============Fichier:CreerListe.c==============
#include <stdio.h>
#include <stdlib.h>
#include "liste.h"

//============== Longueur ===========
int longueurIterative(liste l){
   int lg = 0;
   if(l==NULL){
      return 0;
   }
   while(l!=NULL){
      lg++;
      l=l->suivant;
   }
   return lg;
}

int longueurRecursive(liste l){
   if (l==NULL){
     return 0;
   }else{
      return (1+longueurRecursive(l->suivant));
   }
}

/*=================Maximum=========================*/
int max (liste l) {
   int courant = l->donnee;
   int prochain;

   if (l->suivant == NULL) {
      //La valeur de ce noeud est  is surement la plus grande
      return courant;
   } else {
      //Recur pour trouver le max a partir du reste de la liste
      prochain = max(l->suivant);
   }
```

```
   //Returner le max entre ce noeud et la fin de la liste
   if (courant > prochain) {
      return courant;
   } else {
      return prochain;
   }
}

/*=================Minimum=========================*/
int min (liste l) {
   int courant = l->donnee;
   int prochain;

   if (l->suivant == NULL) {
      //La valeur de ce noeud est  is surement le min
      return courant;
   } else {
      //Recur pour trouver le max a partir du reste de la liste
      prochain = min(l->suivant);
   }

    //Returner le max entre ce noeud et la fin de la liste
   if (courant < prochain) {
      return courant;
   } else {
      return prochain;
   }
}

/*=================RechercheElement=========================*/
liste rechercheElt(liste l, int n){
   if(l==NULL){
      return NULL;
   }else{
      if(l->donnee == n){
         return(l);
      }else{
         rechercheElt(l->suivant, n);
      }
   }
}
//===============Initialiser============================
liste InitialiserListe(){
   return NULL;
}

//===============InsererEntete============================
liste InsererEntete(liste l, int d){
   liste nouveau;
   nouveau = (liste)malloc(sizeof(cellule));
   nouveau->donnee = d;
   nouveau->suivant =l;
```

```
   return nouveau;
}

void AfficherListe(liste debut){
   if(debut != NULL){
      printf("%d\t", debut->donnee);
      AfficherListe(debut->suivant);
   }
}

//===============InsererEnQueue : passage de la liste par valeur=====
liste InsererQueue(liste l, int d){
   liste ptr = l;
   liste nouveau = (liste)malloc(sizeof(cellule));
   nouveau->donnee=d;
   nouveau->suivant = NULL;
   if(ptr == NULL){
      ptr = nouveau;
      return ptr;
   }
   while(ptr->suivant!=NULL){
      ptr = ptr->suivant;
   }
   ptr->suivant = nouveau;
   return l;
}

//===============tabVersLinkedList : ConvertitTableau en linked liste=====
liste tabVersLinkedList(int tab[], int n){
   liste l = NULL;
   for(int i=0; i<n; i++){
     l= InsererQueue(l, tab[i]);
   }
   return l;
}

//===================kieme=========================================
int kieme(liste l, int k){
   int i=1;
   liste ptr=l;
   while((i!=k) && (ptr!=NULL)){
      i++;
      ptr=ptr->suivant;
   }
   if(i==k){
      return (ptr->donnee);
   }else{
      printf("Le rang %d depasse la longueur de la chaine\n", k);
   }
}

//=============Parcour inverss recursive==========
```

```
void afficheInverseRecursive(liste l){
   if(l!=NULL){
      afficheInverseRecursive(l->suivant);
      printf("%d\t", (l->donnee));
   }
}

liste supprimerPremiereOccurrenceRecursive(liste l, int a){
           if(l==NULL) return NULL;
           if(l->donnee==a) {
         liste d=l;
         l=l->suivant;
         free(d);
         return l;
           }
           l->suivant=supprimerPremiereOccurrenceRecursive(l->suivant, a);
           return l;
}

liste supprimerToutеRecursive(liste l, int a){
           if(l==NULL) return l;
           if(l->donnee==a) {
         liste d=l;
         l=l->suivant;
         free(d);
         l=supprimerToutеRecursive(l, a);
           }else{
      l->suivant=supprimerToutеRecursive(l->suivant, a);
           }
}

liste supprimerPremiereOccurrenceIteratif(liste l, int a){
   if(l==NULL){
      return l;
   }
   liste p1 = l, p2=l;
   while(p2!=NULL){
      if(p2->donnee == a){
         if(p2==l){
               l = l->suivant;
               free(p2);
               return l;
         }else{
            p1->suivant=p2->suivant;
            free(p2);
            return l;
         }
      }else{
         p1=p2;
         p2=p2->suivant;
      }
   }
```

```
    return l;
}
liste supprimerToutelteratif(liste l, int a){
    if(l==NULL){
        return l;
    }
    liste p1 = l, p2=l;
    while(p2!=NULL){
        if(p2->donnee == a){
            if(p2==l){
                    l = l->suivant;
                    free(p2);
                    p1=p2=l;
            }else{
                p1->suivant=p2->suivant;
                free(p2);
                p2 = p1->suivant;
            }
        }else{
            p1=p2;
            p2=p2->suivant;
        }
    }
    return l;
}

liste supprimerRepetitionRecursive(liste l){
    if((l==NULL) ||(l->suivant ==NULL))
        return l;
    else{
        if(l->donnee ==l->suivant->donnee){
            liste d=l;
            l=l->suivant;
            free(d);
            l= supprimerRepetitionRecursive(l);
        }else{
            l->suivant = supprimerRepetitionRecursive(l->suivant);
        }
    }
}

liste supprimerRepetitionIterative(liste l){
    if((l==NULL) ||(l->suivant ==NULL))
        return l;
    liste p1=l;
    liste p2 = l->suivant;
    while(p2!=NULL){
        if(p1->donnee == p2->donnee){
            if(p1==l){
                l=p2;
                free(p1);
                p1=p2;
```

