```
 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
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 RANDOM_MAX 32767
16  #define BEFORE 0
17  #define AFTER 1
18
19  char name[256];
20  int keyboards[POP_SIZE][LEN_KEYS];  //染色体(キーボード配列)
21  int fitness[POP_SIZE];              //適合度
22  int max,min,sumfitness;             //適合度の,max,min,sum
23  int n_min;                         //適合度のminの添字
24  int n_max;                         //適合度のmaxの添字
25
26  void PrintKeyboardFitness();
27  void Statistics();
28  void Crossover(int parent1,int parent2,int *child1, int
*child2);
29  void Mutation(int child);
30  int ObjFunc(int i);
31  int Select();
32  void filewrite(int keyboard[],char* phase);
33
34  #define EMPTY -2
35  #define Used -1
36  #define A 0
37  #define B 1
38  #define C 2
39  #define D 3
40  #define E 4
41  #define F 5
42  #define G 6
43  #define H 7
44  #define I 8
```

```
45   #define J 9
46   #define K 10
47   #define L 11
48   #define M 12
49   #define N 13
50   #define O 14
51   #define P 15
52   #define Q 16
53   #define R 17
54   #define S 18
55   #define T 19
56   #define U 20
57   #define V 21
58   #define W 22
59   #define X 23
60   #define Y 24
61   #define Z 25
62   #define Others1 26
63   #define Others2 27
64   #define Others3 28
65   #define Others4 29
66
67   int key_options[LEN_KEYS];    //配置可能なキー
68   int STRINGS = 0;
69   char str[256][10000] = {};   //日本語文字列
70   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',':','<','>','?'};
71
72   //擬似乱数
73   static unsigned long int next = 1;
74
75   int Rand(void){
76     next = next*1103515245 + 12345;
77     return (unsigned int)(next/65536)%(RANDOM_MAX+1);
78   }
79
80   void Srand(unsigned int seed){
81     next = seed;
82   }
83
84   void init_key_options(){
85     int i;
86     for(i=0;i<LEN_KEYS;i++){
87       key_options[i] = i;
88     }
```

```
 89   }
 90
 91   //配列を空に（キー割り当てなしに）
 92   void be_empty(int i){
 93     int j;
 94     for(j=0;j<LEN_KEYS;j++)
 95       keyboards[i][j] = EMPTY;
 96   }
 97
 98   //初期データ設定
 99   void Initialize(){
100     int i,j,n;
101
102     for(i=0;i<POP_SIZE;i++){
103       init_key_options();
104       for(j=0;j<LEN_KEYS;j++){
105         n = Rand()%LEN_KEYS;
106         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
107         keyboards[i][j] = key_options[n];
108         key_options[n] = Used;
109       }
110       fitness[i] = ObjFunc(i);
111     }
112
113     printf("First Position¥n");
114     PrintKeyboardFitness();
115     printf("----------------¥n");
116   }
117
118   //データ表示関数
119   void PrintEachKeyboardFitness(int i){
120     int j;
121     printf("[%d] ",i);
122     for(j=0;j<LEN_KEYS;j++){
123       printf("%d,",keyboards[i][j]);
124     }
125     printf(":%d¥n",fitness[i]);
126   }
127
128   void PrintKeyboardFitness(){
129     int i;
130     for(i=0;i<POP_SIZE;i++)PrintEachKeyboardFitness(i);
131   }
132
133   void PrintStatistics(int gen){
```

```
  134    if(gen==-1){
  135      printf("[gen=%dend] max=%d min=%d sumfitness=%d
ave=%f¥n",
  136
MAX_GEN,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
  137    }else{
  138    printf("[gen=%2d] max=%d min=%d sumfitness=%d ave=%f¥n",
  139
gen,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
  140    }
  141   }
  142
  143   void PrintCrossover(int flag,int parent1,int parent2,int
child1,int child2,int n_cross1, int n_cross2){
  144    switch(flag){
  145    case BEFORE:
  146      printf("parent1 |");PrintEachKeyboardFitness(parent1);
  147      printf("parent2 |");PrintEachKeyboardFitness(parent2);
  148      printf("delete1 |");PrintEachKeyboardFitness(child1);
  149      printf("delete2 |");PrintEachKeyboardFitness(child2);
  150      printf("n_cross1=%d¥n",n_cross1);
  151      printf("n_cross2=%d¥n",n_cross2);
  152
  153      break;
  154    case AFTER:
  155      printf("child1 |");PrintEachKeyboardFitness(child1);
  156      printf("child2 |");PrintEachKeyboardFitness(child2);
