```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include "testkeytime.h"
//#include "keytime.h"
#include<string.h>
#include<stdbool.h>

#define MAX_GEN 700     //最大世代交代   //文字数によって変える
#define POP_SIZE 100      //集団のサイズ
#define LEN_KEYS 30     //遺伝子の長さ
#define GEN_GAP 0.1     //世代交代の割合
#define P_MUTATION 0.05    //突然変異の確率
#define P_CROSSOVER 0.8    //交叉率
#define RANDOM_MAX 32767
#define BEFORE 0
#define AFTER 1

char name[256];
int keyboards[POP_SIZE][LEN_KEYS];  //染色体(キーボード配列)
int fitness[POP_SIZE];             //適合度
int max,min,sumfitness;            //適合度の,max,min,sum
int n_min;                        //適合度のminの添字
int n_max;                        //適合度のmaxの添字

void PrintKeyboardFitness();
void Statistics();
void Crossover(int parent1,int parent2,int *child1, int *child2);
void Mutation(int child);
int ObjFunc(int i);
int Select();
void filewrite(int keyboard[],char* phase);

#define EMPTY -2
#define Used -1
#define A 0
#define B 1
#define C 2
#define D 3
#define E 4
#define F 5
#define G 6
#define H 7
#define I 8
#define J 9
#define K 10
#define L 11
#define M 12
#define N 13
#define O 14
#define P 15
#define Q 16
#define R 17
#define S 18
#define T 19
#define U 20
```

```c
#define V 21
#define W 22
#define X 23
#define Y 24
#define Z 25
#define Others1 26
#define Others2 27
#define Others3 28
#define Others4 29

int key_options[LEN_KEYS];    //配置可能なキー
int STRINGS = 0;
char str[256][10000] = {};  //日本語文字列
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',':','<','>','?'};

//擬似乱数
static unsigned long int next = 1;

int Rand(void){
  next = next*1103515245 + 12345;
  return (unsigned int)(next/65536)%(RANDOM_MAX+1);
}

void Srand(unsigned int seed){
  next = seed;
}

void init_key_options(){
  int i;
  for(i=0;i<LEN_KEYS;i++){
    key_options[i] = i;
  }
}

//配列を空に（キー割り当てなしに）
void be_empty(int i){
  int j;
  for(j=0;j<LEN_KEYS;j++)
    keyboards[i][j] = EMPTY;
}

//初期データ設定
void Initialize(){
  int i,j,n;

  for(i=0;i<POP_SIZE;i++){
    init_key_options();
    for(j=0;j<LEN_KEYS;j++){
      n = Rand()%LEN_KEYS;
      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
      keyboards[i][j] = key_options[n];
      key_options[n] = Used;
    }
    fitness[i] = ObjFunc(i);
  }

  printf("First Position¥n");
  PrintKeyboardFitness();
  printf("----------------¥n");
}

//データ表示関数
void PrintEachKeyboardFitness(int i){
  int j;
  printf("[%d] ",i);
  for(j=0;j<LEN_KEYS;j++){
    printf("%d,",keyboards[i][j]);
  }
  printf(":%d¥n",fitness[i]);
}

void PrintKeyboardFitness(){
  int i;
  for(i=0;i<POP_SIZE;i++)PrintEachKeyboardFitness(i);
}

void PrintStatistics(int gen){
  if(gen==-1){
    printf("[gen=%dend] max=%d min=%d sumfitness=%d ave=%f¥n",
MAX_GEN,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
  }else{
  printf("[gen=%2d] max=%d min=%d sumfitness=%d ave=%f¥n",
      gen,max,min,sumfitness,(double)sumfitness/(double)POP_SIZE);
  }
}

void PrintCrossover(int flag,int parent1,int parent2,int child1,int child2,int n_cross1, int n_cross2){
  switch(flag){
  case BEFORE:
    printf("parent1 |");PrintEachKeyboardFitness(parent1);
    printf("parent2 |");PrintEachKeyboardFitness(parent2);
    printf("delete1 |");PrintEachKeyboardFitness(child1);
    printf("delete2 |");PrintEachKeyboardFitness(child2);
    printf("n_cross1=%d¥n",n_cross1);
    printf("n_cross2=%d¥n",n_cross2);

    break;
  case AFTER:
    printf("child1 |");PrintEachKeyboardFitness(child1);
    printf("child2 |");PrintEachKeyboardFitness(child2);
    printf("------------------¥n");
    break;
  }
}

void PrintMutation(int flag,int child,int n_mutate1,int n_mutate2){
