

”

**UNIDADE CURRICULAR:** Introdução à Programação

**CÓDIGO:** 21173

**DOCENTE:** José Coelho

**A preencher pelo estudante**

**NOME:** José Augusto Oliveira Azevedo

**N.º DE ESTUDANTE:** 2200655

**CURSO:** Licenciatura em Engenharia Informática

**DATA DE ENTREGA:** 5 de Janeiro de 2023

## **TRABALHO / RESOLUÇÃO:**

### **Descrição e explicação do programa 1:**

Na implementação do programa foram utilizadas variáveis alocadas na heap, também foi definido um Buffer de 100 para o comprimento das strings.

Analizando o problema decidi que necessitava de quatro variáveis: nome\_estrela (para guardar no nome da estrela), contadorastros (para contar o número de planetas inseridos), nome\_astro (para receber o nome do planeta), numerocaracteres (guarda o número de caracteres dos nomes, inicialmente da estrela e depois do nome do planeta inserido – controla o ciclo enquanto a informação lida não é nula).

Inicialmente aloca-se memória para as variáveis anteriormente referidas

Verifica-se se a alocação foi bem sucedida e no caso de falha é apresentada uma mensagem de erro.

Efetua-se a leitura do nome da estrela, coloca-se a variável contador de astros a zero e no caso de o nome não estar vazio (numero de caracteres maior do que zero) entra num ciclo.

O programa só sai do ciclo quando o número de caracteres do planeta for igual a zero.

Dentro do ciclo, inicialmente efetua-se a leitura do nome do astro (planeta) e no caso do número de caracteres ser diferente de zero incrementa o contador de astros em uma unidade e avança para a próxima leitura.

Quando o utilizador introduz um nome vazio o programa sai do ciclo, imprime a mensagem de saída e liberta a memória das variáveis alocadas.

Foram efetuados testes no VPL e testes locais com diferentes inputs e verificados os respetivos Outputs.

### **Descrição e explicação do programa 2:**

Na implementação da pergunta dois foi usado o código da pergunta 1. Foram criadas mais quatro variáveis dinâmicas: dados (variável usada para ler a linha como nome do planeta e a distancia), distanciaastro (guarda a distancia inserida), mínimo (guarda o valor mínimo) e máximo (guarda o valor máximo)

O funcionamento é semelhante ao anterior. Dentro do ciclo e se for o primeiro astro iniciamos as variáveis mínimo e máximo com a distancia do astro, depois verifica-se se a distancia é menor que o mínimo e se a distancia é maior que o máximo e atualiza-se as variáveis respetivas.

Foram efetuados testes no VPL e testes locais com diferentes inputs e verificados os respetivos Outputs.

### **Descrição e explicação do programa 3:**

Na implementação da pergunta três foi usada a estrutura do código da pergunta 1, a leitura da estrela e depois a leitura dos planetas e satélites dentro do ciclo.

Por forma a guardar a informação foi definida uma lista encadeada de dados em que o nó inicial é uma estrutura do tipo Estrela que encabeça uma lista de estruturas do tipo Planeta. Cada uma das estruturas do tipo Planeta pode conter ou não uma lista encadeada de estruturas do tipo Satellite.

Foram definidos os tipos Estrela, Planeta e Satellite com as respetivas estruturas, definidas as funções que criam os Nós desses tipos assim como os processos que os inserem na lista encadeada.

Foi criado um método que Filtra uma lista com base nos critérios que são passados por parâmetros. Esse método usa uma função chamada insereNo para verificar se um determinado Nó cumpre os critérios ou não.

Foram implementados métodos de ordenação sobre a lista de planetas e sobre a lista de satélites, que usam o algoritmo de quickSort.

Após a leitura do sistema para uma lista e da leitura dos parâmetros de filtragem a lista é filtrada usando os valores inseridos, depois é ordenada e o output é impresso.

Na rotina de impressão (printListPlanetas) foi necessário identificar se o valor da distância era um valor inteiro ou assim como o número de casas decimais para que a formatação do número fosse de encontro ao pretendido.

Foram efetuados testes no VPL e testes locais com diferentes inputs e verificados os respetivos Outputs.

#### **Descrição e explicação do programa 4:**

Na implementação da pergunta 4 foi usada a implementação da pergunta 3.

Foram removidas as funções de ordenação e de filtragem das listas.

Foi implementada uma função que retorna o NoPlaneta onde se encontrou determinado texto, quer o texto faça parte de nome do Planeta ou do nome de um dos Satelites dele.

Com esta função o programa obtém o NoPlaneta de origem e o NoPlaneta de destino da viagem.

Foi criado um procedimento que efetua o calcula da duração da viagem (entre o NoInicial e o NoFinal) e gera o output.

Nos cálculos é usada a função arrNumero que efetua o arredondamento da duração da viagem para o inteiro acima do valor actual.

Foram efetuados testes no VPL e testes locais com diferentes inputs e verificados os respetivos Outputs.

## ANEXOS

### Testes Programa 1

Testes com valores inseridos manualmente que comprovam que o programa faz o output pretendido.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta1
Sol
Terra
Marte
Jupiter
Urano

Sol, sistema planetario com 4 planetas.

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta1
Sol
Terra
Venus

Sol, sistema planetario com 2 planetas.
```

### Testes Programa 2

Testes com valores inseridos manualmente que comprovam que o programa faz o output pretendido.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta2
Sol
Terra 1
Mercurio 0.05
Ganimeses 0.72
Jupiter 5.2
Dione 12

Sol, sistema planetário com 5 planetas a distâncias entre 0.05 e 12.00 UA.

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta2
Sol
Terra 1
Mercurio 0.39
Venus 0.72
Jupiter 5.2
Saturno 9.58
Urano 21

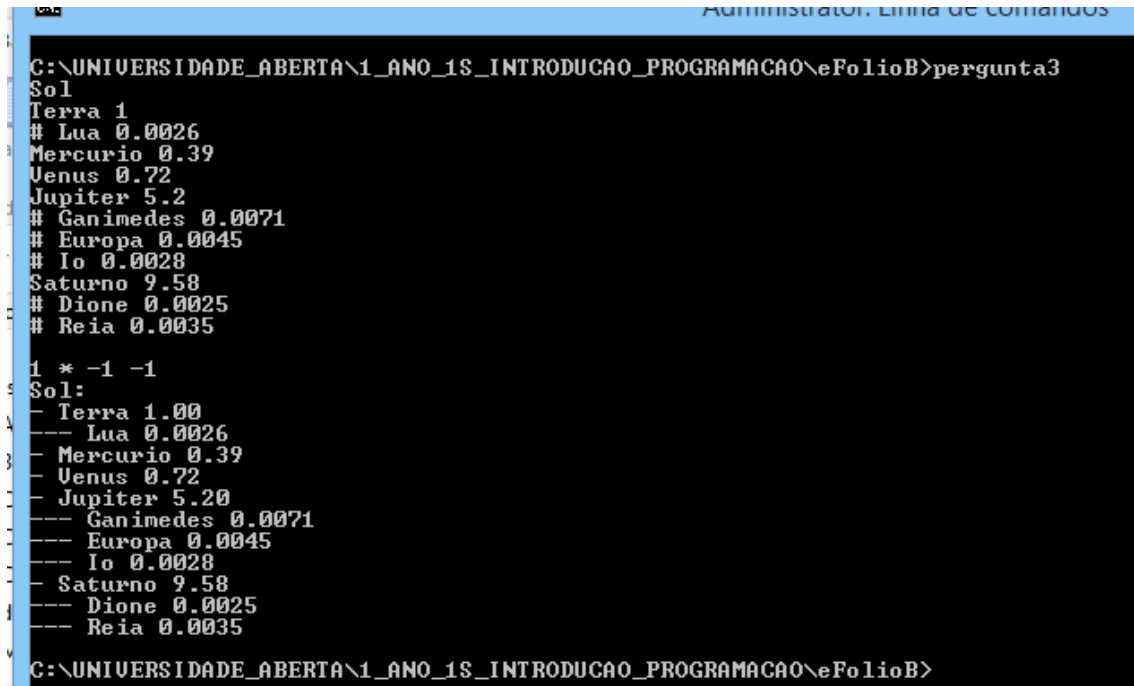
Sol, sistema planetário com 6 planetas a distâncias entre 0.39 e 21.00 UA.

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>
```

### Testes Programa 3

Testes com valores inseridos manualmente que comprovam que o programa faz o output pretendido.