  157      printf("-----------------¥n");
  158      break;
  159    }
  160   }
  161
  162   void PrintMutation(int flag,int child,int n_mutate1,int
n_mutate2){
  163    switch(flag){
  164    case BEFORE:
  165      printf("child(OLD)|");PrintEachKeyboardFitness(child);
  166      printf("n_mutate1=%d <-->
n_mutate2=%d¥n",n_mutate1,n_mutate2);
  167      break;
  168    case AFTER:
  169      printf("child(NEW)|");PrintEachKeyboardFitness(child);
  170      printf("-----------------¥n");
  171      break;
  172    }
  173   }
  174
```

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

```
  }
  return 0;
}

int is_index_finger(int b, int c){
  if(b==3 && c==4) return 1;
  if(b==4 && c==3) return 1;
  if(b==5 && c==6) return 1;
  if(b==6 && c==5) return 1;
  if(b==6 && c==5) return 1;
  if(b==13 && c==14) return 1;
  if(b==14 && c==13) return 1;
  if(b==15 && c==16) return 1;
  if(b==16 && c==15) return 1;
  if(b==23 && c==24) return 1;
  if(b==24 && c==23) return 1;
  if(b==25 && c==26) return 1;
  if(b==26 && c==25) return 1;
  return 0;
}

//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？))
//考える
//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)
int ObjFunc(int i){

  int j,k,bk = 0;
  int ck = -1;
  int count = 0; //指が移動してしまった回数
  int point = 0; //返す評価値
  for(j=0;j<STRINGS;j++){
    int n = 0; //文字列の添字
    int s = 0; //有効な文字のカウント
    while(str[j][n]!='¥0'){
      if(is_keyword(str[j][n])){ //今回考えるキーか確認
      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
        bk = ck;
          for(k=0;k<=29;k++){
          if(alphabet[keyboards[i][k]]==str[j][n]){
            ck=k;
            break;
```

```
   261           }
   262         }
   263            if(k!=30){
   264           if(is_index_finger(bk, k)){count +=
keyweight[k]/2;}
   265           else{count += keyweight[k];}
   266         }
   267         s++;
   268       }
   269       }
   270       n++;
   271     }
   272     point += n*10000; //文字数*10000ポイント加算←全部10秒かかっ
た設定
   273   }
   274   point -= count;
   275   return point; /（全文字数*10000-カウント数）が最終ポイント
   276  }
   277
   278  //fitnessの合計値の計算
   279  void Statistics(){
   280   int i;
   281
   282   max = 0;
   283   min = 2147483647; //int最大値
   284   sumfitness = 0;
   285
   286   for(i=0;i<POP_SIZE;i++){
   287     if(fitness[i]>max){
   288       max = fitness[i];
   289       n_max = i;
   290     }
   291     if(fitness[i]<min){
   292       min = fitness[i];
   293       n_min = i;
   294     }
   295     sumfitness += fitness[i];
   296   }
   297  }
   298
   299  //選択
   300  //ルーレット→ランキング
   301  int Select(int parent_options[]){
   302   int i,j,tmp,rand_n;
   303   double rand;
```

```
   304     int fit_rank[POP_SIZE];
   305     double fit_rank_rate[POP_SIZE] = {};
   306     double max_rate = 0.8;
   307     int rank_limit = 50; //ランク付けする個体数（残りは確率0)
   308
   309     for(i=0;i<POP_SIZE;i++){fit_rank[i]=i;}
   310
   311     //fitnessが高い順に個体の添字を降順ソート
   312     for(i=0;i<POP_SIZE;i++){
   313       for(j=i+1;j<POP_SIZE;j++){
   314         if(fitness[fit_rank[i]] < fitness[fit_rank[j]]){
   315        tmp = fit_rank[i];
   316        fit_rank[i] = fit_rank[j];
   317        fit_rank[j] = tmp;
   318         }
   319       }
   320     }
   321
   322     for(i=0;i<rank_limit;i++){
   323       fit_rank_rate[i] = max_rate -
(double)(i/(rank_limit/(max_rate*10)))/10.0;
   324     }
   325
   326     rand_n =
(int)(((double)Rand()/(double)(RANDOM_MAX+1))*(double)rank_limit);
//0<=num<50とする
   327     rand = (double)Rand()/((double)(RANDOM_MAX+1));
//0<=num<1とする
   328     if(rand < fit_rank_rate[rand_n] &&
parent_options[rand_n]!=1){return rand_n;}
   329     else{return Select(parent_options);}
   330   }
   331   /*
   332   int Select(int parent_options[]){
   333     int i,n=0;
   334     double rand;
   335     double fit_rate_loading[POP_SIZE] = {};
   336     fit_rate_loading[0] =
(double)fitness[0]/(double)sumfitness;
   337
   338     for(i=1;i<POP_SIZE;i++){
