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 cursores

Bernardo Nery Ferreira Cunha [5893]
Erich Pinheiro Amaral [5915]
Tariky Rodrigues Campos [5758]

Florestal - MG
2024

Sumário

1. Introdução

O assunto do nosso trabalho prático do ano de 2024 foi fornecer ajuda para a Agência Espacial para Desenvolvimento Sustentável (AEDS) que deseja enviar sondas para Marte com objetivo de investigar o solo marciano, nossa função nesta agência é catalogar minerais, rochas e sondas, afim de realizar o melhor custo benefício para recolher e identificar o maior número de minerais e rochas desconhecidos. Para a realização do projeto nós escolhemos abordar as técnicas de listas encadeadas, vetores, listas lineares e tipos abstratos de dados, além dessas técnicas, usamos também programação dinâmica, ao juntarmos essas técnicas conseguimos realizar um trabalho que além de funcional ficou organizado e de fácil entendimento.

2. Organização

Na figura 1, é possível perceber uma visão geral da estrutura do projeto, que destaca os TADs e as estruturas de dados utilizadas, com foco nas listas encadeadas.
Ademais, os arquivos foram separados em duas categorias: .h, que contém a declaração das funções e a definição das structs e .c, que se encarrega da implementação das funções previamente declaradas e do uso das structs. Essa divisão visa aprimorar a organização e o funcionamento do código.

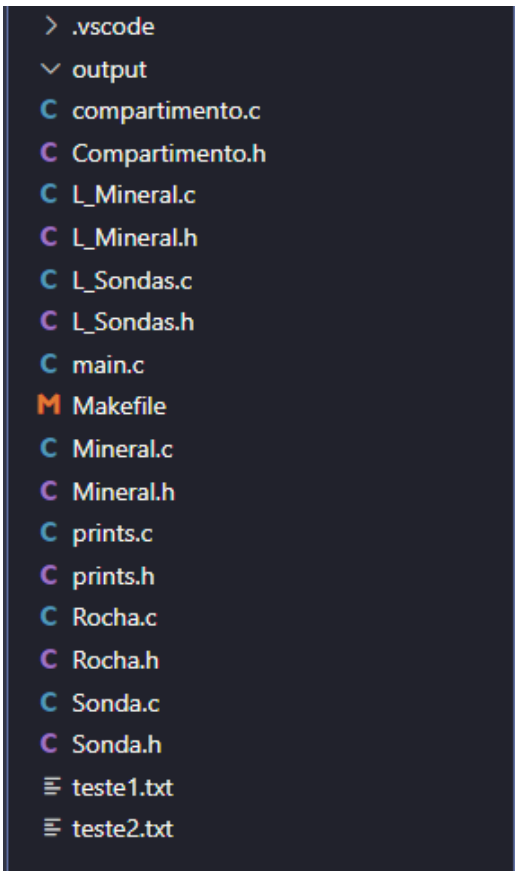


Figura 1 - Repositório do projeto.

3. Desenvolvimento

A aplicação dos TADs foi crucial para a execução do projeto, assegurando que o código operasse de forma adequada. Com isso em vista, é importante salientar as principais implementações do sistema que foi criado:, TAD Mineral, TAD RochaMineral, TAD prints, Main, Operação R, Operação I e Operação E.

TAD Mineral.

Esse TAD tem a finalidade de dar os nomes aos minerais para dar início a busca pela rocha que será capturada pela sonda no planeta Marte.

```c
#define zero 0

void InicializaMineral(Mineral *mineral, char *nome, char *cor, double dureza, double reatividade)
{
    setNome(mineral, nome);
    setCor(mineral, cor);
    setDureza(mineral, reatividade);
    setReatividade(mineral, dureza);
}
void Lis_Minerais(Mineral *mineral, char *nome){
    if (strcmp(nome, "Ferrolita") == zero)
    {
        strcpy(mineral->nome, "Ferrolita");
        strcpy(mineral->cor, "Acinzentado");
        mineral->dureza = 0.5;
        mineral->reatividade = 0.7;
    }
    else if (strcmp(nome, "Solarium") == zero)
    {
        strcpy(mineral->nome, "Solarium");
        strcpy(mineral->cor, "Amarelo");
        mineral->dureza = 0.9;
        mineral->reatividade = 0.2;
    }
    else if (strcmp(nome, "Aquavitae") == zero)
    {
        strcpy(mineral->nome, "Aquavitae");
        strcpy(mineral->cor, "Azulado");
        mineral->dureza = 0.5;
        mineral->reatividade = 0.8;
    }
    else if (strcmp(nome, "Terranita") == zero)
    {
        strcpy(mineral->nome, "Terranita");
        strcpy(mineral->cor, "Marrom");
        mineral->dureza = 0.7;
        mineral->reatividade = 0.6;
    }
```

