E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Introdução à Programação

CÓDIGO: 21173

DOCENTE: José Coelho

A preencher pelo estudante

NOME: Diogo Miguel Palma Sustelo

N.º DE ESTUDANTE: 2201148

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 09/01/2023

TRABALHO / RESOLUÇÃO:

Explicação do programa da alínea A:

- Foram criadas as estruturas de dados “TVPlaneta”(aloca nome do planeta) e “TSPlanetario”(aloca o nome da estrela e um ponteiro array da estrutura de dados “TVPlaneta” já citada), para alocar os dados do sistema planetário.
- A variável “sistemap” do tipo “TSPlanetario” é criada e inicializada chamando a função “LerSistema()”que por sua vez chama a função “SPCriar()”, bem como os procedimentos “SPLeEstrela” e “SPLePlaneta”, que após inicializada a variável, os valores da estrela e dos planetas é recebido enquanto não houver um valor nulo. Através da função “fgets()” recebe o valor numa variável auxiliar e passa-o para a estrutura através das funções “sscanf()” e “strcpy()”, fazendo assim com que não sejam copiadas quebras de linha e o output seja mostrado apenas em uma linha, sem necessidade de retirar o caracter ”\n”.
- Foi criada também uma função que conta os planetas, para auxiliar aquando de imprimir no ecrã a quantidade, bem como auxiliar no procedimento “SPLibertar()”
- É depois mostrado o sistema através do procedimento “MostraSistema()” e depois é libertado o espaço em memória alocado através do procedimento “SPLibertar()”
- Além dos testes efetuados pelo VPL, foram feitos testes locais com diferentes inputs, e verificados os respetivos Outputs

Explicação do programa da alínea B:

- Reutilizando o código da alínea A, foi adicionado à estrutura “TVPlaneta” uma variável para guardar a distância do planeta à estrela.
- Para conter complexidade foi criado o procedimento “AlocaMem()” a fim de alocar a memória dinâmica chamando o procedimento quando necessário.
- Do mesmo modo o procedimento “SPLePlaneta()” foi modificado para receber também a distância, e passá-la para a estrutura de dados, que depois é ordenada por ordem crescente das UA, através do procedimento “OrdenaPlaneta()”, que foi adicionado para esse efeito, a fim de aquando de imprimir as informações no ecrã, se possa imprimir o valor da distância da primeira e da ultima o posição, com auxilio também do procedimento “ContaPlaneta()”.
- As restantes estruturas do programa foram melhoradas para corresponder à realidade do novo programa.
- Além dos testes efetuados pelo VPL, foram feitos testes locais com diferentes inputs, e verificados os respetivos Outputs

Explicação do programa da alínea C:

- Reutilizando o código da alínea B, foi adicionada uma estrutura de dados “TVSatelite”, para acolher os dados dos satélites que orbitam certo planeta, tais como os nomes e as respetivas distâncias ao planeta, a qual foi integrada como um ponteiro de array estrutura já existente “TVPlaneta”, foi também adicionado um procedimento para ler os dados dos satélites e armazená-los na respetiva estrutura já referida.
- Foi criado um procedimento “SPLeOpcoes()” para recolher os dados das opções de filtragem do conteúdo a apresentar, os quais são passados para os também criados procedimentos “OrdenaAstros()”, que por sua vez se reparte em dois novos procedimentos “OrdenaAstrosUA()” e “OrdenaAstrosNome()”, consoante seja a seleção feita, que foram criados com base no procedimento “OrdenaPlaneta()” da alínea anterior, e os procedimentos “FiltraNomeAstros()”, ”FiltraMin() e “FiltraMax()”, que utilizam funções como “strstr()” e a comparação de valores do array, de modo a após as comparações se poder substituir o valor de “n”, pelo “n+1”, caso o valor “n” não seja o desejado, com o intuito de o output ser o desejado quando impresso.
- As restantes estruturas do programa foram melhoradas para corresponder à realidade do novo programa.
- Além dos testes efetuados pelo VPL, foram feitos testes locais com diferentes inputs, e verificados os respetivos Outputs

Explicação do programa da alínea D:

- Reutilizando o código da alínea C, foram removidos os procedimentos que ordenavam os astros e os procedimentos que filtravam os mesmos, pois já não seriam necessários, e foi adicionada uma nova estrutura de dados, “TViagem”, que contempla mais 3 estruturas, “VOrigem”, “VDestino” e “VRota”, por forma a cobrir as necessidades do novo programa e receber nela os dados da estação de origem, estação de destino e rota a traçar para efetuar a viagem, respetivamente.
- Foram criados mais os seguintes procedimentos: “VCriar()”, para inicializar as estruturas de dados, “VProcessaOrigemA()” e “VProcessaOrigem()” para receberem e alojarem os dados de origem, “VProcessaDestinoA()” e “VProcessaDestino()”, para receberem e alojarem os dados do destino, provindos do procedimento “VProcessaEstacoes()”.
- Foram também criados os procedimentos “VProcessaRotaOrigemA()”, para processar a string que irá dar o output da origem da rota, “VProcessaRotaOrigemP()”,para processar a rota caso a origem seja um planeta ou “VProcessaRotaOrigemS()”,caso a origem seja um satélite, todas elas chamadas no procedimento “VProcessaRotaOrigem()”.

- À semelhança dos dados de origem e da rota de origem, foram também criados os “VProcessaRotaDestinoA()”, “VProcessaRotaDestinoP()”, “VProcessaRotaDestinoS()”, “VProcessaRotaDestino()”, para lidar e processar os dados do destino.
- A rota é processada como duas metades, uma na origem, a outra no destino, onde uma começa a ser processada onde acaba a anterior, alojando os dados no array da estrutura de dados, faz o somatório dos dias necessários para deslocação, através da distância, á medida que o programa avança, para mais tarde ser processado para traduzir esse tempo de viagem separado por anos, meses e dias.
- Foi modificado então o procedimento “MostraSistema()” para processar e imprimir esses dados no ecrã, utilizando o mesmo raciocínio do programa “trocos2.c” realizado nas AFS.
- As restantes estruturas do programa foram melhoradas para corresponder à realidade do novo programa.
- Além dos testes efetuados pelo VPL, foram feitos testes locais com diferentes inputs, e verificados os respetivos Outputs

Anexos :

Código da alínea A:

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

#define MAXSTR 255

typedef struct {
    char *nome;
} TVPlaneta;

typedef struct {
    char *estrela;
    TVPlaneta *planeta;
} TSPlanetario;

TSPlanetario SPCriar() {
    TSPlanetario sistemap;
    sistemap.estrela = NULL;
    sistemap.planeta = NULL;
    return sistemap;
}

void SPLeEstrela(TSPlanetario *sistemap) {
    char str[MAXSTR], provisorio[MAXSTR];

    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,]", str);
    sistemap->estrela = (char *) malloc((strlen(str) + 1) * sizeof(char));

    if (sistemap->estrela != NULL) {
        strcpy(sistemap->estrela, str);
    }
}

void SPLePlaneta(TSPlanetario *sistemap, int i) {
    char str[MAXSTR], provisorio[MAXSTR];

