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

/**
 * UFPE - Segundo projeto da disciplina de Calculo Numerico - Turma T7
 *
 * @author Cleber Alberto Cabral e Heloisa Bezerra Neves.
 * @teacher Andre Tiba
 */

/**
 * ---- Structs. ----
 */

/**
 * Struct a tupla (x, f(x))
 */
typedef struct {
    double x;
    double fx;
} Tupla;

/**
 * Struct que representa uma funcao matematica de ajustamento.
 */
typedef struct {
    double matriz[2][3];
    double a;
    double b;
    double erro;
    int ln;
    char* formula;
} Funcao;

/**
 * ---- Interfaces das funcoes auxiliares. ----
 */

int limpar_stdin();
void zerar(double matriz[2][3]);
void imprimir(double matriz[2][3]);
void pivotar(Funcao* funcao);

/**
 * ---- Metodo principal. ----
 */

/**
 * Metodo principal da aplicacao. Oferece ao usuario um menu de selecao de
 * duas opcoes. 1 = Informar o arquivo de dados. 2 = Sair da aplicacao.
 */
int main() {

    // ints utilitarios.
    int opcao, n, i, melhor;
    int nFunc = 10;
    double min, raiz;
    // Arquivo base contendo os dados de leitura
    FILE *file;
    // Variaveis auxiliares de leitura.
    char nomeArquivo[30], x[11], fx[11];
    char c;
```

```
    // Os valores lidos do arquivo.
    Tupla *valores;
    // As funcoes de ajustamento.
    Funcao funcoes[10];

    printf("------------------------------------------------\n");
    printf("-- UFPE - Calculo Numerico - Calculadora v1.0 --\n");
    printf("------------------------------------------------\n");
    printf("-- Prof: Andre Tiba                           --\n");
    printf("-- Alunos: CLEBER ALBERTO CABRAL F DA SILVA   --\n");
    printf("--         HELOISA BEZERRA NEVES              --\n");
    printf("------------------------------------------------\n");

    // Espero ate o usuario selecionar um opcao do menu.
    do {

        printf("Digite o numero da opcao desejada: \n1 - Digitar o nome do arquivo;\n2 -
        Sair do programa.\n");

    } while (((scanf("%d%c", &opcao, &c) != 2 || c != '\n') && limpar_stdin()) || opcao < 1
    || opcao > 2);

    if (opcao == 1) {

        printf("Informe o nome do arquivo ([nome].[extensao]): ");
        scanf(" %s[^\n]", nomeArquivo);

        // Hora de ler o arquivo.
        file = fopen(nomeArquivo, "r");

        // Algum erro ao ler o arquivo.
        if (file == NULL) {

            printf("Impossivel abrir o arquivo.\n");

        } else {

            printf("Processando o arquivo...\n");
            printf("FUNCAO \t\t\t RESIDUO \t\t\t RAIZ\n");

            // Lendo a quantidade de registros.
            fscanf(file, "%d", &n);

            valores = (Tupla*) malloc(n * sizeof(Tupla));

            // leio os numeros e os converto para double
            for (i = 0; i < n; i++) {
                fscanf(file, " %10s,%10s", x, fx);
                valores[i].x = atof(x);
                valores[i].fx = atof(fx);
            }

            // Zero a matriz do sistema de equacoes e o erro de cada funcao.
            for (i = 0; i < nFunc; i++) {
                funcoes[i].erro = 0;
                zerar(funcoes[i].matriz);
            }

            // Informo o corpo de cada funcao.
            funcoes[0].formula = "f_{1}(x) = %.4e*x + %.4e \t %.4e \t %.4e \n";
            funcoes[1].formula = "f_{2}(x) = %.4e*x^2 + %.4e*x \t %.4e \t %.4e \n";
            funcoes[2].formula = "f_{3}(x) = %.4e*x^3 + %.4e*x \t %.4e \t %.4e \n";
            funcoes[3].formula = "f_{4}(x) = %.4e*x^3 + %.4e*x^2 \t %.4e \t %.4e \n";
```

```
    funcoes[4].formula = "f_{5}(x) = %.4e*exp(%.4e*x) \t %.4e \t %.4e \n";
    funcoes[5].formula = "f_{6}(x) = (%.4e)*x^(%.4e) \t %.4e \t %.4e \n";
    funcoes[6].formula = "f_{7}(x) = %.4e*ln(x) + %.4e/x \t %.4e \t %.4e \n";
    funcoes[7].formula = "f_{8}(x) = %.4e*x + %.4e/x \t %.4e \t %.4e \n";
    funcoes[8].formula = "f_{9}(x) = %.4e*cos(x) + %.4e*x \t %.4e \t %.4e \n";
    funcoes[9].formula = "f_{10}(x) = 1/(%.4e*sen(x) + %.4e*exp(x)) \t %.4e \t %.4e
    \n";