Neste teste o output mantém a ordem de entrada e não efetua qualquer filtro.

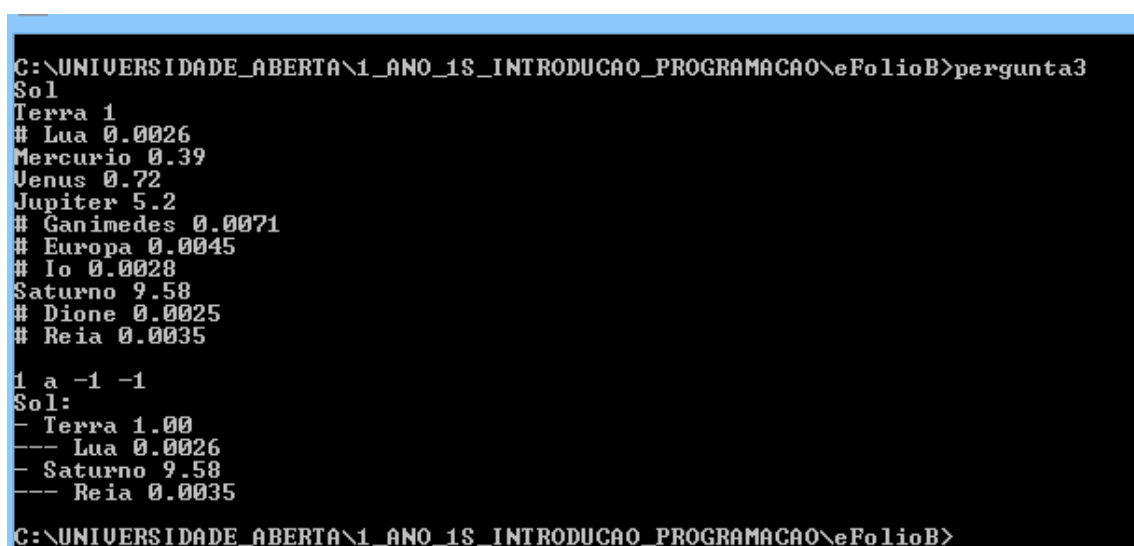


```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta3
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

1 * -1 -1
Sol:
- Terra 1.00
--- Lua 0.0026
- Mercurio 0.39
- Venus 0.72
- Jupiter 5.20
--- Ganimedes 0.0071
--- Europa 0.0045
--- Io 0.0028
- Saturno 9.58
--- Dione 0.0025
--- Reia 0.0035

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>
```

Neste, o output mantém a ordem de entrada e mostra planetas com o texto “a” no nome e satélites deste que também têm o texto “a” no nome



```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta3
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

1 a -1 -1
Sol:
- Terra 1.00
--- Lua 0.0026
- Saturno 9.58
--- Reia 0.0035

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>
```

Neste o output está ordenado alfabeticamente e não efetua qualquer filtro.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta3
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

2 * -1 -1
Sol:
- Jupiter 5.20
--- Europa 0.0045
--- Ganimedes 0.0071
--- Io 0.0028
- Mercurio 0.39
- Saturno 9.58
--- Dione 0.0025
--- Reia 0.0035
- Terra 1.00
--- Lua 0.0026
- Venus 0.72

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>
```

No seguinte, o output está ordenado pela distância e não efetua qualquer filtro.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>pergunta3
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

3 * -1 -1
Sol:
- Mercurio 0.39
- Venus 0.72
- Terra 1.00
--- Lua 0.0026
- Jupiter 5.20
--- Io 0.0028
--- Europa 0.0045
--- Ganimedes 0.0071
- Saturno 9.58
--- Dione 0.0025
--- Reia 0.0035

C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\eFolioB>
```

O output está ordenado ordem de entrada e filtra pela distância mínima, mostra elementos com a distancia superior a 0.5.

```
Administrator: Linha de comandos
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\FolioB>pergunta3
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035
1 * 0.5 -1
Sol:
- Terra 1.00
- Venus 0.72
- Jupiter 5.20
- Saturno 9.58
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\FolioB>
```

## Testes Programa 4

Testes com valores inseridos manualmente que comprovam que o programa faz o output pretendido.

Viagem: Lua para a Terra ( $0.002 \times 30 = 0.078$  – arredonda para 1 dia); Terra para o Sol ( $1 \times 30 = 30$  dias); Sol para Saturno ( $9.58 \times 30 = 287.4$  – arredonda para 288); Saturno para Dione ( $0.0025 \times 30 = 0.75$  – arredonda para 1 dia). Total = 320 dias + 2 dias, que correspondem a 10 meses e 22 dias.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\FolioB>pergunta4
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

Lua Dione
De: Sol-Terra-Lua
Para: Sol-Saturno-Dione
Esta |a| ães: Lua <1> Terra <30> Sol <288> Saturno <1> Dione
Dura |a| úo: 10 meses 22 dias
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\FolioB>pergunta4
```



Viagem: Venus para o Sol ( $0.72 \times 30 = 21.6$  – arredonda para 22 dias); Sol para Saturno ( $9.58 \times 30 = 287.4$  dias – arredonda para 288 dias); Saturno para Reia ( $0.0035 \times 30 = 0.105$  – arredonda para 1 dia). Total = 311 dias + 2 dias, que correspondem a 10 meses e 13 dias.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\efolioB>pergunta4
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

Venus Reia
De: Sol-Venus
Para: Sol-Saturno-Reia
Esta | e | ães: Venus <22> Sol <288> Saturno <1> Reia
Dura | e | ão: 10 meses 13 dias
```

Viagem: Ganimedes para Jupiter ( $0.0071 \times 30 = 0.213$  – arredonda para 1 dia); Jupiter para Io ( $0.0028 \times 30 = 0.084$  – arredonda para 1 dia). Total = 2 dias + 2 dias, que correspondem a 4 dias. Está correto porque os satélites pertencem ao mesmo planeta.

```
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\efolioB>pergunta4
Sol
Terra 1
# Lua 0.0026
Mercurio 0.39
Venus 0.72
Jupiter 5.2
# Ganimedes 0.0071
# Europa 0.0045
# Io 0.0028
Saturno 9.58
# Dione 0.0025
# Reia 0.0035

Ganimedes Io
De: Sol-Jupiter-Ganimedes
Para: Sol-Jupiter-Io
Esta | e | ães: Ganimedes <1> Jupiter <1> Io
Dura | e | ão: 4 dias
C:\UNIVERSIDADE_ABERTA\1_ANO_1S_INTRODUCAO_PROGRAMACAO\efolioB>
```