    *str = '\0';

    if (sistemap->planeta == NULL) {
        sistemap->planeta = (TVPlaneta *) malloc(((i + 1) *
sizeof(TVPlaneta)));
    } else {
        sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta, ((i +
1) * sizeof(TVPlaneta)));
    }

    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,]", str);

    sistemap->planeta[i].nome = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    if (sistemap->planeta != NULL && sistemap->planeta[i].nome != NULL) {
        strcpy(sistemap->planeta[i].nome, str);
    }
}
```

```
int ContaPlaneta(TSPlanetario *sistemap) {
    int i, conta;

    i = 0;
    conta = 0;
    while ((strcmp(sistemap->planeta[i].nome, "\0") != 0)) {
        i++;
        conta++;
    }
    return conta;
}

void SPLibertar(TSPlanetario *sistemap) {
    int i, tamanho;

    if (sistemap->estrela != NULL) {
        free(sistemap->estrela);
        sistemap->estrela = NULL;
    }

    tamanho = ContaPlaneta(sistemap);
    for (i = 0; i <= tamanho; i++) {
        if (sistemap->planeta[i].nome != NULL) {
            free(sistemap->planeta[i].nome);
            sistemap->planeta[i].nome = NULL;
        }
    }

    if (sistemap->planeta != NULL) {
        free(sistemap->planeta);
        sistemap->planeta = NULL;
    }
}

/*---------- Programa ----------*/

TSPlanetario LerSistema() {
    TSPlanetario sistemap;
    int i;

    sistemap = SPCriar();
    SPLeEstrela(&sistemap);

    i = 0;
    do {
        SPLePlaneta(&sistemap, i);
        i++;
    } while (strcmp(sistemap.planeta[i - 1].nome, "\0") != 0);
    return sistemap;
}

void MostraSistema(TSPlanetario *sistemap) {
    printf("%s, sistema planetário com %d planetas", sistemap->estrela,
ContaPlaneta(sistemap));
}

void main() {
    TSPlanetario sistemap;
    sistemap = LerSistema();
    MostraSistema(&sistemap);
```

```
    SPLibertar(&sistemap);
}
```

Código da alínea B:

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

#define MAXSTR 255

typedef struct {
    char *nome;
    float distancia;
} TVPlaneta;

typedef struct {
    char *estrela;
    TVPlaneta *planeta;
} TSPlanetario;

void AlocaMem(TSPlanetario *sistemap, int i, char *str, char *estrutura) {

    if (strcmp(estrutura, "estrela") == 0) {
        sistemap->estrela = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }

    if (strcmp(estrutura, "planeta") == 0) {
        if (sistemap->planeta == NULL) {
            sistemap->planeta = (TVPlaneta *) malloc(((i + 1) *
sizeof(TVPlaneta)));
        } else {
            sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((i + 1) * sizeof(TVPlaneta)));
        }
        sistemap->planeta[i].nome = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }
}

TSPlanetario SPCriar() {
    TSPlanetario sistemap;
    sistemap.estrela = NULL;
    sistemap.planeta = NULL;
    return sistemap;
}

void SPLeEstrela(TSPlanetario *sistemap) {
    char str[MAXSTR], provisorio[MAXSTR];

    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,]", str);
    AlocaMem(sistemap, (int) 0, str, "estrela");

    if (sistemap->estrela != NULL) {
        strcpy(sistemap->estrela, str);
    }
}
```

```
void SPLePlaneta(TSPlanetario *sistemap, int i) {
    char str[MAXSTR], provisorio[MAXSTR];
    float distancia;

    *str = '\0';
    distancia = 0;
    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,0-9] %f", str, &distancia);
    AlocaMem(sistemap, i, str, "planeta");

    if (sistemap->planeta != NULL && sistemap->planeta[i].nome != NULL) {
        strcpy(sistemap->planeta[i].nome, str);
        sistemap->planeta[i].distancia = distancia;
    }
}

int ContaPlaneta(TSPlanetario *sistemap) {
    int i, conta;

    i = 0;
    conta = 0;
    while ((strcmp(sistemap->planeta[i].nome, "\0") != 0)) {
        i++;
        conta++;
    }
    return conta;
}

void OrdenaPlaneta(TSPlanetario *sistemap, int tamanho) {
    int i, j;
    TSPlanetario aux;

    aux.planeta = (TVPlaneta *) malloc(((1) * sizeof(TVPlaneta)));

    if (aux.planeta != NULL) {
        for (i = 0; i < tamanho; i++)
            for (j = i + 1; j < tamanho; j++)
                if (sistemap->planeta[i].distancia > sistemap-
>planeta[j].distancia) {
                    aux.planeta[0].nome = (char *) malloc((strlen(sistemap-
>planeta[i].nome) + 1) * sizeof(char));
                    aux.planeta[0] = sistemap->planeta[i];
                    sistemap->planeta[i] = sistemap->planeta[j];
                    sistemap->planeta[j] = aux.planeta[0];
                }
    }
    free(aux.planeta);
}

void SPLibertar(TSPlanetario *sistemap) {
    int i, tamanho;

    if (sistemap->estrela != NULL) {
        free(sistemap->estrela);
        sistemap->estrela = NULL;
    }

    tamanho = ContaPlaneta(sistemap);
    for (i = 0; i <= tamanho; i++) {
        if (sistemap->planeta[i].nome != NULL) {
            free(sistemap->planeta[i].nome);
```

```
            sistemap->planeta[i].nome = NULL;
            sistemap->planeta[i].distancia = 0;
        }
    }

    if (sistemap->planeta != NULL) {
        free(sistemap->planeta);
        sistemap->planeta = NULL;
    }
}

/*---------- Programa ----------*/

TSPlanetario LerSistema() {
    TSPlanetario sistemap;
    int i;

    sistemap = SPCriar();

    SPLeEstrela(&sistemap);

    i = 0;
    do {
        SPLePlaneta(&sistemap, i);
        i++;
    } while ((strcmp(sistemap.planeta[i - 1].nome, "\0") != 0));

    OrdenaPlaneta(&sistemap, ContaPlaneta(&sistemap));
    return sistemap;
}

void MostraSistema(TSPlanetario *sistemap) {
    printf("%s, sistema planetário com %d planetas a distâncias entre %.2f
e %.2f UA.",
           sistemap->estrela,
           ContaPlaneta(sistemap),
           sistemap->planeta[0].distancia,
           sistemap->planeta[ContaPlaneta(sistemap) - 1].distancia);
}

void main() {
    TSPlanetario sistemap;
    sistemap = LerSistema();
    MostraSistema(&sistemap);
    SPLibertar(&sistemap);

}
```

Código da alínea C:

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

#define MAXSTR 255

typedef struct {
    char *nome;
    float distancia;
} TVSatelite;

typedef struct {
    char *nome;
    float distancia;
    TVSatelite *satelite;
} TVPlaneta;

typedef struct {
    char *estrela;
    TVPlaneta *planeta;
} TSPlanetario;