  switch(flag){
  case BEFORE:
    printf("child(OLD)|");PrintEachKeyboardFitness(child);
    printf("n_mutate1=%d <--> n_mutate2=%d¥n",n_mutate1,n_mutate2);
    break;
  case AFTER:
    printf("child(NEW)|");PrintEachKeyboardFitness(child);
    printf("-----------------¥n");
    break;
  }
}

//世代の処理
void Generation(int gen){
  int parent1,parent2;
  int child1,child2;
  int n_gen;
  int i,j;
  int parent_options[POP_SIZE] = {}; //選択できる親の候補
  int min2;

  //集団の表示
  Statistics();
  PrintStatistics(gen);

  //世代交代
  n_gen=(int)((double)POP_SIZE*GEN_GAP/2.0);
  for(i=0;i<n_gen;i++){
    Statistics();

    //1番小さい値を子供としてセット
    child1 = n_min;
    //2番目に小さい値を見つける
    min2 = 2147483647; //int最大値
    for(j=0;j<POP_SIZE;j++){
      if(j!=child1){
     if(min<=fitness[j]&&fitness[j]<min2){
       min2 = fitness[j]; child2  = j;
     }
      }
    }
    parent_options[child1] = 1;
    parent_options[child2] = 1;
    parent1 = Select(parent_options);
    parent_options[parent1] = 1;
    parent2 = Select(parent_options);
    parent_options[parent2] = 1;
    Crossover(parent1,parent2,&child1,&child2);
    Mutation(child1);
    Mutation(child2);
  }
}

/*今回考えない文字除外関数*/
int is_keyword(char c){
  for (int i=0;i<30;i++){
    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;
          }
        }
           if(k!=30){
          if(is_index_finger(bk, k)){count += keyweight[k]/2;}
          else{count += keyweight[k];}
        }
        s++;
      }
      }
      n++;
    }
    point += n*10000; //文字数*10000ポイント加算←全部10秒かかった設定
  }
  point -= count;
  return point; /（全文字数*10000-カウント数）が最終ポイント
}

//fitnessの合計値の計算
void Statistics(){
  int i;
```

```c

  max = 0;
  min = 2147483647; //int最大値
  sumfitness = 0;

  for(i=0;i<POP_SIZE;i++){
    if(fitness[i]>max){
      max = fitness[i];
      n_max = i;
    }
    if(fitness[i]<min){
      min = fitness[i];
      n_min = i;
    }
    sumfitness += fitness[i];
  }
}

//選択
//ルーレット→ランキング
int Select(int parent_options[]){
  int i,j,tmp,rand_n;
  double rand;
  int fit_rank[POP_SIZE];
  double fit_rank_rate[POP_SIZE] = {};
  double max_rate = 0.8;
  int rank_limit = 50; //ランク付けする個体数（残りは確率0)

  for(i=0;i<POP_SIZE;i++){fit_rank[i]=i;}

  //fitnessが高い順に個体の添字を降順ソート
  for(i=0;i<POP_SIZE;i++){
    for(j=i+1;j<POP_SIZE;j++){
      if(fitness[fit_rank[i]] < fitness[fit_rank[j]]){
     tmp = fit_rank[i];
     fit_rank[i] = fit_rank[j];
     fit_rank[j] = tmp;
      }
    }
  }

  for(i=0;i<rank_limit;i++){
    fit_rank_rate[i] = max_rate - (double)(i/(rank_limit/(max_rate*10)))/10.0;
  }

  rand_n = (int)(((double)Rand()/(double)(RANDOM_MAX+1))*(double)rank_limit);//0<=num<50とする
  rand = (double)Rand()/((double)(RANDOM_MAX+1));   //0<=num<1とする
  if(rand < fit_rank_rate[rand_n] && parent_options[rand_n]!=1){return rand_n;}
  else{return Select(parent_options);}
}
/*
int Select(int parent_options[]){
  int i,n=0;
  double rand;
  double fit_rate_loading[POP_SIZE] = {};
  fit_rate_loading[0] = (double)fitness[0]/(double)sumfitness;

  for(i=1;i<POP_SIZE;i++){
    fit_rate_loading[i] = fit_rate_loading[i-1] + (double)fitness[i]/(double)sumfitness;
  }

  rand = (double)Rand()/((double)(RANDOM_MAX+1));    //0<=num<1とする
  while(fit_rate_loading[n]<rand){
    n++;
  }
  if(parent_options[n]!=1){return n;}
  else{return Select(parent_options);}
}
*/


//交叉
void Crossover(int parent1,int parent2,int *child1, int *child2){
  int min2;
  int n_cross1, n_cross2; //染色体の切断点
  int i,j,n;
  bool _isDuplicate; //重複があるか
  int memory[2][11]; //入れ替えた要素の定義を保存
  int mem_n; //memory[][]まわすための添字
  int parent_elem;
  int x,y,v,z; //ループの添字