    // Minimos quadrados de todas as funcoes.
    // Monto o sistema de equacoes em forma de matriz, na forma:
    //
    // A: somatorio(g0^2) + somatorio(g0g1) = somatorio(f(x)g0)
    // B: somatorio(g1g0) + somatorio(g1^2) = somatorio(f(x)g1)
    for (i = 0; i < n; i++) {
        // f(x) = ax + b
        funcoes[0].matriz[0][0] += valores[i].x * valores[i].x; // g0^2 = x^2
        funcoes[0].matriz[0][1] += valores[i].x; // g0 x g1 = X x 1
        funcoes[0].matriz[0][2] += valores[i].fx * valores[i].x; // f(x) x g0
        funcoes[0].matriz[1][0] += valores[i].x; // g0 x g1 = X x 1
        funcoes[0].matriz[1][1] += 1; // g1^2 = 1^2 = 1
        funcoes[0].matriz[1][2] += valores[i].fx; // f(x) x g1
        // f(x) = ax^2 + bx
        funcoes[1].matriz[0][0] += pow(valores[i].x, 4); // g0^4 = (x^2)^2
        funcoes[1].matriz[0][1] += pow(valores[i].x, 3); // g0 x g1 = X^2 x X = x^3
        funcoes[1].matriz[0][2] += valores[i].fx * pow(valores[i].x, 2); // f(x) x
        g0 = f(x) x x^2
        funcoes[1].matriz[1][0] += pow(valores[i].x, 3); // g0 x g1 = X^2 x X = x^3
        funcoes[1].matriz[1][1] += pow(valores[i].x, 2); // g1^2 = x^2
        funcoes[1].matriz[1][2] += valores[i].fx * valores[i].x; // f(x) x g1
        // f(x) = ax^3 + bx
        funcoes[2].matriz[0][0] += pow(valores[i].x, 6); // g0^6 = (x^3)^2
        funcoes[2].matriz[0][1] += pow(valores[i].x, 4); // g0 x g1 = X^3 x X = x^4
        funcoes[2].matriz[0][2] += valores[i].fx * pow(valores[i].x, 3); // f(x) x
        g0 = f(x) x x^3
        funcoes[2].matriz[1][0] += pow(valores[i].x, 4); // g0 x g1 = X^3 x X = x^4
        funcoes[2].matriz[1][1] += pow(valores[i].x, 2); // g1^2 = x^2
        funcoes[2].matriz[1][2] += valores[i].fx * valores[i].x; // f(x) x g1
        // f(x) = ax^3 + bx^2
        funcoes[3].matriz[0][0] += pow(valores[i].x, 6); // g0^6 = (x^3)^2
        funcoes[3].matriz[0][1] += pow(valores[i].x, 5); // g0 x g1 = X^3 x X^2 =
        x^5
        funcoes[3].matriz[0][2] += valores[i].fx * pow(valores[i].x, 3); // f(x) x
        g0 = f(x) x x^3
        funcoes[3].matriz[1][0] += pow(valores[i].x, 5); // g0 x g1 = X^3 x X^2 =
        x^5
        funcoes[3].matriz[1][1] += pow(valores[i].x, 4); // g1^2 = (x^2)^2 = x^2
        funcoes[3].matriz[1][2] += valores[i].fx * pow(valores[i].x, 2); // f(x) x g1
        // f(x) = ae^bx
        funcoes[4].ln = 1;
        funcoes[4].matriz[0][0] += 1; // g0^2 = 1^2
        funcoes[4].matriz[0][1] += valores[i].x; // g0 x g1 = 1 x X = X
        funcoes[4].matriz[0][2] += log(valores[i].fx); // ln(f(x)) x g0 = ln(f(x))
        x 1 = ln(f(x))
        funcoes[4].matriz[1][0] += valores[i].x; // g0 x g1 = 1 x X = X
        funcoes[4].matriz[1][1] += pow(valores[i].x, 2); // g1^2 = x^2
        funcoes[4].matriz[1][2] += log(valores[i].fx) * valores[i].x; // ln(f(x)) x
        g1 = ln(f(x)) x X
        // f(x) = ax^b
        funcoes[5].ln = 1;
        funcoes[5].matriz[0][0] += 1; // g0^2 = 1^2
        funcoes[5].matriz[0][1] += log(valores[i].x); // g0 x g1 = 1 x ln(x) = ln(x)
        funcoes[5].matriz[0][2] += log(valores[i].fx); // ln(f(x)) x g0 = ln(f(x))
        x 1 = ln(f(x))
```