TSPlanetario SPCriar() {
    TSPlanetario sistemap;
    sistemap.estrela = NULL;
    sistemap.planeta = NULL;
    return sistemap;
}

void AlocaMem(TSPlanetario *sistemap, int i, int j, char *str, char
*estrutura) {
    if (strcmp(estrutura, "estrela") == 0) {
        sistemap->estrela = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }

    if (strcmp(estrutura, "planeta") == 0) {
        if (sistemap->planeta == NULL) {
            sistemap->planeta = (TVPlaneta *) malloc(((i + 1) *
sizeof(TVPlaneta)));
        } else {
            sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((i + 1) * sizeof(TVPlaneta)));
        }
        sistemap->planeta[i].nome = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }

    if (strcmp(estrutura, "satelite") == 0) {
        if (sistemap->planeta[i].satelite == NULL) {
            sistemap->planeta[i].satelite = (TVSatelite *) malloc(((j + 1)
* sizeof(TVSatelite)));
        } else {
            sistemap->planeta[i].satelite = (TVSatelite *)
realloc(sistemap->planeta[i].satelite,
                                                                ((j + 1)
* sizeof(TVSatelite)));
        }
        sistemap->planeta[i].satelite[j].nome = (char *)
```

```
malloc((strlen(str) + 1) * sizeof(char));
    }
}

void SPLeEstrela(TSPlanetario *sistemap) {
    char str[MAXSTR], provisorio[MAXSTR];

    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,]", str);
    AlocaMem(sistemap, (int) 0, (int) 0, str, "estrela");

    if (sistemap->estrela != NULL) {
        strcpy(sistemap->estrela, str);
    }
}

void SPLePlaneta(TSPlanetario *sistemap, int i, char *provisorio) {
    char str[MAXSTR];
    float distancia;

    *str = '\0';
    distancia = 0;
    sscanf(provisorio, "%[^\n, ,] %f", str, &distancia);
    AlocaMem(sistemap, i, (int) 0, str, "planeta");

    if (sistemap->planeta != NULL && sistemap->planeta[i].nome != NULL) {
        strcpy(sistemap->planeta[i].nome, str);
        sistemap->planeta[i].distancia = distancia;
    }
}

void SPLeSatelite(TSPlanetario *sistemap, int i, int j, char *provisorio) {
    char strs[MAXSTR];
    float distancia;

    *strs = '\0';
    distancia = 0;
    sscanf(provisorio, "%*c %[^\n, ] %f", strs, &distancia);
    AlocaMem(sistemap, i, j, strs, "satelite");

    if (sistemap->planeta[i].satelite != NULL && sistemap-
>planeta[i].satelite[j].nome != NULL) {
        strcpy(sistemap->planeta[i].satelite[j].nome, strs);
        sistemap->planeta[i].satelite[j].distancia = distancia;
    }
}

void SPLeSistema(TSPlanetario *sistemap) {
    char provisorio[MAXSTR];
    int iplaneta, isatelite;

    iplaneta = 0;
    do {
        fgets(provisorio, MAXSTR, stdin);
        if (provisorio[0] == '#') {
            SPLeSatelite(sistemap, iplaneta - 1, isatelite, provisorio);
            SPLeSatelite(sistemap, iplaneta - 1, isatelite + 1, "");
            isatelite++;
        } else {
            isatelite = 0;
            SPLePlaneta(sistemap, iplaneta, provisorio);
```

```
            sistemap->planeta[iplaneta].satelite = NULL;
            SPLeSatelite(sistemap, iplaneta, isatelite, "");
            iplaneta++;
        }
    } while ((strcmp(sistemap->planeta[iplaneta - 1].nome, "\0") != 0));
}

int ContaAstros(TSPlanetario *sistemap, int i, char *estrutura) {
    int j, conta;

    conta = 0;
    if (strcmp(estrutura, "planeta") == 0) {
        while ((strcmp(sistemap->planeta[i].nome, "\0") != 0)) {
            i++;
            conta++;
        }
    }

    if (strcmp(estrutura, "satelite") == 0) {
        j = 0;
        while ((strcmp(sistemap->planeta[i].satelite[j].nome, "\0") != 0))
{
            j++;
            conta++;
        }
    }

    return conta;
}

void OrdenaAstrosNome(TSPlanetario *sistemap) {
    int i, j, h, tplaneta, tsatelite;
    TSPlanetario aux;

    tplaneta = ContaAstros(sistemap, 0, "planeta");
    aux.planeta = (TVPlaneta *) malloc(((1) * sizeof(TVPlaneta)));

    if (aux.planeta != NULL) {
        for (i = 0; i < tplaneta - 1; i++) {
            for (j = i + 1; j < tplaneta; j++) {
                if (strcmp(sistemap->planeta[i].nome, sistemap-
>planeta[j].nome) > 0) {
                    aux.planeta[0] = sistemap->planeta[i];
                    sistemap->planeta[i] = sistemap->planeta[j];
                    sistemap->planeta[j] = aux.planeta[0];
                }
            }
            tsatelite = ContaAstros(sistemap, i, "satelite");
            aux.planeta->satelite = (TVSatelite *) malloc(((1) *
sizeof(TVSatelite)));
            for (j = 0; j < tsatelite - 1; j++) {
                for (h = j + 1; h < tsatelite; h++) {
                    if (strcmp(sistemap->planeta[i].satelite[j].nome,
sistemap->planeta[i].satelite[h].nome) > 0) {
                        aux.planeta[0].satelite[0] = sistemap-
>planeta[i].satelite[j];
                        sistemap->planeta[i].satelite[j] = sistemap-
>planeta[i].satelite[h];
                        sistemap->planeta[i].satelite[h] =
aux.planeta[0].satelite[0];
                    }
```

```
                }
            }
        }
    }
    free(aux.planeta->satelite);
    free(aux.planeta);
}

void OrdenaAstrosUA(TSPlanetario *sistemap) {
    int i, j, h, tplaneta, tsatelite;
    TSPlanetario aux;

    tplaneta = ContaAstros(sistemap, 0, "planeta");
    aux.planeta = (TVPlaneta *) malloc(((1) * sizeof(TVPlaneta)));

    if (aux.planeta != NULL) {
        for (i = 0; i < tplaneta - 1; i++) {
            for (j = i + 1; j < tplaneta; j++) {
                if (sistemap->planeta[i].distancia > sistemap-
>planeta[j].distancia) {
                    aux.planeta[0] = sistemap->planeta[i];
                    sistemap->planeta[i] = sistemap->planeta[j];
                    sistemap->planeta[j] = aux.planeta[0];
                }
            }
            tsatelite = ContaAstros(sistemap, i, "satelite");
            aux.planeta->satelite = (TVSatelite *) malloc(((1) *
sizeof(TVSatelite)));
            for (j = 0; j < tsatelite - 1; j++) {
                for (h = j + 1; h < tsatelite; h++) {
                    if (sistemap->planeta[i].satelite[j].distancia >
sistemap->planeta[i].satelite[h].distancia) {
                        aux.planeta[0].satelite[0] = sistemap-
>planeta[i].satelite[j];
                        sistemap->planeta[i].satelite[j] = sistemap-
>planeta[i].satelite[h];
                        sistemap->planeta[i].satelite[h] =
aux.planeta[0].satelite[0];
                    }
                }
            }
        }
    }
    free(aux.planeta->satelite);
    free(aux.planeta);
}

void OrdenaAstros(TSPlanetario *sistemap, int ordem) {
    if (ordem == 2) {
        OrdenaAstrosNome(sistemap);
    }