  int _candidate; //parent_elemのペアのkey(候補)
  double rand; //乱数発生用

  rand = (double)Rand()/((double)(RANDOM_MAX+1));
  if(rand > P_CROSSOVER){
    printf("NO CROSSOVER¥n");
    return;
  }

  //交叉位置
  n_cross1 = Rand()%16+1; //n_cross = 1,...,17 (とりあえずハードコーディング...)
  n_cross2 = n_cross1 + 11;

  //交叉
  PrintCrossover(BEFORE, parent1, parent2, *child1, *child2, n_cross1, n_cross2);
  init_key_options();
  be_empty(*child1);

  mem_n=0;
  for(j=n_cross1; j<n_cross2; j++){
    //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
    keyboards[*child1][j] = keyboards[parent2][j];
    key_options[keyboards[parent2][j]] = Used;

    memory[0][mem_n] = keyboards[parent1][j];
    memory[1][mem_n] = keyboards[parent2][j];
```

```
     mem_n++;
    }
    //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)
    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[parent1][j]] == -1) _isDuplicate = true;
      }
      if(_isDuplicate == false){
        keyboards[*child1][j] = keyboards[parent1][j];
        key_options[keyboards[parent1][j]] = Used;
      }
    }

    //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)
    for(j=n_cross2; j<LEN_KEYS; 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[parent1][j]] == -1) _isDuplicate= true;
      }
      if(_isDuplicate == false){
        keyboards[*child1][j] = keyboards[parent1][j];
        key_options[keyboards[parent1][j]] = Used;
      }
    }

    //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
    for(j=0; j<LEN_KEYS; j++){
      if(keyboards[*child1][j] == -2){
        //EMPTY部分の親の要素を保存
        parent_elem = keyboards[parent1][j];
        //フローチャート始まり
        while(1){
          for(v=0;v<11; v++){
            for(z=0; z<2; z++){
              //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
              if(memory[z][v] == parent_elem){
                if(z == 0){
                  _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;
      }
      if(_isDuplicate == false){
        keyboards[*child2][j] = keyboards[parent2][j];
        key_options[keyboards[parent2][j]] = Used;
      }
    }
    //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)
    for(j=n_cross2; j<LEN_KEYS; 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;
      }
      if(_isDuplicate == false){
        keyboards[*child2][j] = keyboards[parent2][j];
        key_options[keyboards[parent2][j]] = Used;
      }
    }
    //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
    for(j=0; j<LEN_KEYS; j++){
      if(keyboards[*child2][j] == -2){
         //EMPTY部分の親の要素を保存
        parent_elem = keyboards[parent2][j];
         //フローチャート始まり
        while(1){
          for(v=0;v<11; v++){
```

```
          for(z=0; z<2; z++){
             //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
            if(memory[z][v] == parent_elem){
              if(z == 0){
                _candidate = memory[1][v];
                goto OUT2;
              }else{
                _candidate = memory[0][v];
                goto OUT2;
              }
            }
          }
        }
        //_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
        OUT2:
            if(key_options[_candidate] != -1){
              keyboards[*child2][j] = _candidate;
              key_options[keyboards[*child2][j]] = Used;
              break;
            }else{
              parent_elem = _candidate;
              memory[0][v] = -5;
              memory[1][v] = -5;
            }
      }
    }
  }
  fitness[*child1] = ObjFunc(*child1);
  fitness[*child2] = ObjFunc(*child2);