## Código Programa 1

```
/*
 * Programa: Pergunta 1 - IP - Efolio B
 * Descrição:
 *   Após receber informação sobre o sistema planetário na entrada de
 *   dados standard (stdin), apresenta a mensagem:
 *
 *       <nome da estrela>, sistema planetário com <N> planetas.
 *
 *   Dados inseridos:
 *       O nome da estrela seguido dos nomes dos planetas, um nome por
 *       linha.
 *       Os nomes não contêm espaços.
 *
 *       Após a introdução de uma linha vazia, é apresentada a mensagem.
 */

/* BIBLIOTECAS */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
/* MACROS */
#define BUFFER 100
/* PROGRAMA PRINCIPAL */
int main()
{
    /* Alocar memoria para guardar o nome da estrela */
    char *nome_estrela;
    nome_estrela = malloc(sizeof(BUFFER));
    /* Alocar memoria para guardar o nome do astro (planeta) */
    char *nome_astro;
    nome_astro = malloc(sizeof(BUFFER));
    /* Alocar memória para contador de astros*/
    int *contadorastros;
    contadorastros = malloc(sizeof(*contadorastros));
    /* Alocar memória para guardar número de caracteres dos nomes
    inseridos*/
    int *numerocaracteres;
    numerocaracteres = malloc(sizeof(*numerocaracteres));

    /* Verificar se conseguiu alocar memoria para as variaveis */
    if ((nome_estrela != NULL) && (nome_astro != NULL) && (contadorastros
    != NULL) && (numerocaracteres != NULL))
    {
        /* ler o nome da estrela*/
    }
}
```

```

fgets(nome_estrela, BUFFER, stdin);
nome_estrela[strcspn(nome_estrela, "\n")] = 0;
(*contadorastros) = 0;
*numerocaracteres = strlen(nome_estrela);
/* ciclo enquanto tivermos planetas, lemos nomes e contamos */
while (*numerocaracteres != 0)
{
    /* ler o nome do planeta */
    fgets(nome_astro, BUFFER, stdin);
    nome_astro[strcspn(nome_astro, "\n")] = 0;

    *numerocaracteres = strlen(nome_astro);
    if (*numerocaracteres != 0)
    {
        /* se o nome do planeta não for vazio aumentos o contador
*/
        (*contadorastros)++;
    }
}
/* mensagem de saída */
printf("%s, sistema planetário com %d planetas.\n", nome_estrela,
*contadorastros);
/* libertar a memória alocada */
free(nome_estrela);
free(nome_astro);
free(contadorastros);
free(numerocaracteres);
}
else
{
    printf("Erro a alocar memória para executar o programa.");
}
return 0;
}

```

## Código Programa 2

```
/*
 * Programa: Pergunta 2- IP - Efolio B
 * Descrição:
 * Após receber informação sobre o sistema planetário na entrada de
 dados standard (stdin), apresenta a mensagem:
 *
 * <nome da estrela>, sistema planetário com <N> planetas a
 distâncias entre <mínimo> e <máximo> UA.
 *
 * Deve mostrar a distância com precisão a 2 casas decimais.
 *
 * Dados inseridos:
 * O nome da estrela seguido dos nomes dos planetas, um nome por
 linha.
 * Os nomes não contêm espaços.
 * A frente do nome de cada planeta é inserida a distância do
 planeta à estrela, em unidades astronómicas (UA).
 * Após a introdução de uma linha vazia, é apresentada a mensagem.
 */

/* BIBLIOTECAS */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
/* MACROS */
#define BUFFER 100

int main()
{
    /* Alocar memoria para guardar o nome da estrela */
    char *nome_estrela;
    nome_estrela = malloc(sizeof(BUFFER));
    /* Alocar memoria para guardar os dados inseridos */
    char *dados;
    dados = malloc(sizeof(BUFFER));
    /* Alocar memoria para guardar o nome do astro (planeta) */
    char *nome_astro;
    nome_astro = malloc(sizeof(BUFFER));
    /* Alocar memória para guardar a distancia do astro*/
    float *distanciaastro;
    distanciaastro = malloc(sizeof(*distanciaastro));
    /* Alocar memória para contador de astros*/
    int *contadorastros;
    contadorastros = malloc(sizeof(*contadorastros));
    /* Alocar memória para guardar número de caracteres dos dados
 inseridos*/
}
```

```

int *numerocaracteres;
numerocaracteres = malloc(sizeof(*numerocaracteres));
/* Alocar memória para guardar a distancia mínima */
float *minimo;
minimo = malloc(sizeof(*minimo));
/* Alocar memória para guardar a distancia máxima */
float *maximo;
maximo = malloc(sizeof(*maximo));

/* Verificar se conseguiu alocar memoria para as variaveis */
if ((nome_estrela != NULL) && (dados != NULL) && (nome_astro != NULL)
&& (distanciaastro != NULL) && (contadorastros != NULL) &&
(numerocaracteres != NULL) && (maximo != NULL) && (minimo != NULL))
{
    /* ler o nome da estrela*/
    fgets(nome_estrela, BUFFER, stdin);
    nome_estrela[strcspn(nome_estrela, "\n")] = 0;
    (*numerocaracteres) = strlen(nome_estrela);

    /* iniciamos as variaveis */
    (*contadorastros) = 0;
    (*minimo) = 0;
    (*maximo) = 0;

    /* ciclo enquanto tivermos planetas, lemos informação e guardamos
o minimo e o maximo */
    while (*numerocaracteres != 1)
    {
        /* ler dados: nome do planeta e distancia */
        fgets(dados, BUFFER, stdin);
        *numerocaracteres = strlen(dados);
        if (*numerocaracteres != 1)
        {
            dados[strcspn(dados, "\n")] = 0;

            /* separamos a informação pelo espaço */
            char *pt = (char *)strtok(dados, " ");
            strcpy(nome_astro, pt);

            pt = (char *)strtok(NULL, " ");
            (*distanciaastro) = atof(pt);

            if (*numerocaracteres != 0)
            {
                /* dados inseridos não são vazios, aumentos o
contador */
                (*contadorastros)++;
            }
        }
    }
}

```

```

        if (*contadorastros == 1)
        {
            /* primeiro astro inserido iniciamos as variaveis
minimo e maximo */
            (*minimo) = *distanciaastro;
            (*maximo) = *distanciaastro;
        }
        else
        {
            /* verifica se distancia do astro é menor que o
minimo */
            if (*minimo > (*distanciaastro))
            {
                /* atualiza minimo */
                (*minimo) = *distanciaastro;
            }
            /* verifica se distancia do astro é maior que o
máximo */
            if (*maximo < (*distanciaastro))
            {
                /* atualiza maximo */
                (*maximo) = *distanciaastro;
            }
        }
    }

    /* mensagem de saida */
    printf("%s, sistema planetário com %d planetas a distâncias entre
%.2f e %.2f UA.\n", nome_estrela, *contadorastros, *minimo, *maximo);
    /* libertar a memoria alocada */
    free(nome_estrela);
    free(dados);
    free(nome_astro);
    free(distanciaastro);
    free(contadorastros);
    free(numerocaracteres);
    free(minimo);
    free(maximo);
}
else
{
    printf("Erro a alocar memória para executar o programa.");
}
return 0;
}

