INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS SÃO JOÃO EVANGELISTA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GABRIEL KÁICON E JOÃO VITOR MENDES CAMPOS

TRABALHO PRÁTICO IV

SÃO JOÃO EVANGELISTA
2022

GABRIEL KÁICON E JOÃO VITOR MENDES CAMPOS

SISTEMA PARA UMA LINHA DE ÔNIBUS INTERMUNICIPAL

SÃO JOÃO EVANGELISTA
2022

SUMÁRIO

1. INTRODUÇÃO

Este trabalho prático foi documentado para que seja avaliado em conjunto com os códigos na linguagem C/C++, exigido pelo docente Eduardo Augusto da Costa Trindade, dentro da disciplina de Algoritmos e Estruturas de Dados I, ministrada pelo mesmo. Porém a documentação tem cunho expositivo, onde é descrito as funcionalidades do programa, com testes, e desenvolvimento de novas linhas de raciocínio lógico para realização do trabalho prático.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral apresentar na prática os conhecimentos adquiridos nas aulas de Algoritmos e Estruturas de Dados I, a respeito de lista duplamente encadeada, Tipos Abstratos de Dados (TADs), utilizando a linguagem de C++ para escrita dos códigos.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Objetivo 1. Fixar conceitos sobre estruturas de dados e modularização.
- Objetivo 2. Trabalhar com a TAD de Lista duplamente encadeada circular.
- Objetivo 3. Estimular o raciocínio lógico e a proposição para solução de problemas.

1.3. Justificativa

Ao iniciar os estudos de Algoritmos e Estruturas de Dados I, vemos os conteúdos de Ponteiros, TADs, e estruturas de dados, porém o que nos interessa nesse momento é a lista duplamente encadeada circular.

Vemos em ponteiros, a manipulação de valores da variável por meio do endereço de memória utilizando ponteiros. Por último vimos o conteúdo de lista duplamente encadeada circular, que consiste na inserção de itens em uma lista, e também na remoção por meio de suas posições. Temos o conceito de lista, lista

encadeada, lista duplamente encadeada, lista circular,e lista duplamente encadeada circular, dentre outras, e todas essas citadas podem ser com alocação estática(possui um número limitado de itens) ou com alocação dinâmica (possui um número ilimitado de itens). Estamos interessados na lista duplamente encadeada circular, que não possui um limite pré-definido podendo ter um número infinito de itens. O conceito de lista é simples, de acordo com o primeiro que é inserido, você vai inserindo itens, de forma que o item a ser inserido possa vir no começo, no fim ou após outro item . Na lista duplamente encadeada circular é a mesma, porém há novas coisas como agora o fim não aponta para NULL, mas para o início e também contém mais um apontador o anterior. No trabalho, é exigido que utilizemos a lista duplamente encadeada circular, porém utilizei de uns conceitos novos apresentados em sala de aula recentemente, que juntos resultaram na criação do minissistema de gerenciamento da linha de ônibus intermunicipal. Tendo isso tudo em vista, o trabalho foi exigido para que seja possível desenvolver o raciocínio lógico quanto a aplicação dessa estrutura, e com o bônus de novamente aplicá-la em conjunto.

2. DESENVOLVIMENTO

Nesta seção do documento é apresentado, os conceitos aprendidos e o desenvolvimento do trabalho em si, na linguagem C++.

2.1. Conceitos Aplicados

Explicação sucinta dos conceitos de Lista duplamente encadeada circular e Tipos Abstratos de Dados (TADs).

2.2. Tipos Abstratos de Dados

As estruturas utilizadas como registro para criação de objetos, e as funções de manipulação dessas estruturas, ambas compõem uma TAD. Declaramos esses modelos como structs, elas são representações de qualquer coisa no mundo real, sendo ela lógica, abstrata ou física, como por exemplo uma pessoa, que é algo físico, ou um filme digital, que é algo lógico/abstrato, veja o exemplo na figura 1. Cada um tem suas características específicas, uma pessoa nome, sexo, idade, CPF, altura, dentre outras, e um filme, título, linguagem, elenco, personagens, duração, categoria, ano de lançamento, dentre outros, e tudo isso pode ser definido dentro de uma struct para cada um deles. Resumindo uma Struct é uma espécie de variável modelo para cadastrar diferentes itens, dentro de um software escrito em C/C++. Acompanhado das structs temos as funções para manipulação dos dados dessa lista, e desses itens, que será visto no próximo tópico.

