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

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

/**
 * Struct representando a Maquina da calculadora.
 */
typedef struct {
    int nSignificativos;
    int expoenteMinimo;
    int expoenteMaximo;
    int qtdOperacoes;
} Maquina;

/**
 * Struct representando um Numero qualquer.
 */
typedef struct {
    int sinal;
    int expoente;
    int mantissa[15];
} Numero;

/**
 * ---- Constantes gerais. ----
 */

const int BASE_PADRAO = 10; /* Base padrao da maquina */
const int MIN_DIGITOS = 2; /* Limite inferior dos digitos significativos */
const int MAX_DIGITOS = 7; /* Limite superior dos digitos significativos */
const int MIN_EXPOENTE_MINIMO = -9; /* Limite inferior do expoente minimo */
const int MAX_EXPOENTE_MINIMO = -4; /* Limite superior do expoente minimo */
const int MIN_EXPOENTE_MAXIMO = 4; /* Limite inferior do expoente maximo */
const int MAX_EXPOENTE_MAXIMO = 9; /* Limite superior do expoente maximo */
const char SINAL_POSITIVO =  '+';
const char SINAL_NEGATIVO =  '-';

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

void somar(Numero* n1, Numero* n2, Numero* resultado, int n);
void subtrair(Numero* n1, Numero* n2, Numero* resultado, int n);
void imprimir(Numero numero, int nsig);
void normalizar(Numero* num, int n);
void efetuarOperacoes(char operador, char sinalN2, Numero* n1, Numero* n2,
  Numero* resultado, int n);
void igualarExpoentes(Numero* n1, Numero* n2, Numero* resultado, int n);
void distribuir(Numero* num, int p);
void arredondar(Numero* num, int nsig, int n);
void zerar(Numero* numero, int n);
void padding(int p, int* m, int n);
int validarParametrosMaquina(Maquina maquina);
int validar(Numero num, Maquina maq);
int validarNumeros(Numero n1, Numero n2, Maquina maq);
int comparar(int* n1, int* n2, int n);
int limpar_stdin();

/**
 * ---- 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;
    int n = 15, expoente;
    // Arquivo base contendo os dados de leitura
    FILE *file;
    // Variaveis auxiliares de leitura.
    char nomeArquivo[30];
    char c, s1, s2, operador;
    // Maquina que sera lida.
    Maquina m;
    // Os numeros em si
    Numero n1, n2, resultado;

    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");

            // Lendo os parametros iniciais da maquina.
            fscanf(file, "%d;%d;%d;%d", &m.nSignificativos,
                   &m.expoenteMinimo,
                   &m.expoenteMaximo,
                   &m.qtdOperacoes);

            // Validando os paramentros lidos.
            if (validarParametrosMaquina(m)) {

                printf("F(%d,%d,%d,%d)\n", BASE_PADRAO,
                       m.nSignificativos,
                       m.expoenteMinimo,
                       m.expoenteMaximo);

                printf("numero de oper.: %d\n", m.qtdOperacoes);

                int nLinhas = m.qtdOperacoes;

                while (nLinhas-- > 0) {

                    zerar(&n1, n);
```

```
                zerar(&n2, n);
                zerar(&resultado, n);

                // Le cada linha do arquivo preenchendo os structs que representam os
                numeros.
                fscanf(file, "
                %c%1d.%1d%1d%1d%1d%1d%1dE%d;%c%1d.%1d%1d%1d%1d%1d%1dE%d;%c",
                       &s1, &n1.mantissa[0], &n1.mantissa[1], &n1.mantissa[2], &n1.
                       mantissa[3], &n1.mantissa[4], &n1.mantissa[5], &n1.mantissa[6], &
                       n1.expoente,
                       &s2, &n2.mantissa[0], &n2.mantissa[1], &n2.mantissa[2], &n2.
                       mantissa[3], &n2.mantissa[4], &n2.mantissa[5], &n2.mantissa[6], &
                       n2.expoente,
                       &operador);

                // Especifico o sinal de acordo com o char lido.
                n1.sinal = s1 == SINAL_POSITIVO ? 1 : -1;
                n2.sinal = s2 == SINAL_POSITIVO ? 1 : -1;

                // Arredondando e exibindo o primeiro numero.
                arredondar(&n1, m.nSignificativos, n);
                imprimir(n1, m.nSignificativos);
                printf(";");

                // Arredondando e exibindo o segundo numero.
                arredondar(&n2, m.nSignificativos, n);
                imprimir(n2, m.nSignificativos);
                printf(";");

                printf("%c", operador);
                printf("; resultado: ");