Figura 3.1 Lista Minerais

TAD Rocha Mineral

Esse TAD tem a função de buscar a rocha pela sua característica, basicamente você precisará da lista de minerais e buscar a rocha que as suas características representam.

```c
#define zero 0
#define um 1
#define dois 2
#define tres 3

void InicializaRocha(rochamineral *rocha, int id, float peso, char *categoria, char *data, double latitude, double longitude)
{
    setIdentificador(rocha, id);
    setPeso(rocha, peso);
    setCategoria(rocha, categoria);
    setData(rocha, data);
    setLocalizacao(rocha, latitude, longitude);
}

char *Categoria(rochamineral *rocha)
{
    int NumMine = rocha->L_Mineral.pUltimo;
    if (NumMine == zero)
    {
        strcpy(rocha->categoria, "Sem Minerais");
        return rocha->categoria;
    }
    int ferrolita = zero, solarium = zero, aquavitae = zero, terranita = zero, calaris = zero;

    for (int i = rocha->L_Mineral.pPrimeiro; i < rocha->L_Mineral.pUltimo; i++)
    {
        if (strcmp(rocha->L_Mineral.ListaMINERAIS[i].nome, "Ferrolita") == zero)
        {
            ferrolita = um;
        }
        else if (strcmp(rocha->L_Mineral.ListaMINERAIS[i].nome, "Solarium") == zero)
        {
            solarium = um;
        }
        else if (strcmp(rocha->L_Mineral.ListaMINERAIS[i].nome, "Aquavitae") == zero)
        {
            aquavitae = um;
        }
        else if (strcmp(rocha->L_Mineral.ListaMINERAIS[i].nome, "Terranita") == zero)
        {
            terranita = um;
        }
```

Figura 3.2 Função Definir Categoria.

TAD prints

Basicamente esse TAD e apenas uma implementação de textos no código, utilizaremos na main.c, onde deixará o código mais abstrato possível.

```c
#include "prints.h"
#include <stdio.h>

void printInicio(){
    printf("\n=============================================================================================================\n");
    printf("Ola, somos a equipe de desenvolvimento que fara o sistema de controle de sondas e catalogacao de rochas minerais.\nMembros do grupo: Tariky, Erich e Bernado.\n");
    printf("=============================================================================================================\n\n");
    printf("Primeiro insira a opcao de entrada.\n");
}
void printEntrada(){
    printf("Arquivo de entrada 1-Terminal, 2-Arquivo: \n");
}
void printNumOper(){
    printf("A Sonda ja esta disponivel para realizar as operacoes.\nAgora insira o numero de operacoes: ");

}
void printDadosSonda(){
    printf("Insira os dados das sondas: Latitude, longitude, capacidade maxima, velocidade e combustivel\n");
}
void printNumSondas(){
    printf("Agora insira o numero de sondas: ");
}
void printSucesDados(){
    printf("Dados da sonda(s) inseridos com sucesso!! Elas ja estao prontas para irem para Marte.\n\n");
}
```