  PrintCrossover(AFTER, parent1, parent2, *child1, *child2, n_cross1, n_cross2);
}

//一定確率でキー入れ替わり
void Mutation(int child){

  int n_mutate1;
  int n_mutate2;
  int x;
  double rand;

  rand = (double)Rand()/((double)(RANDOM_MAX+1));    //0<=num<1とする
  if(rand<P_MUTATION){
    //突然変異位置
    n_mutate1 = Rand()%LEN_KEYS;  //n_mutate1=0,...,29
    n_mutate2 = Rand()%LEN_KEYS;  //n_mutate2=0,...,29
    //突然変異
    PrintMutation(BEFORE,child,n_mutate1,n_mutate2);
    x = keyboards[child][n_mutate1];
    keyboards[child][n_mutate1] = keyboards[child][n_mutate2];
    keyboards[child][n_mutate2] = x;
    fitness[child] = ObjFunc(child);
    PrintMutation(AFTER,child,n_mutate1,n_mutate2);
  }
}

```

```
//ファイルから文字列入力
void fileread(){
  FILE *fp;
  int i=0;
     char fname[] = "learning.txt";
     char text[256];

     fp = fopen(fname, "r");
     if(fp == NULL) {
       exit(1);
     }

     for (i = 0; fgets(text, 256, fp) != NULL; i++){
          strcpy(str[i], text);
     }
     STRINGS = i;

     fclose(fp);
}

//ファイルに結果出力
void filewrite(int keyboard[],char* phase){
  int i;
  char filename[256];
  strcpy(filename,name);
  FILE* f = fopen(strcat(strcat(filename,phase),"_result.txt"), "w");

  for(i=0;i<LEN_KEYS;i++){
    fprintf(f, "%c¥n", alphabet[keyboard[i]]);
  }

  fclose(f);
}

//CSVファイルに結果出力

void filewrite_csv(int gen){
  int i;
  char filename[256];
  strcpy(filename,name);
  FILE* f;

  if(gen==0){
    f = fopen(strcat(filename,"_Maxfitness_result.csv"),"w");
    fprintf(f, "gen,Maxfitness¥n");
    //f = fopen(strcat(filename,"_fitnessAve_result.csv"),"w");
    //fprintf(f, "gen,fitnessAve¥n");
  }else{
    f = fopen(strcat(filename,"_Maxfitness_result.csv"),"a");
    fprintf(f, "%d,%d¥n", gen,max);
    //f = fopen(strcat(filename,"_fitnessAve_result.csv"),"a");
    //fprintf(f, "%d,%f¥n", gen,(double)sumfitness/(double)POP_SIZE);
  }

  fclose(f);
}

//メイン関数
int main(int argc,char **argv){
  int gen,i;

  Srand((unsigned) time(NULL)); //seed値変更

  printf("名前を入力してください -> ");
  scanf("%s",name);
  fileread();

  keyweightcal(); //キーの重み付け
  Initialize(); //初期化

  filewrite_csv(0);

  for(gen=1;gen<=MAX_GEN;gen++){
    Generation(gen);
    if(gen==1)
      filewrite(keyboards[n_max],"_first");
    if(gen==MAX_GEN/2)
      filewrite(keyboards[n_max],"_intermediate");
    if(gen==MAX_GEN)
      filewrite(keyboards[n_max],"_final");
    filewrite_csv(gen);
  }
  Statistics();
  PrintStatistics(-1);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "conio.h"


struct eachkey{
    int key_num; //キーの位置番号
    double keytimes; //各キー入力時間保存用
};

struct eachkey keys[30];
int keyweight[30]; //各キーの重み付け保存用
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',',','.','/'};

double getETime(){
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec + (double)tv.tv_usec*1e-6;
}

/*指定されたキーが押される時間を計測*/
double keytime(char c){
    double start, end, time;
    char getkey;
    int ec;
    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");
    while(1){
        int space = getch();
        if(space == ' ') break;
    }
    start = getETime();
    printf("「%c」を入力してください>> ", c);
    while(1){
        ec = getch();
        if(ec == c) break;
    }
    printf("%c¥n", ec);
    end = getETime();
    time = end - start;
    printf("キー入力時間：%lf秒¥n", time);
    return time;
}

int GetRandom(int min,int max){
     return min + (int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}

/*ランダムにキー入力を受付＋時間計測*/
void keyweightcal(){
    int i, n, k, h;
    struct eachkey tmp;
    for(i=0;i<30;i++) keys[i].key_num = i; //キーの位置番号設定
    for(i=0;i<30;i++) keys[i].keytimes = -1; //keytimes初期化