   339       fit_rate_loading[i] = fit_rate_loading[i-1] +
(double)fitness[i]/(double)sumfitness;
   340     }
   341
   342     rand = (double)Rand()/((double)(RANDOM_MAX+1));
```

```
//0<=num<1とする
   343     while(fit_rate_loading[n]<rand){
   344       n++;
   345     }
   346     if(parent_options[n]!=1){return n;}
   347     else{return Select(parent_options);}
   348   }
   349   */
   350
   351
   352   //交叉
   353   void Crossover(int parent1,int parent2,int *child1, int
*child2){
   354     int min2;
   355     int n_cross1, n_cross2; //染色体の切断点
   356     int i,j,n;
   357     bool _isDuplicate; //重複があるか
   358     int memory[2][11]; //入れ替えた要素の定義を保存
   359     int mem_n; //memory[][]まわすための添字
   360     int parent_elem;
   361     int x,y,v,z; //ループの添字
   362     int _candidate; //parent_elemのペアのkey(候補)
   363
   364     //交叉位置
   365     n_cross1 = Rand()%16+1; //n_cross = 1,...,17 (とりあえずハ
ードコーディング...)
   366     n_cross2 = n_cross1 + 11;
   367
   368     //交叉
   369     PrintCrossover(BEFORE, parent1, parent2, *child1,
*child2, n_cross1, n_cross2);
   370     init_key_options();
   371     be_empty(*child1);
   372
   373     mem_n=0;
   374     for(j=n_cross1; j<n_cross2; j++){
   375       //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
   376       keyboards[*child1][j] = keyboards[parent2][j];
   377       key_options[keyboards[parent2][j]] = Used;
   378
   379       memory[0][mem_n] = keyboards[parent1][j];
   380       memory[1][mem_n] = keyboards[parent2][j];
   381       mem_n++;
   382     }
```

```
  383     //EMPTY=-2,Used=-1, 親1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）
  384     for(j=0; j<n_cross1; j++){
  385       //使われていない要素か探索
  386       _isDuplicate = false;
  387       for(n=0; n<LEN_KEYS; n++){
  388         if(key_options[keyboards[parent1][j]] == -1)
_isDuplicate = true;
  389       }
  390       if(_isDuplicate == false){
  391         keyboards[*child1][j] = keyboards[parent1][j];
  392         key_options[keyboards[parent1][j]] = Used;
  393       }
  394     }
  395
  396     //EMPTY=-2,Used=-1, 親1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）
  397     for(j=n_cross2; j<LEN_KEYS; j++){
  398       //使われていない要素か探索
  399       _isDuplicate = false;
  400       for(n=0; n<LEN_KEYS; n++){
  401         if(key_options[keyboards[parent1][j]] == -1)
_isDuplicate= true;
  402       }
  403       if(_isDuplicate == false){
  404         keyboards[*child1][j] = keyboards[parent1][j];
  405         key_options[keyboards[parent1][j]] = Used;
  406       }
  407     }
  408
  409     //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
  410     for(j=0; j<LEN_KEYS; j++){
  411       if(keyboards[*child1][j] == -2){
  412         //EMPTY部分の親の要素を保存
  413         parent_elem = keyboards[parent1][j];
  414         //フローチャート始まり
  415         while(1){
  416           for(v=0;v<11; v++){
  417             for(z=0; z<2; z++){
  418               //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
  419               if(memory[z][v] == parent_elem){
  420                 if(z == 0){
  421                   _candidate = memory[1][v];
```

```
                goto OUT1;
              }else{
                _candidate = memory[0][v];
                goto OUT1;
              }
            }
          }
        }
        //_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
        OUT1:
            if(key_options[_candidate] != -1){
              keyboards[*child1][j] = _candidate;
              key_options[keyboards[*child1][j]] = Used;
              break;
            }else{
              parent_elem = _candidate;
              memory[0][v] = -5;
              memory[1][v] = -5;
            }
      }
    }
  }

  init_key_options();
  be_empty(*child2);
  mem_n = 0;
  for(j=n_cross1; j<n_cross2; j++){
    //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
    keyboards[*child2][j] = keyboards[parent1][j];
    key_options[keyboards[parent1][j]] = Used;