```

### Código Programa 3

```
/*
 * Programa: Pergunta 3 - IP - Efolio B
 * Descrição:
 *   Após receber informação sobre o sistema planetário, composto por
 *   estrela, planetas e luas, na entrada de dados standard (stdin), apresenta
 *   os dados ordenados e filtrados.
 *
 *   <estrela>:
 *   - <planeta 1> <UA do planeta 1>
 *   --- <satélite do planeta 1> <UA do satélite>
 *
 *   Dados inseridos:
 *   O nome da estrela seguido dos nomes dos planetas, um nome por
 *   linha.
 *   Os nomes não contêm espaços.
 *   A frente do nome de cada planeta é inserida a distância do
 *   planeta à estrela, em unidades astronómicas (UA).
 *   As linhas que começam com # contem a informação de um satélite do
 *   planeta.
 *   A distância referida no satélite, é relativamente ao planeta que
 *   orbita, tendo 4 casas decimais.
 *   Após a introdução do último planeta/satélite, segue-se uma linha
 *   em branco, seguida da linha com quatro elementos, correspondendo à
 *   solicitação:
 *   <ordenação> <filtro nome> <filtro mínimo> <filtro máximo>
 *
 *   Os filtros não se aplicam à estrela, apenas aos planetas e
 *   satélites.
 *
 *   O elemento <ordenação> é um número inteiro com o seguinte
 *   significado:
 *
 *   1 - manter ordem de entrada de dados (este deve ser o
 *   procedimento base, caso seja fornecido um inteiro sem significado)
 *   2 - ordenar alfabeticamente, por ordem crescente (aplicar
 *   primeiro aos planetas e depois aos satélites dos planetas)
 *   3 - ordenar por UA, por ordem crescente (aplicar primeiro aos
 *   planetas e depois aos satélites dos planetas)
 *
 *   O elemento <filtro nome> é um texto sem espaços . Se for "*" não
 *   causa nenhum impacto na solução,
 *   caso contrário deverá mostrar apenas os astros que contenham o
 *   texto (distingue entre maiúsculas e minúsculas).
 */
```

```

/* BIBLIOTECAS */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
/* MACROS */
#define BUFFER 100

/* Enumeração do tipo de Ordenação */
enum tipo_ordenacao
{
    ORDEM_ENTRADA = 1,
    ORDEM_ALFABETICA = 2,
    ORDEM_UA = 3
};

/* TIPOS */
/* Satellite: Tipo que permite criar uma lista encadeada de satelites */
typedef struct SSatelite
{
    char *nome;
    float distancia;
    struct SSatelite *nextsat;
} NoSatelite;
/* Planeta: Tipo que permite criar uma lista encadeada de planetas que
podem conter ou nao uma lista de satelites */
typedef struct SPlaneta
{
    char *nome;
    float distancia;
    struct SSatelite *nextsat;
    struct SPlaneta *nextpla;
} NoPlaneta;
/* Estrela: Tipo que permite criar uma lista encabeçada em que o nó
inicial é uma estrela e depois contem ou nao uma lista de planetas */
typedef struct SEstrela
{
    char *nome;
    struct SPlaneta *nextpla;
} NoEstrela;

/* CRIA ESTANCIAS DAS ESTRUTURAS*/

// Descrição: Cria instância de uma Estrela
NoEstrela *createNoEstrela(char *nome)
{
    char *aux;

```



```

    aux = malloc(sizeof(nome));
    strcpy(aux, nome);

    NoEstrela *newNode = (NoEstrela *)malloc(sizeof(NoEstrela));
    newNode->nome = aux;
    newNode->nextpla = NULL;
    return newNode;
}

// Descrição: Cria instância de um Planeta
NoPlaneta *createNoPlaneta(char *nome, float distancia)
{
    char *aux;
    aux = malloc(sizeof(nome));
    strcpy(aux, nome);

    NoPlaneta *newNode = (NoPlaneta *)malloc(sizeof(NoPlaneta));
    newNode->nome = aux;
    newNode->distancia = (float)distancia;
    newNode->nextsat = NULL;
    newNode->nextpla = NULL;
    return newNode;
}

// Descrição: Cria instância de um Satelite
NoSatelite *createNoSatelite(char *nome, float distancia)
{
    char *aux;
    aux = malloc(sizeof(nome));
    strcpy(aux, nome);

    NoSatelite *newNode = (NoSatelite *)malloc(sizeof(NoSatelite));
    newNode->nome = aux;
    newNode->distancia = distancia;
    newNode->nextsat = NULL;
    return newNode;
}

/* INSERE INFORMACAO NA LISTA */
// Insere uma Estrela na lista
void insertNoEstrela(NoEstrela **link, NoEstrela *newNode)
{
    // newNode->nextpla = *link;
    *link = newNode;
}

// Insere um Planeta na lista
void insertNoPlaneta(NoPlaneta **link, NoPlaneta *newNode)
{
    newNode->nextpla = *link;
    *link = newNode;
}

```

```

// Insere um Satellite na lista
void insertNoSatelite(NoSatelite **link, NoSatelite *newNode)
{
    newNode->nextsat = *link;
    *link = newNode;
}

/* FUNÇÕES AUXILIARES */
// Função que retorna se o valor de um float é um inteiro ou não
int isInteger(float numero)
{
    return (floor(numero) == numero);
}
// Função que efetua o trim de uma string
char *trimString(char *str)
{
    char *end;

    while (isspace((unsigned char)*str))
        str++;

    if (*str == 0)
        return str;

    end = str + strlen(str) - 1;
    while (end > str && isspace((unsigned char)*end))
        end--;

    end[1] = '\0';

    return str;
}

// Retorna o último Planeta da lista
NoPlaneta *getLastPlaneta(NoPlaneta *cur)
{
    while (cur != NULL &&
           cur->nextpla != NULL)
        cur = cur->nextpla;
    return cur;
}

// Retorna o último Satellite da lista
NoSatelite *getLastSatelite(NoSatelite *cur)
{
    while (cur != NULL &&
           cur->nextsat != NULL)
        cur = cur->nextsat;
    return cur;
}

```

```

// Função que compara dois planetas e retorna se efetua a troca ou nao
// com base no Tipo de Ordenacao
int OrdemPlanetas(int TipoOrdenacao, NoPlaneta *planeta1, NoPlaneta
*planeta2)
{
    if (TipoOrdenacao == ORDEM_ALFABETICA)
    {
        return strcmp(planeta2->nome, planeta1->nome);
    }
    if (TipoOrdenacao == ORDEM_UA)
    {
        if ((planeta1->distancia - planeta2->distancia) > 0)
        {
            return 0;
        }
        else
        {
            return 1;
        }
    }
}

// Função que compara dois Satelites e retorna se efetua a troca ou nao
// com base no Tipo de Ordenacao
float OrdemSatelites(int TipoOrdenacao, NoSatelite *satelite1, NoSatelite
*satelite2)
{
    if (TipoOrdenacao == ORDEM_ALFABETICA)
    {
        return strcmp(satelite2->nome, satelite1->nome);
    }
    if (TipoOrdenacao == ORDEM_UA)
    {
        if ((satelite1->distancia - satelite2->distancia) > 0)
        {
            return 0;
        }
        else
        {
            return 1;
        }
    }
}

/* FUNÇÕES DE ORDENAÇÃO */

// Particiona a lista de Planetas tomando o
// último elemento como pivô
NoPlaneta *partitionPlanetas(NoPlaneta *head,

```

```

        NoPlaneta *end,
        NoPlaneta **newHead,
        NoPlaneta **newEnd,
        int TipoOrdenacao)
{
    NoPlaneta *pivot = end;
    NoPlaneta *prev = NULL,
        *cur = head, *tail = pivot;

    // Durante a partição, tanto o head quanto o
    // final da lista podem ser alterados,
    // atualizando as variáveis newHead e newEnd
    while (cur != pivot)
    {
        // verificar se é para trocar segundo o tipo de ordenacao
        if (OrdemPlanetas(TipoOrdenacao, cur, pivot) > 0)
        {
            // Primeiro nó que possui um valor
            // menor que o pivô - torna-se
            // a nova cabeça (newHead)
            if ((*newHead) == NULL)
                (*newHead) = cur;

            prev = cur;
            cur = cur->nextpla;
        }

        // Se o nó cur é maior que o pivô
        else
        {
            // Move o nó cur para o próximo da cauda,
            // e muda a cauda
            if (prev)
                prev->nextpla = cur->nextpla;
            NoPlaneta *tmp = cur->nextpla;
            cur->nextpla = NULL;
            tail->nextpla = cur;
            tail = cur;
            cur = tmp;
        }
    }

    // Se o pivô for o menor elemento
    // na lista atual, o pivô se torna a cabeça
    if ((*newHead) == NULL)
        (*newHead) = pivot;

    // Atualiza newEnd para o último nó atual
    (*newEnd) = tail;
}

