```c
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
//pour afficher les colour
#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define RESET "\033[0m"

// cette foncion  verifie que mot 1 inferieur a mot 2 on ordre alphabetique
bool inf(char * mot1,char * mot2)
{
int i=-1;
        do{i++;
                if(mot1[i]<mot2[i]) return 1;
                if(mot1[i]>mot2[i]) return 0;
        }while(mot1[i]!=0 && mot2[i]!=0);
return 0;
}
// fonction de permutaion entre deux mots
void permut(char * mot1,char * mot2)
{
        int i;
        char c;
        for(i=0;i<80;i++)
        {
        c=mot1[i];
        mot1[i]=mot2[i];
        mot2[i]=c;
        }
}
// cette foncion  verifie que deux mot sont egaux
bool equal(char* mot1,char * mot2)
{
int i=-1;
        do{i++;
        if(mot1[i]!=mot2[i]) return 0;
        }while(mot1[i]!=0 && mot2[i]!=0);
return 1;
}
// cette foncion  verifie que le tableau est on ordre alphabetique
bool ordr(char** Y,int N)
        {
        int i;
        for(i=0;i<N-1;i++)
        if (!(inf(Y[i],Y[i+1]) || equal(Y[i],Y[i+1])) ) return 0;
        return 1;
        }
//la foction de la recherche  séquentielle
int fctsec(char** Y,int N,char* x)
        {
        int i;
        for(i=0;i<N;i++)
        if (equal(Y[i],x)) return i;
        return -1;
        }
//la foction de la recherche  dichotomique
int fctdich(char** Y,int N,char* x)
{
int g=0,d=N-1,m;
do
{
        m=(g+d)/2;
        //printf("g=%i d=%i m=%i \n",g,d,m);
        if(equal(Y[m],x)) return m;
```

```
        if(inf(x,Y[m])) d=m-1;
        else g=m+1;
}while(g<=d);
return -1;
}
//la foction d'indexation 1 foctione bien mais elle est abondoné car elle foctione seulement avec les mots a caracter muniscule
/*
//la foction d'indexation 1
int* findex1(char** T,int N)
{       int* index=malloc(26*sizeof(int));
        int h=0,i;
        index[0]=0;
        index[26]=N;
        for(i=0;i<N-1;i++)
        {
                printf("\r %.2f ",(1.0*i/(N-2))*100 );
                if (T[i][0]!= T[i+1][0])
                {h++;
                index[h]=i+1;
                }
        }
return index;
}
//la foction de la recherche  par indexation 1
int fctind(char** Y,int N,char* x,int* Index)
{
int i;
        for(i=Index[(x[0]-97)];i<(Index[(x[0]-96)]);i++)
        if (equal(Y[i],x)) return i;
        return -1;
}
*/
//*************************************//
struct lim{
int deb;
int fin;
};
typedef struct lim limit;
//la foction d'indexation 2
limit* findex2(char** T,int N)
{       int i;
        limit* index=malloc(126*sizeof(limit));
        index[(T[1][0])].deb=0;
        index[(T[N-1][0])].fin=N-1;
        for(i=0;i<N-1;i++)
        {
                printf("\r %.2f ",(1.0*i/(N-2))*100 );
                if (T[i][0]!= T[i+1][0])
                {
                index[(T[i][0])].fin=i;
                index[(T[i+1][0])].deb=i+1;
                }
        }
return index;
}


//la foction de la recherche  par indexation 2
int fctind2(char** Y,int N,char* x,limit* Index)
{
int i;
        if(Index[(x[0])].fin==0) return -1;
        for(i=Index[(x[0])].deb;i<(Index[(x[0])].fin);i++)
        if (equal(Y[i],x)) return i;
        return -1;
}
//la fonction qui associé a un mot un chifre
long long int mchif(char* mt)
{
```

```c
long long int ch=0,po;
int i,j;
        for(i=0;mt[i]!=0 && i<6;i++)
        {
                po=1;
                for(j=0;j<5-i;j++)
                {po=po*126;}
        ch=ch+(mt[i])*po;
        }
return ch;
}
//la foction de la recherche  par interpolation
int fctintp(char** Y,int N,char* x)
{
int i,ist=1.0*(mchif(x)-mchif(Y[0]))/(mchif(Y[N-1])-mchif(Y[0]))*(N-1);
//printf("ist= %i \n",ist);
if(ist<0 || ist>N-1) return -1;
if(equal(Y[ist],x)) return ist;
if(inf(x,Y[ist]))
        {
        for(i=ist-1;(inf(x,Y[i]) || equal(Y[i],x))&& i>-1;i--)
        if (equal(Y[i],x)) return i;
        return -1;
        }
else
{
        for(i=ist-1;(inf(Y[i],x) || equal(Y[i],x))&& i<N;i++)
        if (equal(Y[i],x)) return i;
        return -1;
        }
}
//la foction de la recherche  par istimation
int fctistm(char** Y,int N,char* x)
{
long long int star=0,end=N-1,istm;
istm=1.0*(mchif(x)-mchif(Y[star]))*(end-star)/(mchif(Y[end])-mchif(Y[star]))+star+0.5;
do
{       /*istm=1.0*(mchif(x)-mchif(Y[star]))*(end-star)/(mchif(Y[end])-mchif(Y[star]))+star+0.5;
        printf("g=%lli d=%lli m=%lli \n",star,end,istm);*/
        if(equal(Y[istm],x)) return istm;
        if(inf(x,Y[istm])) end=istm-1;
        else star=istm+1;
        istm=(star+end)/2;
}while(star<=end);
return -1;
}

main ()
{
        char mot[80],A[80],B[80],c;
        int n=0,i=0,j;
        FILE* dicr=fopen("persn.txt","r+");
        FILE* defr=fopen("num.txt","r+");
        if(dicr==NULL || defr==NULL)
        {
