



SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV  
CAMPUS FLORESTAL

## **Trabalho 1 - AEDS 1**

**Sistema de gerenciamento de processos utilizando lista de  
cursos**

Pedro Paulo Paz - 5937

Bruno Vicentini Ribeiro - 5907

Vitor Mathias Andrade - 5901

Florestal - MG

2024

# Sumário

<b>1. Introdução</b>	<b>2</b>
<b>2. Organização</b>	<b>2</b>
<b>3. Desenvolvimento</b>	<b>3</b>
3.1 Categorização da rocha mineral	3
3.2 TAD Compartimento	3
3.3 Main	4
3.4 Operação R	5
3.5 Operação I	5
3.6 Operação E	6
<b>4. Compilação e Execução</b>	<b>6</b>
<b>5. Resultados</b>	<b>6</b>
<b>6. Conclusão</b>	<b>9</b>
<b>7. Referências</b>	<b>10</b>

## 1. Introdução

Este trabalho tem como objetivo o desenvolvimento de um sistema de controle de sondas espaciais e de catalogação de rochas minerais encontradas em Marte. Para isso, foram utilizadas a linguagem de programação C e o método de Programação Dinâmica — técnica de construção de sistemas baseada no fracionamento de um problema principal em outros menores —, por meio de Tipos Abstratos de Dados (TADs), os quais foram implementados com auxílio de listas lineares, para a organização do código e a resolução de possíveis problemas.

Em síntese, o sistema visa controlar as sondas enviadas para Marte, verificando o peso presente no compartimento de cada sonda disparada e as rochas coletadas, além de analisar a localização de cada sonda por meio de suas coordenadas.

## 2. Organização

Na figura 1, observa-se o panorama geral da organização do projeto, no qual são elencados os TADs, bem como as estruturas de dados presentes, sendo utilizada a de listas encadeadas.

Além disso, dividiram-se os arquivos em .h (responsável pela declaração das funções e pela criação das structs) e .c (responsável pela implementação das funções previamente declaradas e pela utilização das structs), para garantir uma melhor organização e funcionamento do código.

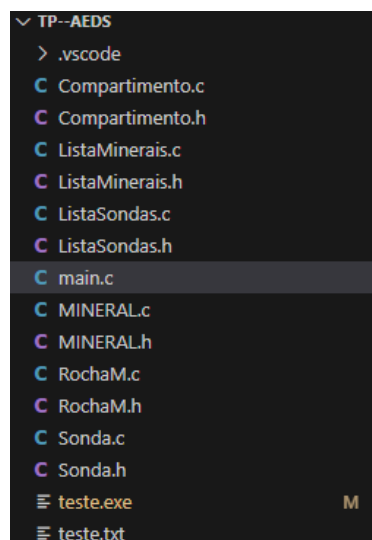


Figura 1: Repositório do projeto.