    if (ordem == 3) {
        OrdenaAstrosUA(sistemap);
    }
}

void FiltraNomeAstros(TSPlanetario *sistemap, char *str) {
    int i, j, h, tplaneta, tsatelite;

    if (strcmp(str, "*") != 0) {
```

```
        tplaneta = ContaAstros(sistemap, 0, "planeta");
        for (i = 0; i < tplaneta; i++) {
            while (i < tplaneta && strstr(sistemap->planeta[i].nome, str)
== NULL) {
                for (j = i; j <= tplaneta; j++)
                    sistemap->planeta[j] = sistemap->planeta[j + 1];
                tplaneta--;
            }

            tsatelite = ContaAstros(sistemap, i, "satelite");
            for (j = 0; j < tsatelite; j++) {
                while (j < tsatelite && strstr(sistemap-
>planeta[i].satelite[j].nome, str) == NULL) {
                    for (h = j; h <= tsatelite; h++)
                        sistemap->planeta[i].satelite[h] = sistemap-
>planeta[i].satelite[h + 1];
                    tsatelite--;
                }
            }
            sistemap->planeta[i].satelite = (TVSatelite *)
realloc(sistemap->planeta[i].satelite,

((tsatelite + 1) * sizeof(TVSatelite)));
        }
        sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((tplaneta + 1) * sizeof(TVPlaneta)));
    }
}

void FiltraMin(TSPlanetario *sistemap, float minimo) {
    int i, j, h, tplaneta, tsatelite;

    if (minimo >= 0) {
        tplaneta = ContaAstros(sistemap, 0, "planeta");
        for (i = 0; i < tplaneta; i++) {
            while (i < tplaneta && sistemap->planeta[i].distancia < minimo)
{
                for (j = i; j <= tplaneta; j++)
                    sistemap->planeta[j] = sistemap->planeta[j + 1];
                tplaneta--;
            }

            tsatelite = ContaAstros(sistemap, i, "satelite");
            for (j = 0; j < tsatelite; j++) {
                while (j < tsatelite && sistemap-
>planeta[i].satelite[j].distancia < minimo) {
                    for (h = j; h <= tsatelite; h++)
                        sistemap->planeta[i].satelite[h] = sistemap-
>planeta[i].satelite[h + 1];
                    tsatelite--;
                }
            }
            sistemap->planeta[i].satelite = (TVSatelite *)
realloc(sistemap->planeta[i].satelite,

((tsatelite + 1) * sizeof(TVSatelite)));
        }
        sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((tplaneta + 1) * sizeof(TVPlaneta)));
    }
}
```

```
void FiltraMax(TSPlanetario *sistemap, float maximo) {
    int i, j, h, tplaneta, tsatelite;

    if (maximo >= 0) {
        tplaneta = ContaAstros(sistemap, 0, "planeta");
        for (i = 0; i < tplaneta; i++) {
            while (i < tplaneta && sistemap->planeta[i].distancia > maximo)
{
                for (j = i; j <= tplaneta; j++)
                    sistemap->planeta[j] = sistemap->planeta[j + 1];
                tplaneta--;
            }

            tsatelite = ContaAstros(sistemap, i, "satelite");
            for (j = 0; j < tsatelite; j++) {
                while (j < tsatelite && sistemap-
>planeta[i].satelite[j].distancia > maximo) {
                    for (h = j; h <= tsatelite; h++)
                        sistemap->planeta[i].satelite[h] = sistemap-
>planeta[i].satelite[h + 1];
                    tsatelite--;
                }
            }
            sistemap->planeta[i].satelite = (TVSatelite *)
realloc(sistemap->planeta[i].satelite,

((tsatelite + 1) * sizeof(TVSatelite)));
        }
        sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((tplaneta + 1) * sizeof(TVPlaneta)));
    }
}

void SPLeOpcoes(TSPlanetario *sistemap) {
    char provisorio[MAXSTR], str[MAXSTR];
    int ordem;
    float minimo, maximo;
    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%d %[^\n, ] %f %f", &ordem, str, &minimo, &maximo);
    OrdenaAstros(sistemap, ordem);
    FiltraNomeAstros(sistemap, str);
    FiltraMin(sistemap, minimo);
    FiltraMax(sistemap, maximo);

}

void SPLibertar(TSPlanetario *sistemap) {
    int i, j, tplaneta, tsatelite;

    if (sistemap->estrela != NULL) {
        free(sistemap->estrela);
        sistemap->estrela = NULL;
    }

    tplaneta = ContaAstros(sistemap, 0, "planeta");
    for (i = 0; i <= tplaneta; i++) {
        tsatelite = ContaAstros(sistemap, i, "satelite");
        for (j = 0; j <= tsatelite; j++) {
            if (sistemap->planeta[i].satelite[j].nome != NULL) {
                free(sistemap->planeta[i].satelite[j].nome);
```

```
                sistemap->planeta[i].satelite[j].nome = NULL;
                sistemap->planeta[i].satelite[j].distancia = 0;
            }
        }

        if (sistemap->planeta[i].satelite != NULL) {
            free(sistemap->planeta[i].satelite);
            sistemap->planeta[i].satelite = NULL;
        }

        if (sistemap->planeta[i].nome != NULL) {
            free(sistemap->planeta[i].nome);
            sistemap->planeta[i].nome = NULL;
            sistemap->planeta[i].distancia = 0;
        }
    }

    if (sistemap->planeta != NULL) {
        free(sistemap->planeta);
        sistemap->planeta = NULL;
    }
}

/*---------- Programa ----------*/

TSPlanetario ProcessaSistema() {
    TSPlanetario sistemap;

    sistemap = SPCriar();
    SPLeEstrela(&sistemap);
    SPLeSistema(&sistemap);
    SPLeOpcoes(&sistemap);
    return sistemap;
}

void MostraSistema(TSPlanetario *sistemap) {
    int i, j, tplaneta, tsatelite;
    tplaneta = ContaAstros(sistemap, 0, "planeta");
    printf("%s:\n", sistemap->estrela);
    for (i = 0; i < tplaneta; i++) {
        printf("- %s %.2f\n", sistemap->planeta[i].nome, sistemap-
>planeta[i].distancia);
        tsatelite = ContaAstros(sistemap, i, "satelite");
        for (j = 0; j < tsatelite; j++) {
            printf("--- %s %.4f\n", sistemap->planeta[i].satelite[j].nome,
sistemap->planeta[i].satelite[j].distancia);
        }
    }
}