    memory[0][mem_n] = keyboards[parent1][j];
    memory[1][mem_n] = keyboards[parent2][j];
    mem_n++;
  }
  //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)
  for(j=0; j<n_cross1; j++){
    //使われていない要素か探索
    _isDuplicate = false;
    for(n=0; n<LEN_KEYS; n++){
      if(key_options[keyboards[parent2][j]] == -1) _isDuplicate = true;
```

```
  463       }
  464       if(_isDuplicate == false){
  465         keyboards[*child2][j] = keyboards[parent2][j];
  466         key_options[keyboards[parent2][j]] = Used;
  467       }
  468     }
  469     //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)
  470     for(j=n_cross2; j<LEN_KEYS; j++){
  471       //使われていない要素か探索
  472       _isDuplicate = false;
  473       for(n=0; n<LEN_KEYS; n++){
  474         if(key_options[keyboards[parent2][j]] == -1)
_isDuplicate= true;
  475       }
  476       if(_isDuplicate == false){
  477         keyboards[*child2][j] = keyboards[parent2][j];
  478         key_options[keyboards[parent2][j]] = Used;
  479       }
  480     }
  481     //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
  482     for(j=0; j<LEN_KEYS; j++){
  483       if(keyboards[*child2][j] == -2){
  484          //EMPTY部分の親の要素を保存
  485         parent_elem = keyboards[parent2][j];
  486          //フローチャート始まり
  487         while(1){
  488           for(v=0;v<11; v++){
  489             for(z=0; z<2; z++){
  490                //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
  491               if(memory[z][v] == parent_elem){
  492                 if(z == 0){
  493                   _candidate = memory[1][v];
  494                   goto OUT2;
  495                 }else{
  496                   _candidate = memory[0][v];
  497                   goto OUT2;
  498                 }
  499               }
  500             }
  501           }
  502           //_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
   503          OUT2:
   504              if(key_options[_candidate] != -1){
   505                keyboards[*child2][j] = _candidate;
   506                key_options[keyboards[*child2][j]] = Used;
   507                break;
   508              }else{
   509                parent_elem = _candidate;
   510                memory[0][v] = -5;
   511                memory[1][v] = -5;
   512              }
   513          }
   514        }
   515      }
   516      fitness[*child1] = ObjFunc(*child1);
   517      fitness[*child2] = ObjFunc(*child2);
   518      PrintCrossover(AFTER, parent1, parent2, *child1,
*child2, n_cross1, n_cross2);
   519    }
   520
   521    //一定確率でキー入れ替わり
   522    void Mutation(int child){
   523
   524      int n_mutate1;
   525      int n_mutate2;
   526      int x;
   527      double rand;
   528
   529      rand = (double)Rand()/((double)(RAND_MAX+1));
//0<=num<1とする
   530      if(rand<P_MUTATION){
   531        //突然変異位置
   532        n_mutate1 = Rand()%LEN_KEYS;  //n_mutate1=0,...,29
   533        n_mutate2 = Rand()%LEN_KEYS;  //n_mutate2=0,...,29
   534        //突然変異
   535        PrintMutation(BEFORE,child,n_mutate1,n_mutate2);
   536        x = keyboards[child][n_mutate1];
   537        keyboards[child][n_mutate1] =
keyboards[child][n_mutate2];
   538        keyboards[child][n_mutate2] = x;
   539        fitness[child] = ObjFunc(child);
   540        PrintMutation(AFTER,child,n_mutate1,n_mutate2);
   541      }
   542    }
   543
   544    //ファイルから文字列入力
```

```
545   void fileread(){
546     FILE *fp;
547     int i=0;
548        char fname[] = "learning.txt";
549        char text[256];
550
551        fp = fopen(fname, "r");
552        if(fp == NULL) {
553          exit(1);
554        }
555
556        for (i = 0; fgets(text, 256, fp) != NULL; i++){
557            strcpy(str[i], text);
558        }
559        STRINGS = i;
560
561        fclose(fp);
562   }
563
564   //ファイルに結果出力
565   void filewrite(int keyboard[],char* phase){
566     int i;
567     char filename[256];
568     strcpy(filename,name);