                //
                int valido = validarNumeros(n1, n2, m);

                if (valido) {

                    // Sigo o fluxo descrito em sala.
                    igualarExpoentes(&n1, &n2, &resultado, n);
                    efetuarOperacoes(operador, s2, &n1, &n2, &resultado, n);
                    normalizar(&resultado, n);
                    arredondar(&resultado, m.nSignificativos, n);

                    int invalido = validar(resultado, m);

                    // Caso o resultado nao pertenca a maquina.
                    if (invalido != 0) {

                        if (invalido > 0) {
                            printf("overflow");
                        } else {
                            printf("underflow");
                        }

                    } else {

                        // Tudo OK. Pode imprimir o resultado
                        imprimir(resultado, m.nSignificativos);
                    }
                }

                printf("\n");
            }

            // All done. Celebrate, drink coffee.
            printf("Fim do processamento.\n");

        } else {
            printf("Parametros incorretos. Corrija os erros listados acima.\n");
        }
    }
```

```
        fclose(file);
    }

    return 0;
}

/**
 * ---- Implementacoes das funcoes auxiliares. ----
 */

/**
 * Valida os parametros de entrada da maquina lida no arquivo.
 *
 * @param Maquina: A maquina lida no arquivo.
 */
int validarParametrosMaquina(Maquina maquina) {

    int valida = 1;

    if (maquina.nSignificativos < MIN_DIGITOS
            || maquina.nSignificativos > MAX_DIGITOS) {

        printf("O num. digitos significativos deve obedecer %d <= N <= %d. Valor atual: %d\n",
               MIN_DIGITOS,
               MAX_DIGITOS,
               maquina.nSignificativos);
        valida = 0;
    }

    if (maquina.expoenteMinimo < MIN_EXPOENTE_MINIMO
            || maquina.expoenteMinimo > MAX_EXPOENTE_MINIMO) {

        printf("O expoente minimo deve obedecer %d <= N <= %d. Valor atual: %d\n",
               MIN_EXPOENTE_MINIMO,
               MAX_EXPOENTE_MINIMO,
               maquina.expoenteMinimo);
        valida = 0;
    }

    if (maquina.expoenteMaximo < MIN_EXPOENTE_MAXIMO
            || maquina.expoenteMaximo > MAX_EXPOENTE_MAXIMO) {

        printf("O expoente maximo deve obedecer %d <= N <= %d. Valor atual: %d\n",
               MIN_EXPOENTE_MAXIMO,
               MAX_EXPOENTE_MAXIMO,
               maquina.expoenteMaximo);
        valida = 0;
    }

    return valida;
}

/**
 * Valida se um determinado numero esta contigo no intervalo da maquina.
 *
 * @param Maquina: A maquina lida no arquivo.
 * @param Numero: O numero que sera validado.
 * @return -1 para UNDERFLOW, 1 para OVERFLOW, 0 para numero valido.
 */
int validar(Numero num, Maquina maq) {

    int invalido = 0;

    if (num.expoente < maq.expoenteMinimo) {
        invalido = -1;
    } else if (num.expoente > maq.expoenteMaximo) {
        invalido = 1;
    }

    return invalido;
```

```
}

/**
 * Valida se os numeros lidos sao validos de acordo com a Maquina.
 *
 * @param n1: O primeiro numero lido.
 * @param n2: O segundo numero lido.
 * @param maq: A maquina lida no arquivo.
 * @return 1 para valido e 0 para invalido.
 */
int validarNumeros(Numero n1, Numero n2, Maquina maq) {

    int invalido1 = validar(n1, maq);

    if (invalido1 != 0) {
        printf("parcela 1 invalida, ");
        if (invalido1 > 0) {
            printf("overflow");
        } else {
            printf("underflow");
        }
    }

    int invalido2 = validar(n2, maq);

    if (invalido2 != 0) {
        printf("parcela 2 invalida, ");
        if (invalido2 > 0) {
            printf("overflow.");
        } else {
            printf("underflow.");
        }
    }

    return !invalido1 && !invalido2;
}

/**
 * Atribui ZERO para todos os numeros da mantissa, e demais atributos do numero.
 *
 * @param numero: O Struct que representa o nummero.
 * @param n: O tamanho do array.
 */
void zerar(Numero* numero, int n) {

    numero->expoente = 0;
    numero->sinal = 0;

    int i;

    for (i = 0; i < n; i++) {
        numero->mantissa[i] = 0;
    }
}

/**
 * Executa um padding a esquerda dos numeros da mantissa,
 * transportando-os para a direita 'p' casas.
 *
 * @param p: A quantidade de casas que os numeros devem andar para a direita.
 * @param m: O array de int que representa a mentissa
 * @param n: O tamanho do array.
 */
void padding(int p, int* m, int n) {

    int i;

    for (i = n - 1; i >= 0; i--) {
        if (i + p <= n - 1) {
            m[i + p] = m[i];
        }
```