void main() {
    TSPlanetario sistemap;
    sistemap = ProcessaSistema();
    MostraSistema(&sistemap);
    SPLibertar(&sistemap);
    exit(0);
}
```

Código da alínea D:

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

#define MAXSTR 255

typedef struct {
    char *nome;
    float distancia;
} TVSatelite;

typedef struct {
    char *nome;
    float distancia;
    TVSatelite *satelite;
} TVPlaneta;

typedef struct {
    char *estrela;
    TVPlaneta *planeta;
} TSPlanetario;

typedef struct {
    char *astro;
    char *origem;
    int indexi;
    int indexj;
} TOrigem;

typedef struct {
    char *astro;
    char *destino;
    int indexi;
    int indexj;
} TDestino;

typedef struct {
    char *astro;
    float distancia;
} TRota;

typedef struct {
    TOrigem origem;
    TDestino destino;
    TRota *rota;
    float distancia;
} TViagem;

TViagem VCriar() {
    TViagem viagem;
    viagem.rota = NULL;
    viagem.origem.origem = NULL;
    viagem.destino.destino = NULL;
    return viagem;
}

TSPlanetario SPCriar() {
    TSPlanetario sistemap;
```

```
    sistemap.estrela = NULL;
    sistemap.planeta = NULL;
    return sistemap;
}

void AlocaMem(TSPlanetario *sistemap, TViagem *viagem, int i, int j, char
*str, char *estrutura) {
    if (strcmp(estrutura, "estrela") == 0) {
        sistemap->estrela = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }

    if (strcmp(estrutura, "planeta") == 0) {
        if (sistemap->planeta == NULL) {
            sistemap->planeta = (TVPlaneta *) malloc(((i + 1) *
sizeof(TVPlaneta)));
        } else {
            sistemap->planeta = (TVPlaneta *) realloc(sistemap->planeta,
((i + 1) * sizeof(TVPlaneta)));
        }
        sistemap->planeta[i].nome = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }

    if (strcmp(estrutura, "satelite") == 0) {
        if (sistemap->planeta[i].satelite == NULL) {
            sistemap->planeta[i].satelite = (TVSatelite *) malloc(((j + 1)
* sizeof(TVSatelite)));
        } else {
            sistemap->planeta[i].satelite = (TVSatelite *)
realloc(sistemap->planeta[i].satelite,
                                                                    ((j + 1)
* sizeof(TVSatelite)));
        }
        sistemap->planeta[i].satelite[j].nome = (char *)
malloc((strlen(str) + 1) * sizeof(char));
    }

    if (strcmp(estrutura, "viagem") == 0) {
        if (viagem->rota == NULL) {
            viagem->rota = (TRota *) malloc(((i + 1) * sizeof(TRota)));
        } else {
            viagem->rota = (TRota *) realloc(viagem->rota, ((i + 1) *
sizeof(TRota)));
        }
        viagem->rota[i].astro = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    }
}

void SPLeEstrela(TSPlanetario *sistemap) {
    char str[MAXSTR], provisorio[MAXSTR];

    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ,]", str);
    AlocaMem(sistemap, 0, (int) 0, (int) 0, str, "estrela");

    if (sistemap->estrela != NULL) {
        strcpy(sistemap->estrela, str);
    }
}
```

```
void SPLePlaneta(TSPlanetario *sistemap, int i, char *provisorio) {
    char str[MAXSTR];
    float distancia;

    *str = '\0';
    distancia = 0;
    sscanf(provisorio, "%[^\n, ,] %f", str, &distancia);
    AlocaMem(sistemap, 0, i, (int) 0, str, "planeta");

    if (sistemap->planeta != NULL && sistemap->planeta[i].nome != NULL) {
        strcpy(sistemap->planeta[i].nome, str);
        sistemap->planeta[i].distancia = distancia;
    }
}

void SPLeSatelite(TSPlanetario *sistemap, int i, int j, char *provisorio) {
    char strs[MAXSTR];
    float distancia;

    *strs = '\0';
    distancia = 0;
    sscanf(provisorio, "%*c %[^\n, ] %f", strs, &distancia);
    AlocaMem(sistemap, 0, i, j, strs, "satelite");

    if (sistemap->planeta[i].satelite != NULL && sistemap-
>planeta[i].satelite[j].nome != NULL) {
        strcpy(sistemap->planeta[i].satelite[j].nome, strs);
        sistemap->planeta[i].satelite[j].distancia = distancia;
    }
}

void SPLeSistema(TSPlanetario *sistemap) {
    char provisorio[MAXSTR];
    int iplaneta, isatelite;

    iplaneta = 0;
    do {
        fgets(provisorio, MAXSTR, stdin);
        if (provisorio[0] == '#') {
            SPLeSatelite(sistemap, iplaneta - 1, isatelite, provisorio);
            SPLeSatelite(sistemap, iplaneta - 1, isatelite + 1, "");
            isatelite++;
        } else {
            isatelite = 0;
            SPLePlaneta(sistemap, iplaneta, provisorio);
            sistemap->planeta[iplaneta].satelite = NULL;
            SPLeSatelite(sistemap, iplaneta, isatelite, "");
            iplaneta++;
        }
    } while ((strcmp(sistemap->planeta[iplaneta - 1].nome, "\0") != 0));
}

int ContaAstros(TSPlanetario *sistemap, TViagem *viagem, int i, char
*estrutura) {
    int j, conta;

    conta = 0;
    if (strcmp(estrutura, "planeta") == 0) {
        while ((strcmp(sistemap->planeta[i].nome, "\0") != 0)) {
            i++;
```

```
            conta++;
        }
    }

    if (strcmp(estrutura, "satelite") == 0) {
        j = 0;
        while ((strcmp(sistemap->planeta[i].satelite[j].nome, "\0") != 0))
{
            j++;
            conta++;
        }
    }

    if (strcmp(estrutura, "viagem") == 0) {
        while ((strcmp(viagem->rota[i].astro, "\0") != 0)) {
            i++;
            conta++;
        }
    }
    return conta;
}

void VProcessaOrigemA(TViagem *viagem, char *str, int i, int j) {
    viagem->origem.astro = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    strcpy(viagem->origem.astro, str);
    viagem->origem.indexi = i;
    viagem->origem.indexj = j;
}

void VProcessaOrigem(TSPlanetario *sistemap, TViagem *viagem, char *origem)
{
    int i, j, tplaneta, tsatelite;

    VProcessaOrigemA(viagem, "estrela", 0, 0);

    tplaneta = ContaAstros(sistemap, 0, 0, "planeta");
    for (i = 0; i < tplaneta; i++) {
        if (strcmp(sistemap->planeta[i].nome, origem) == 0) {
            VProcessaOrigemA(viagem, "planeta", i, 0);
        } else {
            tsatelite = ContaAstros(sistemap, 0, i, "satelite");
            for (j = 0; j < tsatelite; j++) {
                if (strcmp(sistemap->planeta[i].satelite[j].nome, origem)
== 0) {
                    VProcessaOrigemA(viagem, "satelite", i, j);
                }
            }
        }
    }
}

void VProcessaDestinoA(TViagem *viagem, char *str, int i, int j) {
    viagem->destino.astro = (char *) malloc((strlen(str) + 1) *
sizeof(char));
    strcpy(viagem->destino.astro, str);
    viagem->destino.indexi = i;
    viagem->destino.indexj = j;
}

void VProcessaDestino(TSPlanetario *sistemap, TViagem *viagem, char
```

```
*destino) {
    int i, j, tplaneta, tsatelite;

    VProcessaDestinoA(viagem, "estrela", 0, 0);
    tplaneta = ContaAstros(sistemap, 0, 0, "planeta");
    for (i = 0; i < tplaneta; i++) {
        if (strcmp(sistemap->planeta[i].nome, destino) == 0) {
            VProcessaDestinoA(viagem, "planeta", i, 0);
        } else {
            tsatelite = ContaAstros(sistemap, 0, i, "satelite");
            for (j = 0; j < tsatelite; j++) {
                if (strcmp(sistemap->planeta[i].satelite[j].nome, destino)
== 0) {
                    VProcessaDestinoA(viagem, "satelite", i, j);
                }
            }
        }
    }
}

void VProcessaRotaOrigemA(TSPlanetario *sistemap, TViagem *viagem, char
*str) {
    if (viagem->origem.origem == NULL) {
        viagem->origem.origem = (char *) malloc((strlen(sistemap->estrela)
+ 1) * sizeof(char));
        strcpy(viagem->origem.origem, sistemap->estrela);
    } else {
        viagem->origem.origem = (char *) realloc(viagem->origem.origem,
                                                 ((strlen(viagem-
>origem.origem) + strlen(str)) + 1) * sizeof(char));
        strcat(viagem->origem.origem, str);
    }
}

int VProcessaRotaOrigemP(TSPlanetario *sistemap, TViagem *viagem, int
irota) {
    int auxplaneta;
    AlocaMem(0, viagem, irota, 0, sistemap->planeta[viagem-
>origem.indexi].nome, "viagem");
    strcpy(viagem->rota[irota].astro, sistemap->planeta[viagem-
>origem.indexi].nome);
    viagem->rota[irota].distancia = ceil(((sistemap->planeta[viagem-
>origem.indexi].distancia) * 30));
    viagem->distancia += viagem->rota[irota].distancia;
    VProcessaRotaOrigemA(sistemap, viagem, sistemap->planeta[viagem-
>origem.indexi].nome);
    irota++;
    auxplaneta = irota;
    return auxplaneta;
}

int VProcessaRotaOrigemS(TSPlanetario *sistemap, TViagem *viagem, int
irota) {
    int auxsatelite;

    AlocaMem(0, viagem, irota, 0, sistemap->planeta[viagem-