```

```

    // Retorna o nó pivô
    return pivot;
}
// Particiona a lista de Satelites tomando o
// último elemento como pivô
NoSatelite *partitionSatelites(NoSatelite *head,
                                NoSatelite *end,
                                NoSatelite **newHead,
                                NoSatelite **newEnd,
                                int TipoOrdenacao)
{
    NoSatelite *pivot = end;
    NoSatelite *prev = NULL,
                *cur = head, *tail = pivot;

    // Durante a partição, tanto o head quanto o
    // final da lista podem ser alterados,
    // atualizando as variaveis newHead e newEnd
    while (cur != pivot)
    {
        // verificar se é para trocar segundo o tipo de ordenacao
        if (OrdemSatelites(TipoOrdenacao, cur, pivot) > 0)
        {
            // Primeiro nó que possui um valor
            // menor que o pivô - torna-se
            // a nova cabeça (newHead)
            if ((*newHead) == NULL)
                (*newHead) = cur;

            prev = cur;
            cur = cur->nextsat;
        }

        // Se o nó cur é maior que o pivô
        else
        {
            // Move o nó cur para o próximo da cauda,
            // e muda a cauda
            if (prev)
                prev->nextsat = cur->nextsat;
            NoSatelite *tmp = cur->nextsat;
            cur->nextsat = NULL;
            tail->nextsat = cur;
            tail = cur;
            cur = tmp;
        }
    }
}

```

```

    // Se o pivô for o menor elemento
    // na lista atual, o pivô se torna a cabeça
    if ((*newHead) == NULL)
        (*newHead) = pivot;

    // Atualiza newEnd para o último nó atual
    (*newEnd) = tail;

    // Retorna o nó pivô
    return pivot;
}

// Ordenação dos Planetas, excepto o nó final
NoPlaneta *quickSortRecurPlanetas(NoPlaneta *head,
                                   NoPlaneta *end,
                                   int TipoOrdenacao)
{
    // condição base
    if (!head || head == end)
        return head;

    NoPlaneta *newHead = NULL, *newEnd = NULL;

    // Particiona a lista, newHead e newEnd
    // são atualizados pela função de partição
    NoPlaneta *pivot = partitionPlanetas(head, end,
                                           &newHead, &newEnd,
TipoOrdenacao);

    // Se o pivô for o menor elemento - não há necessidade
    // de verificar a parte esquerda.
    if (newHead != pivot)
    {
        // Define o nó antes do nó pivô como NULL
        NoPlaneta *tmp = newHead;
        while (tmp->nextpla != pivot)
            tmp = tmp->nextpla;
        tmp->nextpla = NULL;

        // Recursivo para a lista antes do pivô
        newHead = quickSortRecurPlanetas(newHead, tmp, TipoOrdenacao);

        // Muda próximo do último nó da metade esquerda para usar como
pivô
        tmp = getLastPlaneta(newHead);
        tmp->nextpla = pivot;
    }

    // Recursivo para a lista após o elemento pivô

```

```

        pivot->nextpla = quickSortRecurPlanetas(pivot->nextpla,
                                                newEnd,
                                                TipoOrdenacao);

        return newHead;
    }
    // Ordenação dos Satelites, excepto o nó final
    NoSatelite *quickSortRecurSatelites(NoSatelite *head,
                                        NoSatelite *end,
                                        int TipoOrdenacao)
    {
        // condição base
        if (!head || head == end)
            return head;

        NoSatelite *newHead = NULL, *newEnd = NULL;

        // Particiona a lista, newHead e newEnd
        // são atualizados pela função de partição
        NoSatelite *pivot = partitionSatelites(head, end,
                                                &newHead, &newEnd,
TipoOrdenacao);

        // Se o pivô for o menor elemento - não há necessidade
        // de verificar a parte esquerda.
        if (newHead != pivot)
        {
            // Define o nó antes do nó pivô como NULL
            NoSatelite *tmp = newHead;
            while (tmp->nextsat != pivot)
                tmp = tmp->nextsat;
            tmp->nextsat = NULL;

            // Recursivo para a lista antes do pivô
            newHead = quickSortRecurSatelites(newHead, tmp, TipoOrdenacao);

            // Muda próximo do último nó da metade esquerda para usar como
pivô
            tmp = getLastSatelite(newHead);
            tmp->nextsat = pivot;
        }

        // Recursivo para a lista após o elemento pivô
        pivot->nextsat = quickSortRecurSatelites(pivot->nextsat,
                                                newEnd,
                                                TipoOrdenacao);

        return newHead;
    }

```

```

// função principal para ordenação rápida dos Planetas.
// Este é um wrapper sobre a função recursiva quickSortRecurPlanetas()
void quickSortPlanetas(NoPlaneta **headRef, int TipoOrdenacao)
{
    (*headRef) = quickSortRecurPlanetas(*headRef,
                                         getLastPlaneta(*headRef),
                                         TipoOrdenacao);
    return;
}

// função principal para ordenação rápida dos Satélites.
// Este é um wrapper sobre a função recursiva quickSortRecurSatelites()
void quickSortSatelites(NoSatelite **headRef, int TipoOrdenacao)
{
    (*headRef) = quickSortRecurSatelites(*headRef,
                                         getLastSatelite(*headRef),
                                         TipoOrdenacao);
    return;
}

void printList(NoEstrela *head)
{
    NoPlaneta *pla = NULL;
    NoSatelite *sat = NULL;
    /* Imprime a Estrela */
    printf("%s:\n", head->nome);
    pla = head->nextpla;
    while (pla != NULL)
    {
        if (isInteger(pla->distancia) == 1)
        {
            printf("- %s %.2lf\n", pla->nome, pla->distancia);
        }
        else
        {
            printf("- %s %g\n", pla->nome, pla->distancia);
        }

        sat = pla->nextsat;
        while (sat != NULL)
        {
            printf("--- %s %g\n", sat->nome, sat->distancia);
            sat = sat->nextsat;
        }

        pla = pla->nextpla;
    }
}

```



```

// Função que imprime a saída de dados
void printListPlanetas(NoEstrela *estrela, NoPlaneta *head)
{
    NoSatelite *sat = NULL;
    /* Imprime a Estrela */
    printf("%s:\n", trimString(estrela->nome));

    while (head != NULL)
    {
        if (isInteger(head->distancia) == 1)
        {
            printf("- %s %.2lf\n", head->nome, head->distancia);
        }
        else
        {
            int integer_part = (int)(head->distancia);
            float decimal_part = head->distancia - integer_part;
            if (decimal_part * 100 < 100)
            {
                printf("- %s %.2lf\n", head->nome, head->distancia);
            }
            else
            {
                printf("- %s %.4g\n", head->nome, head->distancia);
            }
        }
        sat = head->nextsat;
        while (sat != NULL)
        {
            printf("--- %s %.4lf\n", sat->nome, sat->distancia);
            sat = sat->nextsat;
        }
        head = head->nextpla;
    }
}

// Funcao que verifica os criterios e retorna se é para usar o Nó da
lista ou não
int insereNo(char *no_nome, float no_distancia, char *texto, float
minimo, float maximo)
{
    int filtratexto = strcmp(texto, "") == 0 ? 0 : 1;
    int filtraminimo = minimo > 0 ? 1 : 0;
    int filtramaximo = maximo > 0 ? 1 : 0;
    int inseretxt = 0;
    int inseremin = 0;
    int inseremax = 0;
    int testou = 0;

