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

GABRIEL KÁICON BATISTA HILÁRIO

TRABALHO PRÁTICO I

SÃO JOÃO EVANGELISTA
SETEMBRO - 2022

GABRIEL KÁICON BATISTA HILÁRIO

SISTEMA DE CADASTRO DE FUNCIONÁRIOS E DE PROJETOS DENTRO DE FUNCIONÁRIOS

SÃO JOÃO EVANGELISTA

SETEMBRO - 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 Listas com Arranjo, Listas Encadeadas, Tipos Abstratos de Dados (TADs), Manipulação de Arquivos e Ponteiros, utilizando a linguagem de C++ para escrita dos códigos.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Apresentar conhecimentos em lista encadeada;
- Apresentar conhecimentos em lista com arranjo;
- Apresentar conhecimentos em manipulação de arquivos;
- Aplicar conhecimentos adquiridos em um minissistema, em uma visão empresarial.

1.3. Justificativa

Ao iniciar os estudos de Algoritmos e Estruturas de Dados I, vemos os seguintes conteúdos:

- Ponteiros
- TADs
- Manipulação de Arquivos
- Listas:
  - Lista com Arranjo
  - Lista Encadeada

Vemos em ponteiros, a manipulação de valores da variável por meio do endereço de memória, utilizando ponteiros. Vemos em arquivos, os comandos básicos de leitura e gravação de dados em um arquivo por meio de objetos da biblioteca *fstream*, o *ifstream* para leitura e o *ofstream* para gravação.

Posteriormente vemos um conteúdo mais amplo de listas, que consiste na inserção de itens em uma lista, uma estrutura sendo inserida dentro de outra, semelhante a POO (Programação Orientada a Objetos). Inicialmente são apresentados dois tipos de lista:

- A lista com arranjo, também chamada de lista sequencial que é com alocação estática, ou seja, possui um limite pré-definido, podendo ter um número limitado de itens.
- A lista encadeada ou lista com ponteiro que é a com alocação dinâmica, ou seja, não possui um limite pré-definido podendo ter um número infinito de itens.

A princípio ambas são aplicados de forma individual, porém no trabalho, é exigido que usemos as duas juntas, antes devemos cadastrar funcionários em uma lista encadeada, e devemos criar uma lista sequencial dentro de cada funcionário, para cadastrar projetos para esse funcionário que está na lista encadeada.

Os conteúdos foram aplicados para realização do trabalho, criando um minissistema que nos permite cadastrar e excluir funcionários, adicionar e retirar projetos ao funcionário, e calcular e imprimir o contracheque.

Tendo isso tudo em vista, o trabalho foi exigido para que seja possível desenvolver o raciocínio lógico quanto a aplicação dessas estruturas 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 Encadeada, Lista sequencial, arquivos e Tipos Abstratos de Dados(TADs).

2.1.1. Tipos Abstratos de Dados

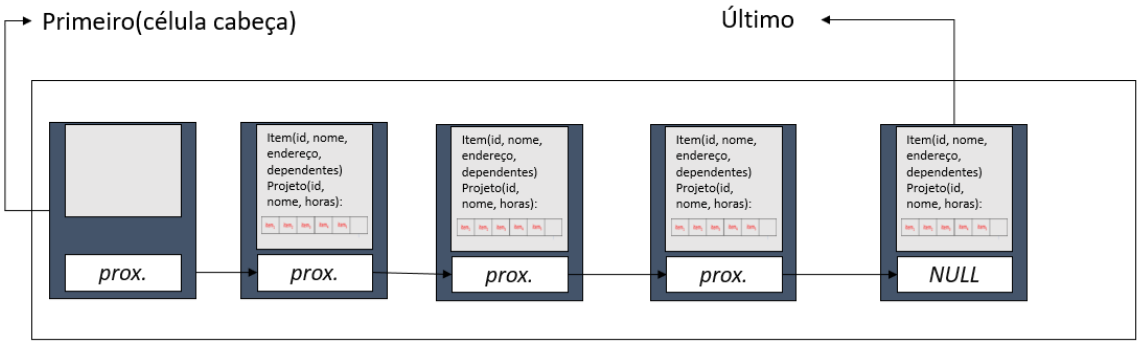
As estruturas utilizadas como registro para criação de objetos, e as funções de manipulação dessas estruturas, ambas compõem uma TAD. Declaramos esses modelos como *structs*, elas são representações de qualquer coisa no mundo real, sendo ela lógica, abstrata ou física, como por exemplo uma pessoa, que 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 definida 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, sem ligação com o trabalho

```
6   typedef struct Horario{
7       int hora;
8       int min;
9   };
10
11  typedef struct Data{
12      int dia;
13      int mes;
14      int ano;
15  };
16
17  typedef struct Compromisso{
18      char descricao[100];
19      Data dt;
20      Horario hr;
21  };
```

