

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<time.h>
4  #include<math.h>
5  #include "testkeytime.h"
6  // #include "keytime.h"
7  #include<string.h>
8  #include<stdbool.h>
9
10 #define MAX_GEN 700      //最大世代交代    //文字数によって変える
11 #define POP_SIZE 100     //集団のサイズ
12 #define LEN_KEYS 30      //遺伝子の長さ
13 #define GEN_GAP 0.1      //世代交代の割合
14 #define P_MUTATION 0.05   //突然変異の確率
15 #define P_CROSSOVER 0.8   //交叉率
16 #define RANDOM_MAX 32767
17 #define BEFORE 0
18 #define AFTER 1
19
20 char name[256];
21 int keyboards[POP_SIZE][LEN_KEYS]; //染色体(キーボード配列)
22 int fitness[POP_SIZE];              //適合度
23 int max,min,sumfitness;              //適合度の,max,min,sum
24 int n_min;                          //適合度のminの添字
25 int n_max;                          //適合度のmaxの添字
26
27 void PrintKeyboardFitness();
28 void Statistics();
29 void Crossover(int parent1,int parent2,int *child1, int *child2);
30 void Mutation(int child);
31 int ObjFunc(int i);
32 int Select();
33 void filewrite(int keyboard[],char* phase);
34
35 #define EMPTY -2
36 #define Used -1
37 #define A 0
38 #define B 1
39 #define C 2
40 #define D 3
41 #define E 4
42 #define F 5
43 #define G 6
44 #define H 7
45 #define I 8
46 #define J 9
47 #define K 10
48 #define L 11
49 #define M 12
50 #define N 13
51 #define O 14
52 #define P 15
53 #define Q 16
54 #define R 17
55 #define S 18
56 #define T 19
57 #define U 20

```

```

58  #define V 21
59  #define W 22
60  #define X 23
61  #define Y 24
62  #define Z 25
63  #define Others1 26
64  #define Others2 27
65  #define Others3 28
66  #define Others4 29
67
68  int key_options[LEN_KEYS];    //配置可能なキー
69  int STRINGS = 0;
70  char str[256][10000] = {};   //日本語文字列
71  char alphabet[30] =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T',
',','U','V','W','X','Y','Z','.', '<','>','?'};
72
73  //擬似乱数
74  static unsigned long int next = 1;
75
76  int Rand(void){
77      next = next*1103515245 + 12345;
78      return (unsigned int)(next/65536)%(RANDOM_MAX+1);
79  }
80
81  void Srand(unsigned int seed){
82      next = seed;
83  }
84
85  void init_key_options(){
86      int i;
87      for(i=0;i<LEN_KEYS;i++){
88          key_options[i] = i;
89      }
90  }
91
92  //配列を空に（キー割り当てなしに）
93  void be_empty(int i){
94      int j;
95      for(j=0;j<LEN_KEYS;j++){
96          keyboards[i][j] = EMPTY;
97      }
98
99  //初期データ設定
100 void Initialize(){
101     int i,j,n;
102
103     for(i=0;i<POP_SIZE;i++){
104         init_key_options();
105         for(j=0;j<LEN_KEYS;j++){
106             n = Rand()%LEN_KEYS;
107             while(key_options[n]==Used){n++;if(n>LEN_KEYS-1)n=A;} //まだ使われ
ていないキーの検索
108             keyboards[i][j] = key_options[n];
109             key_options[n] = Used;
110         }
111         fitness[i] = ObjFunc(i);
112     }

```