Figura 1 - Struct exemplo

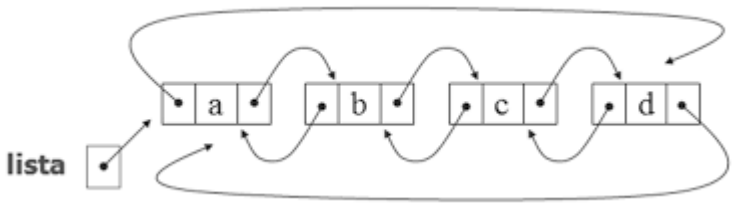
```
//Duplamente encadeada
typedef struct TipoItemD{
    int id;
    string nome;
    string saida;
    double valor_passagem;
}TipoLinha;

typedef struct ElementoL* ApontadorL;
typedef struct ElementoL{
    ApontadorL ant;
    ApontadorL prox;
    TipoItemD item;
}ElementoL;

typedef struct TipoListaD{
    ApontadorL primeiro;
    ApontadorL ultimo;
    int tamanho = 0;
}TipoListaRota;
```

2.3. Lista duplamente encadeada circular

Figura 2 – Lista duplamente encadeada circular



Na figura 2, vemos um desenho esquemático de como é a estrutura de dados da lista duplamente encadeada.

2.4. Implementação

Na parte de implementação foi utilizada a TAD disponibilizada.

Apêndice B - Nele é apresentado a função (Pesquisa Linha por Origem e Destino) o objetivo dessa função é encontrar uma linha que tenha a origem e o destino que o usuário determinou. A lista é percorrida por meio de um while que percorre a lista de cidades, uma variável auxiliar é criada para armazenar a lista de cidades que estão dentro de uma linha, em auxp. Usa-se outro While para percorrer a lista de cidades na ida. Utilizando um if para achar o destino, se o destino for encontrado pegamos o horário de chegada, depois do if percorremos outra lista por meio do while, a lista a ser percorrida é a lista de cidades na volta. Criamos uma variável para armazenar o valor da passagem (soma o valor da passagem de cidade em cidade). Por meio de outro if verificamos se a origem foi encontrada, se ela for encontrada pegamos o horário de saída. O auxp é atualizado. Se a origem e o destino forem encontrados, será mostrado na tela o nome da companhia, o horário de chegada e de saída e o valor da passagem.

Apêndice C - Nele é apresentado a função (Pesquisa Linha por Código) o objetivo é encontrar a linha por meio de código. O usuário digita o código da linha se a linha for encontrada ela será exibida na tela.

Apêndice D - Nele é apresentado a função (Insere Linha) o objetivo é inserir uma nova linha. Para a inserção o usuário deverá digitar o id, o nome da companhia, após isso ele deverá informar o nome da primeira parada, o valor da passagem da parada e o horário de saída da parada. Assim inserindo uma nova linha.

Apêndice E - Nele é apresentado a função (Incluir Parada) o objetivo incluir uma nova parada. Para a inclusão de uma nova parada o usuário deverá informar o id da linha, após a linha ser encontrada o usuário deverá digitar o nome da parada, o valor da passagem e o horário da saída da parada. Essa parada será incluída após outra parada que foi previamente incluída.

Apêndice F - Nele é apresentado a função (Altera Parada) o objetivo é alterar algum dado de alguma parada. Para que haja essa alteração o usuário deve informar o id da linha e depois informar o nome da parada que deseja alterar. Após isso será pedido o novo nome da parada, o valor da passagem e o horário de saída. Assim alterando a parada.

Apêndice G - Nele é apresentado a função (Elimina Parada) o objetivo eliminar uma parada. Para a eliminação de uma parada é preciso que o usuário informe o id da linha e depois o nome da parada a ser excluída. Assim eliminando essa parada.

Apêndice H - Nele é apresentado a função (Elimina linha) o objetivo é eliminar uma linha. Para a eliminação é preciso que o usuário informe o id da linha a ser eliminada. Assim eliminando a linha por inteiro.

A modularização foi feita por meio da main.cpp, sistema.hpp, sistema.cpp , listasencadeadas.cpp, cabeclistasencadeadas.hpp e um main.exe. A TAD se localiza no Apêndice A.