                printf("ereur : impossible de d'acces aux données\n");
                exit(1);
        }
// affichage des ficher
        while (!feof(dicr))
        {
        fscanf(dicr, "%s\n", A);
        fscanf(defr, "%s\n", B);
        printf(" mot(%d) || %s ---> %s \n",i,A,B);
        /*printf(" mot(%d) || %s || %lli ||---> %s \n",i,A,mchif(A),B);*/
        i++;
        }
        n=i;
        char** dic=malloc(n*sizeof(char*));
        char** def=malloc(n*sizeof(char*));
```

```
        for(i=0;i<n;i++)
        {dic[i]=malloc(80*sizeof(char));
         def[i]=malloc(80*sizeof(char));}
// saisie les donnes dans la ram
        fseek(dicr,0,SEEK_SET);
        fseek(defr,0,SEEK_SET);
        i=0;
        while (!feof(dicr))
        {
        fscanf(dicr, "%s\n", dic[i]);
        fscanf(defr, "%s\n", def[i]);
        i++;
        }
// traitement
        if(ordr(dic,n))
        {
                printf("les doonez sont on ordre alphabetique\n");
        }
        else
        {
                printf("les doonez ne sont pas on ordre alphabetique\n");
                printf("         \% rendre en ordre alphabetique");
                for(i=0; i<n-1; i++ )
                {
                printf("\r %.2f ",(1.0*i/(n-2))*100 );
                for(j=i+1; j<n; j++ )
                {
                        if(inf(dic[j],dic[i]))
                        {
                        permut(dic[i],dic[j]);
                        permut(def[i],def[j]);
                        }
                }
                }
//reconstruction des fichier
                fseek(dicr,0,SEEK_SET);
                fseek(defr,0,SEEK_SET);
                i=0;
                for (i=0;i<n;i++)
                {
                fprintf(dicr, "%s\n", dic[i]);
                fprintf(defr, "%s\n", def[i]);
                }
//afichage des fichiers
                i=0;
                fseek(dicr,0,SEEK_SET);
                fseek(defr,0,SEEK_SET);
                printf("\n");
                while (!feof(dicr))
                {
                fscanf(dicr, "%s\n", A);
                fscanf(defr, "%s\n", B);
                printf(" mot(%d) || %s ---> %s \n",i,A,B);
                i++;
                }
        }
/*
//indexation:
        printf("        \% , (Indexation 1)");
        int* index=findex1(dic,n);
//afichage de lindex
        printf("\n_____________________________________________________________\n");
        for(i=0;i<26;i++)
        printf("%c ---> %i \n",i+97,index[i]);*/
//indexation 2:
        printf("        \% , (Indexation 2)");
        limit* index2=findex2(dic,n);
//afichage de lindex
        printf("\n_____________________________________________________________\n");
        for(i=32;i<126;i++)
```

```
280          printf("%c ---> [ %i , %i ] \n",i,index2[i].deb,index2[i].fin);
281  // la recheche dans le tableau:
282          do{
283                  printf("\ndonnez le mot a cherché : ");
284                  scanf("%s",mot);
285                  printf("recherche dicotomique :\n");
286                  int m=fctdich(dic,n,mot);
287                  if(m==-1) printf(KRED "le mot n'existe pas\n" RESET);
288                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,m,def[m]);
289                  printf("recherche séquentielle :\n");
290                  int z=fctsec(dic,n,mot);
291                  if(z==-1) printf(KRED "le mot n'existe pas\n" RESET);
292                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,z,def[z]);
293                  /*printf("recherche séquentielle indexé:\n");
294                  int k=fctind(dic,n,mot,index);
295                  if(k==-1) printf("le mot n'existe pas\n");
296                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,k,def[k]);*/
297                  printf("recherche par séquentielle indexé 2:\n");
298                  int k2=fctind2(dic,n,mot,index2);
299                  if(k2==-1) printf(KRED "le mot n'existe pas\n" RESET);
300                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,k2,def[k2]);
301                  printf("recherche par interpolation :\n");
302                  int f=fctintp(dic,n,mot);
303                  if(f==-1) printf(KRED "le mot n'existe pas\n" RESET);
304                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,f,def[f]);
305                  printf("recherche par istimation :\n");
306                  int l=fctistm(dic,n,mot);
307                  if(l==-1) printf(KRED "le mot n'existe pas\n" RESET);
308                  else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,l,def[l]);
309                  printf("est ce que vous vouler cherché un autre mot ? (taper O/N pour Non) : ");
310                  scanf("%c",&c);
311                  scanf("%c",&c);
312          }while(c!='N');
313
314  }
```