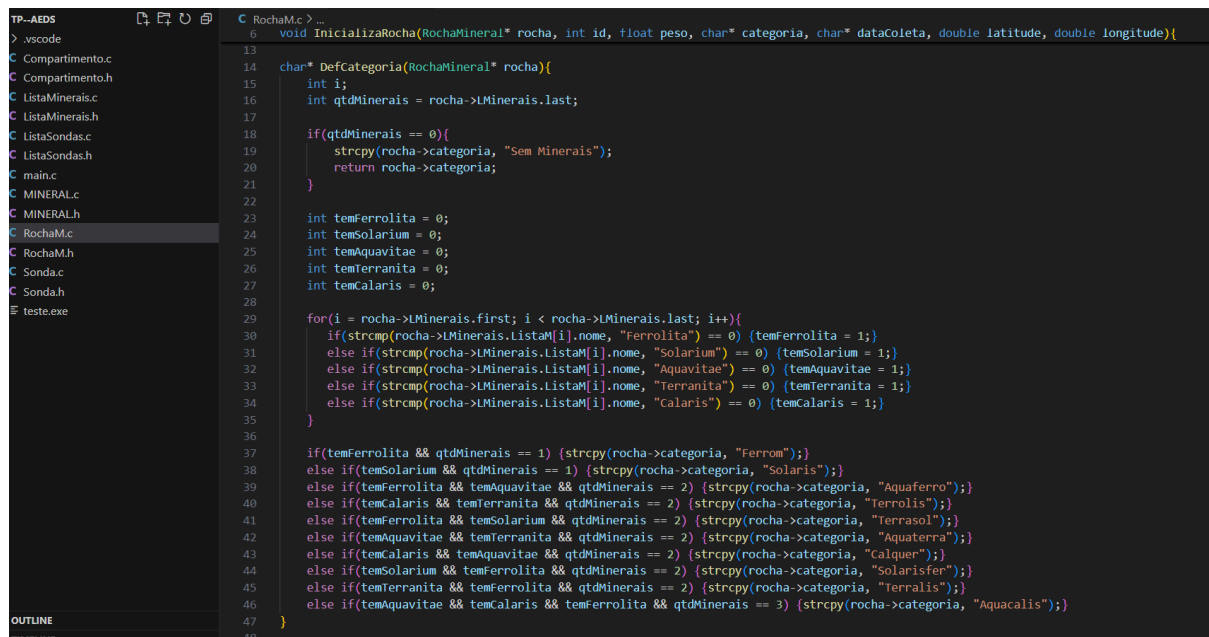
## 3. Desenvolvimento

A utilização dos TADs foi essencial para a realização do trabalho, garantindo o correto funcionamento do código. Tendo isso em mente, destacam-se as principais implementações do sistema desenvolvido: (3.1) Categorização da rocha mineral, (3.2) TAD Compartimento e (3.3) Main.

### 3.1 Categorização da rocha mineral

A partir da figura 2, visualiza-se a função DefCategoria, localizada no TAD RochaM.c. Essa função é responsável por analisar os minerais presentes em cada rocha e, em seguida, categorizá-la. Para isso, tendo acesso à ListaMinerais.c, bem como ao seu tamanho, a DefCategoria atribui ao último uma variável e, a partir disso, define qual o tipo de rocha captada.

A importância dessa função está em sua necessidade prévia para a execução de outras operações, já que, para identificar quais rochas estão presentes na sonda e substituir uma rocha por outra de menor peso, por exemplo, é preciso conhecer a sua classificação. Assim, essa função é fundamental para o correto desenvolvimento de etapas subsequentes.



```

13 void InicializaRocha(RochaMineral* rocha, int id, float peso, char* categoria, char* dataColeta, double latitude, double longitude){
14
15     char* DefCategoria(RochaMineral* rocha){
16         int i;
17         int qtdMinerais = rocha->LMinerais.last;
18
19         if(qtdMinerais == 0){
20             strcpy(rocha->categoria, "Sem Minerais");
21             return rocha->categoria;
22         }
23
24         int temFerrolita = 0;
25         int temSolarium = 0;
26         int temAquavitae = 0;
27         int temTerrarita = 0;
28         int temCalaris = 0;
29
30         for(i = rocha->LMinerais.first; i < rocha->LMinerais.last; i++){
31             if(strcmp(rocha->LMinerais.Lista[i].nome, "Ferrolita") == 0) {temFerrolita = 1;}
32             else if(strcmp(rocha->LMinerais.Lista[i].nome, "Solarium") == 0) {temSolarium = 1;}
33             else if(strcmp(rocha->LMinerais.Lista[i].nome, "Aquavitae") == 0) {temAquavitae = 1;}
34             else if(strcmp(rocha->LMinerais.Lista[i].nome, "Terrarita") == 0) {temTerrarita = 1;}
35             else if(strcmp(rocha->LMinerais.Lista[i].nome, "Calaris") == 0) {temCalaris = 1;}
36         }
37
38         if(temFerrolita && qtdMinerais == 1) {strcpy(rocha->categoria, "Ferrom");}
39         else if(temSolarium && qtdMinerais == 1) {strcpy(rocha->categoria, "Solaris");}
40         else if(temFerrolita && temAquavitae && qtdMinerais == 2) {strcpy(rocha->categoria, "Aquaferro");}
41         else if(temCalaris && temTerrarita && qtdMinerais == 2) {strcpy(rocha->categoria, "Terralis");}
42         else if(temFerrolita && temSolarium && qtdMinerais == 2) {strcpy(rocha->categoria, "Terrasol");}
43         else if(temAquavitae && temTerrarita && qtdMinerais == 2) {strcpy(rocha->categoria, "Aquaterra");}
44         else if(temCalaris && temAquavitae && qtdMinerais == 2) {strcpy(rocha->categoria, "Calquer");}
45         else if(temSolarium && temFerrolita && qtdMinerais == 2) {strcpy(rocha->categoria, "Solarisfer");}
46         else if(temTerrarita && temFerrolita && qtdMinerais == 2) {strcpy(rocha->categoria, "Terralis");}
47         else if(temAquavitae && temCalaris && temFerrolita && qtdMinerais == 3) {strcpy(rocha->categoria, "Aquacalis");}
48     }
49 }