```

```

char *found = NULL;
if (filtratexto)
{
    testou = 1;
    found = strstr(no_nome, texto);
    inseretxt = (found != NULL) ? 1 : 0;
}
if (filtraminimo)
{
    testou += 2;
    inseremin = (no_distancia > minimo) ? 2 : 0;
}
if (filtramaximo)
{
    testou += 4;
    inseremax = (no_distancia < maximo) ? 4 : 0;
}
if (testou == (inseretxt + inseremin + inseremax))
{
    return 1;
}
else
{
    return 0;
}
}

// Funcao que cria uma lista nova usando os critérios
NoPlaneta *filtraLista(NoPlaneta *head, char *texto, float minimo, float
maximo)
{
    NoPlaneta *head2 = NULL;
    NoPlaneta *tail2 = NULL;
    NoPlaneta *n = NULL;
    NoPlaneta *aux = NULL;
    NoSatelite *nsat = NULL;
    NoSatelite *auxsat = NULL;
    NoSatelite *tailsat = NULL;
    char *found = NULL;
    n = head;

    while (n != NULL)
    {
        if (head2 == NULL)
        {
            if (insereNo(n->nome, n->distancia, texto, minimo, maximo))
            {

```

```

        aux = createNoPlaneta(n->nome, n->distancia);
        insertNoPlaneta(&head2, aux);
        tail2 = aux;
        if (n->nextsat != NULL)
        {
            // Temos satelites
            nsat = n->nextsat;
            while (nsat != NULL)
            {
                if (insereNo(nsat->nome, nsat->distancia, texto,
minimo, maximo))
                {
                    auxsat = createNoSatelite(nsat->nome, nsat-
>distancia);
                    insertNoSatelite(&tail2->nextsat, auxsat);
                }
                nsat = nsat->nextsat;
            }
        }
        n = n->nextpla;
    }
    else
    {
        if (insereNo(n->nome, n->distancia, texto, minimo, maximo))
        {
            aux = createNoPlaneta(n->nome, n->distancia);
            insertNoPlaneta(&tail2->nextpla, aux);
            tail2 = aux;
            if (n->nextsat != NULL)
            {
                // Temos satelites
                nsat = n->nextsat;
                while (nsat != NULL)
                {
                    if (insereNo(nsat->nome, nsat->distancia, texto,
minimo, maximo))
                    {
                        auxsat = createNoSatelite(nsat->nome, nsat-
>distancia);
                        insertNoSatelite(&tail2->nextsat, auxsat);
                    }
                    nsat = nsat->nextsat;
                }
            }
        }
    }
}

```

```

        n = n->nextpla;
    }
}
return head2;
}

// funcao que ordena a lista segundo o tipo de ordenacao
NoPlaneta *ordenaLista(NoPlaneta *head, int tipo_ordenacao)
{
    NoPlaneta *n;
    NoSatelite *auxsat;
    if ((tipo_ordenacao != ORDEM_ALFABETICA) && (tipo_ordenacao !=
ORDEM_UA))
    {
        tipo_ordenacao = ORDEM_ENTRADA;
    }
    if (tipo_ordenacao != ORDEM_ENTRADA)
    {
        /* ordena pelo nome */
        quickSortPlanetas(&head, tipo_ordenacao);
        /* vamos ordenar os satelites */
        n = head;
        while (n != NULL)
        {
            if (n->nextsat != NULL)
            {
                auxsat = n->nextsat;
                quickSortSatelites(&auxsat, tipo_ordenacao);
                n->nextsat = auxsat;
            }
            n = n->nextpla;
        }
    }
    return head;
}

// Limpar de memoria a lista
int freeLista(NoEstrela *head)
{
    NoPlaneta *pla = NULL;
    NoSatelite *sat = NULL;

    NoPlaneta *curpla = NULL;
    NoSatelite *cursat = NULL;

    pla = head->nextpla;
    free(head);
    while (pla != NULL)
    {

```

```

        sat = pla->nextsat;
        while (sat != NULL)
        {
            cursat = sat;
            sat = sat->nextsat;
            free(cursat);
        }
        curpla = pla;
        pla = pla->nextpla;
        free(curpla);
    }
}

int main()
{
    NoEstrela *head = NULL;
    NoEstrela *star;

    NoPlaneta *tail = NULL;
    NoPlaneta *n;

    NoSatelite *sat = NULL;

    char dados[BUFFER];
    fgets(dados, BUFFER, stdin);
    dados[strcspn(dados, "\n")] = 0;

    int numerocaracteres;
    char nome[BUFFER];
    float distancia;

    numerocaracteres = strlen(dados);
    if (numerocaracteres != 0)
    {
        star = createNoEstrela(dados);
        // Primeiro nó no início da lista - head é atualizado.
        insertNoEstrela(&head, star);

        while (numerocaracteres != 0)
        {

            fgets(dados, BUFFER, stdin);
            dados[strcspn(dados, "\n")] = 0;

            numerocaracteres = strlen(dados);
            if (numerocaracteres != 0)
            {
                dados[strcspn(dados, "\n")] = 0;

```

```

char *pt = (char *)strtok(dados, " ");

strcpy(nome, pt);
if (strcmp(nome, "#") != 0)
{
    // entrada de um planeta
    pt = (char *)strtok(NULL, " ");
    distancia = atof(pt);

    n = createNoPlaneta(nome, distancia);
    if (tail == NULL)
    {
        // primeiro planeta
        // insere depois da estrela
        insertNoPlaneta(&star->nextpla, n);
    }
    else
    {
        // Inserir No na cauda da lista
        // no ultimo planeta
        insertNoPlaneta(&tail->nextpla, n);
    }

    // Atualizar a cauda da lista
    tail = n;
}
else
{
    // entrada de um satellite
    char *pt = (char *)strtok(NULL, " ");
    strcpy(nome, pt);

    pt = (char *)strtok(NULL, " ");
    distancia = atof(pt);

    sat = createNoSatelite(nome, distancia);
    // Inserir No na cauda da lista dos satelites
    insertNoSatelite(&tail->nextsat, sat);
}
}

// leitura dos dados de ordenação, filtro, mínimo e máximo
fgets(dados, BUFFER, stdin);
dados[strcspn(dados, "\n")] = 0;

numerocaracteres = strlen(dados);

```

```

    int ordenacao;
    char filtro_nome[BUFFER];
    float filtro_minimo;
    float filtro_maximo;

    if (numerocaracteres != 0)
    {
        // tratamento dos dados de ordenação, filtro, mínimo e máximo
        char *pt = (char *)strtok(dados, " ");
        ordenacao = atoi(pt);

        pt = (char *)strtok(NULL, " ");
        strcpy(filtro_nome, pt);

        pt = (char *)strtok(NULL, " ");
        filtro_minimo = atof(pt);

        pt = (char *)strtok(NULL, " ");
        filtro_maximo = atof(pt);
        // guarda o no inicial da lista, a estrela
        NoEstrela *estrela2;
        estrela2 = createNoEstrela(head->nome);

        NoPlaneta *head3 = NULL;
        n = head->nextpla;
        // filtra a lista depois da estrela, só os planetas e
satelitesc
        head3 = filtraLista(n, filtro_nome, filtro_minimo,
filtro_maximo);
        n = head3;
        // ordena a lista de planetas e satelites
        head3 = ordenaLista(n, ordenacao);
        // imprime a informação
        printListPlanetas(estrela2, head3);
    }
}
if (head != NULL)
{
    /* liberta memoria*/
    freeLista(head);
    head = NULL;
}

return 0;
}

