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

void imprimeAviso();

double **alocaMatriz(int n) {
    /* Caso haja memória disponível, aloca dinamicamente
    uma matriz de double com n linhas e n + 1 colunas e
    devolve um ponteiro para essa matriz; caso não haja
    memória disponível, devolve um ponteiro nulo. */
    int i, j;
    double **M;

    M = malloc(sizeof(double*) * n);

    if (M == NULL) // Sem memória disponível
        return NULL;

    for (i = 0; i < n; ++i) {
        M[i] = malloc(sizeof(double) * (n + 1));
        if (M[i] == NULL) { // Sem memória disponível
            for (j = 0; j < i; ++j)
                free(M[j]);
            free(M);
            return NULL;
                }

    }
    return M;
} /* ================= Fim alocaMatriz ================= */

void desalocaMatriz(double **M, int n) {
    // Libera a memória ocupada pela matriz
    // de double M de n linhas e n + 1 colunas.
    int i;

    for (i = 0; i < n; ++i)
        free(M[i]);
    free(M);
} /* ============== Fim desalocaMatriz ============== */

void leMatriz(double **M, int n, FILE *f) {
    // Lê valores para uma matriz de double alocada
    // dinamicamente com n linhas e n+1 colunas.
    int i, j;
    double diag, somaLin = 0, somaCol = 0;

    for (i = 0; i < n; ++i){
        for (j = 0; j <= n; ++j){
            fscanf(f, "%lf", &M[i][j]);
        }
    }
} /* ================= Fim leMatriz ================= */

int verificaCriterios(double **M, int n) {
    /* Lê valores para uma matriz de double alocada
        dinamicamente com n linhas e n + 1 colunas. */
    int i, j, flagLin = 0, flagCol = 0, tipo = 0;
    double somaLin = 0, somaCol = 0, diag = 0;

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            diag = fabs(M[i][i]);

            if (j != i){
                somaLin += fabs(M[i][j]);
                somaCol += fabs(M[j][i]);
            }
        }

        if((somaLin / diag) > 1)
            flagLin = 1;
        if((somaCol / diag) > 1)
            flagCol = 1;
        somaLin = 0;
        somaCol = 0;
    }

    /* Verifica se os criterios das linhas ou
    colunas é satisfeito (de acordo com a matriz
    de coeficientes) utilizando de uma flag para
    verificar essas condicoes */
    if(flagLin == 0 && flagCol == 0){
        printf("Os criterios foram satisfeitos\n\n");
        return tipo;
    }else{
        if(flagLin){
            printf("O criterio das linhas não foi satisfeito\n");
            tipo++;
        }
        if(flagCol){
            printf("O criterio das colunas não foi satisfeito\n\n");
            tipo++;
        }
        if(tipo == 2){
            printf("Os criterios nao foram satisfeitos, logo Gauss-Seidel nao ira convergir\n");
        }
        return tipo;
    }
} /* ================= Fim verificaCriterios ================= */

void imprimeMatriz(double **M, int n) {
    /* Imprime uma matriz de double alocada dinamicamente,
    com n linhas e n + 1 colunas. */
    int i, j;

    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j)
            printf("%10.3lf ", M[i][j]);
        printf("\n");
    }
} /* ================= Fim imprimeMatriz ================= */

/* Sessão da Opção C (conversao de bases) do menu */

char *conversao(double decimal, int base) {
    /* Se houver memória disponível, retorna um
    ponteiro para uma string contendo o número
    decimal convertido para a base indicada
    no parâmetro; caso contrário, retorna um
    ponteiro nulo. */

    const int tamanhoResposta = 50;
    const int maxCasasDecimais = 20;
    int i, j, pos = 0;
    int parteInteira = decimal;
    double parteFracionaria = decimal - parteInteira;
    char *resposta = malloc(sizeof(char) * (tamanhoResposta + 1));

    if (!resposta)
        return NULL;

