

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<stdbool.h>
4  //pour afficher les colour
5  #define KNRM "\x1B[0m"
6  #define KRED "\x1B[31m"
7  #define KGRN "\x1B[32m"
8  #define KYEL "\x1B[33m"
9  #define KBLU "\x1B[34m"
10 #define KMAG "\x1B[35m"
11 #define KCYN "\x1B[36m"
12 #define KWHT "\x1B[37m"
13 #define RESET "\033[0m"
14
15 // cette fonction verifie que mot 1 inferieur a mot 2 on ordre alphabetique
16 bool inf(char * mot1,char * mot2)
17 {
18     int i=-1;
19     do{i++;
20         if(mot1[i]<mot2[i]) return 1;
21         if(mot1[i]>mot2[i]) return 0;
22     }while(mot1[i]!=0 && mot2[i]!=0);
23     return 0;
24 }
25 // fonction de permutaion entre deux mots
26 void permut(char * mot1,char * mot2)
27 {
28     int i;
29     char c;
30     for(i=0;i<80;i++)
31     {
32         c=mot1[i];
33         mot1[i]=mot2[i];
34         mot2[i]=c;
35     }
36 }
37 // cette fonction verifie que deux mot sont egaux
38 bool equal(char* mot1,char * mot2)
39 {
40     int i=-1;
41     do{i++;
42         if(mot1[i]!=mot2[i]) return 0;
43     }while(mot1[i]!=0 && mot2[i]!=0);
44     return 1;
45 }
46 // cette fonction verifie que le tableau est on ordre alphabetique
47 bool ord(char** Y,int N)
48 {
49     int i;
50     for(i=0;i<N-1;i++)
51         if (!(inf(Y[i],Y[i+1]) || equal(Y[i],Y[i+1])) ) return 0;
52     return 1;
53 }
54 //la foction de la recherche séquentielle
55 int fctsec(char** Y,int N,char* x)
56 {
57     int i;
58     for(i=0;i<N;i++)
59         if (equal(Y[i],x)) return i;
60     return -1;
61 }
62 //la foction de la recherche dichotomique
63 int fctdich(char** Y,int N,char* x)
64 {
65     int g=0,d=N-1,m;
66     do
67     {
68         m=(g+d)/2;
69         //printf("g=%i d=%i m=%i \n",g,d,m);
70         if(equal(Y[m],x)) return m;

```

```

71         if(inf(x,Y[m])) d=m-1;
72         else g=m+1;
73     }while(g<=d);
74     return -1;
75 }
76 //la foction d'indexation 1 foctione bien mais elle est abandonné car elle foctione seulement avec
les mots a caracter muniscule
77 /*
78 //la foction d'indexation 1
79 int* findex1(char** T,int N)
80 {
81     int* index=malloc(26*sizeof(int));
82     int h=0,i;
83     index[0]=0;
84     index[26]=N;
85     for(i=0;i<N-1;i++)
86     {
87         printf("\r %.2f ",(1.0*i/(N-2))*100 );
88         if (T[i][0]!= T[i+1][0])
89         {h++;
90          index[h]=i+1;
91         }
92     }
93     return index;
94 }
95 //la foction de la recherche par indexation 1
96 int fctind(char** Y,int N,char* x,int* Index)
97 {
98     int i;
99     for(i=Index[(x[0]-97)];i<(Index[(x[0]-96)]);i++)
100     if (equal(Y[i],x)) return i;
101     return -1;
102 }
103 /*
104 //*****//
105 struct lim{
106     int deb;
107     int fin;
108 };
109 typedef struct lim limit;
110 //la foction d'indexation 2
111 limit* findex2(char** T,int N)
112 {
113     int i;
114     limit* index=malloc(126*sizeof(limit));
115     index[(T[1][0]).deb]=0;
116     index[(T[N-1][0]).fin]=N-1;
117     for(i=0;i<N-1;i++)
118     {
119         printf("\r %.2f ",(1.0*i/(N-2))*100 );
120         if (T[i][0]!= T[i+1][0])
121         {
122             index[(T[i][0]).fin]=i;
123             index[(T[i+1][0]).deb]=i+1;
124         }
125     }
126     return index;
127 }
128 //la foction de la recherche par indexation 2
129 int fctind2(char** Y,int N,char* x,limit* Index)
130 {
131     int i;
132     if(Index[(x[0]).fin]==0) return -1;
133     for(i=Index[(x[0]).deb];i<(Index[(x[0]).fin]);i++)
134     if (equal(Y[i],x)) return i;
135     return -1;
136 }
137 //la fonction qui associé a un mot un chiffre
138 long long int mchif(char* mt)
139 {

```