3. CONCLUSÃO

Ao longo do trabalho tive várias ideias de como realizar o problema, a parte de lista duplamente encadeada circular pode ser feita de várias formas. Utilizamos os códigos padrões das TADs e fizemos as devidas alterações, utilizamos um pouco do código do último trabalho que fizemos, também fizemos uso dos slides disponibilizados pelo professor para sanar dúvidas a respeito, e usuários externos para tirar dúvidas sobre a criação e uso de novas funções. Creio que nosso desenvolvimento foi bom nesse trabalho, desenvolvemos novas formas de raciocínio e tivemos que abstrair e diminuir muitas coisas como de costume, para que assim o trabalho ficasse leve, sem ser muito extenso. Tivemos que buscar ajuda externa e conhecimento externo para realização do trabalho, mas mesmo assim conseguimos atingir nossos objetivos estipulados no início do trabalho, explicar bem o que nós estávamos fazendo dentro de cada função, apresentar conhecimentos em Lista, e com confiança aprimorar nossos conhecimentos. Estamos felizes e satisfeitos com o resultado, e ele atendeu às nossas expectativas além do que nós esperávamos.

4. REFERÊNCIAS

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Fila com ponteiro. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/149269/mod_resource/content/1/Aula 12 - Listas Duplamente Encadeadas.pdf . Acesso em: 12 de dezembro de 2022.

COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados - Listas utilizando Ponteiro. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146500/mod_resource/content/1/Aula%207%20-%20Listas%20Encadeadas.pdf . Acesso em: 12 de Dezembro de 2022.

COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados – Listas Duplamente Encadeada. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/149269/mod_resource/content/1/Aula%2012%20-%20Listas%20Duplamente%20Encadeadas.pdf . Acesso em: 12 de Dezembro de 2022.

Link no Github: https://github.com/gKaicon/Arquivos_C_and_C-withClasses/tree/main/AEDs%20I/TP/TP%20-%204

5. APÊNDICES

5.1. APÊNDICE A – (Diagrama TAD)

```
//Duplamente encadeada
typedef struct TipoItemD{
    int id;
    string nome;
    string saida;
    double valor_passagem;
}TipoLinha;

typedef struct ElementoL* ApontadorL;
typedef struct ElementoL{
    ApontadorL ant;
    ApontadorL prox;
    TipoItemD item;
}ElementoL;

typedef struct TipoListaD{
    ApontadorL primeiro;
    ApontadorL ultimo;
    int tamanho = 0;
}TipoListaRota;

void criaListaVaziaD(TipoListaRota *lista);
bool listaVaziaD(TipoListaRota *lista);
void insereItemUltimoD(TipoListaRota *lista, TipoLinha item);
void insereItemPrimeiroD(TipoListaRota *lista, TipoLinha item);
ApontadorL localizaItemPorIdD(TipoListaRota *lista ,int id);
void insereItemAposElementoD(TipoListaRota *lista, TipoLinha item, int id);
TipoLinha retiraItemPorIdD(TipoListaRota *lista, int id);
void imprimeItemD(TipoLinha item);
void imprimeListaD(TipoListaRota *lista);
void imprimeListaReversaD(TipoListaRota *lista);
```

```
//Encadeada
typedef struct TipoItem{
    string id;
    TipoListaD cidades;
    string companhia;
}TipoRota;

typedef struct Elemento* Apontador;
typedef struct Elemento{
    TipoItem item;
    struct Elemento* prox;
}Elemento;

typedef struct TipoLista{
    Apontador primeiro;
    Apontador ultimo;
    int tamanho = 0;
}TipoListaLinha;

void CriaListaVazia(TipoListaLinha *lista);
bool verificaListaVazia(TipoListaLinha lista);
void atualizaUltimo(TipoListaLinha *lista);
void insereListaPrimeiro(TipoListaLinha *lista, TipoRota *item);
void insereListaUltimo(TipoListaLinha *lista, TipoRota *item);
void insereListaAposElemento(TipoListaLinha *lista, TipoRota *item, string id);
void imprimeLista(TipoListaLinha *lista);
string pesquisaItem(TipoListaLinha *lista, string id);
void imprimeItem(TipoListaLinha *lista, string id);
void removeListaPrimeiro(TipoListaLinha *lista);
void removeListaUltimo(TipoListaLinha *lista);
void removeItemPorId(TipoListaLinha *lista, string id);
```