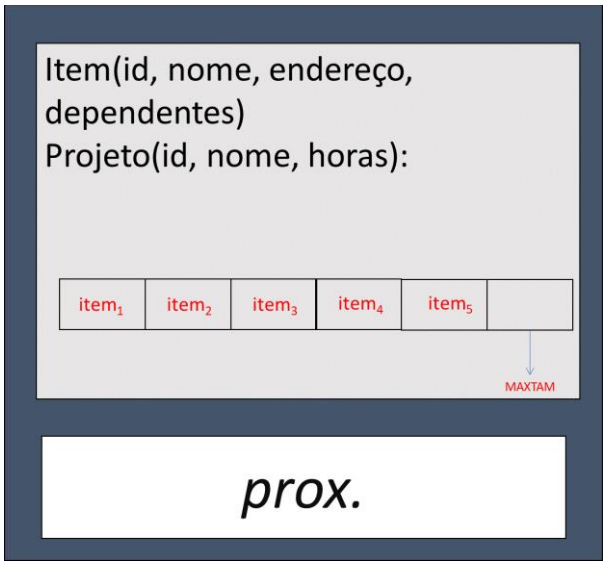
2.1.2. Lista Encadeada

Figura 2 – Lista Encadeada com Sequencial



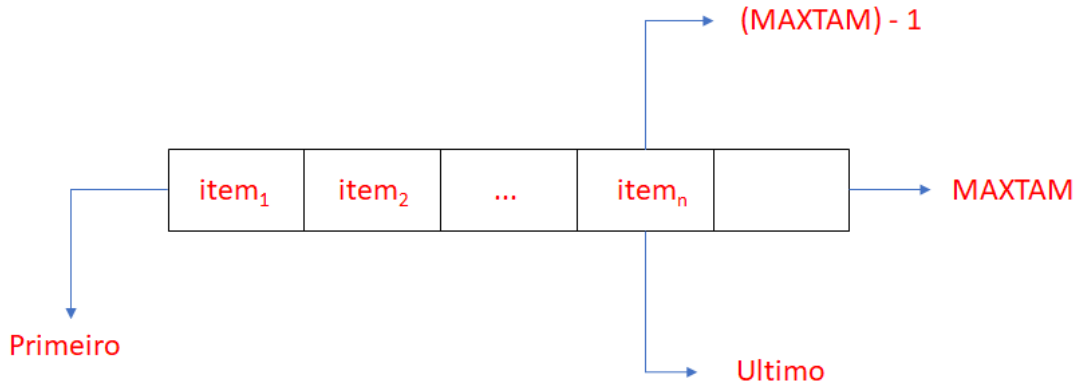
Na figura 2, vemos um desenho esquemático de como seria uma lista encadeada. Cada espaço (em cinza azulado) da lista, tem um apontador, que aponta para a célula seguinte, e uma célula (em cinza claro), essa célula tem o Funcionário (lá está como item) que é uma struct e o projeto, a nossa lista sequencial, que é uma struct, com isso temos, uma struct dentro de um campo de outra struct, comom vemos na figura 3. O primeiro, recebe a célula como NULL, pois ele deve apontar para o primeiro item, e o último, tem o apontador NULL, pois não há uma célula seguinte. Com isso, podemos inserir no inicio ou no fim da lista, e também após um item.

Figura 3 – Célula da Lista



2.1.3. Lista Sequencial

Figura 4 – Lista Sequencial



Na figura 4, vemos um desenho esquemático de como seria uma lista sequencial, e o limite dela, é o MAXTAM, uma variável didática muito comum de ser usada, e sua função é delimitar o tamanho da lista. Ela é um vetor, porém a tipagem de dados a ser inserida podem ser as structs, podendo armazenar mais de um tipo de variável dentro de uma posição. Os elementos são inseridos dentro do índice do vetor no ultimo, que seria o apontador para ultima posição até o momento, e assim sucessivamente, até que a Lista alcance o tamanho máximo, que seria o valor definido para o MAXTAM.

2.1.4. Ponteiros

Ponteiros são variáveis especiais, que podem armazenar um endereço de memória, e manipular o conteúdo desses endereços de memória. Tem aplicações como manipulação de vetores, passagem de parâmetro por referência, desenvolvimento de estruturas de dados complexos. Resumindo, um tipo de variável que nos permite exibir uma struct, ou o conteúdo de um vetor ou variável, apenas pelo endereço de memória dele.