    // Conversão da parte inteira
    int negativo = (decimal < 0);

    if (negativo) {
        resposta[pos++] = '-';
        parteInteira *= -1;
        parteFracionaria *= -1;
    } do {
        int resto = parteInteira % base;
        char digito = (resto <= 9 ? '0' + resto : 'A' + resto - 10);
        resposta[pos++] = digito;
        parteInteira /= base;
    } while (parteInteira > 0 && pos < tamanhoResposta);

    for (i = negativo, j = pos - 1; i < j; ++i, --j) {
        char aux = resposta[i];
        resposta[i] = resposta[j];
        resposta[j] = aux;
    }

    // Conversão da parte racional
    if (parteFracionaria != 0) {
        resposta[pos++] = '.';
        int tamanho = 0;

        while (parteFracionaria != 0 && tamanho < maxCasasDecimais && pos < tamanhoResposta) {
            parteFracionaria *= base;
            parteInteira = (int) parteFracionaria;
            char digito = (parteInteira <= 9 ? '0' + parteInteira : 'A' + parteInteira - 10);
            resposta[pos++] = digito;
            ++tamanho;
            if (parteFracionaria >= 1.0)
                parteFracionaria -= parteInteira;
        }
    }
    resposta[pos] = '\0';

    return resposta;
}

void imprimeConversao() {
    // Exibe os numeros ja convertidos para as bases binaria, octal e hexadecimal.

    double decimal;
    int i, bases[] = {2, 8, 16};
    char *nomes[] = {"Binario", "Octal", "Hexadecimal"};
    char *resultado;

    printf("\nInsira o numero decimal a ser convertido");
    printf("\n(utilize um ponto para numeros com parte fracionaria): ");
    scanf("%lf", &decimal);
    getchar();
    printf("\nDecimal: %lf\n", decimal);

    for (i = 0; i < 3; ++i) {
        resultado = conversao(decimal, bases[i]);
        if (!resultado) {
            printf("Memoria insuficiente!\n");
            imprimeAviso();
            return;
        }
        printf("%s: %s\n", nomes[i], resultado);
        free(resultado);
    }
    imprimeAviso();
}

/* Opção S */

double novoResultado(double **M, double x[1000], int j, int i, int n){

    int a, k;
    double soma = 0;

    for (a = n-1, k = 1; a >= 0; a--){
        if (a != j){
            soma = soma + (M[j][(i-k)%n] * -x[i-k]);
            k++;
        }
    }
    soma = (soma + M[j][n]) / M[j][j];

    return soma;
}

void aplicaGaussSeidel(double **M, int n) {
    /* Recebe M (a matriz aumentada de um SL com n variáveis
    e n equações) e a transforma em uma matriz aumentada de
    um SL diagonal equivalente */

    int i, j, k, l, flag = 1;
    double x[100000] = {0};
    double aux;
    int a;
    i = n;
    int iteracoes = 1000;

    while(iteracoes--){
        for (j = 0; j < n; j++){
            x[i] = novoResultado(M, x, j, i, n);
            i++;
        }
        for (k = i-n; k < n; k++){
            flag = 0;
            aux = x[k];
            if ((aux - ((int)aux)) > 0.000000009)
                flag = 1;
        }
        if (!flag)
            iteracoes = 0;
    }
    j = i - n;

    printf("Atraves do metodo de Gauss-Seidel:\n");

    for (l = 0; l < n; l++, j++)
        printf("x%d = %lf\n", l + 1, x[j]);
}

void resolveSL() {
    /* Funcao que resolve o sistema linear contido
       no arquivo a ser lido */
    FILE *f;
    char nome[100];
    double **M, *x;
    int n, i, j, tipo;
    int criterio;

    printf("\nDigite o nome do arquivo de texto contendo o sistema linear: ");
    scanf("%s", nome);
    getchar();
    f = fopen(nome, "r");

    if (f == NULL) {
        printf("\nArquivo nao encontrado!\n");
        imprimeAviso();
        return;
    }

    fscanf(f, "%d", &n);
    M = alocaMatriz(n);
    x = malloc(sizeof(double) * n);

    if (M == NULL || x == NULL) {
        printf("Memoria insuficiente para alocar a matriz!\n");
        if (M != NULL)
            desalocaMatriz(M, n);
        if (x != NULL)
            free(x);
        return;
    }