```

113
114     printf("First Position¥n");
115     PrintKeyboardFitness();
116     printf("-----¥n");
117 }
118
119 //データ表示関数
120 void PrintEachKeyboardFitness(int i){
121     int j;
122     printf("[%d] ",i);
123     for(j=0;j<LEN_KEYS;j++){
124         printf("%d,",keyboards[i][j]);
125     }
126     printf(":%d¥n",fitness[i]);
127 }
128
129 void PrintKeyboardFitness(){
130     int i;
131     for(i=0;i<POP_SIZE;i++)PrintEachKeyboardFitness(i);
132 }
133
134 void PrintStatistics(int gen){
135     if(gen==-1){
136         printf("[gen=%dend] max=%d min=%d sumfitness=%d ave=%f¥n",
137             MAX_GEN,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
138     }else{
139         printf("[gen=%2d] max=%d min=%d sumfitness=%d ave=%f¥n",
140             gen,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
141     }
142 }
143
144 void PrintCrossover(int flag,int parent1,int parent2,int child1,int
child2,int n_cross1, int n_cross2){
145     switch(flag){
146     case BEFORE:
147         printf("parent1 |");PrintEachKeyboardFitness(parent1);
148         printf("parent2 |");PrintEachKeyboardFitness(parent2);
149         printf("delete1 |");PrintEachKeyboardFitness(child1);
150         printf("delete2 |");PrintEachKeyboardFitness(child2);
151         printf("n_cross1=%d¥n",n_cross1);
152         printf("n_cross2=%d¥n",n_cross2);
153
154         break;
155     case AFTER:
156         printf("child1 |");PrintEachKeyboardFitness(child1);
157         printf("child2 |");PrintEachKeyboardFitness(child2);
158         printf("-----¥n");
159         break;
160     }
161 }
162
163 void PrintMutation(int flag,int child,int n_mutate1,int n_mutate2){
164     switch(flag){
165     case BEFORE:
166         printf("child(OLD)|");PrintEachKeyboardFitness(child);
167         printf("n_mutate1=%d <--> n_mutate2=%d¥n",n_mutate1,n_mutate2);
168         break;
169     case AFTER:

```

```

170     printf("child(NEW) |"); PrintEachKeyboardFitness(child);
171     printf("-----¥n");
172     break;
173 }
174 }
175
176 //世代の処理
177 void Generation(int gen){
178     int parent1,parent2;
179     int child1,child2;
180     int n_gen;
181     int i,j;
182     int parent_options[POP_SIZE] = {}; //選択できる親の候補
183     int min2;
184
185     //集団の表示
186     Statistics();
187     PrintStatistics(gen);
188
189     //世代交代
190     n_gen=(int)((double)POP_SIZE*GEN_GAP/2.0);
191     for(i=0;i<n_gen;i++){
192         Statistics();
193
194         //1番小さい値を子供としてセット
195         child1 = n_min;
196         //2番目に小さい値を見つける
197         min2 = 2147483647; //int最大値
198         for(j=0;j<POP_SIZE;j++){
199             if(j!=child1){
200                 if(min<=fitness[j]&&fitness[j]<min2){
201                     min2 = fitness[j]; child2 = j;
202                 }
203             }
204         }
205         parent_options[child1] = 1;
206         parent_options[child2] = 1;
207         parent1 = Select(parent_options);
208         parent_options[parent1] = 1;
209         parent2 = Select(parent_options);
210         parent_options[parent2] = 1;
211         Crossover(parent1,parent2,&child1,&child2);
212         Mutation(child1);
213         Mutation(child2);
214     }
215 }
216
217 /*今回考えない文字除外関数*/
218 int is_keyword(char c){
219     for (int i=0;i<30;i++){
220         if(c==alphabet[i]) return 1;
221     }
222     return 0;
223 }
224
225 int is_index_finger(int b, int c){
226     if(b==3 && c==4) return 1;
227     if(b==4 && c==3) return 1;

```