```

## Código Programa 4

```
/*
 * Programa: Pergunta 4 - IP - Efolio B
 * Descrição:
 *   Após receber informação sobre o sistema planetário, composto por estrela,
 *   planetas e luas, na entrada de dados standard (stdin), apresenta a informacao de
 *   uma viagem entre dois pontos (satelite ou planeta).
 *
 *   De: <estrela>-<planeta>-<satélite>
 *   Para: <estrela>-<planeta>-<satélite>
 *   Estações: <satélite> (<dias>) <planeta> ... (<dias>) <satélite>
 *   Duração: <anos> anos <meses> meses <dias> dias
 *
 *
 * Dados inseridos:
 *   O nome da estrela seguido dos nomes dos planetas, um nome por linha.
 *   Os nomes não contêm espaços.
 *   A frente do nome de cada planeta é inserida a distância do planeta à estrela,
 *   em unidades astronómicas (UA).
 *   As linhas que comecem com # contem a informação de um satelite do planeta.
 *   A distância referida no satélite, é relativamente ao planeta que orbita,
 *   tendo 4 casas decimais.
 *   Após a introdução do último planeta/satélite, segue-se uma linha em branco,
 *   seguida da linha com dois elementos, astro origem e astro destino.
 *
 *   A empresa assegura um elevado número de viagens, com uma saída por dia para
 *   todas as ligações asseguradas. No entanto não consegue encurtar a
 *   duração das viagens. As viagens de e para a superfície têm a duração de 1
 *   dia. No entanto as viagens entre estações orbitais, dependem da
 *   distância do satélite ao planeta (ou do planeta à estrela), correspondendo a
 *   30 dias vezes UA, arredondando ao dia para cima.
 *
 *   Pretende-se para quaisquer dois astros seja calculada a duração total da
 *   viagem, no formato anos/meses/dias, bem como o plano de viagem.
 *
 *   A identificação do astro pode ter entre 1 e 3 elementos, dependente do astro
 *   ser estrela, planeta ou satélite. As estações correspondem ao
 *   caminho necessário efetuar para ir de um astro para o outro, tendo entre
 *   parêntesis o número de dias de cada viagem. No final temos a duração total,
 *   no formato # anos # meses # dias, onde # representa o valor. No entanto, a
 *   versão plural e singular de cada unidade deve ser tida em consideração, e
 *   não se deve mostrar uma unidade para 0 elementos. Os anos devem ser
 *   considerados de 30*12 dias, e um mês de 30 dias.
 *
 */
```



```

/* BIBLIOTECAS */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
/* MACROS */
#define BUFFER 100

/* TIPOS */
/* Satellite: Tipo que permite criar uma lista encadeada de satelites */
typedef struct SSatelite
{
    char *nome;
    float distancia;
    struct SSatelite *nextsat;
} NoSatelite;
/* Planeta: Tipo que permite criar uma lista encadeada de planetas que
podem conter ou nao uma lista de satelites */
typedef struct SPlaneta
{
    char *nome;
    float distancia;
    struct SSatelite *nextsat;
    struct SPlaneta *nextpla;
} NoPlaneta;

/* CRIA ESTANCIAS DAS ESTRUTURAS*/

// Descrição: Cria instância de um Planeta
NoPlaneta *createNoPlaneta(char *nome, float distancia)
{
    char *aux;
    aux = malloc(sizeof(nome));
    strcpy(aux, nome);

    NoPlaneta *newNode = (NoPlaneta *)malloc(sizeof(NoPlaneta));
    newNode->nome = aux;
    newNode->distancia = (float)distancia;
    newNode->nextsat = NULL;
    newNode->nextpla = NULL;
    return newNode;
}

// Descrição: Cria instância de um Satellite
NoSatelite *createNoSatelite(char *nome, float distancia)
{
    char *aux;
    aux = malloc(sizeof(nome));
    strcpy(aux, nome);

```

```

    NoSatelite *newNode = (NoSatelite *)malloc(sizeof(NoSatelite));
    newNode->nome = aux;
    newNode->distancia = distancia;
    newNode->nextsat = NULL;
    return newNode;
}

/* INSERE INFORMACAO NA LISTA */
// Insere um Planeta na lista
void insertNoPlaneta(NoPlaneta **link, NoPlaneta *newNode)
{
    newNode->nextpla = *link;
    *link = newNode;
}
// Insere um Satellite na lista
void insertNoSatelite(NoSatelite **link, NoSatelite *newNode)
{
    newNode->nextsat = *link;
    *link = newNode;
}

/* FUNÇÕES AUXILIARES */
// Função que retorna se o valor de um float é um inteiro ou não
int isInteger(float numero)
{
    return (floor(numero) == numero);
}
// Função que efetua o trim de uma string
char *trimString(char *str)
{
    char *end;

    while (isspace((unsigned char)*str))
        str++;

    if (*str == 0)
        return str;

    end = str + strlen(str) - 1;
    while (end > str && isspace((unsigned char)*end))
        end--;

    end[1] = '\0';

    return str;
}
// Função que arredonda o numero para cima
double arrNumero(double numero)

```

```

{
    double aux = numero;
    if ((numero - (double)floor(numero) == 0))
    {
        // numero inteiro retorna o proprio
        aux = (double)numero;
    }
    else
    {
        aux = (numero - (double)floor(numero) < 0.5) ?
(double)floor(numero + 1) : (double)floor(numero + 0.5);
    }
    return aux;
}

// Retorna o último Planeta da lista
NoPlaneta *getLastPlaneta(NoPlaneta *cur)
{
    while (cur != NULL &&
        cur->nextpla != NULL)
        cur = cur->nextpla;
    return cur;
}

// Retorna o último Satellite da lista
NoSatelite *getLastSatelite(NoSatelite *cur)
{
    while (cur != NULL &&
        cur->nextsat != NULL)
        cur = cur->nextsat;
    return cur;
}

// funcao que retorna o planeta aonde se encontra o texto, que pode estar
no planeta ou num satelite
NoPlaneta *findPlaneta(NoPlaneta *head, char *texto)
{
    NoPlaneta *head2 = NULL;

    NoPlaneta *posicao2 = NULL;

    NoPlaneta *n = NULL;
    NoSatelite *nsat = NULL;

    NoPlaneta *no_planeta = NULL;
    NoSatelite *no_satelite = NULL;
    int found;
    n = head;

    while (n != NULL)
    {

```

```

        found = strcmp(n->nome, texto);
        if (found == 0)
        {
            no_planeta = createNoPlaneta(n->nome, n->distancia);
            insertNoPlaneta(&posicao2->nextpla, no_planeta);
            if (head2 == NULL)
            {
                head2 = posicao2;
            }
            n = getLastPlaneta(n);
        }
        else
        {
            if (n->nextsat != NULL)
            {
                // Temos satelites
                nsat = n->nextsat;
                while (nsat != NULL)
                {
                    found = strcmp(nsat->nome, texto);
                    if (found == 0)
                    {
                        no_planeta = createNoPlaneta(n->nome, n-
>distancia);
                        insertNoPlaneta(&posicao2->nextpla, no_planeta);
                        no_satelite = createNoSatelite(nsat->nome, nsat-
>distancia);
                        insertNoSatelite(&no_planeta->nextsat,
no_satelite);
                        nsat = getLastSatelite(nsat);
                        n = getLastPlaneta(n);
                    }
                    nsat = nsat->nextsat;
                }
            }
            else
            {
                if (head2 == NULL)
                {
                    // insere a estrela
                    no_planeta = createNoPlaneta(n->nome, n->distancia);
                    insertNoPlaneta(&head2, no_planeta);

                    posicao2 = getLastPlaneta(head2);
                    ;
                }
            }
        }
    }
}