    /* Verifica o criterio das linhas ou colunas, caso
    seja possivel, aplica o metodo de gauss-seidel*/
    leMatriz(M, n, f);
    criterio = verificaCriterios(M, n);
    fclose(f);

    if (criterio != 2)
        aplicaGaussSeidel(M, n);

    imprimeAviso();
    desalocaMatriz(M, n);
    free(x);
} /* =============== Fim resolve SistemaLinear =============== */

/* =============== Opção E (equacao algebrica) =============== */

double aplicaLagrange(double *polinomio, int n, int positivas, int negativas) {
    // Aplica o teorema de Lagrange ao polinômio p e exibe
    // os intervalos onde se encontram as raízes reais negativas
    // e as raízes reais positivas da equação.
    int i, j, k[4];
    int a0negativo = polinomio[0] > 0.0 ? 1 : -1;
    int nImpar = (n & 1) ? -1 : 1;
    double L[4], B[4];
    // A primeira linha da matriz 'coeficientes' equivale a
    // p, a segunda a p1, a terceira a p2 e a quarta a p3.
    double coeficientes[4][n + 1];

    for (i = 0; i <= n; i++) {
        int indiceImpar = (i & 1) ? -1 : 1;
        coeficientes[0][i] = polinomio[i];
        coeficientes[1][n - i] = polinomio[i] * a0negativo;
        coeficientes[2][i] = polinomio[i] * indiceImpar * nImpar;
        coeficientes[3][n - i] = -polinomio[i] * indiceImpar * nImpar * a0negativo;
    }
    for (i = 0; i < 4; i++)
        k[i] = B[i] = -1;
    for (i = 0; i < 4; i++) {
        for (j = n; j >= 0; j--)
            if (coeficientes[i][j] < 0) {
                if (k[i] == -1)
                    k[i] = j;
                if (fabs(coeficientes[i][j]) > B[i])
                    B[i] = fabs(coeficientes[i][j]);
            }
        L[i] = 1 + pow((B[i] / coeficientes[i][n]), 1.0 / (n - k[i]));
    }
    printf("\nLimites para as raizes do polinomio:\n");
    if (positivas)
        printf("Positivas: %lf <= x+ <= %lf\n", 1.0 / L[1], L[0]);
    else
        printf("Nao ha raizes positivas para o polinomio.\n");
    if (negativas)
        printf("Negativas: %lf <= x- <= %lf\n", -L[2], -1.0 / L[3]);
    else
        printf("Nao ha raizes negativas para o polinomio.\n");

    return L[0];

} /* =============== Fim aplicaLagrange =============== */

int *calculaVariacoesEPermanencias(double *p, int n) {
    /* Recebe p, o vetor de coeficientes de um
    polinômio, e n, o grau de p. Se houver
    memória disponível, retorna um ponteiro
    para um vetor com o número de variações
    e de permanências de sinal de p. Caso
    contrário, retorna um ponteiro nulo. */
    int variacoes = 0, permanencias = 0, i;
    int sinal = p[0] > 0 ? 1 : -1;
    int *resultado = malloc(sizeof(int) * 2);

    if (!resultado)
        return NULL;

    for (i = 1; i <= n; i++)
        if (p[i] != 0) {
            int atual = p[i] > 0 ? 1 : -1;
            if (sinal != atual)
                variacoes++;
            else
                permanencias++;
            sinal = atual;
        }
    resultado[0] = variacoes;
    resultado[1] = permanencias;

    return resultado;
} /* ========== Fim calculaVariacoesEPermanencias ========== */

double metodoDeNewton(int grau, double vcoef[], double limiteSuperior){
    /* Recebe o grau do polinomio, juntamente com
    o vetor de polinomios(invertido) e o limite
    positivo superior para aplicar como x0 o limite
    superior positivo as iteracoes de newton.    */

    int i, count = 0;
    double *limites;
    double aux, aux_linha, xi = 0.0, xiprox = 0.0;

    xi = limiteSuperior;