```

228     if(b==5 && c==6) return 1;
229     if(b==6 && c==5) return 1;
230     if(b==6 && c==5) return 1;
231     if(b==13 && c==14) return 1;
232     if(b==14 && c==13) return 1;
233     if(b==15 && c==16) return 1;
234     if(b==16 && c==15) return 1;
235     if(b==23 && c==24) return 1;
236     if(b==24 && c==23) return 1;
237     if(b==25 && c==26) return 1;
238     if(b==26 && c==25) return 1;
239     return 0;
240 }
241
242 //目的関数(各文字列sを打つときに指が移動した回数/文字数 が少ない方が優れている(指
ごとに重み付け?))
243 //考える
244 //現状：ホームポジションにあるキーが入力されたらcount++(簡単だったから)
245 int ObjFunc(int i){
246
247     int j,k,bk = 0;
248     int ck = -1;
249     int count = 0; //指が移動してしまった回数
250     int point = 0; //返す評価値
251     for(j=0;j<STRINGS;j++){
252         int n = 0; //文字列の添字
253         int s = 0; //有効な文字のカウント
254         while(str[j][n]!='¥0'){
255             if(is_keyword(str[j][n])){ //今回考えるキーか確認
256                 if(!(n!=0 && str[j][n]==str[j][n-1])){ //1つ前の文字と同じ時はカウン
トしない
257                     bk = ck;
258                     for(k=0;k<=29;k++){
259                         if(alphabet[keyboards[i][k]]==str[j][n]){
260                             ck=k;
261                             break;
262                         }
263                     }
264                     if(k!=30){
265                         if(is_index_finger(bk, k)){count += keyweight[k]/2;}
266                         else{count += keyweight[k];}
267                     }
268                     s++;
269                 }
270             }
271             n++;
272         }
273         point += n*10000; //文字数*10000ポイント加算←全部10秒かかった設定
274     }
275     point -= count;
276     return point; / (全文字数*10000-カウント数) が最終ポイント
277 }
278
279 //fitnessの合計値の計算
280 void Statistics(){
281     int i;

```

```

282
283     max = 0;
284     min = 2147483647; //int最大値
285     sumfitness = 0;
286
287     for(i=0;i<POP_SIZE;i++){
288         if(fitness[i]>max){
289             max = fitness[i];
290             n_max = i;
291         }
292         if(fitness[i]<min){
293             min = fitness[i];
294             n_min = i;
295         }
296         sumfitness += fitness[i];
297     }
298 }
299
300 //選択
301 //ルーレット→ランキング
302 int Select(int parent_options[]){
303     int i,j,tmp,rand_n;
304     double rand;
305     int fit_rank[POP_SIZE];
306     double fit_rank_rate[POP_SIZE] = {};
307     double max_rate = 0.8;
308     int rank_limit = 50; //ランク付けする個体数（残りは確率0）
309
310     for(i=0;i<POP_SIZE;i++){fit_rank[i]=i;}
311
312     //fitnessが高い順に個体の添字を降順ソート
313     for(i=0;i<POP_SIZE;i++){
314         for(j=i+1;j<POP_SIZE;j++){
315             if(fitness[fit_rank[i]] < fitness[fit_rank[j]]){
316                 tmp = fit_rank[i];
317                 fit_rank[i] = fit_rank[j];
318                 fit_rank[j] = tmp;
319             }
320         }
321     }
322
323     for(i=0;i<rank_limit;i++){
324         fit_rank_rate[i] = max_rate -
(double)(i/(rank_limit/(max_rate*10)))/10.0;
325     }
326
327     rand_n =
(int)(((double)Rand()/((double)(RANDOM_MAX+1)))*(double)rank_limit);//0<=num<50と
する
328     rand = (double)Rand()/((double)(RANDOM_MAX+1)); //0<=num<1とする
329     if(rand < fit_rank_rate[rand_n] && parent_options[rand_n] != 1){return
rand_n;}
330     else{return Select(parent_options);}
331 }
332 /*
333 int Select(int parent_options[]){
334     int i,n=0;
335     double rand;

```