    for(i=0;i<30;i++){
      n = GetRandom(0,29);
      while(keys[n].keytimes!=-1){ //記録されていないキーを探すs
        n = GetRandom(0,29);
      }
      keys[n].keytimes = keytime(keyplace[n]);
      keyweight[n] = keys[n].keytimes * 1000; //小数点をなくした数値
    }
    /*結果確認用*/
    for(i=0;i<30;i++){
      printf("キー番号(%d)の重み:%d¥n",i,keyweight[i]);
    }
}
```

```html
<!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>
    &nbsp;&nbsp;<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>
    <button type="button" id="pos15">H</button>
    <button type="button" id="pos16">J</button>
    <button type="button" id="pos17">K</button>
    <button type="button" id="pos18">L</button>
    <button type="button" id="pos19">；</button>
    <br>
    &nbsp;&nbsp;&nbsp;&nbsp;<button type="button" id="pos20">Z</button>
    <button type="button" id="pos21">X</button>
    <button type="button" id="pos22">C</button>
    <button type="button" id="pos23">V</button>
    <button type="button" id="pos24">B</button>
    <button type="button" id="pos25">N</button>
    <button type="button" id="pos26">M</button>
    <button type="button" id="pos27">,</button>
    <button type="button" id="pos28">.</button>
    <button type="button" id="pos29">?</button>
    <br><br>
    </div>
    <input type="file" id="selfile"><br>

    <script>
      var obj1 = document.getElementById("selfile");
      //ダイアログでファイルが選択された時
      obj1.addEventListener("change",function(evt){
        var file = evt.target.files;
        //FileReaderの作成
        var reader = new FileReader();
        //テキスト形式で読み込む
        reader.readAsText(file[0]);
        //読込終了後の処理
        reader.onload = function(ev){
          var K = reader.result.split(/¥n/);
          document.getElementById("pos0").innerText = K[0];
          document.getElementById("pos1").innerText = K[1];
          document.getElementById("pos2").innerText = K[2];
          document.getElementById("pos3").innerText = K[3];
          document.getElementById("pos4").innerText = K[4];
          document.getElementById("pos5").innerText = K[5];
          document.getElementById("pos6").innerText = K[6];
          document.getElementById("pos7").innerText = K[7];
          document.getElementById("pos8").innerText = K[8];
          document.getElementById("pos9").innerText = K[9];
          document.getElementById("pos10").innerText = K[10];
          document.getElementById("pos11").innerText = K[11];
          document.getElementById("pos12").innerText = K[12];
          document.getElementById("pos13").innerText = K[13];
          document.getElementById("pos14").innerText = K[14];
          document.getElementById("pos15").innerText = K[15];
          document.getElementById("pos16").innerText = K[16];
          document.getElementById("pos17").innerText = K[17];
          document.getElementById("pos18").innerText = K[18];
          document.getElementById("pos19").innerText = K[19];
          document.getElementById("pos20").innerText = K[20];
          document.getElementById("pos21").innerText = K[21];
          document.getElementById("pos22").innerText = K[22];
          document.getElementById("pos23").innerText = K[23];
          document.getElementById("pos24").innerText = K[24];
          document.getElementById("pos25").innerText = K[25];
          document.getElementById("pos26").innerText = K[26];
          document.getElementById("pos27").innerText = K[27];
          document.getElementById("pos28").innerText = K[28];
          document.getElementById("pos29").innerText = K[29];
        }
      },false);
    </script>

  </body>
</html>
```