>origem.indexi].satelite[viagem->origem.indexj].nome,
             "viagem");
    strcpy(viagem->rota[irota].astro, sistemap->planeta[viagem-
>origem.indexi].satelite[viagem->origem.indexj].nome);
    viagem->rota[irota].distancia = ceil(
```

```
            ((sistemap->planeta[viagem->origem.indexi].satelite[viagem-
>origem.indexj].distancia) * 30));
    viagem->distancia += viagem->rota[irota].distancia;
    irota++;

    auxsatelite = VProcessaRotaOrigemP(sistemap, viagem, irota);
    VProcessaRotaOrigemA(sistemap, viagem, "-");
    VProcessaRotaOrigemA(sistemap, viagem,
                         sistemap->planeta[viagem-
>origem.indexi].satelite[viagem->origem.indexj].nome);
    return auxsatelite;
}

int VProcessaRotaOrigem(TSPlanetario *sistemap, TViagem *viagem) {
    int irota;
    irota = 0;

    viagem->distancia = 0;
    VProcessaRotaOrigemA(sistemap, viagem, sistemap->estrela);
    VProcessaRotaOrigemA(sistemap, viagem, "-");

    if (strcmp(viagem->origem.astro, "satelite") == 0) {
        irota = VProcessaRotaOrigemS(sistemap, viagem, irota);
    }

    if (strcmp(viagem->origem.astro, "planeta") == 0) {
        irota = VProcessaRotaOrigemP(sistemap, viagem, irota);
    }

    return irota;
}

void VProcessaRotaDestinoA(TSPlanetario *sistemap, TViagem *viagem, char
*str) {
    if (viagem->destino.destino == NULL) {
        viagem->destino.destino = (char *) malloc((strlen(sistemap-
>estrela) + 1) * sizeof(char));
        strcpy(viagem->destino.destino, sistemap->estrela);
    } else {
        viagem->destino.destino = (char *) realloc(viagem->destino.destino,
                                                   ((strlen(viagem-
>destino.destino) + strlen(str)) + 1) *
                                                   sizeof(char));
        strcat(viagem->destino.destino, str);
    }
}

int VProcessaRotaDestinoP(TSPlanetario *sistemap, TViagem *viagem, int
irota) {
    int auxplaneta;
    AlocaMem(0, viagem, irota, 0, sistemap->estrela, "viagem");
    strcpy(viagem->rota[irota].astro, sistemap->estrela);
    viagem->rota[irota].distancia = ceil(((sistemap->planeta[viagem-
>destino.indexi].distancia) * 30));
    viagem->distancia += viagem->rota[irota].distancia;
    irota++;

    AlocaMem(0, viagem, irota, 0, sistemap->planeta[viagem-
>destino.indexi].nome, "viagem");
    strcpy(viagem->rota[irota].astro, sistemap->planeta[viagem-
>destino.indexi].nome);
```

```
    viagem->rota[irota].distancia = 0;
    irota++;
    auxplaneta = irota;
    return auxplaneta;
}

int VProcessaRotaDestinoS(TSPlanetario *sistemap, TViagem *viagem, int
irota) {
    int auxsatelite;

    auxsatelite = irota;
    if (viagem->origem.indexi != viagem->destino.indexi) {
        auxsatelite = VProcessaRotaDestinoP(sistemap, viagem, irota);
    }

    AlocaMem(0, viagem, auxsatelite, 0, sistemap->planeta[viagem-
>destino.indexi].satelite[viagem->destino.indexj].nome,
             "viagem");
    strcpy(viagem->rota[auxsatelite].astro,
           sistemap->planeta[viagem->destino.indexi].satelite[viagem-
>destino.indexj].nome);
    viagem->distancia -= viagem->rota[auxsatelite - 1].distancia;
    viagem->rota[auxsatelite - 1].distancia = ceil(
            ((sistemap->planeta[viagem->destino.indexi].satelite[viagem-
>destino.indexj].distancia) * 30));
    viagem->rota[auxsatelite].distancia = 0;
    viagem->distancia += viagem->rota[auxsatelite - 1].distancia;
    auxsatelite++;

    VProcessaRotaDestinoA(sistemap, viagem, sistemap->planeta[viagem-
>destino.indexi].nome);
    VProcessaRotaDestinoA(sistemap, viagem, "-");
    VProcessaRotaDestinoA(sistemap, viagem,
                          sistemap->planeta[viagem-
>destino.indexi].satelite[viagem->destino.indexj].nome);
    return auxsatelite;
}

void VProcessaRotaDestino(TSPlanetario *sistemap, TViagem *viagem, int
irota) {
    VProcessaRotaDestinoA(sistemap, viagem, sistemap->estrela);
    VProcessaRotaDestinoA(sistemap, viagem, "-");

    if (strcmp(viagem->destino.astro, "satelite") == 0) {
        irota = VProcessaRotaDestinoS(sistemap, viagem, irota);
    }

    if (strcmp(viagem->destino.astro, "planeta") == 0) {
        irota = VProcessaRotaDestinoP(sistemap, viagem, irota);
        VProcessaRotaDestinoA(sistemap, viagem, sistemap->planeta[viagem-
>destino.indexi].nome);
    }
    AlocaMem(0, viagem, irota, 0, "\0", "viagem");
    strcpy(viagem->rota[irota].astro, "\0");
}

void VProcessaRota(TSPlanetario *sistemap, TViagem *viagem) {
    int irota;
    irota = VProcessaRotaOrigem(sistemap, viagem);
    VProcessaRotaDestino(sistemap, viagem, irota);
}
```

```c
void VProcessaEstacoes(TSPlanetario *sistemap, TViagem *viagem) {
    char provisorio[MAXSTR], origem[MAXSTR], destino[MAXSTR];
    fgets(provisorio, MAXSTR, stdin);
    sscanf(provisorio, "%[^\n, ] %[^\n, ]", origem, destino);
    VProcessaOrigem(sistemap, viagem, origem);
    VProcessaDestino(sistemap, viagem, destino);
    VProcessaRota(sistemap, viagem);

}


void SPLibertar(TSPlanetario *sistemap, TViagem *viagem) {
    int i, j, tplaneta, tsatelite, trota;

    if (sistemap->estrela != NULL) {
        free(sistemap->estrela);
        sistemap->estrela = NULL;
    }

    tplaneta = ContaAstros(sistemap, 0, 0, "planeta");
    for (i = 0; i <= tplaneta; i++) {
        tsatelite = ContaAstros(sistemap, 0, i, "satelite");
        for (j = 0; j <= tsatelite; j++) {
            if (sistemap->planeta[i].satelite[j].nome != NULL) {
                free(sistemap->planeta[i].satelite[j].nome);
                sistemap->planeta[i].satelite[j].nome = NULL;
                sistemap->planeta[i].satelite[j].distancia = 0;
            }
        }

        if (sistemap->planeta[i].satelite != NULL) {
            free(sistemap->planeta[i].satelite);
            sistemap->planeta[i].satelite = NULL;
        }

        if (sistemap->planeta[i].nome != NULL) {
            free(sistemap->planeta[i].nome);
            sistemap->planeta[i].nome = NULL;
            sistemap->planeta[i].distancia = 0;
        }
    }

    if (sistemap->planeta != NULL) {
        free(sistemap->planeta);
        sistemap->planeta = NULL;
    }

    trota = ContaAstros(0, viagem, 0, "viagem");
    for (i = 0; i <= trota; i++) {
        if (viagem->rota[i].astro != NULL) {
            free(viagem->rota[i].astro);
            viagem->rota[i].astro = NULL;
            viagem->rota[i].distancia = 0;
        }
    }

    if (viagem->origem.astro != NULL) {
        free(viagem->origem.astro);
        free(viagem->origem.origem);
        viagem->origem.astro = NULL;
```

```
        viagem->origem.origem = NULL;
        viagem->origem.indexi = 0;
        viagem->origem.indexj = 0;
    }

    if (viagem->destino.astro != NULL) {
        free(viagem->destino.astro);
        free(viagem->destino.destino);
        viagem->destino.astro = NULL;
        viagem->destino.destino = NULL;
        viagem->destino.indexi = 0;
        viagem->destino.indexj = 0;
    }
}

TSPlanetario ProcessaSistema() {
    TSPlanetario sistemap;

    sistemap = SPCriar();
    SPLeEstrela(&sistemap);
    SPLeSistema(&sistemap);
    return sistemap;
}

TViagem ProcessaViagem(TSPlanetario *sistemap) {
    TViagem viagem;

    viagem = VCriar();
    VProcessaEstacoes(sistemap, &viagem);
    return viagem;
}

void MostraSistema(TSPlanetario *sistemap, TViagem *viagem) {
    int i, trota, auxdistancia, auxdias, dias[] = {360, 30, 1, 0};
    char *singular[] = {"ano", "mês", "dia"}, *plural[] = {"anos", "meses",
"dias"};
    auxdistancia = (int) viagem->distancia + 2;
    trota = ContaAstros(0, viagem, 0, "viagem");

    printf("De: %s\n", viagem->origem.origem);
    printf("Para: %s\n", viagem->destino.destino);
    printf("Estações:");
    for (i = 0; i < trota; i++) {
        printf(" %s", viagem->rota[i].astro);
        if (viagem->rota[i].distancia > 0) {
            printf(" (%d)", (int) viagem->rota[i].distancia);
        }
    }

    printf("\nDuração: ");
    for (i = 0; dias[i] > 0; i++) {
        auxdias = (int) (auxdistancia / dias[i]);
        auxdistancia -= dias[i] * auxdias;
        if (auxdias > 0 && auxdias <= 1) {
            printf(" %d %s", auxdias, singular[i]);
        }
        if (auxdias > 1) {
            printf(" %d %s", auxdias, plural[i]);
        }
    }
}
```

```
void main() {
    TSPlanetario sistemap;
    TViagem viagem;
    sistemap = ProcessaSistema();
    viagem = ProcessaViagem(&sistemap);
    MostraSistema(&sistemap, &viagem);
    SPLibertar(&sistemap, &viagem);
    exit(0);
}
```