```

140 long long int ch=0,po;
141 int i,j;
142 for(i=0;mt[i]!=0 && i<6;i++)
143 {
144     po=1;
145     for(j=0;j<5-i;j++)
146     {po=po*126;}
147     ch=ch+(mt[i])*po;
148 }
149 return ch;
150 }
151 //la foction de la recherche par interpolation
152 int fctintp(char** Y,int N,char* x)
153 {
154     int i,ist=1.0*(mchif(x)-mchif(Y[0]))/(mchif(Y[N-1])-mchif(Y[0]))*(N-1);
155     //printf("ist= %i \n",ist);
156     if(ist<0 || ist>N-1) return -1;
157     if(equal(Y[ist],x)) return ist;
158     if(inf(x,Y[ist]))
159     {
160         for(i=ist-1;(inf(x,Y[i]) || equal(Y[i],x))&& i>-1;i--)
161             if (equal(Y[i],x)) return i;
162         return -1;
163     }
164     else
165     {
166         for(i=ist-1;(inf(Y[i],x) || equal(Y[i],x))&& i<N;i++)
167             if (equal(Y[i],x)) return i;
168         return -1;
169     }
170 }
171 //la foction de la recherche par istimation
172 int fctistm(char** Y,int N,char* x)
173 {
174     long long int star=0,end=N-1,istm;
175     istm=1.0*(mchif(x)-mchif(Y[star]))*(end-star)/(mchif(Y[end])-mchif(Y[star]))+star+0.5;
176     do
177     {
178         /*istm=1.0*(mchif(x)-mchif(Y[star]))*(end-star)/(mchif(Y[end])-mchif(Y[star]))+star+0.5;
179         printf("g=%lli d=%lli m=%lli \n",star,end,istm);*/
180         if(equal(Y[istm],x)) return istm;
181         if(inf(x,Y[istm])) end=istm-1;
182         else star=istm+1;
183         istm=(star+end)/2;
184     }while(star<=end);
185     return -1;
186 }
187 main ()
188 {
189     char mot[80],A[80],B[80],c;
190     int n=0,i=0,j;
191     FILE* dicr=fopen("persn.txt","r+");
192     FILE* defr=fopen("num.txt","r+");
193     if(dicr==NULL || defr==NULL)
194     {
195         printf("erreur : impossible de d'accès aux données\n");
196         exit(1);
197     }
198     // affichage des fichier
199     while (!feof(dicr))
200     {
201         fscanf(dicr, "%s\n", A);
202         fscanf(defr, "%s\n", B);
203         printf(" mot(%d) || %s ---> %s \n",i,A,B);
204         /*printf(" mot(%d) || %s || %lli ||---> %s \n",i,A,mchif(A),B);*/
205         i++;
206     }
207     n=i;
208     char** dic=malloc(n*sizeof(char*));
209     char** def=malloc(n*sizeof(char*));

```

```

210     for(i=0;i<n;i++)
211     {dic[i]=malloc(80*sizeof(char));
212     def[i]=malloc(80*sizeof(char));}
213 // saisie les donnees dans la ram
214     fseek(dicr,0,SEEK_SET);
215     fseek(defr,0,SEEK_SET);
216     i=0;
217     while (!feof(dicr))
218     {
219         fscanf(dicr, "%s\n", dic[i]);
220         fscanf(defr, "%s\n", def[i]);
221         i++;
222     }
223 // traitement
224     if(ordr(dic,n))
225     {
226         printf("les doonez sont on ordre alphabetique\n");
227     }
228     else
229     {
230         printf("les doonez ne sont pas on ordre alphabetique\n");
231         printf("        \n rendre en ordre alphabetique");
232         for(i=0; i<n-1; i++ )
233         {
234             printf("\r %.2f ",(1.0*i/(n-2))*100 );
235             for(j=i+1; j<n; j++ )
236             {
237                 if(inf(dic[j],dic[i]))
238                 {
239                     permut(dic[i],dic[j]);
240                     permut(def[i],def[j]);
241                 }
242             }
243         }
244 //reconstruction des fichier
245         fseek(dicr,0,SEEK_SET);
246         fseek(defr,0,SEEK_SET);
247         i=0;
248         for (i=0;i<n;i++)
249         {
250             fprintf(dicr, "%s\n", dic[i]);
251             fprintf(defr, "%s\n", def[i]);
252         }
253 //affichage des fichiers
254         i=0;
255         fseek(dicr,0,SEEK_SET);
256         fseek(defr,0,SEEK_SET);
257         printf("\n");
258         while (!feof(dicr))
259         {
260             fscanf(dicr, "%s\n", A);
261             fscanf(defr, "%s\n", B);
262             printf(" mot(%d) || %s ---> %s \n",i,A,B);
263             i++;
264         }
265     }
266 /*
267 //indexation:
268     printf("        \n , (Indexation 1)");
269     int* index=findex1(dic,n);
270 //affichage de lindex
271     printf("\n_____ \n");
272     for(i=0;i<26;i++)
273     printf("%c ---> %i \n",i+97,index[i]);*/
274 //indexation 2:
275     printf("        \n , (Indexation 2)");
276     limit* index2=findex2(dic,n);
277 //affichage de lindex
278     printf("\n_____ \n");
279     for(i=32;i<126;i++)

```

```

280     printf("%c ---> [ %i , %i ] \n",i,index2[i].deb,index2[i].fin);
281 // la recherche 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 0/N pour Non) : ");
310         scanf("%c",&c);
311         scanf("%c",&c);
312     }while(c!='N');
313
314 }

```