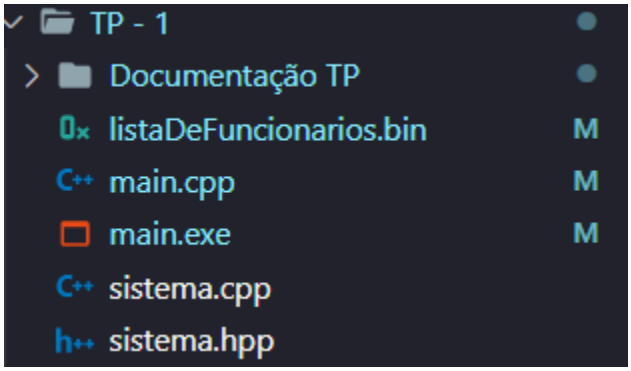
2.1.5. Arquivos

A parte de manipulação de arquivos é mais abstrata, não é convencional para ilustrar ela, mas podemos usar uma analogia para compreender, imagine-se estudando para uma prova, onde se usa um livro e uma folha de papel em branco, o papel é seu arquivo, e o livro é seu programa, quando você quer guardar algo importante do livro, você lê o que está nele e escreve na folha, isso seria semelhante

às funções do ofstream, que pega os dados digitados no programa e escreve no arquivo, independente da extensão deste. Imagine novamente, com um caderno e uma folha apenas, o caderno é o programa, e a folha é o arquivo com o que você leu dele, e não estava em branco quando começou tendo algo escrito na folha, você realiza a leitura e exibe isso no caderno, seria semelhante às funções do ifstream, que pega os dados do arquivo e exibe na tela.

2.2. Implementação

Figura 5 – Organização



O trabalho foi dividido em 5 arquivos, incluindo a main, e o arquivo bin, como apresentado na figura 4. E o arquivo sistema.cpp com seu arquivo sistema.hpp, e o *.exe. E a pasta de documentação, com os arquivos que serão enviados.

O sistema.cpp possui uma parte para as funções básicas para manipulação das TADs(Apêndice A) da Lista Encadeada e da Lista Sequencial.

- void menu(); - Menu que é exibido na tela(Apêndice D)

- void incluiNovoFuncionario(ListaEncadeada *lista); - função para a inclusão de funcionário (Apêndice E)

- Funcionario *criaFuncionario(); - função com cout e cin que retorna o funcionário, para ser inserido(Apêndice F)

- void incluiNovosProjetos(ListaEncadeada *lista); - incluir novos projetos, é para perguntar, se quer incluir projetos, e quantos projetos quer incluir, para inserir de fato o projeto com o incluiNovoProjeto(Apêndice G)

- void incluiNovoProjeto(Funcionario *funcionario); - insere o projeto ao funcionário (Apêndice G)

- void criaProjeto(ListaSequencial *lista, Projeto *projeto); - pega o projeto como parâmetro, e faz alterações nele dentro da função (Apêndice G)

- Funcionario *FuncionarioPorID(ListaEncadeada *lista); - função para pesquisar um funcionário por id dentro da função, e retornar, um funcionário, para manipulação de outras funções (Apêndice H)

- void excluiProjetos(ListaEncadeada *lista); - pesquisa o funcionário na lista encadeada, usa das funções da TAD, para remover o projeto do funcionário (Apêndice I)

- void excluiFuncionariosSemProjetos(ListaEncadeada *lista); - se existir funcionários sem projetos, ou seja com a Lista Sequencial com tamanho = 0, esse funcionário é excluído (Apêndice J)

- void ImprimeListaFunc(ListaEncadeada *lista); - Imprime a lista de funcionários (Apêndice K)

- void consultaFuncionario(ListaEncadeada *lista); - pesquisa um funcionário pelo id dele, e exibe na tela as informações dele (Apêndice K)

- int totalHorasSemanais(ListaSequencial lista); - calcula o somatório de horas semanais, de um funcionário (Apêndice L)