Figura 3.3 Funções prints

Main

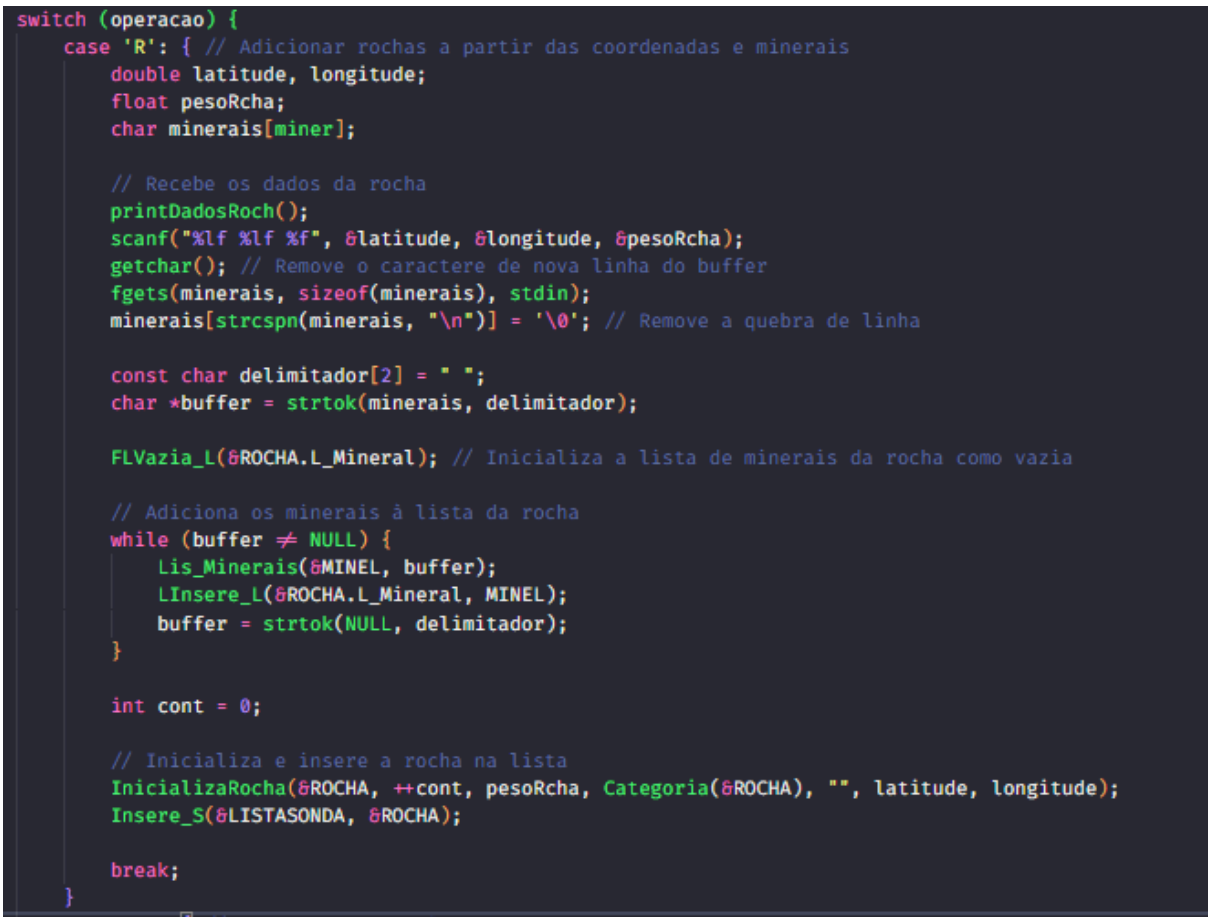
A função Main desempenha um papel crucial na interação entre o usuário e o programa, recolhendo informações e invocando funções disponíveis, conforme as demandas do usuário. Em síntese, sua relevância reside na habilidade de organizar as chamadas a outras funções e garantir que as interações aconteçam de maneira eficaz.

```c
int main() {
    int N_Sondas, comando = 0; // Número de sondas e comando do método de entrada
    double lat_i, long_i; // Coordenadas iniciais da sonda
    float c_i, v_i, nc_i; // Capacidade, velocidade, e combustível inicial das sondas
    char linha[Maxtam]; // Buffer para leitura de strings
    L_Sondas LISTASONDA; // Lista de sondas
    rochamineral ROCHA; // Estrutura que representa uma rocha mineral
    Mineral MINEL; // Estrutura que representa um mineral
    DadosSonda SONDA; // Estrutura que representa uma sonda


    printInicio();
    while (comando != 1 && comando != 2) {
        printEntrada();
        scanf("%d", &comando);
    }
    // Entrada de dados via terminal
    if (comando == 1) {
        FLVazia_S(&LISTASONDA); // Inicializa a lista de sondas como vazia

        // Solicita o número de sondas e seus dados
        printNumSondas();
        scanf("%d", &N_Sondas);

        printDadosSonda();
        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(&SONDA, (i + 1), lat_i, long_i, c_i, "Sim"); // Inicializa uma nova sonda
            LInsere_S(&LISTASONDA, &SONDA); // Insere a sonda na lista
        }
        printSucesDados();

        int numOperacoes; // Número de operações
        printNumOper();
        scanf("%d", &numOperacoes);
        for (int i = 0; i < numOperacoes; i++) {
            char operacao;
            printOper();
            scanf(" %c", &operacao);
```

Figura 3.4 Função Principal (Main)