```

Figura 2: Função DefCategoria.

### 3.2 TAD Compartimento

O TAD Compartimento possui a função de armazenar as informações acerca das rochas minerais, sendo de extrema importância para a substituição de uma rocha por outra de menor peso e de mesma categoria, por meio da função LTrocaR. Essa funcionalidade foi crucial, à medida que garantiu que o peso máximo suportado por cada sonda não fosse ultrapassado e que o peso total carregado por cada dispositivo de exploração fosse o menor possível, para carregar o maior número de amostras de rochas.

```

89
90 void LTrocaR(RCompartmento* rLista) {
91     if (LEhVazia(rLista) || rLista->pProx == NULL) {return;}
92
93     ApontadorRocha pAux = rLista->pPrimeiro->pProx;
94     ApontadorRocha maisPesada = NULL;
95     float maiorPeso = 0;
96
97     while (pAux != NULL) {
98         if (pAux->rocha.peso > maiorPeso) {
99             maiorPeso = pAux->rocha.peso;
100            maisPesada = pAux;
101        }
102        pAux = pAux->pProx;
103    }
104
105    if (maisPesada == NULL) {return;}
106
107    char categoriaMaisPesada[40];
108    strcpy(categoriaMaisPesada, maisPesada->rocha.categoria);
109
110    ApontadorRocha maisLeve = NULL;
111    ApontadorRocha anteriorMaisLeve = NULL;
112    float menorPeso = -1;
113
114    pAux = rLista->pPrimeiro->pProx;
115    ApontadorRocha pAnt = rLista->pPrimeiro;
116    while (pAux != NULL) {
117        if (strcmp(pAux->rocha.categoria, categoriaMaisPesada) == 0) {
118            if (menorPeso == -1 || pAux->rocha.peso < menorPeso) {
119                menorPeso = pAux->rocha.peso;
120                maisLeve = pAux;
121                anteriorMaisLeve = pAnt;
122            }
123        }
124        pAnt = pAux;
125        pAux = pAux->pProx;

```

Figura 3: Função LTrocaR.

### 3.3 - Main

A função Main é responsável pela comunicação entre o programa e o usuário, coletando dados e solicitando funções existentes no primeiro, conforme as requisições do último. Em resumo, a sua importância está na coordenação de chamadas de outras funções e na garantia de que as interações ocorram de forma eficiente.

```

int main()
{
    int Escolha = 0;
    while (Escolha != 1 && Escolha != 2)
    {
        printf("Arquivo de entrada (1 - Terminal, 2 - Arquivo): \n");
        scanf("%d", &Escolha);
    }

    if (Escolha == 1)
    {
        int N_Sondas;
        double lat_i, long_i;
        float c_i, v_i, nc_i;
        Slista ListaSonda;
        RochaMineral rocha;
        Mineral minel;
        Sonda sondai;

        FlVaziaSonda(&ListaSonda);

        printf("Digite o numero de sondas: ");
        scanf("%d", &N_Sondas);
        printf("Digite os dados das sondas: (latitude, longitude, capacidade maxima, velocidade e combustivel)\n");
        for (int i = 0; i < N_Sondas; i++)
        {
            scanf("%lf %lf %f %f %f", &lat_i, &long_i, &c_i, &v_i, &nc_i);
            InicializaSonda(&sondai, (i + 1), lat_i, long_i, c_i, "Sim");
            LinsereSonda(&ListaSonda, &sondai);
        }

        int N_op;
        printf("Digite o numero de operacoes: ");
        scanf("%d", &N_op);
        for (int i = 0; i < N_op; i++)
        {
            char operacao;
            printf("Digite a operacao: ");
            scanf(" %c", &operacao);
            switch (operacao) {
                case 'R': {
                    double lat_r, long_r;
                    float p_r;
                    char cat_r[20];
                    char minerais_str[100];

                    printf("\nDigite a latitude, longitude, peso e ate 3 minerais (SEPARAR POR ESPACO):\n");
                    scanf("%lf %lf %f", &lat_r, &long_r, &p_r);
                    getchar();