```

```

    }
}
n = n->nextpla;
}
return head2;
}

// procedimento que calcula e imprime o output da viagem
void duracaoViagem(NoPlaneta *inicial, NoPlaneta *final)
{
    double result = 0;
    double auxduracao = 0;

    printf("\nEstações: ");

    if (inicial->nextpla->nextsat != NULL)
    {
        auxduracao = arrNumero((inicial->nextpla->nextsat->distancia) *
30);
        result += auxduracao;
        printf("%s (%.01f) ", inicial->nextpla->nextsat->nome,
auxduracao);
    }

    if (inicial->nextpla != NULL)
    {
        if (final->nextpla != NULL)
        {
            if (strcmp(final->nextpla->nome, inicial->nextpla->nome) !=
0)
            {
                auxduracao = arrNumero((inicial->nextpla->distancia) *
30);
                result += auxduracao;
                printf("%s (%.01f) ", inicial->nextpla->nome,
auxduracao);
            }
            else
            {
                printf("%s ", inicial->nextpla->nome);
            }
        }
    }

    if (final->nextpla != NULL)
    {
        if (strcmp(final->nextpla->nome, inicial->nextpla->nome) != 0)
        {

```

```

        auxduracao = arrNumero((final->nextpla->distancia) * 30);
        result += auxduracao;
        printf("%s (%.0lf) %s ", final->nome, auxduracao, final-
>nextpla->nome);
    }
}

if (final->nextpla->nextsat != NULL)
{
    auxduracao = arrNumero((final->nextpla->nextsat->distancia) *
30);
    result += auxduracao;
    printf("(%.0lf) %s", auxduracao, final->nextpla->nextsat->nome);
}
result += 2;
printf("\nDuração: ");
int aux = result;
int numeroanos = 0;
int numeromeses = 0;
int numerodias = 0;
/*anos: 30 dias * 12 (360 dias) meses mes: 30 dias */

if (aux > 360)
{
    // mais do que 1 ano
    numeroanos = (int)aux / 360;
    aux = (int)aux % 360;
    if (aux > 30)
    {
        // mais do que 1 mes
        numeromeses = (int)aux / 30;
        numerodias = (int)aux % 30;
    }
    else
    {
        numeromeses = 0;
        numerodias = aux;
    }
}
else
{
    numeroanos = 0;
    if (aux > 30)
    {
        // mais do que 1 mes
        numeromeses = (int)aux / 30;
        numerodias = (int)aux % 30;
    }
    else

```

```

        {
            numeromeses = 0;
            numerodias = aux;
        }
    }
    if (numeroanos != 0)
    {
        numeroanos == 1 ? printf("%d ano ", numeroanos) : printf("%d anos
", numeroanos);
    }
    if (numeromeses != 0)
    {
        numeromeses == 1 ? printf("%d mês ", numeromeses) : printf("%d
meses ", numeromeses);
    }
    if (numerodias != 0)
    {
        numerodias == 1 ? printf("%d dia ", numerodias) : printf("%d dias
", numerodias);
    }
}

// Limpar de memoria a lista
int freeLista(NoPlaneta *head)
{
    NoPlaneta *pla = NULL;
    NoSatelite *sat = NULL;

    NoPlaneta *curpla = NULL;
    NoSatelite *cursat = NULL;

    pla = head->nextpla;
    free(head);
    while (pla != NULL)
    {
        sat = pla->nextsat;
        while (sat != NULL)
        {
            cursat = sat;
            sat = sat->nextsat;
            free(cursat);
        }
        curpla = pla;
        pla = pla->nextpla;
        free(curpla);
    }
}

```

```

int main()
{
    NoPlaneta *head = NULL;
    NoPlaneta *tail = NULL;
    NoPlaneta *n;

    NoSatelite *sat = NULL;

    char dados[BUFFER];
    fgets(dados, BUFFER, stdin);
    dados[strcspn(dados, "\n")] = 0;

    int numerocaracteres;
    char nome[BUFFER];
    float distancia;

    numerocaracteres = strlen(dados);
    if (numerocaracteres != 0)
    {
        n = createNoPlaneta(dados, 0);
        // Primeiro nó no início da lista - head é atualizado.
        insertNoPlaneta(&head, n);
        // Atualizar a cauda da lista
        tail = n;
        while (numerocaracteres != 0)
        {

            fgets(dados, BUFFER, stdin);
            dados[strcspn(dados, "\n")] = 0;

            numerocaracteres = strlen(dados);
            if (numerocaracteres != 0)
            {
                dados[strcspn(dados, "\n")] = 0;

                char *pt = (char *)strtok(dados, " ");

                strcpy(nome, pt);
                if (strcmp(nome, "#") != 0)
                {
                    // entrada de um planeta
                    pt = (char *)strtok(NULL, " ");
                    distancia = atof(pt);

                    n = createNoPlaneta(nome, distancia);

                    // Inserir No na cauda da lista
                    // no último planeta

```



```

        insertNoPlaneta(&tail->nextpla, n);

        // Atualizar a cauda da lista
        tail = n;
    }
    else
    {

        // entrada de um satellite
        char *pt = (char *)strtok(NULL, " ");
        strcpy(nome, pt);

        pt = (char *)strtok(NULL, " ");
        distancia = atof(pt);

        sat = createNoSatelite(nome, distancia);
        // Inserir No na cauda da lista dos satelites
        insertNoSatelite(&tail->nextsat, sat);
    }
}

fgets(dados, BUFFER, stdin);
dados[strcspn(dados, "\n")] = 0;

numerocaracteres = strlen(dados);

char origem[BUFFER];
char destino[BUFFER];

if (numerocaracteres != 0)
{
    char *pt = (char *)strtok(dados, " ");
    strcpy(origem, pt);

    pt = (char *)strtok(NULL, " ");
    strcpy(destino, pt);

    // A viagem é entre dois astros (planeta ou satellite)
    // procurar o planeta inicial e o planeta final

    // procurar o no inicial
    NoPlaneta *inicial;
    inicial = findPlaneta(head, origem);

    // imprimir origem
    printf("De: ");
    printf("%s", trimString(inicial->nome));
    if (inicial->nextpla != NULL)

```

```

        {
            printf("-%s", trimString(inicial->nextpla->nome));
            if (inicial->nextpla->nextsat != NULL)
            {
                printf("-%s", trimString(inicial->nextpla->nextsat-
>nome));
            }
        }

        // procurar o no final
        NoPlaneta *final;
        final = findPlaneta(head, destino);

        // imprimir destino
        printf("\nPara: ");
        printf("%s", trimString(final->nome));
        if (final->nextpla != NULL)
        {
            printf("-%s", trimString(final->nextpla->nome));
            if (final->nextpla->nextsat != NULL)
            {
                printf("-%s", trimString(final->nextpla->nextsat-
>nome));
            }
        }

        // imprimir a duracao de viagem entre o no inicial e o no
final
        duracaoViagem(inicial, final);
    }
}

if (head != NULL)
{
    /* liberta memoria*/
    freeLista(head);
    head = NULL;
}

return 0;
}

```