Operação R

A operação R responsável por indicar a coleta de uma nova rocha, nós fizemos por meio da inicialização de uma lista vazia, logo após fizemos a adição das rochas coletadas na lista anteriormente feita.



```
switch (operacao) {
    case 'R': { // Adicionar rochas a partir das coordenadas e minerais
        double latitude, longitude;
        float pesoRcha;
        char minerais[miner];

        // Recebe os dados da rocha
        printDadosRoch();
        scanf("%lf %lf %f", &latitude, &longitude, &pesoRcha);
        getchar(); // Remove o caractere de nova linha do buffer
        fgets(minerais, sizeof(minerais), stdin);
        minerais[strcspn(minerais, "\n")] = '\0'; // Remove a quebra de linha

        const char delimitador[2] = " ";
        char *buffer = strtok(minerais, delimitador);

        FLVazia_L(&ROCHA.L_Mineral); // Inicializa a lista de minerais da rocha como vazia

        // Adiciona os minerais à lista da rocha
        while (buffer != NULL) {
            Lis_Minerais(&MINEL, buffer);
            LInsere_L(&ROCHA.L_Mineral, MINEL);
            buffer = strtok(NULL, delimitador);
        }

        int cont = 0;

        // Inicializa e insere a rocha na lista
        InicializaRocha(&ROCHA, ++cont, pesoRcha, Categoria(&ROCHA), "", latitude, longitude);
        Insere_S(&LISTASONDA, &ROCHA);

        break;
    }
```

Figura 3.5 Operação R

Operação I

A operação I, que imprime a situação atual das sondas, ou seja, as informações de cada sonda, como: peso e rochas coletadas, nós fizemos a partir de uma lista encadeada.

```c
void OperacaoI(L_Sondas *pLista_Sonda) {
    Apontador_S pAux = pLista_Sonda->pPrimeiro->pProx;

    while (pAux != NULL)
    {
        DadosSonda *pSonda = &pAux->sonda;
        printf("\nSonda: %d\n", pSonda->idSonda);

        if (LEhVazia_R(&pSonda->Compar_Rocha)){
            printf("Compartimento vazio!\n");
        }
        else{
            Apontador_R rAux = pSonda->Compar_Rocha.pPrimeiro->pProx;
            while (rAux != NULL){
                printf("%s %.2f\n", rAux->rocha.categoria, rAux->rocha.peso);
                rAux = rAux->pProx;
            }
        }
        pAux = pAux->pProx;
    }
}
```

Figura 3.6 Operação I

Operação E

Na imagem acima, representamos a operação E, que tinha como objetivo redistribuir as rochas coletadas, a fim de distribuir uniformemente entre as sondas, feitas a partir de um laço “IF” e “While” e listas, além de retornarem todas as sondas para o ponto de origem (0,0).

```c
// Função principal para redistribuir rochas e mover sondas para a origem
void OperacaoE(L_Sondas *pLista_S) {
    int numRochas = 0;
    // Extrai as rochas de todas as sondas
    rochamineral *rochas = NovaListaTempRocha(pLista_S, &numRochas);

    if (rochas != NULL) {
        // Redistribui as rochas entre as sondas
        Redistribuir(pLista_S, rochas, numRochas);
        free(rochas); // Libera a memória do array de rochas
    } else {
        printf("Nenhuma rocha para redistribuir.\n");
    }

    // Move todas as sondas para a origem
    MoveOrigem(pLista_S);
    printf("\nRochas redistribuidas com sucesso!!\n");
    printf("\n");

}
```

