```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
{
```

```
68          m=(g+d)/2;
69          if(equal(Y[m],x)) return m;
70          if(inf(x,Y[m])) d=m-1;
71          else g=m+1;
72  }while(g<=d);
73  return -1;
74  }
75  //la foction d'indexation 1 foctione bien mais elle est abondoné car elle foctione seulement avec
    les mots a caracter muniscule
76  /*
77  //la foction d'indexation 1
78  int* findex1(char** T,int N)
79  {       int* index=malloc(26*sizeof(int));
80          int h=0,i;
81          index[0]=0;
82          index[26]=N;
83          for(i=0;i<N-1;i++)
84          {
85                  printf("\r %.2f ",(1.0*i/(N-2))*100 );
86                  if (T[i][0]!= T[i+1][0])
87                  {h++;
88                  index[h]=i+1;
89                  }
90          }
91  return index;
92  }
93  //la foction de la recherche  par indexation 1
94  int fctind(char** Y,int N,char* x,int* Index)
95  {
96  int i;
97          for(i=Index[(x[0]-97)];i<(Index[(x[0]-96)]);i++)
98          if (equal(Y[i],x)) return i;
99          return -1;
100 }
101 */
102 //*********************************//
103 struct lim{
104 int deb;
105 int fin;
106 };
107 typedef struct lim limit;
108 //la foction d'indexation 2
109 limit* findex2(char** T,int N)
110 {       int i;
111         limit* index=malloc(126*sizeof(limit));
112         index[(T[1][0])].deb=0;
113         index[(T[N-1][0])].fin=N-1;
114         for(i=0;i<N-1;i++)
115         {
116                 printf("\r %.2f ",(1.0*i/(N-2))*100 );
117                 if (T[i][0]!= T[i+1][0])
118                 {
119                 index[(T[i][0])].fin=i;
120                 index[(T[i+1][0])].deb=i+1;
121                 }
122         }
123 return index;
124 }
125
126
127 //la foction de la recherche  par indexation 2
128 int fctind2(char** Y,int N,char* x,limit* Index)
129 {
130 int i;
131         if(Index[(x[0])].fin==0) return -1;
132         for(i=Index[(x[0])].deb;i<(Index[(x[0])].fin);i++)
133         if (equal(Y[i],x)) return i;
```

```
134         return -1;
135 }
136 //la fonction qui associé a un mot un chifre
137 int mchif(char* mt)
138 {
139 int ch=0,po;
140 int i,j;
141         for(i=0;mt[i]!=0 && i<4;i++)
142         {
143                 po=1;
144                 for(j=0;j<3-i;j++)
145                 po=po*126;
146         ch=ch+(mt[i])*po;
147         }
148 return ch;
149 }
150 //la foction de la recherche  par interpolation
151 int fctintp(char** Y,int N,char* x)
152 {
153 int i,ist=1.0*(mchif(x)-mchif(Y[0]))/(mchif(Y[N-1])-mchif(Y[0]))*(N-1);
154 printf("ist= %i \n",ist);
155 if(ist<0 || ist>N-1) return -1;
156 if(equal(Y[ist],x)) return ist;
157 if(inf(x,Y[ist]))
158         {
159         for(i=ist-1;(inf(x,Y[i]) || equal(Y[i],x))&& i>-1;i--)
160         if (equal(Y[i],x)) return i;
161         return -1;
162         }
163 else
164 {
165         for(i=ist-1;(inf(Y[i],x) || equal(Y[i],x))&& i<N;i++)
166         if (equal(Y[i],x)) return i;
167         return -1;
168         }
169 }
170
171 main ()
172 {
173         char mot[80],A[80],B[80],c;
174         int n=0,i=0,j;
175         FILE* dicr=fopen("persn.txt","r+");
176         FILE* defr=fopen("num.txt","r+");
177         if(dicr==NULL || defr==NULL)
178         {
179                 printf("ereur : impossible de d'acces aux données\n");
180                 exit(1);
181         }
182 // affichage des ficher
183         while (!feof(dicr))
184         {
185         fscanf(dicr, "%s\n", A);
186         fscanf(defr, "%s\n", B);
187         printf(" mot(%d) || %s ---> %s \n",i,A,B);
188         /*printf(" mot(%d) || %s || %i ||---> %s \n",i,A,mchif(A),B);*/
189         i++;
190         }
191         n=i;
192         char** dic=malloc(n*sizeof(char*));
193         char** def=malloc(n*sizeof(char*));
194         for(i=0;i<n;i++)
195         {dic[i]=malloc(80*sizeof(char));
196          def[i]=malloc(80*sizeof(char));}
197 // saisie des dans la ram
198         fseek(dicr,0,SEEK_SET);
199         fseek(defr,0,SEEK_SET);
200         i=0;
```

```