```
            funcoes[5].matriz[1][0] += log(valores[i].x); // g0 x g1 = 1 x ln(x) = ln(x)
            funcoes[5].matriz[1][1] += pow(log(valores[i].x), 2); // g1^2 = ln(x)^2
            funcoes[5].matriz[1][2] += log(valores[i].fx) * log(valores[i].x); //
            ln(f(x)) x g1 = ln(f(x)) x ln(x)
            // f(x) = aln(x) + b/x
            funcoes[6].matriz[0][0] += pow(log(valores[i].x), 2); // g0^2 = ln(x)^2
            funcoes[6].matriz[0][1] += log(valores[i].x) / valores[i].x; // g0 x g1 =
            ln(x) x 1/x = ln(x)/x
            funcoes[6].matriz[0][2] += valores[i].fx * log(valores[i].x); // f(x) x g0
            = f(x) x ln(x)
            funcoes[6].matriz[1][0] += log(valores[i].x) / valores[i].x; // g0 x g1 =
            ln(x) x 1/x = ln(x)/x
            funcoes[6].matriz[1][1] += pow(1/valores[i].x, 2); // g1^2 = (1/x)^2
            funcoes[6].matriz[1][2] += valores[i].fx / valores[i].x; // f(x) x g1 =
            f(x)/x
            // f(x) = ax + b/x
            funcoes[7].matriz[0][0] += pow(valores[i].x, 2); // g0^2 = x^2
            funcoes[7].matriz[0][1] += 1; // g0 x g1 = x x 1/x = x/x = 1
            funcoes[7].matriz[0][2] += valores[i].fx * valores[i].x; // f(x) x g0 =
            f(x) x ln(x)
            funcoes[7].matriz[1][0] += 1; // g0 x g1 = x x 1/x = x/x = 1
            funcoes[7].matriz[1][1] += pow(1/valores[i].x, 2); // g1^2 = (1/x)^2
            funcoes[7].matriz[1][2] += valores[i].fx / valores[i].x; // f(x) x g1 =
            f(x)/x
            // f(x) = acos(x) + bx
            funcoes[8].matriz[0][0] += pow(cos(valores[i].x), 2); // g0^2 = cos(x)^2
            funcoes[8].matriz[0][1] += cos(valores[i].x) * valores[i].x; // g0 x g1 =
            cos(x) x X
            funcoes[8].matriz[0][2] += valores[i].fx * cos(valores[i].x); // f(x) x g0
            = f(x) x cos(x)
            funcoes[8].matriz[1][0] += cos(valores[i].x) * valores[i].x; // g0 x g1 =
            cos(x) x X
            funcoes[8].matriz[1][1] += pow(valores[i].x, 2); // g1^2 = x^2
            funcoes[8].matriz[1][2] += valores[i].fx * valores[i].x; // f(x) x g1 =
            f(x) x X
            // f(x) = 1/(asen(x) + be^x)
            funcoes[9].matriz[0][0] += pow(sin(valores[i].x), 2); // g0^2 = sen(x)^2
            funcoes[9].matriz[0][1] += exp(valores[i].x) * sin(valores[i].x); // g0 x
            g1 = e^(x) x sen(x)
            funcoes[9].matriz[0][2] += sin(valores[i].x)/valores[i].fx; // 1/f(x) x g0
            = sin(x)/f(x)
            funcoes[9].matriz[1][0] += exp(valores[i].x) * sin(valores[i].x); // g0 x
            g1 = e^(x) x sen(x)
            funcoes[9].matriz[1][1] += pow(exp(valores[i].x), 2); // g1^2 = (e^x)^2
            funcoes[9].matriz[1][2] += exp(valores[i].x)/valores[i].fx; // 1/f(x) x g1
            = x/f(x)
        }

        // Pivoto parcialmente cada matriz.
        for (i = 0; i < nFunc; i++) {
            pivotar(&funcoes[i]);
            if (funcoes[i].ln == 1) {
                funcoes[i].a = exp(funcoes[i].a);
            }
        }