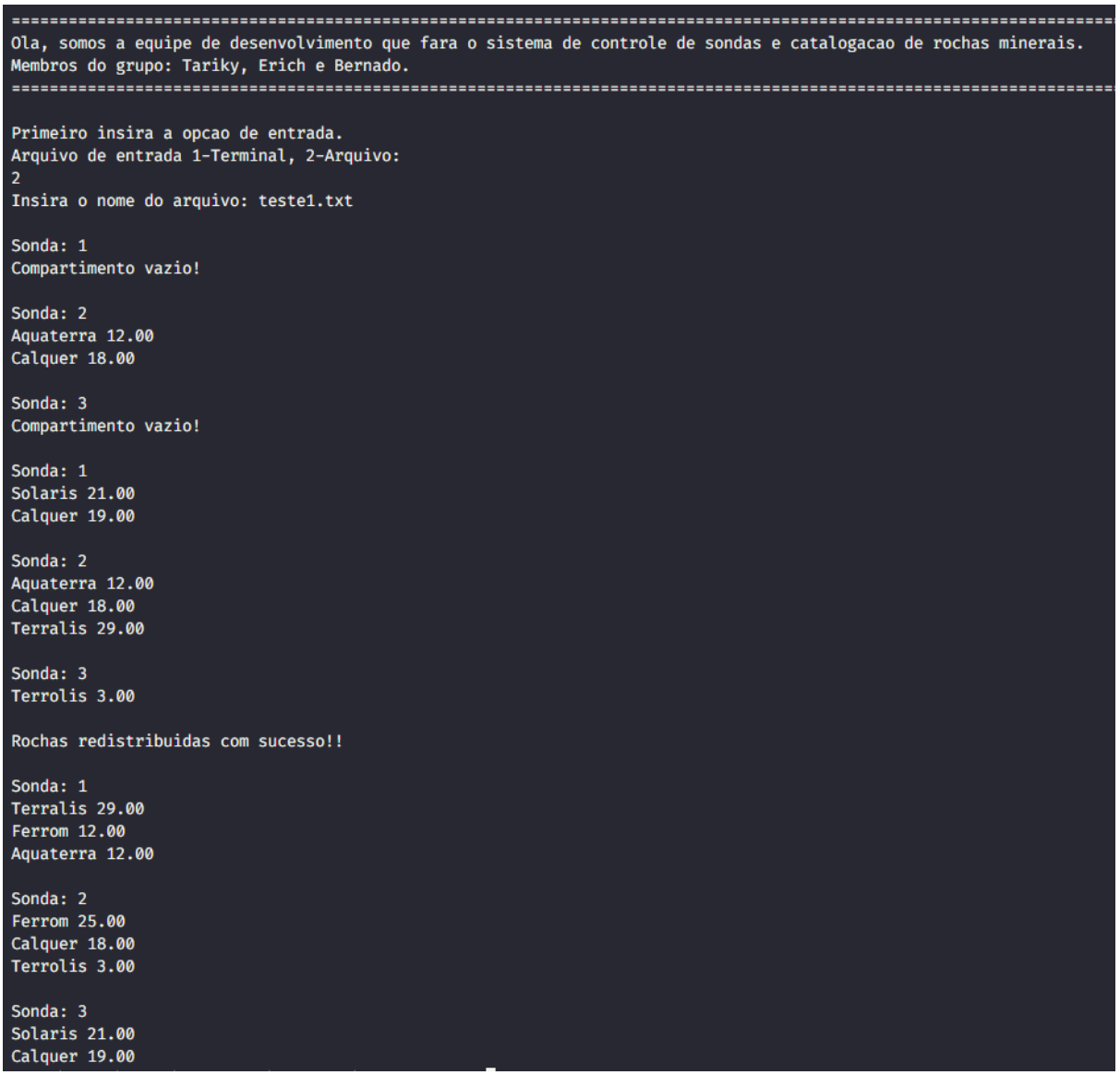
Figura 3.7 Operação E

4. Compilação e Execução

Para a compilação e execução do projeto, todos do grupo usaram o Windows 11, realizamos o “Make a File”, usando o gcc compilando todos os arquivos .c e usando um .exe para a execução do código, no caso nosso gcc de compilação usamos um compilador genérico para Windows fornecido pelo monitor.

5. Resultados

Nosso projeto foi realizado de forma que as saídas dependem das entradas do usuário, criamos uma menu de utilização no qual o usuário escolhe o'que fazer digitando letras e números do teclado, como este print do nosso código em execução de um teste:



Na figura acima mostramos o resultado do nosso código, com os resultados corretos com o teste feito pelo pvanet.

6. Conclusão

Para concluir, vemos que durante a realização do trabalho prático, nós tivemos algumas escolhas que nos ajudaram muito na hora de programar os códigos, como a organização por meio de listas encadeadas e definição de estruturas bem nomeadas, utilização do github Desktop para envio e commit dos arquivos, nosso código foi coerente acerca do que nós imaginávamos no começo, passamos por algumas obstruções no caminho, porém serviram de aprendizado para outras atividades em grupo, algumas escolhas no âmbito de ambientes também foram muito importantes, pelo fato de que o Visual Studio Code nos deu muitas ferramentas de auxílio, como extensões que nos permitiam diferenciar variáveis, tudo isso ornou para que o projeto ficasse da forma que nos era imaginado.

7. Referências

[1] Github. Disponível em: <https://github.com/tariky-campos/TP-AEDS> Último acesso em: 17 de novembro de 2024.

[2] PROGRAMAÇÃO DESCOMPLICADA | LINGUAGEM C. Estrutura de Dados em Linguagem C | Lista Dinâmica Encadeada. 2013. Disponível <https://abrir.link/CLVFO>. Último acesso em: 16 de novembro de 2024.

[2] LINGUAGEM C DESCOMPLICADA | Portal de vídeo-aulas para estudo de programação. <https://programacaodescomplicada.wordpress.com/>