```
        m[i] = 0;
    }
}

/**
 * Informa se as mantissas de nois numeros de mesma potencia sao iguais, menor ou maior
 * um que o outro.
 *
 * @param n1: a mantissa do primeiro numero.
 * @param n2: a mantissa do segundo numero.
 * @param n: a quantidade de itens das mantissas.
 * @return 0 para iguais, 1 para n1 > n2, -1 para n1 < n2.
 */
int comparar(int* n1, int* n2, int n) {

    int i;

    for (i = 0; i < n; i++) {
        if (n1[i] > n2[i]) return 1;
        if (n1[i] < n2[i]) return -1;
    }

    return 0;
}

/**
 * Soma dois numeros de acordo com suas bases e os arrays de int que representam
 * o numero.
 * @param n1: Struct do primeiro numero lido.
 * @param n2: Struct do segundo numero lido.
 * @param resultado: Struct do resultado.
 * @param n: O numero de itens que contem em cada array de int.
 */
void somar(Numero* n1, Numero* n2, Numero* resultado, int n) {

    int i, soma;

    for (i = n - 1; i >= 0; i--) {

        soma = n1->mantissa[i] + n2->mantissa[i];

        if (soma > 9 && (i - 1) >= 0) {
            soma = soma % 10;
            n1->mantissa[i - 1]++;
        }

        resultado->mantissa[i] = soma;

        if (i == 0 && soma > 9) {
            padding(1, resultado->mantissa, n);
            resultado->mantissa[i + 1] = soma % 10;
            resultado->mantissa[i]++;
            resultado->expoente++;
        }
    }
}

/**
 * Subtrai dois numeros de acordo com suas bases e os arrays de int que representam
 * o numero.
 * @param n1: Struct do primeiro numero lido.
 * @param n2: Struct do segundo numero lido.
 * @param resultado: Struct do resultado.
 * @param n: O numero de itens que contem em cada array de int.
 */
void subtrair(Numero* n1, Numero* n2, Numero* resultado, int n) {

    int i, sub;

    for (i = n - 1; i >= 0; i--) {
```

```
        if (n1->mantissa[i] < n2->mantissa[i]) {
            n1->mantissa[i] += 10;
            n2->mantissa[i - 1]++;
        }

        resultado->mantissa[i] = n1->mantissa[i] - n2->mantissa[i];
    }
}

/**
 * Exibe um numero no formato de notacao cientifica.
 *
 * @param numero: Struct do numero lido.
 * @param nsig: A quantidade de numeros significativos que deve ser exibida.
 */
void imprimir(Numero numero, int nsig) {

    int i;

    if (numero.sinal == 1) {
        printf("%c", SINAL_POSITIVO);
    } else if (numero.sinal == -1) {
        printf("%c", SINAL_NEGATIVO);
    } else {
        // quando o numero for zero.
    }

    for (i = 0; i < nsig; i++) {
        printf("%d", numero.mantissa[i]);
        if (i == 0) {
            printf(".");
        }
    }

    printf("E%d", numero.expoente);
}

/**
 * Varre um numero e reordena sua mantissa para que o primeiro numero diferente de
 * zero seja o primero item da mantissa. Alem disso, atualiza o expoente de acordo com a nova
 * posicao do primeiro algarismo.
 *
 * @param num: O numero que sera reordenado.
 * @param n: A quantidade de algarismos da mantissa.
 */
void normalizar(Numero* num, int n) {

    // primeiro devo trazer o primero numero diferente de zero para a primeira posicao do
    array.
    // para isso, procuro onde o primeiro elemento positivo aparece.
    int i, j;
    int encontrou = 0;

    for (i = 0; i < n; i++) {
        if (num->mantissa[i] != 0) {
            encontrou = 1;
            break;
        }
    }

    // apos encontra-lo, devo realocar todos os numerais para as posicoes iniciais, de acordo
    // a partir da posicao inicial que encontrei anteriormente.
    if (encontrou && i > 0) {

        for (j = i; j < n; j++) {
            num->mantissa[j - i] = num->mantissa[j];
            num->mantissa[j] = 0;
        }

        num->expoente += i * -1;
    }
```

```
}

/**
 * Efetuas as operacoes de ADICAO ou SUBTRACAO a partir de dois numeros informados.
 *
 * @param operador: O sinal da operacao que sera efetuada (+ ou -).
 * @param sinalN2: O sinal do segundo numero.
 * @param n1: O primeiro numero lido.
 * @param n2: O segundo numero lido.
 * @param resultado: O numero que recebera o resultado.
 * @param n: A quantidade de algarismos da mantissa dos numeros.
 */
void efetuarOperacoes(char operador, char sinalN2, Numero* n1, Numero* n2, Numero* resultado,
 int n) {