```

Figura 4: Função Main.

### 3.4 - Operação R

A operação R consiste em indicar uma nova rocha e armazená-la na sonda mais próxima, com base na comparação de coordenadas. Isso é feito por meio da coleta das seguintes informações sobre a rocha: latitude, longitude, peso e minerais constituintes.

```
case 'R': {
    double lat_r, long_r;
    float p_r;
    char minerais_str[100];

    printf("\nDigite a latitude, longitude, peso e ate 3 minerais (SEPARE POR ESPACO):\n");
    scanf("%lf %lf %f", &lat_r, &long_r, &p_r);
    getchar();
    fgets(minerais_str, sizeof(minerais_str), stdin);
    minerais_str[strcspn(minerais_str, "\n")] = '\0';

    const char delim[2] = " ";
    char *buffer = strtok(minerais_str, delim);

    FLVaziaMine(&rocha.LMinerais);

    while (buffer != NULL)
    {
        RetornaMineral(&min1, buffer);
        LInserirMine(&rocha.LMinerais, min1);
        buffer = strtok(NULL, delim);
    }

    int cont = 0;

    InicializaRocha(&rocha, ++cont, p_r, DefCategoria(&rocha), "", lat_r, long_r);

    InsereRochaS(&ListaSonda, &rocha);

    break;
}
```

Figura 5: Implementação da Operação R.

### 3.5 - Operação I

A operação I tem como objetivo principal imprimir o identificador de cada sonda, as rochas coletadas e seus respectivos pesos. Caso uma sonda não tenha coletado nenhuma rocha, será exibida a mensagem “Compartimento vazio!”.

```
void OperacaoI(SLista *ListaSonda) {
    ApontadorSonda pAux = ListaSonda->pPrimeiro->pProx;

    while (pAux != NULL)
    {
        Sonda *pSonda = &pAux->sonda;
        printf("%d\n", pSonda->id);

        if (LEhVazia(&pSonda->cRocha))
        {
            printf("Compartimento vazio!\n");
            /* Se o compartimento estiver vazio, imprime o id da sonda e
            'compartimento vazio!' */
        }
        else
        {
            ApontadorRocha rAux = pSonda->cRocha.pPrimeiro->pProx;

            while (rAux != NULL)
            {
                printf("%s %.2f\n", rAux->rocha.categoria, rAux->rocha.peso);
                rAux = rAux->pProx;
            }
        }
        pAux = pAux->pProx;
    }
}
```

Figura 6: Implementação da Operação I.

### 3.6 - Operação E

A Operação E tem como função redistribuir todas as rochas coletadas entre as sondas, equilibrando o peso carregado por cada uma. Para isso, todas as sondas são movidas para as coordenadas (0,0), as rochas são armazenadas em uma lista temporária e, posteriormente, redistribuídas.

```
void OperacaoE(SLista *ListaS) {
    int numRochas = 0;
    RochaMineral *rochas = extraiRochas(ListaS, &numRochas);

    if (rochas != NULL) {
        distribuirRochas(ListaS, rochas, numRochas);
        free(rochas); // Libera a memória alocada para as rochas
    } else {
        printf("Nenhuma rocha para redistribuir.\n");
    }

    // Move todas as sondas para a origem (0,0)
    MoveOrigem(ListaS);
}
```

Figura 7: Implementação da Operação E.