5.2. APÊNDICE B – (Pesquisa Por Origem e Destino)

```
void pesquisaLinhaPorOrigemDestino(TipoLista *lista){
    Apontador aux = lista->primeiro->prox;
    string origem, destino;
    bool procuraLinha = false;

    system("cls");
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "**                                                     **" << endl;
    cout << "**            PROCURA LINHA POR ORIGEM/DESTINO         **" << endl;
    cout << "**                                                     **" << endl;
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "Digite a origem: ";
    getline(cin,origem);
    cin.ignore();
    cout << "Digite o destino: ";
    getline(cin,destino);

    while(aux != NULL){ //percorre a lista de cidades
        ApontadorL auxP = aux->item.cidades.primeiro->prox; //armazena a lista de paradas(cidades) que estão dentro de uma linha, em auxP
        string chegada, saida;
        bool dest = false, orig = false;
        double valor_real = 0;

        while(auxP != aux->item.cidades.primeiro){//percorre a lista de cidades na ida
            if(auxP->item.nome == destino){ //para achar o destino
                dest = true;
                chegada = auxP->item.saida; //insere o horario da chegada
                while(auxP != aux->item.cidades.primeiro){ //percorre a lista de cidades na volta
                    valor_real += auxP->item.valor_passagem; //soma ao valor da passagem de cidade em cidade
                    auxP = auxP->ant;
                    if(auxP->item.nome == origem){//para achar a origem
                        orig = true;
                        saida = auxP->item.saida; //insere o horario da saida
                        break;
                    }
                }
                break;
            }
            auxP = auxP->prox;
        }
        if(orig && dest == false) cout << "Nenhuma linha foi encontrada!";
        Sleep(1500);
        if(orig && dest) procuraLinha = true;
        if(procuraLinha){//origem e destino foram encontrados, ele mostra
            cout << "ID: " << aux->item.id << endl;
            cout << "Companhia: " << aux->item.companhia << endl;
            cout << "Chegada/Saida: " << chegada << "/" << saida << endl;
            cout << "Valor da Passagem: " << valor_real << endl;
            system("PAUSE");
        }
        aux = aux->prox;
    }
}
```

5.3. APÊNDICE C – (Pesquisa Linha Por Código)

```
//Pesquisa linhas de acordo com o código.
void pesquisaLinhaPorCodigo(TipoLista *lista){
    string codigo;
    system("cls");
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "**                                                     **" << endl;
    cout << "**            PESQUISA DE LINHA POR CÓDIGO             **" << endl;
    cout << "**                                                     **" << endl;
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "Digite o código a ser pesquisado: ";
    cin >> codigo;
    cin.ignore();
    imprimeItem(lista, codigo);
}
```

5.4. APÊNDICE D – (Insere Linha)

```
void insereLinha(TipoLista *lista){
    TipoLinha parada;
    TipoRota linha;
    string comp, id;
    char op = 's';
     int i = 0;
    system("cls");
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "**                                                     **" << endl;
    cout << "**                   CADASTRA LINHA                    **" << endl;
    cout << "**                                                     **" << endl;
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "Digite o id da linha: ";
    getline(cin, id);
    cout << "Digite a companhia: ";
    getline(cin, comp);

    linha.id = id;
    linha.companhia = comp;

    insereListaUltimo(lista, &linha);
    do{
        string nome, hora;
        double valor;

        cout << "Digite o nome da " << i <<"ª parada: ";
        getline(cin,nome);

        cout << "Digite o valor da passagem da parada: ";
        cin >> valor;
        cin.ignore();
        cout << "Digite o horário de saida da parada(hh:mm): ";
        getline(cin,hora);

        parada.nome = nome;
```

```
        cout << "Digite o horario de saida da parada(hh:mm): ";
        getline(cin,hora);

        parada.nome = nome;
        parada.valor_passagem = valor;
        parada.saida = hora;
        parada.id = i + 1;

        cout << "Deseja cadastrar mais paradas?(s/n)";
        cin >> op;
        cin.ignore();
    }while(op == 's');
}
```