        // Calculo o erro de cada funcao.
        for (i = 0; i < n; i++) {
            // f(x) = ax + b
            funcoes[0].erro += pow((funcoes[0].a * valores[i].x + funcoes[0].b) -
            valores[i].fx, 2);
            // f(x) = ax^2 + bx
            funcoes[1].erro += pow((funcoes[1].a * pow(valores[i].x, 2) + funcoes[1].b *
```

```
                    valores[i].x) - valores[i].fx, 2);
                // f(x) = ax^3 + bx
                funcoes[2].erro += pow((funcoes[2].a * pow(valores[i].x, 3) + funcoes[2].b *
                    valores[i].x) - valores[i].fx, 2);
                // f(x) = ax^3 + bx^2
                funcoes[3].erro += pow((funcoes[3].a * pow(valores[i].x, 3) + funcoes[3].b *
                    pow(valores[i].x, 2)) - valores[i].fx, 2);
                // f(x) = ae^bx
                funcoes[4].erro += pow((funcoes[4].a * exp(funcoes[4].b * valores[i].x)) -
                valores[i].fx, 2);
                // f(x) = ax^b
                funcoes[5].erro += pow((funcoes[5].a * pow(valores[i].x, funcoes[5].b)) -
                valores[i].fx, 2);
                // f(x) = aln(x) + b/x
                funcoes[6].erro += pow((funcoes[6].a * log(valores[i].x) + funcoes[6].b /
                valores[i].x) - valores[i].fx, 2);
                // f(x) = ax + b/x
                funcoes[7].erro += pow((funcoes[7].a * valores[i].x + funcoes[7].b / valores
                [i].x) - valores[i].fx, 2);
                // f(x) = acos(x) + bx
                funcoes[8].erro += pow((funcoes[8].a * cos(valores[i].x) + funcoes[8].b *
                valores[i].x) - valores[i].fx, 2);
                // f(x) = 1/(asen(x) + be^x)
                funcoes[9].erro += pow((1/(funcoes[9].a * sin(valores[i].x) + funcoes[9].b *
                    exp(valores[i].x))) - valores[i].fx, 2);
            }

            // Agora busco encontrar a funcao que possui o menor erro.
            min = sqrt(funcoes[0].erro);
            melhor = 0;

            for (i = 0; i < nFunc; i++) {

                raiz = sqrt(funcoes[i].erro);

                if (raiz < min) {
                    min = raiz;
                    melhor = i;
                }

                // exibo cada funcao.
                printf(funcoes[i].formula, funcoes[i].a, funcoes[i].b, funcoes[i].erro, raiz
                );
            }

            // Econtrada a funcao com o menor erro, agora a exibo.
            printf("\n");
            printf("melhor funcao de ajustamento: \n");
            printf(funcoes[melhor].formula, funcoes[melhor].a, funcoes[melhor].b, funcoes[
            melhor].erro, min);
            free(valores);
        }

        fclose(file);
    }

    return 0;
}

/**
 * Realiza a pivotacao parcial da matriz associada a uma funcao, e enconto
 * seus indices A e B.
 */
```

```c
void pivotar(Funcao* funcao) {

    int i, j;
    double fator, aux;

    // Troca de linhas.
    if (fabs(funcao->matriz[0][0]) < fabs(funcao->matriz[1][0])) {
        for (i = 0; i < 3; i++) {
            aux = funcao->matriz[1][i];
            funcao->matriz[1][i] = funcao->matriz[0][i];
            funcao->matriz[0][i] = aux;
        }
    }

    // Fator de subtracao das linhas.
    fator = funcao->matriz[1][0] / funcao->matriz[0][0];

    // L2 = L2 - fator * L1
    for (j = 0; j < 3; j++) {
        funcao->matriz[1][j] = funcao->matriz[1][j] - funcao->matriz[0][j] * fator;
    }

    // Os indices A e B.
    funcao->b = funcao->matriz[1][2] / funcao->matriz[1][1];
    funcao->a = (funcao->matriz[0][2] - funcao->b * funcao->matriz[0][1]) / funcao->matriz[0
    ][0];
}

/**
 * Zero todas as posicoes de uma matriz.
 */
void zerar(double matriz[2][3]) {
    int i, j;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            matriz[i][j] = 0;
        }
    }
}

/**
 * Imprimo uma matriz na tela.
 */
void imprimir(double matriz[2][3]) {
    int i, j;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%lf ", matriz[i][j]);
        }
        printf("\n");
    }
}

/**
 * Limpa os resquicios de memoria do stdin.
 */
int limpar_stdin() {
    while (getchar() != '\n');
    return 1;
}
```