    // Loop que garante as condicoes maximas das operacoes
    while(count < 1000 && fabs(xi - xiprox) > 0.00000001){
        aux = 0;
        aux_linha = 0;

        if(xiprox != 0.0)
            xi = xiprox;
        for (i = 0; i <= grau; i++){
            aux += vcoef[i] * pow(xi, i);                  // soma da f(x)
            aux_linha += i * vcoef[i] * pow(xi, (i - 1)); // soma de f'(x)
        }
        xiprox = xi - (aux/aux_linha);
        count += 1;
    }
    printf("\nPor meio do metodo de Newton, a aproximacao da raiz é: %lf\n", xiprox);

    return 0.0;
} /* =============== Fim metodoDeNewton =============== */

void resolveEquacaoAlgebrica() {
    /* Recebe um polinômio como entrada, aplica o teorema
    de Lagrange e usa do método de Newton calculo da
    aproximacao de uma raiz, usando do limite superior
    intervalo indicado positivo pelo metodo de Lagrange */

    int grauPolinomio, i, raizesImpares;
    double *polinomio, limiteEsq, limiteDir, aux;

    printf("\nQual o grau do polinomio? ");
    scanf("%d", &grauPolinomio);

    if (grauPolinomio < 1) {
        printf("\nO grau do polinomio deve ser no minimo 1!\n");
        getchar();
        imprimeAviso();
        return;
    }
    polinomio = malloc(sizeof(double) * (grauPolinomio + 1));
    if (polinomio == NULL) {
        printf("\nMemoria insuficiente para alocar os coeficientes!\n");
        imprimeAviso();
        return;
    }
    printf("\nDigite os coeficientes a partir de an ate a0: ");

    for (i = grauPolinomio; i >= 0; i--)
        scanf("%lf", &polinomio[i]);

    if (polinomio[grauPolinomio] <= 0.0) {
        printf("\nan deve ser maior do que zero!\n");
        getchar();
        imprimeAviso();
        return;
    } else if (polinomio[0] == 0.0) {
        printf("\na0 deve ser diferente de zero!\n");
        getchar();
        imprimeAviso();

        return;
    }
    int *variacoesEPermanencias = calculaVariacoesEPermanencias(polinomio, grauPolinomio);
    if (!variacoesEPermanencias) {
        printf("\nMemoria insuficiente!\n");
        imprimeAviso();
        return;
    }

    int var = variacoesEPermanencias[0];
    int per = variacoesEPermanencias[1];
    double limiteSuperior;

    limiteSuperior = aplicaLagrange(polinomio, grauPolinomio, var, per);

    metodoDeNewton(grauPolinomio, polinomio, limiteSuperior);

    free(variacoesEPermanencias);
    free(polinomio);
} /* =============== Fim resolveEquacaoAlgebrica =============== */

void imprimeAviso() {
    printf("\nAperte ENTER para voltar para o menu\n");
    getchar(); // Limpa o buffer
}

int menu() {
    /* O menu é um switch que chama funcoes correspondentes
       as opcoes inseridas pelo usuario */
    char op = getchar();

    switch(op) {
        case 'C':
        case 'c':
            imprimeConversao();
            break;
        case 'S':
        case 's':
            resolveSL();
            break;
        case 'E':
        case 'e':
            resolveEquacaoAlgebrica();
            break;
        case 'F':
        case 'f':
            return 0;
        default:
            getchar();
            printf("\n\n Opcao invalida!\n");
            imprimeAviso();
    }
    return 1;
} /* =============== Fim Menu =============== */

void imprimeMenu() {
    //Exibe as opcoes do menu

    printf("\n\n========================= Menu =========================\n\n");
    printf("\tC --> Conversao\n");
    printf("\tS --> Sistema Linear\n");
    printf("\tE --> Equacao Algebrica\n");
    printf("\tF --> Finalizar\n\n");
    printf("Escolha uma das opcoes acima digitando a letra correspondente: ");
    printf("\n\n=======================================================\n\n");
} /* =============== Fim imprimeMenu =============== */

int main() {
    /* Um loop que mantem o programa em execucao
       ate o momento em que o usuario o encerra */
    do
        imprimeMenu();
    while (menu());
    return 0;
}
```