Testes à alínea A:

Pre-check do VPL

Testes com valores introduzidos manualmente, que comprovam que o programa faz o expectável

```
"C:\Users\dsust\Programas em C\220148A\cmake-build
Zeus
America
Alentejo

Zeus, sistema planetário com 2 planetas
Process finished with exit code 0
```

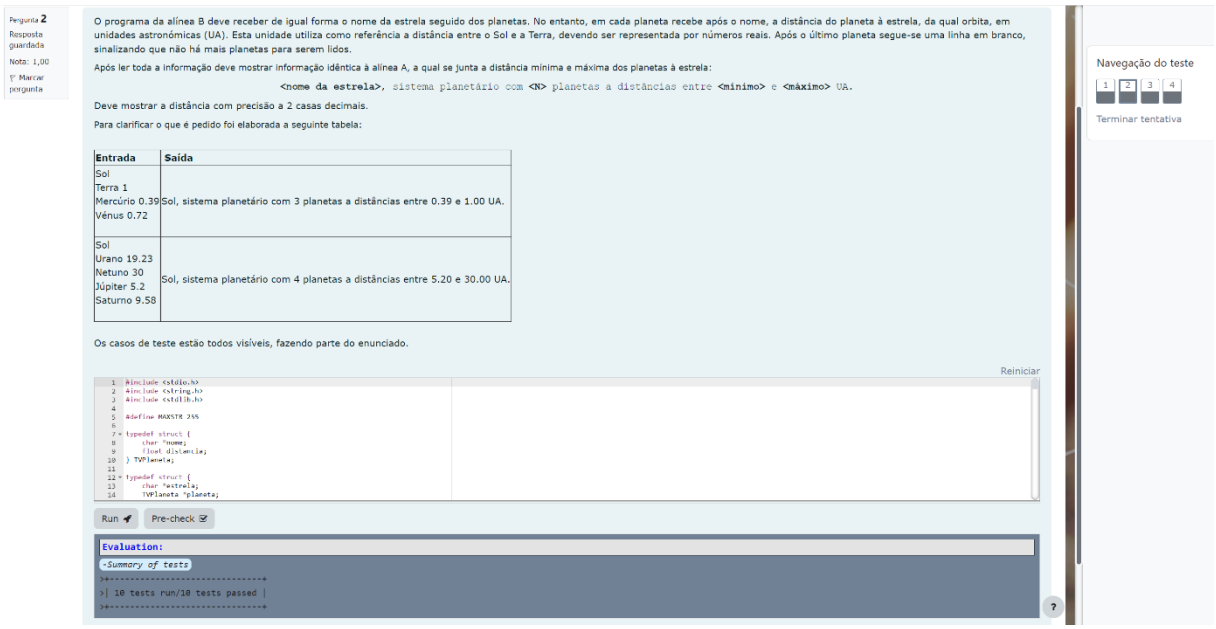
Foi inserido 1 estrela e 2 planetas no primeiro teste e 1 estrela e 4 planetas no segundo teste, o output corresponde.

```
"C:\Users\dsust\Programas em C\220148A\cmake-build-debug\220148A.exe"
Sol
Terra
Marte
Venus
Sagitario

Sol, sistema planetário com 4 planetas
Process finished with exit code 0
```