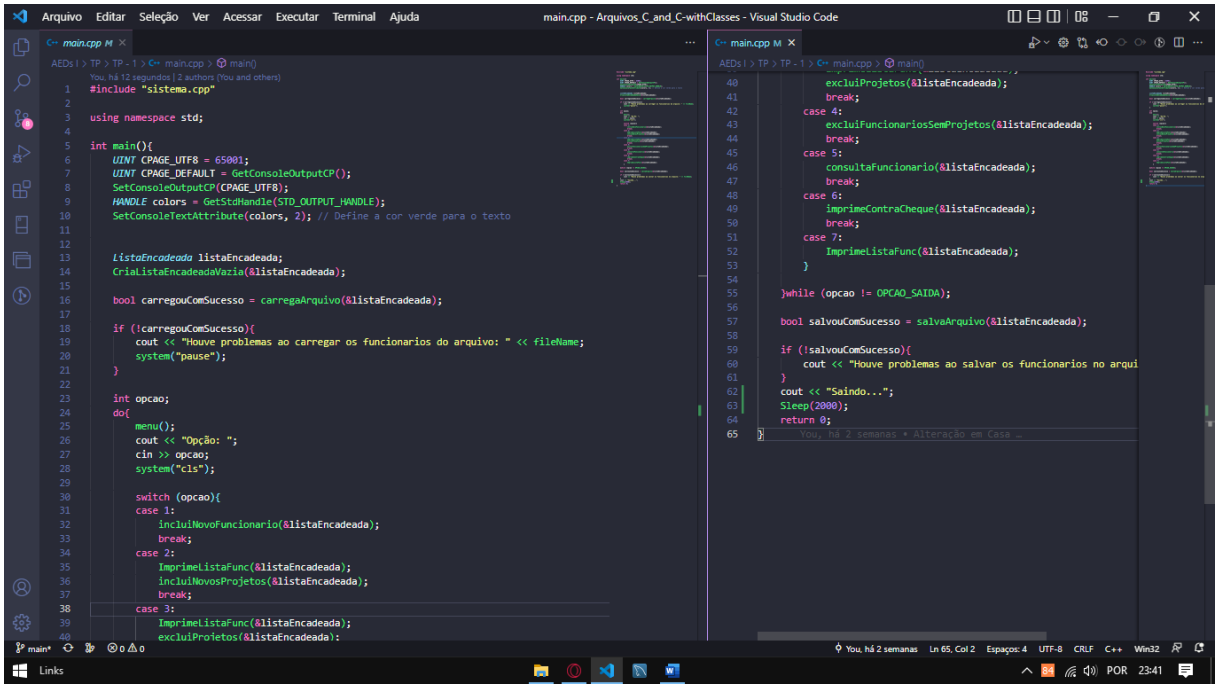
- double calculaSalarioBruto(Funcionario *funcionario); - calcula o salário bruto de acordo com base nas horas semanais (Apêndice L)

- void imprimeContraCheque(ListaEncadeada *lista); - calcula o salário líquido, com base nos descontos no salário bruto e imprime o contracheque (Apêndice L)

- // FUNÇÕES DE MANIPULAÇÃO DE ARQUIVOS
- bool carregaArquivo(ListaEncadeada *lista); - ele pega o que foi cadastrado na struct, e insere na última posição da lista, no momento em que o programa é iniciado (Apêndice C)
- bool salvaArquivo(ListaEncadeada *lista); - ele pega o que foi cadastrado durante a execução e salva no arquivo bin, e deleta ele da memória (Apêndice C)

Após todas as funções, chamo as funções de carregamento e salvamento do arquivo, e as funções exigidas pelo trabalho, como na figura 6.

Figura 6 – Main



3. CONCLUSÃO

Ao longo do trabalho encontrei dificuldades para interpretar o que estava sendo pedido, fui a monitoria para sanar minhas dúvidas, conversei com poucos

colegas de classe, para me auxiliarem na interpretação e na escrita do código do trabalho, fiz uso dos slides disponibilizados pelo professor para sanar dúvidas a respeito das TADs (Tipos Abstratos de Dados), criação de funções de manipulação de arquivos, e também utilizei as informações da biblioteca, sites e fóruns.

Coloquei o trabalho em modularização separando-o em 5 arquivos, 2 arquivos *.cpp, o sistema.cpp, com as funções e a main.cpp com a execução das funções, 1 arquivo *.hpp, o sistema.hpp, onde tinha as structs e o cabeçalho das funções, 1 arquivo *.bin, listaDeFuncionarios.bin onde os funcionários eram salvos, e 1 arquivo *.exe, a main.exe, que seria o executável do código, onde as funções compilavam.

Creio eu que meu desenvolvimento apesar das dificuldades de interpretação, foram ótimos, aprendi coisas novas como funções que retornam um item, manipular arquivos, usar duas Estruturas de Dados para compor 1 só. Achei bastante semelhante ao que fiz no Ensino Médio Técnico Integrado, onde fizemos um sistema para atender à uma pequena empresa. Acho que consegui atingir meus objetivos, e explicar bem o que cada função faz, apresentar conhecimentos em lista encadeada, lista sequencial, manipulação de arquivos, e aplicar isso em um software, que atendesse o mínimo que uma empresa precisaria. Estou feliz com o resultado, e ele atendeu às minhas expectativas além do que eu esperava.

