```c
//
// 9_2.cpp
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const int data = 2000;      // データ点数
const int M = 10;           // 行列の行数
const double EPS = 1e-6;    // ピボットが 0 かどうか判定する値

typedef double Matrix[M+1][M+2];    // 行列の型を定義

typedef struct{
    double x;       // 測定データ x
    double y;       // 測定データ y
}input_data;

// ファイルを読み込む関数
int scan_file(const char* ifile, input_data* Indata){
    int i = 0;
    FILE *ifp = NULL;
    if((ifp = fopen(ifile, "r")) == NULL){
        fprintf(stderr, "Can't open file %s\n", ifile);
        exit(-1);
    }
    while(fscanf(ifp, "%lf,%lf", &Indata[i].x, &Indata[i].y) != EOF){
        i++;
        if(i==data){
            printf("Warning: Too many data points(n > %d). Stop the process.",data);
            exit(0);
        }
    }
    fclose(ifp);
    return i;   // ファイルのデータ点の個数を返す
}

// べき乗計算をする関数
double ipow(double p, int n){
    double s = 1;
    for (int k=0; k<n; k++){
        s *= p;
    }
    return s;
}

// 最大値を検索し、その行数を返す関数
int search_max(Matrix A, int k, const int N){
    double max = 0.0;   // 最大値を格納する変数
    int max_row = k;    // 最大値がある行数を格納する変数
    for(int i=0; i<N-k; i++){
        if(fabs(max)<fabs(A[i+k][k]) && fabs(A[i+k][k])>EPS){
            max = A[i+k][k];
            max_row = i+k;
        }
    }
    return max_row; // 最大値がある行数を返す
}

// 行を入れ替える関数
void change_row(Matrix A, int i, int max_row, const int N){
    for(int j=0; j<N+1; j++){
        double tmp = A[i][j];   // 一時的に格納する変数
        A[i][j] = A[max_row][j];
        A[max_row][j] = tmp;
    }
}

// 前進消去
void forward_elimination(Matrix A, const int N){
    for(int i=0; i<N; i++){
        int max_row = search_max(A, i, N);  // 最大値がある行数を検索
        if(i != max_row){
            change_row(A, i, max_row, N);   // 最大値がある行数と行を入れ替え
        }
```

```
        double pivot = A[i][i];
        if(fabs(pivot) > EPS){          // ピボットが 0 でなければ
            for(int j=i; j<N+1; j++){   // 各列をピボットで除算
                A[i][j] /= pivot;
            }
            for(int k=i+1; k<N; k++){   // 掃き出し計算
                double tmp = A[k][i];   // 一時的に値を保存する変数
                for(int j=i; j<N+1; j++){
                    A[k][j] -= tmp * A[i][j];
                }
            }
        }
    }
}

// 後退代入
void backward_substitution(Matrix A, const int N){
    for(int i=N-2; i>=0; i--){
        for(int j=N-1; j>i; j--){
            A[i][N] -= A[i][j]*A[j][N];
        }
    }
}

// ガウス消去法
void gauss_elimination(Matrix A, const int N){
    forward_elimination(A, N);      // 前進消去
    backward_substitution(A, N);    // 後退代入
}

// 解を表示する関数
void print_ans(Matrix A, const int N){
    printf("\n----- Answers : m = %d -----\n",N-1);
    for(int i=0; i<N; i++){
        printf("a_%2d = %9.6f\n", i, A[i][N]);  // 連立方程式の解が近似多項式の係数
    }
}

// 最小二乗法を行う関数
void least_squares_method(input_data* Indata, Matrix A, int m, int datanum){
    double s[2*M + 1] = {0.0};
    double t[M + 1] = {0.0};

    // s の計算
    for(int i=0; i<2*m+1; i++){
        for(int j=0; j<datanum; j++){
            s[i] += ipow(Indata[j].x,i);
        }
    }
    // t の計算
    for(int i=0; i<m+1; i++){
        for(int j=0; j<datanum; j++){
            t[i] += Indata[j].y * ipow(Indata[j].x,i);
        }
    }
    // 係数行列 A の作成
    for(int i=0; i<m+1; i++){
        A[i][m + 1] = t[i];
        for(int j=0; j<m+1; j++){
            A[i][j] = s[j + i];
        }
    }
    // ガウス消去法
    gauss_elimination(A, m+1);
}

double calc_acod(input_data* Indata, Matrix A, int m, int datanum){
    double sst = 0.0; // 全変動の平方根
    double sse = 0.0; // 残差平方根
    double y_m = 0.0; // yの平均

    for(int i=0; i<datanum; i++){
        y_m += Indata[i].y;
    }
    y_m /= datanum;

    for(int i=0; i<datanum; i++){
```

```
        double y_p = 0.0; // 近似多項式のy値
        for(int j=0; j<m+1; j++){
            y_p += A[j][m + 1] * ipow(Indata[i].x, j);
        }
        sse += ipow(Indata[i].y - y_p, 2);
        sst += ipow(Indata[i].y - y_m, 2);
    }
    return 1.0 - ((sse/(datanum-m-1))/(sst/(datanum-1))); // 自由度調整済み決定係数の算出
}

// 各次数における決定係数の計算結果を画面に表示する関数
void print_cod_calculation(int m_min, int m_max, const int datanum1, input_data* Data_mat1){
    printf("\n----- adjusted R^2 of the calculation results <measurement data> -----\n");
    double max = 0;
    int degree = 1;
    for(int m = m_min;m<=m_max;m++){
        Matrix A = {0.0};   // 拡大行列を定義
        least_squares_method(Data_mat1, A, m, datanum1);    // 最小二乗法によって近似多項式の係数を算出
        double acod = calc_acod(Data_mat1, A, m, datanum1); // 計測データに対する 自由度調整済み決定係数 を計算
        printf("m =%2d\t%9.7f\t\n",m,acod);
        if(acod>max){
            max = acod;
            degree = m;
        }
    }
    printf("\n----- degree of adjusted R^2 at maximum  -----\n");
    printf("m =%2d\n",degree);
}

// メイン関数
int main(void){
    input_data Data_mat1[data] = {0.0};     // 計測データ用の配列を定義

    const char* ifile1 = "measurement_data.csv";    // 計測データのファイル名
    const int datanum1 = scan_file(ifile1, Data_mat1);  // ファイルを読み込みデータを配列に格納

    const int m_min = 1;    // 次数 m の最小値
    const int m_max = 10;   // 次数 m の最大値

    print_cod_calculation(m_min,m_max,datanum1,Data_mat1);  // 各次数における決定係数の計算結果を画面に表示する

    return 0;
}
```