Testes à alínea B:

Pre-check do VPL



Pergunta **2**
Resposta guardada
Nota: 1,00
Marcar pergunta

O programa da alínea B deve receber de igual forma o nome da estrela seguido dos planetas. No entanto, em cada planeta recebe após o nome, a distância do planeta à estrela, da qual orbita, em unidades astronómicas (UA). Esta unidade utiliza como referência a distância entre o Sol e a Terra, devendo ser representada por números reais. Após o último planeta segue-se uma linha em branco, sinalizando que não há mais planetas para serem lidos.

Após ler toda a informação deve mostrar informação idêntica à alínea A, a qual se junta a distância mínima e máxima dos planetas à estrela:

<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.

Para clarificar o que é pedido foi elaborada a seguinte tabela:

Entrada	Saída
Sol Terra 1 Mercúrio 0.39 Vénus 0.72	Sol, sistema planetário com 3 planetas a distâncias entre 0.39 e 1.00 UA.
Sol Urano 19.23 Netuno 30 Júpiter 5.2 Saturno 9.58	Sol, sistema planetário com 4 planetas a distâncias entre 5.20 e 30.00 UA.

Os casos de teste estão todos visíveis, fazendo parte do enunciado.

Reiniciar

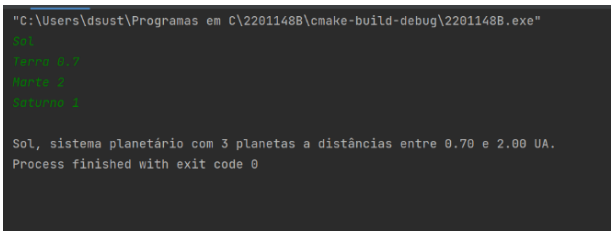
```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #define MAXSTR 255
6
7  typedef struct {
8      char *nome;
9      float distancia;
10 } TVPlaneta;
11
12 typedef struct {
13     char *estrela;
14     TVPlaneta *planeta;
```

Run Pre-check

```
Evaluation:
-Summary of tests
>+------------------------------+
>| 10 tests run/10 tests passed |
>+------------------------------+
```

Navegação do teste
1 2 3 4
Terminar tentativa

Testes com valores introduzidos manualmente, que comprovam que o programa faz o expectável



```
"C:\Users\dsust\Programas em C\2201148B\cmake-build-debug\2201148B.exe"
Sol
Terra 0.7
Marte 2
Saturno 1

Sol, sistema planetário com 3 planetas a distâncias entre 0.70 e 2.00 UA.
Process finished with exit code 0
```

Foram inseridos 1 estrela e 3 planetas, com distancias diferentes e desordenados, e verifica-se que o output corresponde, tanto no primeiro como no segundo caso.

```
"C:\Users\dsust\Programas em C\2201148B\cmake-build-debug\2201148B.exe"
Kalius
Vlaus 6
Galius 3
Helius 0.3

Kalius, sistema planetário com 3 planetas a distâncias entre 0.30 e 6.00 UA.
Process finished with exit code 0
```

Testes à alínea C:

Pre-check do VPL

Testes com valores introduzidos manualmente, que comprovam que o programa faz o expectável

```
"C:\Users\dsust\Programas em C\2201148C\cmake-build-debug\2201148C.exe"
Sol
Terra 0.7
Marte 0.3
Saturno 5
# Kalius 1.4
# Kaleidoscopius 0.6

2 e 0.7 3
Sol:
- Terra 0.70

Process finished with exit code 0
```

Foram inseridos 1 estrela e vários planetas e satélites, com distancias diferentes e desordenadas nos 3 testes de exemplo, e o output corresponde aos filtros e ordenação pedidos nos 3 testes.

```
"C:\Users\dsust\Programas em C\2201148C\cmake-build-debug\2201148C.exe"
Los
Arret 1
# Aul 0.32
Sunev 4
# Klops 0.23
# Dibala 0.45

3 * -1 -1
Los:
- Arret 1.00
--- Aul 0.3200
- Sunev 4.00
--- Klops 0.2300
--- Dibala 0.4500

Process finished with exit code 0
```

```
"C:\Users\dsust\Programas em C\2201148C\cmake-build-debug\2201148C.exe"
Sol
Terra 1
# Luo 0.8
Klaus 4
# CR7 0.2
# Mbappe 0.3
Plutao 9
# Messi 0.1

1 s 0.2 8
Sol:
- Klaus 4.00

Process finished with exit code 0
```

Testes à alínea D:

Pre-check do VPL

Testes com valores introduzidos manualmente, que comprovam que o programa faz o expectável

```
"C:\Users\dsust\Programas em C\2201148D\cmake-build-debug\2201148D.exe"
Sol
Terra 4
# Lua 0.2
Klaus 2
# Maradona 0.2
# Pele 0.5
Barcelona 2
# Figo 0.2

Figo Lua
De: Sol-Barcelona-Figo
Para: Sol-Terra-Lua
Estações: Figo (6) Barcelona (60) Sol (120) Terra (6) Lua
Duração:  6 meses 14 dias
Process finished with exit code 0
```

Foram inseridos diferentes planetas e satélites com diversas distancias distintas, e escolhidos origens e destinos diferentes, e o output sempre correspondeu ao espectado.

```
"C:\Users\dsust\Programas em C\2201148D\cmake-build-debug\2201148D.exe"
FML
Amazonas 3
# Lopus 0.6
# Plaus 0.3
Asia 8
# Rosa
# Mar

Mar Lopus
De: FML-Asia-Mar
Para: FML-Amazonas-Lopus
Estações: Mar Asia (240) FML (90) Amazonas (18) Lopus
Duração:  11 meses 20 dias
Process finished with exit code 0
```

```
"C:\Users\dsust\Programas em C\2201148D\cmake-build-debug\2201148D.exe"
Champions
Chelsea 1
# Mourinho 0.3
# JJ 0.5
Maritimo 4
# Coentrao 0.2

JJ Mourinho
De: Champions-Chelsea-JJ
Para: Champions-Chelsea-Mourinho
Estações: JJ (15) Chelsea (9) Mourinho
Duração:  26 dias
Process finished with exit code 0
```

```
"C:\Users\dsust\Programas em C\2201148D\cmake-build-debug\2201148D.exe"
Sol
Terra 1
# Lua 0.3
Marte 5
# Klaus 0.3
# Glis 0.7

Glis Terra
De: Sol-Marte-Glis
Para: Sol-Terra
Estações: Glis (21) Marte (150) Sol (30) Terra
Duração:  6 meses 23 dias
Process finished with exit code 0
```