    // Se o operador eh igual ao sinal do segundo numero,
    // mudo o sinal do numero para positivo.
    // + com + = +
    // - com - = +
    n2->sinal = sinalN2 == operador ? 1 : -1;

    // Se os sinais dos dois numeros forem iguais,
    // aplico a soma e ja atribuo o sinal ao resultado, que eh igual ao de qualquer um dos
    numeros.
    if (n1->sinal == n2->sinal) {

        resultado->sinal = n1->sinal;
        somar(n1, n2, resultado, n);

    } else {
        // Como os sinais sao diferentes, devo encontrar qual eh o maior numero,
        // para deixa-lo como o minuendo.
        int comp = comparar(n1->mantissa, n2->mantissa, n);

        if (comp > 0) { // n1 > n2

            subtrair(n1, n2, resultado, n);
            resultado->sinal = n1->sinal; // O sinal do resultado eh o mesmo do primeiro
            numero.

        } else if (comp < 0) { // n1 < n2

            subtrair(n2, n1, resultado, n);
            resultado->sinal = n2->sinal; // O sinal do resultado eh o mesmo do segundo
            numero.

        } else { // os numeros sao iguais.
            // como a mantissa ja esta zerada por padrao, preciso apenas setar o sinal e
            expoente.
            resultado->sinal = 0;
            resultado->expoente = 0;
        }
    }
}

/**
 * Iguala o expoente de dois numeros. Alem disso, utiliza a regra de padding left para
 * adicionar zeros a esquerda de acordo com a quantidade de casas da diferenca entre
 expoentes.
 *
 * @param n1: O primeiro numero lido.
 * @param n2: O segundo numero lido.
 * @param resultado: O numero que representa o resultado.
 * @param n: O numero de itens das mantissas dos numeros.
 */
void igualarExpoentes(Numero* n1, Numero* n2, Numero* resultado, int n) {

    // Comparo os dois numeros e ponho os dois no mesmo expoente,
    // usando uma regra de padding no array de int, como descrito no livro-texto.
    resultado->expoente = n1->expoente;
```

```
    if (n1->expoente != n2->expoente) {

        if (n1->expoente < n2->expoente) {

            padding(abs(n1->expoente - n2->expoente), n1->mantissa, n);
            resultado->expoente = n1->expoente = n2->expoente;
        } else {

            padding(abs(n2->expoente - n1->expoente), n2->mantissa, n);
            resultado->expoente = n2->expoente = n1->expoente;
        }
    }
}

/**
 * Funcao recursiva que distribui para os algarismos anteriors a uma posicao p,
 * o resto do modulo de 10 de um numero nesta posicao.
 *
 * @param num: O numero que possui algarismos que precisam ser distribuidos.
 * @param p: A posicao do numero que precisa ser distribuido.
 */
void distribuir(Numero* num, int p) {

    int i = p - 1;

    if (i > 0 && num->mantissa[i] > 9) {

        num->mantissa[i] = num->mantissa[i] % 10;
        num->mantissa[i - 1]++;
        distribuir(num, i);
    }
}

/**
 * Arredonda um determinado numero de acordo com os numeros
 * significativos da maquina.
 *
 * @param num: O numero que sera arredondado.
 * @param nsig: O numero de algarismos significativos.
 * @param n: O numero de itens da mantissa do numero.
 */
void arredondar(Numero* num, int nsig, int n) {

    int i;

    if (nsig > 0) {

        if (num->mantissa[nsig] > 5) {

            num->mantissa[nsig - 1]++;
            distribuir(num, nsig);

        } else if (num->mantissa[nsig] == 5 && nsig + 1 < n) {

            for (i = nsig + 1; i < n; i++) {

                if (num->mantissa[i] != 0) {

                    num->mantissa[nsig - 1]++;
                    distribuir(num, nsig);
                    break;
                }
            }
        }
    }

    if (num->mantissa[0] > 9) {

        padding(1, num->mantissa, n);
        num->mantissa[1] = num->mantissa[1] % 10;
        num->mantissa[0]++;
```

```
        num->expoente++;
    }

    for (i = nsig; i < n; i++) {
        num->mantissa[i] = 0;
    }
}

/**
 * Limpa os resquicios de memoria do stdin.
 */
int limpar_stdin() {

    while (getchar() != '\n');

    return 1;
}
```