## 4. Compilação e Execução

A compilação e a execução do código são feitas mediante o terminal Visual Studio Code a partir da inserção dos seguintes comandos:

- gcc MINERAL.c ListaSondas.c ListaMineral.c Sonda.c Compartimento.c RochaM.c main.c -o teste.exe
- .\teste.exe

Para realizar a entrada por meio de um arquivo, é necessário colocar o arquivo na mesma pasta de todo o código (o arquivo deve ser do tipo .txt).

## 5. Resultados

Após a execução dos comandos de compilação e execução, o código será iniciado. A primeira mensagem exibida solicita ao usuário que escolha se os dados serão inseridos via terminal ou por meio de um arquivo, conforme ilustrado na figura 5.

```
PS D:\GIT HUB\TP--AEDS> gcc MINERAL.c ListaSondas.c ListaMinerais.c Sonda.c Compartimento.c RochaM.c main.c -o teste.exe
PS D:\GIT HUB\TP--AEDS> .\teste.exe
Arquivo de entrada (1 - Terminal, 2 - Arquivo):
```

Figura 8: Mensagem para a inserção de arquivo.

## 1) Resultados da inserção pelo terminal

Após selecionar o terminal como a forma de entrada de arquivos, serão requeridas a quantidade de sondas a serem criadas e as informações de cada sonda, sendo elas: latitude, longitude, capacidade máxima, velocidade e combustível.

```
Arquivo de entrada (1 - Terminal, 2 - Arquivo):
1
Digite o numero de sondas: 2
Digite os dados das sondas: (latitude, longitude, capacidade maxima, velocidade e combustivel)
0.0 50 50 15 80
50 0.0 30 10 70
```

**Figura 9: Mensagem para a inserção dos dados das sondas.**

A seguir, será solicitado o número de operações utilizadas e, posteriormente, o modo de operação desejado. É válido ressaltar que a mensagem solicitando a operação desejada será exibida apenas o número de vezes definido previamente pelo usuário.

```
Digite o numero de operacoes: 5
Digite a operacao: R
```

**Figura 10: Número de operações e operação requisitada.**

- **Operação 'R':** A operação 'R' exibe uma mensagem requisitando a latitude, a longitude, o peso e os minerais a serem coletados. Em seguida, será informado qual sonda coletou a rocha. Além disso, a mensagem solicitando uma nova operação aparecerá novamente na tela.

```
Digite a operacao: R
Digite a latitude, longitude, peso e ate 3 minerais (SEPARE POR ESPACO):
-4.6 137.5 20 Ferrolita Aquavitae
Rocha coletada pela sonda: 1
Digite a operacao: I
```

**Figura 11: Operação 'R'.**

- **Operação 'I':** Ao selecionar a operação 'I', serão exibidas no terminal as rochas e seus respectivos pesos em cada uma das sondas espaciais criadas. Na sequência, a mensagem solicitando a escolha da operação será apresentada ao usuário.

```
Digite a operacao: I
1
Aquaferro 20.00
2
Compartimento vazio!
Digite a operacao: R
```

**Figura 12: Operação 'I'.**



- **Operação 'E':** Ao selecionar a operação 'E', o peso de cada sonda após a redistribuição das rochas será exibido. Em seguida, será solicitada a escolha da próxima operação.

```
Digite a operacao: E
Peso da sonda 1: 20.00
Peso da sonda 2: 13.00
Digite a operacao: I
```

**Figura 13: Operação 'E'.**

Por fim, após a conclusão de todas as operações requisitadas, uma mensagem é exibida informando os integrantes do grupo responsável pela criação do sistema.

```
-----
Trabalho feito por Bruno Vicentini, Pedro Paulo Paz e Vitor Mathias
-----
```

**Figura 14: Mensagem final.**

## **2) Resultados da inserção por arquivo**

Após selecionar o arquivo de entrada, será exibida uma mensagem solicitando o nome do arquivo para continuar o processo.

```
Arquivo de entrada (1 - Terminal, 2 - Arquivo):
2
Nome do arquivo de entrada: teste.txt
```