```

336     double fit_rate_loading[POP_SIZE] = {};
337     fit_rate_loading[0] = (double)fitness[0]/(double)sumfitness;
338
339     for(i=1;i<POP_SIZE;i++){
340         fit_rate_loading[i] = fit_rate_loading[i-1] +
(double)fitness[i]/(double)sumfitness;
341     }
342
343     rand = (double)Rand()/((double)(RANDOM_MAX+1));    //0<=num<1とする
344     while(fit_rate_loading[n]<rand){
345         n++;
346     }
347     if(parent_options[n]!=1){return n;}
348     else{return Select(parent_options);}
349 }
350 */
351
352
353 //交叉
354 void Crossover(int parent1,int parent2,int *child1, int *child2){
355     int min2;
356     int n_cross1, n_cross2; //染色体の切断点
357     int i,j,n;
358     bool _isDuplicate; //重複があるか
359     int memory[2][11]; //入れ替えた要素の定義を保存
360     int mem_n; //memory[][]まわすための添字
361     int parent_elem;
362     int x,y,v,z; //ループの添字
363     int _candidate; //parent_elemのペアのkey(候補)
364     double rand; //乱数発生用
365
366     rand = (double)Rand()/((double)(RANDOM_MAX+1));
367     if(rand > P_CROSSOVER){
368         printf("NO CROSSOVER¥n");
369         return;
370     }
371
372     //交叉位置
373     n_cross1 = Rand()%16+1; //n_cross = 1,...,17 (とりあえずハードコーディン
グ...)
374     n_cross2 = n_cross1 + 11;
375
376     //交叉
377     PrintCrossover(BEFORE, parent1, parent2, *child1, *child2, n_cross1,
n_cross2);
378     init_key_options();
379     be_empty(*child1);
380
381     mem_n=0;
382     for(j=n_cross1; j<n_cross2; j++){
383         //親2の切断点間の要素を子に配置・要素のペア定義を記憶
384         keyboards[*child1][j] = keyboards[parent2][j];
385         key_options[keyboards[parent2][j]] = Used;
386
387         memory[0][mem_n] = keyboards[parent1][j];
388         memory[1][mem_n] = keyboards[parent2][j];

```



```

442         key_options[keyboards[*child1][j]] = Used;
443         break;
444     }else{
445         parent_elem = _candidate;
446         memory[0][v] = -5;
447         memory[1][v] = -5;
448     }
449 }
450 }
451 }
452
453 init_key_options();
454 be_empty(*child2);
455 mem_n = 0;
456 for(j=n_cross1; j<n_cross2; j++){
457     //親1の切断点間の要素を子に配置/ 要素のペア定義を記憶
458     keyboards[*child2][j] = keyboards[parent1][j];
459     key_options[keyboards[parent1][j]] = Used;
460
461     memory[0][mem_n] = keyboards[parent1][j];
462     memory[1][mem_n] = keyboards[parent2][j];
463     mem_n++;
464 }
465 //EMPTY=-2,Used=-1, 親2に子で使われていない要素があれば、そのまま子に配置 (切
断点より前)
466 for(j=0; j<n_cross1; j++){
467     //使われていない要素か探索
468     _isDuplicate = false;
469     for(n=0; n<LEN_KEYS; n++){
470         if(key_options[keyboards[parent2][j]] == -1) _isDuplicate = true;
471     }
472     if(_isDuplicate == false){
473         keyboards[*child2][j] = keyboards[parent2][j];
474         key_options[keyboards[parent2][j]] = Used;
475     }
476 }
477 //EMPTY=-2,Used=-1, 親2に子で使われていない要素があれば、そのまま子に配置 (切
断点より後)
478 for(j=n_cross2; j<LEN_KEYS; j++){
479     //使われていない要素か探索
480     _isDuplicate = false;
481     for(n=0; n<LEN_KEYS; n++){
482         if(key_options[keyboards[parent2][j]] == -1) _isDuplicate= true;
483     }
484     if(_isDuplicate == false){
485         keyboards[*child2][j] = keyboards[parent2][j];
486         key_options[keyboards[parent2][j]] = Used;
487     }
488 }
489 //残りのEMPTYにはペア定義を参照して衝突しないように要素を配置
490 for(j=0; j<LEN_KEYS; j++){
491     if(keyboards[*child2][j] == -2){
492         //EMPTY部分の親の要素を保存
493         parent_elem = keyboards[parent2][j];
494         //フローチャート始まり
495         while(1){
496             for(v=0;v<11; v++){

```