201          while (!feof(dicr))
202          {
203          fscanf(dicr, "%s\n", dic[i]);
204          fscanf(defr, "%s\n", def[i]);
205          i++;
206          }
207  // traitement
208          if(ordr(dic,n))
209          {
210                  printf("les doonez sont on ordre alphabetique\n");
211          }
212          else
213          {
214                  printf("les doonez ne sont pas on ordre alphabetique\n");
215                  printf("         \% rendre en ordre alphabetique");
216                  for(i=0; i<n-1; i++ )
217                  {
218                  printf("\r %.2f ",(1.0*i/(n-2))*100 );
219                  for(j=i+1; j<n; j++ )
220                  {
221                          if(inf(dic[j],dic[i]))
222                          {
223                          permut(dic[i],dic[j]);
224                          permut(def[i],def[j]);
225                          }
226                  }
227                  }
228  //reconstruction des fichier
229                  fseek(dicr,0,SEEK_SET);
230                  fseek(defr,0,SEEK_SET);
231                  i=0;
232                  for (i=0;i<n;i++)
233                  {
234                  fprintf(dicr, "%s\n", dic[i]);
235                  fprintf(defr, "%s\n", def[i]);
236                  }
237  //afichage des fichiers
238                  i=0;
239                  fseek(dicr,0,SEEK_SET);
240                  fseek(defr,0,SEEK_SET);
241                  printf("\n");
242                  while (!feof(dicr))
243                  {
244                  fscanf(dicr, "%s\n", A);
245                  fscanf(defr, "%s\n", B);
246                  printf(" mot(%d) || %s ---> %s \n",i,A,B);
247                  i++;
248                  }
249          }
250  /*
251  //indexation:
252          printf("        \% , (Indexation 1)");
253          int* index=findex1(dic,n);
254  //afichage de lindex
255          printf("\n_____________________________________________________\n");
256          for(i=0;i<26;i++)
257          printf("%c ---> %i \n",i+97,index[i]);*/
258  //indexation 2:
259          printf("        \% , (Indexation 2)");
260          limit* index2=findex2(dic,n);
261  //afichage de lindex
262          printf("\n_____________________________________________________\n");
263          for(i=32;i<126;i++)
264          printf("%c ---> [ %i , %i ] \n",i,index2[i].deb,index2[i].fin);
265  // la recheche dans le tableau:
266          do{
267                  printf("\ndonnez le mot a cherché : ");
```

```c
                scanf("%s",mot);
                printf("recherche dicotomique :\n");
                int m=fctdich(dic,n,mot);
                if(m==-1) printf(KRED "le mot n'existe pas\n" RESET);
                else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,m,def[m]);
                printf("recherche séquentielle :\n");
                int z=fctsec(dic,n,mot);
                if(z==-1) printf(KRED "le mot n'existe pas\n" RESET);
                else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,z,def[z]);
                /*printf("recherche séquentielle indexé:\n");
                int k=fctind(dic,n,mot,index);
                if(k==-1) printf("le mot n'existe pas\n");
                else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,k,def[k]);*/
                printf("recherche par séquentielle indexé 2:\n");
                int k2=fctind2(dic,n,mot,index2);
                if(k2==-1) printf(KRED "le mot n'existe pas\n" RESET);
                else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,k2,def[k2]);
                printf("recherche par interpolation :\n");
                int f=fctintp(dic,n,mot);
                if(f==-1) printf(KRED "le mot n'existe pas\n" RESET);
                else printf("l'indice de '%s' est : %d --En-->> %s \n",mot,f,def[f]);
                printf("est ce que vous vouler cherché un autre mot ? (taper O/N pour Non) : ");
                scanf("%c",&c);
                scanf("%c",&c);
        }while(c!='N');

}
```