**Figura 15: Nome do arquivo.**

Em seguida, o programa coleta todas as entradas e exibe todas as saídas, incluindo a mensagem com os integrantes do grupo responsável pela criação do sistema, como pode ser visto na figura 13.

```
Nome do arquivo de entrada: teste.txt
```

```
Rocha coletada pela sonda: 2
```

```
Rocha coletada pela sonda: 2
```

```
1
```

```
Compartimento vazio!
```

```
2
```

```
Aquaterra 12.00
```

```
Calquer 18.00
```

```
3
```

```
Compartimento vazio!
```

```
Rocha coletada pela sonda: 1
```

```
Rocha coletada pela sonda: 2
```

```
Rocha coletada pela sonda: 1
```

```
Rocha coletada pela sonda: 3
```

```
1
```

```
Solaris 21.00
```

```
Calquer 19.00
```

```
2
```

```
Aquaterra 12.00
```

```
Calquer 18.00
```

```
Terralis 29.00
```

```
3
```

```
Terrolis 3.00
```

```
Rocha coletada pela sonda: 3
```

```
Rocha coletada pela sonda: 1
```

```
Peso da sonda 1: 53.00
```

```
Peso da sonda 2: 46.00
```

```
Peso da sonda 3: 40.00
```

```
1
```

```
Terralis 29.00
```

```
Ferrom 12.00
```

```
Aquaterra 12.00
```

```
2
```

```
Ferrom 25.00
```

```
Calquer 18.00
```

```
Terrolis 3.00
```

```
3
```

```
Solaris 21.00
```

```
Calquer 19.00
```

```
-----  
Trabalho feito por Bruno Vicentini, Pedro Paulo Paz e Vitor Mathias
```

**Figura 16: Saídas.**

## 6. Conclusão

Diante do exposto, observa-se a relevância do uso de Tipos Abstratos de Dados (TADs) e listas encadeadas para o desenvolvimento do projeto, dado que proporcionaram soluções eficientes, como a criação de uma lista de sondas dinâmica, adaptada à demanda, evitando desperdício de memória.

A utilização de listas encadeadas mostrou-se mais eficiente que os vetores, que podem ocupar mais espaço do que necessário ou serem insuficientes para armazenar valores, pois possuem tamanho fixo. Além disso, os TADs contribuíram para uma melhor organização do projeto, facilitando a associação de valores, como a atribuição de minerais às rochas.

Em consonância, a implementação de um terminal interativo e a opção de entrada de dados por arquivo permitiram uma melhor experiência ao usuário, adaptando o sistema às suas necessidades. Ademais, os resultados obtidos foram coerentes com os casos de teste fornecidos, atendendo corretamente às expectativas.

Por fim, além de alcançar os objetivos propostos, o trabalho desempenhou um papel crucial na aprendizagem do grupo, reforçando a importância de TADs e listas encadeadas, bem como exemplificando boas práticas de programação, vitais para a realização de um projeto em equipe.

## 7. Referências

- [1] Github. Disponível em: <<https://github.com/VMathiasAndrade/TP--AEDS>>. Último acesso em: 21 de novembro de 2024.
- [2] SILVA, Thais. Aula2CCF211\_ListasTH2023. Florestal, MG: Universidade Federal de Viçosa, 2024. Último acesso em: 16 de novembro de 2024.
- [3] PROGRAMAÇÃO DESCOMPLICADA | LINGUAGEM C. Estrutura de Dados em Linguagem C | Lista Dinâmica Encadeada. 2013. Disponível em: <<https://abrir.link/CLVfO>>. Último acesso em: 16 de novembro de 2024.
- [4] w3schools. Disponível em: <[https://www.w3schools.com/c/ref\\_stdlib\\_qsort.php](https://www.w3schools.com/c/ref_stdlib_qsort.php)>. Último acesso em: 20 de novembro de 2024.
- [5] PROGRAMAÇÃO DESCOMPLICADA | LINGUAGEM C. Estrutura de Dados em Linguagem C | Aula 55 - Ordenação - Usando a função qsort(). 2013. Disponível em: <<https://youtu.be/HtvfgqO0IM4?si=2D3O2vDJfqiX2TGD>>. Último acesso em: 20 de novembro de 2024.