```

497         for(z=0; z<2; z++){
498             //要素がメモリ内にあったとき、ペアを_candidateに入れてループ抜ける
499             if(memory[z][v] == parent_elem){
500                 if(z == 0){
501                     _candidate = memory[1][v];
502                     goto OUT2;
503                 }else{
504                     _candidate = memory[0][v];
505                     goto OUT2;
506                 }
507             }
508         }
509     }
510     //_candidateが配列にあるかないか。 あり-> parent_elemに_candidate入
    れてフローチャート頭に戻る{このときmemoryを潰す(無限ループ対策)}/ なし-> _candidateを染色
    体に配置してループ抜ける
511     OUT2:
512         if(key_options[_candidate] != -1){
513             keyboards[*child2][j] = _candidate;
514             key_options[keyboards[*child2][j]] = Used;
515             break;
516         }else{
517             parent_elem = _candidate;
518             memory[0][v] = -5;
519             memory[1][v] = -5;
520         }
521     }
522 }
523 }
524 fitness[*child1] = ObjFunc(*child1);
525 fitness[*child2] = ObjFunc(*child2);
526 PrintCrossover(AFTER, parent1, parent2, *child1, *child2, n_cross1,
n_cross2);
527 }
528
529 //一定確率でキー入れ替わり
530 void Mutation(int child){
531
532     int n_mutate1;
533     int n_mutate2;
534     int x;
535     double rand;
536
537     rand = (double)Rand()/((double)(RANDOM_MAX+1));    //0<=num<1とする
538     if(rand<P_MUTATION){
539         //突然変異位置
540         n_mutate1 = Rand()%LEN_KEYS; //n_mutate1=0,...,29
541         n_mutate2 = Rand()%LEN_KEYS; //n_mutate2=0,...,29
542         //突然変異
543         PrintMutation(BEFORE,child,n_mutate1,n_mutate2);
544         x = keyboards[child][n_mutate1];
545         keyboards[child][n_mutate1] = keyboards[child][n_mutate2];
546         keyboards[child][n_mutate2] = x;
547         fitness[child] = ObjFunc(child);
548         PrintMutation(AFTER,child,n_mutate1,n_mutate2);
549     }
550 }
551

```

```

552 //ファイルから文字列入力
553 void fileread(){
554     FILE *fp;
555     int i=0;
556     char fname[] = "learning.txt";
557     char text[256];
558
559     fp = fopen(fname, "r");
560     if(fp == NULL) {
561         exit(1);
562     }
563
564     for (i = 0; fgets(text, 256, fp) != NULL; i++){
565         strcpy(str[i], text);
566     }
567     STRINGS = i;
568
569     fclose(fp);
570 }
571
572 //ファイルに結果出力
573 void filewrite(int keyboard[],char* phase){
574     int i;
575     char filename[256];
576     strcpy(filename,name);
577     FILE* f = fopen(strcat(strcat(filename,phase),"_result.txt"), "w");
578
579     for(i=0;i<LEN_KEYS;i++){
580         fprintf(f, "%c¥n", alphabet[keyboard[i]]);
581     }
582
583     fclose(f);
584 }
585
586 //CSVファイルに結果出力
587
588 void filewrite_csv(int gen){
589     int i;
590     char filename[256];
591     strcpy(filename,name);
592     FILE* f;
593
594     if(gen==0){
595         f = fopen(strcat(filename,"_Maxfitness_result.csv"),"w");
596         fprintf(f, "gen,Maxfitness¥n");
597         //f = fopen(strcat(filename,"_fitnessAve_result.csv"),"w");
598         //fprintf(f, "gen,fitnessAve¥n");
599     }else{
600         f = fopen(strcat(filename,"_Maxfitness_result.csv"),"a");
601         fprintf(f, "%d,%d¥n", gen,max);
602         //f = fopen(strcat(filename,"_fitnessAve_result.csv"),"a");
603         //fprintf(f, "%d,%f¥n", gen,(double)sumfitness/(double)POP_SIZE);
604     }
605
606     fclose(f);
607 }
608
609 //メイン関数
610 int main(int argc,char **argv){

```