569     FILE* f =
fopen(strcat(strcat(filename,phase),"_result.txt"), "w");
570
571     for(i=0;i<LEN_KEYS;i++){
572       fprintf(f, "%c¥n", alphabet[keyboard[i]]);
573     }
574
575     fclose(f);
576   }
577
578   //CSVファイルに結果出力
579   void filewrite_csv(int gen){
580     int i;
581     char filename[256];
582     strcpy(filename,name);
583     FILE* f;
584
585     if(gen==0){
586       f =
fopen(strcat(filename,"_Maxfitness_result.csv"),"w");
587       fprintf(f, "世代,最大評価値¥n");
588     }else{
```

```
  589       f =
fopen(strcat(filename,"_Maxfitness_result.csv"),"a");
  590       fprintf(f, "%d,%d¥n", gen,max);
  591     }
  592
  593     fclose(f);
  594   }
  595
  596   //メイン関数
  597   int main(int argc,char **argv){
  598     int gen,i;
  599
  600     Srand((unsigned) time(NULL)); //seed値変更
  601
  602     printf("名前を入力してください -> ");
  603     scanf("%s",name);
  604     fileread();
  605
  606     keyweightcal(); //キーの重み付け
  607     Initialize(); //初期化
  608
  609     filewrite_csv(0);
  610
  611     for(gen=1;gen<=MAX_GEN;gen++){
  612       Generation(gen);
  613       if(gen==1)
  614         filewrite(keyboards[n_max],"_first");
  615       if(gen==MAX_GEN/2)
  616         filewrite(keyboards[n_max],"_intermediate");
  617       if(gen==MAX_GEN)
  618         filewrite(keyboards[n_max],"_final");
  619       filewrite_csv(gen);
  620     }
  621     Statistics();
  622     PrintStatistics(-1);
  623   }
```

```
 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] =
{'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',',','.','/'};
16
17  double getETime(){
18      struct timeval tv;
19      gettimeofday(&tv, NULL);
20      return tv.tv_sec + (double)tv.tv_usec*1e-6;
21  }
22
23  /*指定されたキーが押される時間を計測*/
24  double keytime(char c){
25      double start, end, time;
26      char getkey;
27      int ec;
28      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");
29      while(1){
30          int space = getch();
31          if(space == ' ') break;
32      }
33      start = getETime();
34      printf("「%c」を入力してください>> ", c);
35      while(1){
36          ec = getch();
37          if(ec == c) break;
38      }
39      printf("%c¥n", ec);
40      end = getETime();
41      time = end - start;
42      printf("キー入力時間：%lf秒¥n", time);
```

```
    43      return time;
    44   }
    45
    46   int GetRandom(int min,int max){
    47       return min + (int)(rand()*(max-
min+1.0)/(1.0+RAND_MAX));
    48   }
    49
    50   /*ランダムにキー入力を受付＋時間計測*/
    51   void keyweightcal(){
    52      int i, n, k, h;
    53      struct eachkey tmp;
    54      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 }
```

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8">
    <title>キー配列オーダーメイド</title>
    <style>
      body {
        background: #ffd700;
        font-family: Meiryo;
      }
      div {
        background: #ffffff;
        padding: 10px;
        text-align: center;
        border: 5px solid #cccccc;
        margin: 30px auto;
      }
      button {
        width: 50px;
        height: 50px;
      }
    </style>
  </head>
  <body>
    <div>
    <h1>オリジナルキーボード</h1>
    <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>
    <button type="button" id="pos0">Q</button>
    <button type="button" id="pos1">W</button>
    <button type="button" id="pos2">E</button>
    <button type="button" id="pos3">R</button>
    <button type="button" id="pos4">T</button>
    <button type="button" id="pos5">Y</button>
    <button type="button" id="pos6">U</button>
    <button type="button" id="pos7">I</button>
    <button type="button" id="pos8">O</button>
    <button type="button" id="pos9">P</button>
    <br>
      <button type="button" id="pos10">A</button>
    <button type="button" id="pos11">S</button>
    <button type="button" id="pos12">D</button>
    <button type="button" id="pos13">F</button>
    <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          <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>
```