5.5. APÊNDICE E – (Incluir Parada)

```
void incluirParada(TipoListaLinha *lista){
    string id;
    system("cls");
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "**                                                      **" << endl;
    cout << "**                   CADASTRA PARADA                    **" << endl;
    cout << "**                                                      **" << endl;
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    imprimeLista(lista);
    cout << "Qual o id da linha que deseja incluir uma parada?";
    getline(cin, id);
    Apontador aux = lista->primeiro->prox;
    while(aux != NULL){
        if(aux->item.id == id){
            imprimeListaD(&aux->item.cidades);
            TipoItemD parada;
            string nome, hora, pAnterior;
            double valor;

            cout << "Digite o nome da parada: ";
            getline(cin, nome);
            cout << "Digite o valor da passagem da parada: ";
            cin >> valor;
            cin.ignore();
            cout << "Digite o horário de saida da parada(hh:mm): ";
            getline(cin, hora);
            cout << "Digite o nome da parada antes da que está sendo adicionada: ";
            getline(cin, pAnterior);

            int id =  aux->item.cidades.ultimo->item.id + 1;
            parada.nome = nome;
            parada.valor_passagem = valor;
            parada.saida = hora;
            parada.id = id;
```

```
            ApontadorL auxP = aux->item.cidades.primeiro->prox;
            while(auxP != aux->item.cidades.primeiro){
                if(auxP->item.nome == pAnterior){
                    insereItemAposElementoD(&aux->item.cidades,parada,auxP->item.id);
                    cout << "Parada inserida com sucesso";
                    Sleep(1800);
                }
                auxP = auxP->prox;
            }
            cout << "Parada não encontrada!";
            return;
        }
        aux = aux->prox;
    }
    cout << "Linha não encontrada";
    Sleep(1800);
}
```

5.6. APÊNDICE F – (Alterar Parada)

```
void alterarParada(TipoLista *lista){
    Apontador aux = lista->primeiro->prox;
    string id;
    system("cls");
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "**                                                     **" << endl;
    cout << "**                   EDITAR PARADA                     **" << endl;
    cout << "**                                                     **" << endl;
    cout << "*********************************************************" << endl;
    cout << "*********************************************************" << endl;
    cout << "Lista de Linhas" << endl;
    imprimeLista(lista);
    cout << "\nQual o id da linha que deseja alterar uma parada?";
    getline(cin, id);

    while(aux != NULL){
        if(aux->item.id == id){
            ApontadorL auxP = aux->item.cidades.primeiro->prox;
            string nome;

            imprimeListaD(&aux->item.cidades);
            cout << "Qual o nome da parada que deseja alterar?";
            getline(cin, nome);

            while(auxP != aux->item.cidades.primeiro){
                if(auxP->item.nome == nome){
                    cout << "Insira o novo nome da parada: ";
                    getline(cin, auxP->item.nome);
                    cout << "Digite o novo valor da passagem da parada: ";
                    cin >> auxP->item.valor_passagem;
                    cin.ignore();
                    cout << "Digite o novo horário de saida da parada(hh:mm): ";
                    getline(cin, auxP->item.saida);
                    cout << "PARADA ALTERADA COM SUCESSO!";
                    return;
                }
                auxP = auxP->prox;
            }
            cout << "Parada não encontrada!";
            return;
        }
        aux = aux->prox;
    }
    cout << "Linha não encontrada";
}
```

5.7. APÊNDICE G - (Eliminar Parada)

```
void eliminarParada(TipoLista *lista){
    Apontador aux = lista->primeiro->prox;
    string id;
    system("cls");
    cout << "*****************************************************" << endl;
    cout << "*****************************************************" << endl;
    cout << "**                                                 **" << endl;
    cout << "**                  EXCLUIR PARADA                 **" << endl;
    cout << "**                                                 **" << endl;
    cout << "*****************************************************" << endl;
    cout << "*****************************************************" << endl;
    cout << "Lista de Linhas" << endl;
    imprimeLista(lista);
    cout << "\nQual o id da Linha que deseja excluir uma parada?";
    getline(cin, id);

    while(aux != NULL){
        if(aux->item.id == id){
            ApontadorL auxP = aux->item.cidades.primeiro->prox;
            string nome;

            imprimeListaD(&aux->item.cidades);
            cout << "Qual o nome da parada que deseja excluir?";
            getline(cin, nome);

            while(auxP != aux->item.cidades.primeiro){
                if(auxP->item.nome == nome){
                    retiraItemPorIdD(&aux->item.cidades,auxP->item.id);
                    cout << "Parada excluída!";
                    return;
                }
                else cout << "Parada não encontrada";
                auxP = auxP->prox;
            }
        }
        else cout << "Linha não encontrada";
        aux = aux->prox;
```

5.8. APÊNDICE H - (Eliminar Linha)

```
void eliminarLinha(TipoLista *lista){ //Elimina toda uma linha.
    string id;
    system("cls");
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "**                                                      **" << endl;
    cout << "**                   EXCLUIR LINHA                      **" << endl;
    cout << "**                                                      **" << endl;
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "Lista de Linhas" << endl;
    imprimeLista(lista);
    cout << "Qual o id da Linha que deseja excluir?";
    getline(cin, id);
    removeItemPorId(lista, id);
}
```