```
611     int gen,i;
612
613     Srand((unsigned) time(NULL)); //seed値変更
614
615     printf("名前を入力してください -> ");
616     scanf("%s",name);
617     fileread();
618
619     keyweightcal(); //キーの重み付け
620     Initialize(); //初期化
621
622     fwrite_csv(0);
623
624     for(gen=1;gen<=MAX_GEN;gen++){
625         Generation(gen);
626         if(gen==1)
627             fwrite(keyboards[n_max],"_first");
628         if(gen==MAX_GEN/2)
629             fwrite(keyboards[n_max],"_intermediate");
630         if(gen==MAX_GEN)
631             fwrite(keyboards[n_max],"_final");
632         fwrite_csv(gen);
633     }
634     Statistics();
635     PrintStatistics(-1);
636 }
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/time.h>
5  #include "conio.h"
6
7
8  struct eachkey{
9      int key_num; //キーの位置番号
10     double keytimes; //各キー入力時間保存用
11 };
12
13 struct eachkey keys[30];
14 int keyweight[30]; //各キーの重み付け保存用
15 char keyplace[30] =
16 {'q','w','e','r','t','y','u','i','o','p','a','s','d','f','g','h','j','k','l',';','z','x','c','v','b','n','m','.',',','/'};
17
18 double getETime(){
19     struct timeval tv;
20     gettimeofday(&tv, NULL);
21     return tv.tv_sec + (double)tv.tv_usec*1e-6;
22 }
23
24 /*指定されたキーが押される時間を計測*/
25 double keytime(char c){
26     double start, end, time;
27     char getkey;
28     int ec;
29     printf("手をホームポジションにおいて、スペースキーを押してください¥n");
30     while(1){
31         int space = getch();
32         if(space == ' ') break;
33     }
34     start = getETime();
35     printf("「%c」を入力してください>> ", c);
36     while(1){
37         ec = getch();
38         if(ec == c) break;
39     }
40     printf("%c¥n", ec);
41     end = getETime();
42     time = end - start;
43     printf("キー入力時間：%lf秒¥n", time);
44     return time;
45 }
46
47 int GetRandom(int min,int max){
48     return min + (int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
49 }
50
51 /*ランダムにキー入力を受付+時間計測*/
52 void keyweightcal(){
53     int i, n, k, h;
54     struct eachkey tmp;
55     for(i=0;i<30;i++) keys[i].key_num = i; //キーの位置番号設定

```

```
55     for(i=0;i<30;i++) keys[i].keytimes = -1; //keytimes初期化
56     for(i=0;i<30;i++){
57         n = GetRandom(0,29);
58         while(keys[n].keytimes!=-1){ //記録されていないキーを探す s
59             n = GetRandom(0,29);
60         }
61         keys[n].keytimes = keytime(keyplace[n]);
62         keyweight[n] = keys[n].keytimes * 1000; //小数点をなくした数値
63     }
64     /*結果確認用*/
65     for(i=0;i<30;i++){
66         printf("キー番号(%d)の重み:%d¥n",i,keyweight[i]);
67     }
68 }
```