```
            p2=p2->suivant;
         }else{
            p1->suivant = p2->suivant;
            liste d = p2;
            p2=p2->suivant;
            free(d);
         }
      }else{
         p1=p2;
         p2=p2->suivant;
      }
   }
   return l;
}

//========== Fichier: main.c================================
#include <stdio.h>
#include <stdlib.h>
#include "liste.h"
int main()
{
   liste debut, p;
   int donnee, rang, choix;
   int *tab, nbElt;

   do{
      printf("\n\n****** Menu**********\n");
      printf("1-Initialiser la liste\n");
      printf("2-Insere au debut de la liste\n");
      printf("3-Affichier la liste\n");
      printf("4-Inserer en Queue passage liste par valeur\n");
      printf("5-Inserer en Queue passage liste par reference : ==Non faite==\n");
      printf("-----------------------------------------------\n");
      printf("6-Longueur de la liste : methode iterative\n");
      printf("7-Longueur de la liste : methode recursive\n");
      printf("8-Maximum de liste : methode recursive\n");
      printf("9-Minimum de liste : methode recursive\n");
      printf("10-Recherche du pointeur vers une donne dans la liste : methode recursive\n");
      printf("11-Convertir un tableau en liste chainnee\n");
      printf("12-Trouver le kieme element de la liste chainnee\n");
      printf("13-Affichage inversee recursive la liste chainee\n");
      printf("14-Suppression de la 1ere occurrence d'une donnee dans la liste(recursive)\n");
      printf("15-Suppression de la 1ere occurrence d'une donnee dans la liste(iterative)\n");
      printf("16-Suppression toutes occurrences d'une donnee dans la liste(recursive)\n");
      printf("17-Suppression toutes occurrences d'une donnee dans la liste(iterative)\n");
       printf("18-Suppression repetitions dans une liste ordonnee(recursive)\n");
      printf("19-Suppression repetitions dans une liste ordonnee(iterative)\n");

      printf("20-Quitter le programme\n");
      printf("\n\Donner votre choix : ");
      scanf("%d", & choix); getchar();
      switch(choix){
```

```
        case 1 : debut = InitialiserListe(); break;
        case 2 : printf("\n\nDonner un entier :");
                 scanf("%d", &donnee); getchar();
                 debut = InsererEntete(debut, donnee);
                 break;
        case 3 : AfficherListe(debut);break;
        case 4 : printf("\n\nDonner un entier :");
                 scanf("%d", &donnee); getchar();
                 debut = InsererQueue(debut, donnee);
                 break;
        case 5 : printf("\n\nNon faite :");
                 break;
        case 6 : printf("Longueur iterative de la liste : %d\n", longueurIterative(debut));
                 break;
        case 7 : printf("Longueur recursive de la liste : %d\n", longueurRecursive(debut));
                 break;
        case 8 : printf("Maximun de la liste : Methode recursive: %d\n", max(debut));
                 break;
        case 9 : printf("Minimun de la liste : %d\n", min(debut));
                 break;
        case 10 : printf("Donner la donnee a chercher : ");
                  scanf("%d", &donnee); getchar();
                  p = rechercheElt(debut, donnee);
                  if(p ==NULL){
                      printf("La donnee %d ne se trouve pas dans la liste\n", donnee);
                  }else{
                    printf("la donnee %d se trouve dans la liste et pointee par %p\n",p->donnee, p);
                  }
                  break;

        case 11 : printf("Donner la dimension du tableau\n");
                  scanf("%d", &nbElt);
                  tab = (int *)malloc(nbElt*sizeof(int));
                  printf("Donner les elements du tableau:\n");
                  for(int i=0; i<nbElt; i++){
                     printf("Elt %d : ", i);
                     scanf("%d", (tab+i));
                  }
                  p=tabVersLinkedList(tab, nbElt);
                  printf("\nListe chainee resultat : \n");
                  AfficherListe(p);
                  break;
        case 12 : printf("Donner le rang de la donnee a chercher : ");
                  scanf("%d", &rang); getchar();
                  printf("la donnee qui se trouve au rang %d dans la liste est %d\n", rang, kieme(debut,
rang));
                  break;
        case 13 : afficheInverseRecursive(debut);
                  break;
        case 14 : printf("Donner la donnee a supprimer : ");
                  scanf("%d", &donnee);
                  debut = supprimerPremiereOccurrenceRecursive(debut, donnee);
```

```
                break;
        case 15 : printf("Donner la donnee a supprimer : ");
                scanf("%d", &donnee);
                debut = supprimerPremiereOccurrenceIteratif(debut, donnee);
                break;
        case 16 : printf("Donner la donnee a supprimer : ");
                scanf("%d", &donnee);
                debut = supprimerToutеRecursive(debut, donnee);
                break;
        case 17 : printf("Donner la donnee a supprimer : ");
                scanf("%d", &donnee);
                debut = supprimerToutеIteratif(debut, donnee);
                break;
        case 18 : debut=supprimerRepetitionRecursive(debut);
                break;
        case 19 : debut=supprimerRepetitionIterative(debut);
                break;

      }
   }while(choix !=20);

   return 0;
}
```