```

1  <!DOCTYPE html>
2  <html lang="ja">
3    <head>
4      <meta charset="UTF-8">
5      <title>キー配列オーダーメイド</title>
6      <style>
7        body {
8          background: #ffd700;
9          font-family: Meiryo;
10       }
11       div {
12         background: #ffffff;
13         padding: 10px;
14         text-align: center;
15         border: 5px solid #cccccc;
16         margin: 30px auto;
17       }
18       button {
19         width: 50px;
20         height: 50px;
21       }
22     </style>
23   </head>
24   <body>
25     <div>
26       <h1>オリジナルキーボード</h1>
27       <p>手の癖調査、普段の文章を元にGAを行った結果、<br>あなたに適したキー配列はこの
ようになりました！</p>
28       <button type="button" id="pos0">Q</button>
29       <button type="button" id="pos1">W</button>
30       <button type="button" id="pos2">E</button>
31       <button type="button" id="pos3">R</button>
32       <button type="button" id="pos4">T</button>
33       <button type="button" id="pos5">Y</button>
34       <button type="button" id="pos6">U</button>
35       <button type="button" id="pos7">I</button>
36       <button type="button" id="pos8">O</button>
37       <button type="button" id="pos9">P</button>
38       <br>
39       &nbsp;&nbsp;&nbsp;<button type="button" id="pos10">A</button>
40       <button type="button" id="pos11">S</button>
41       <button type="button" id="pos12">D</button>
42       <button type="button" id="pos13">F</button>
43       <button type="button" id="pos14">G</button>
44       <button type="button" id="pos15">H</button>
45       <button type="button" id="pos16">J</button>
46       <button type="button" id="pos17">K</button>
47       <button type="button" id="pos18">L</button>
48       <button type="button" id="pos19"> ; </button>
49       <br>
50       &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<button type="button" id="pos20">Z</button>
51       <button type="button" id="pos21">X</button>
52       <button type="button" id="pos22">C</button>
53       <button type="button" id="pos23">V</button>
54       <button type="button" id="pos24">B</button>
55       <button type="button" id="pos25">N</button>
56       <button type="button" id="pos26">M</button>

```

```

57 <button type="button" id="pos27">, </button>
58 <button type="button" id="pos28">.</button>
59 <button type="button" id="pos29">?</button>
60 <br><br>
61 </div>
62 <input type="file" id="selfile"><br>
63
64 <script>
65   var obj1 = document.getElementById("selfile");
66   //ダイアログでファイルが選択された時
67   obj1.addEventListener("change",function(evt){
68     var file = evt.target.files;
69     //FileReaderの作成
70     var reader = new FileReader();
71     //テキスト形式で読み込む
72     reader.readAsText(file[0]);
73     //読込終了後の処理
74     reader.onload = function(ev){
75       var K = reader.result.split(/\\n/);
76       document.getElementById("pos0").innerText = K[0];
77       document.getElementById("pos1").innerText = K[1];
78       document.getElementById("pos2").innerText = K[2];
79       document.getElementById("pos3").innerText = K[3];
80       document.getElementById("pos4").innerText = K[4];
81       document.getElementById("pos5").innerText = K[5];
82       document.getElementById("pos6").innerText = K[6];
83       document.getElementById("pos7").innerText = K[7];
84       document.getElementById("pos8").innerText = K[8];
85       document.getElementById("pos9").innerText = K[9];
86       document.getElementById("pos10").innerText = K[10];
87       document.getElementById("pos11").innerText = K[11];
88       document.getElementById("pos12").innerText = K[12];
89       document.getElementById("pos13").innerText = K[13];
90       document.getElementById("pos14").innerText = K[14];
91       document.getElementById("pos15").innerText = K[15];
92       document.getElementById("pos16").innerText = K[16];
93       document.getElementById("pos17").innerText = K[17];
94       document.getElementById("pos18").innerText = K[18];
95       document.getElementById("pos19").innerText = K[19];
96       document.getElementById("pos20").innerText = K[20];
97       document.getElementById("pos21").innerText = K[21];
98       document.getElementById("pos22").innerText = K[22];
99       document.getElementById("pos23").innerText = K[23];
100      document.getElementById("pos24").innerText = K[24];
101      document.getElementById("pos25").innerText = K[25];
102      document.getElementById("pos26").innerText = K[26];
103      document.getElementById("pos27").innerText = K[27];
104      document.getElementById("pos28").innerText = K[28];
105      document.getElementById("pos29").innerText = K[29];
106    }
107    },false);
108 </script>
109
110 </body>
111 </html>

```