4. REFERÊNCIAS

COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados - Arquivos. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146487/mod_resource/content/1/Aula%203%20-%20Arquivos.pdf . Acesso em: 25 de setembro de 2022.

COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados - Listas. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146499/mod_resource/content/1/Aula%206%20-%20Listas.pdf . Acesso em: 24 de setembro de 2022.

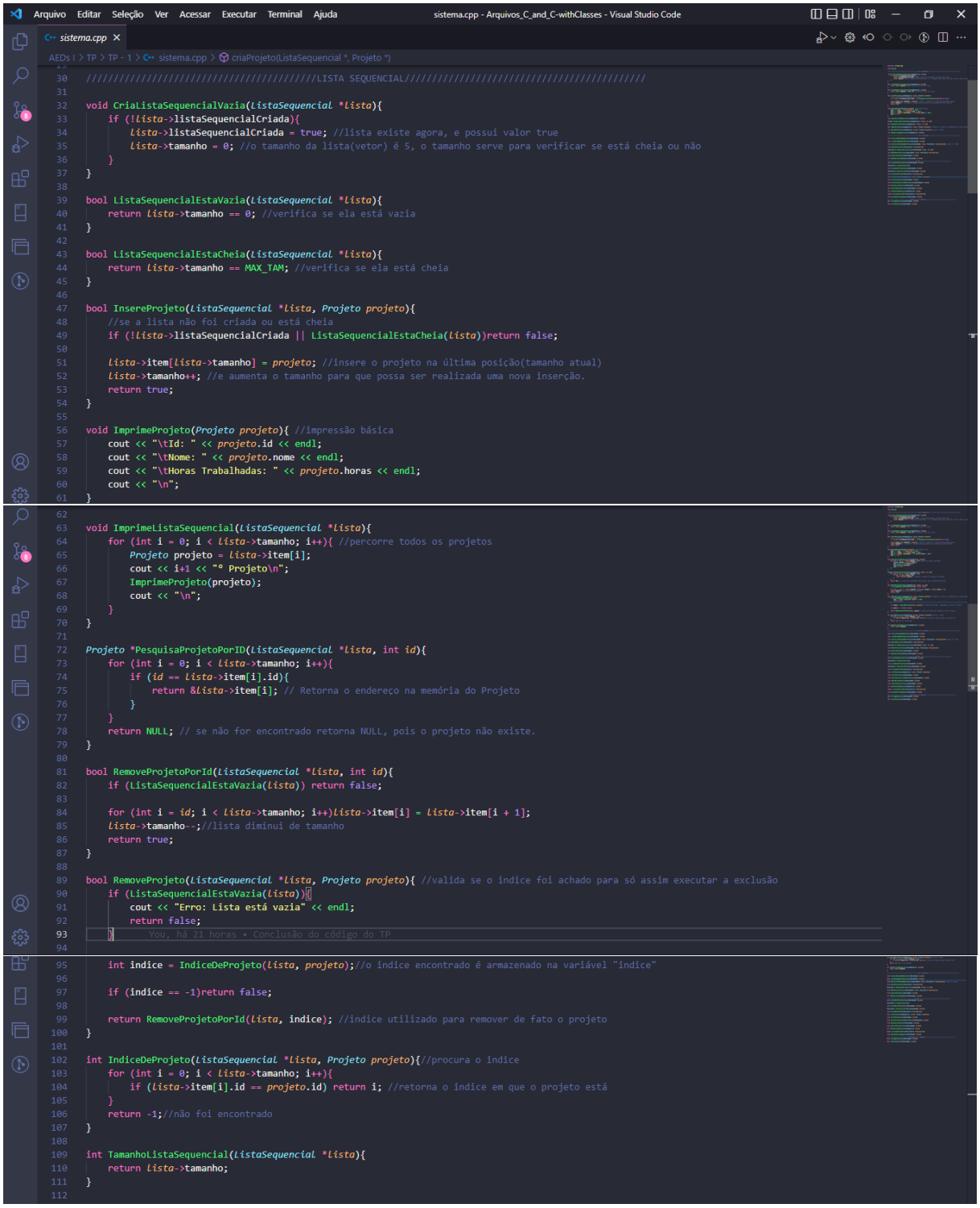
COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados - Listas utilizando Ponteiro. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146500/mod_resource/content/1/Aula%207%20-%20Listas%20Encadeadas.pdf . Acesso em: 26 de setembro de 2022.

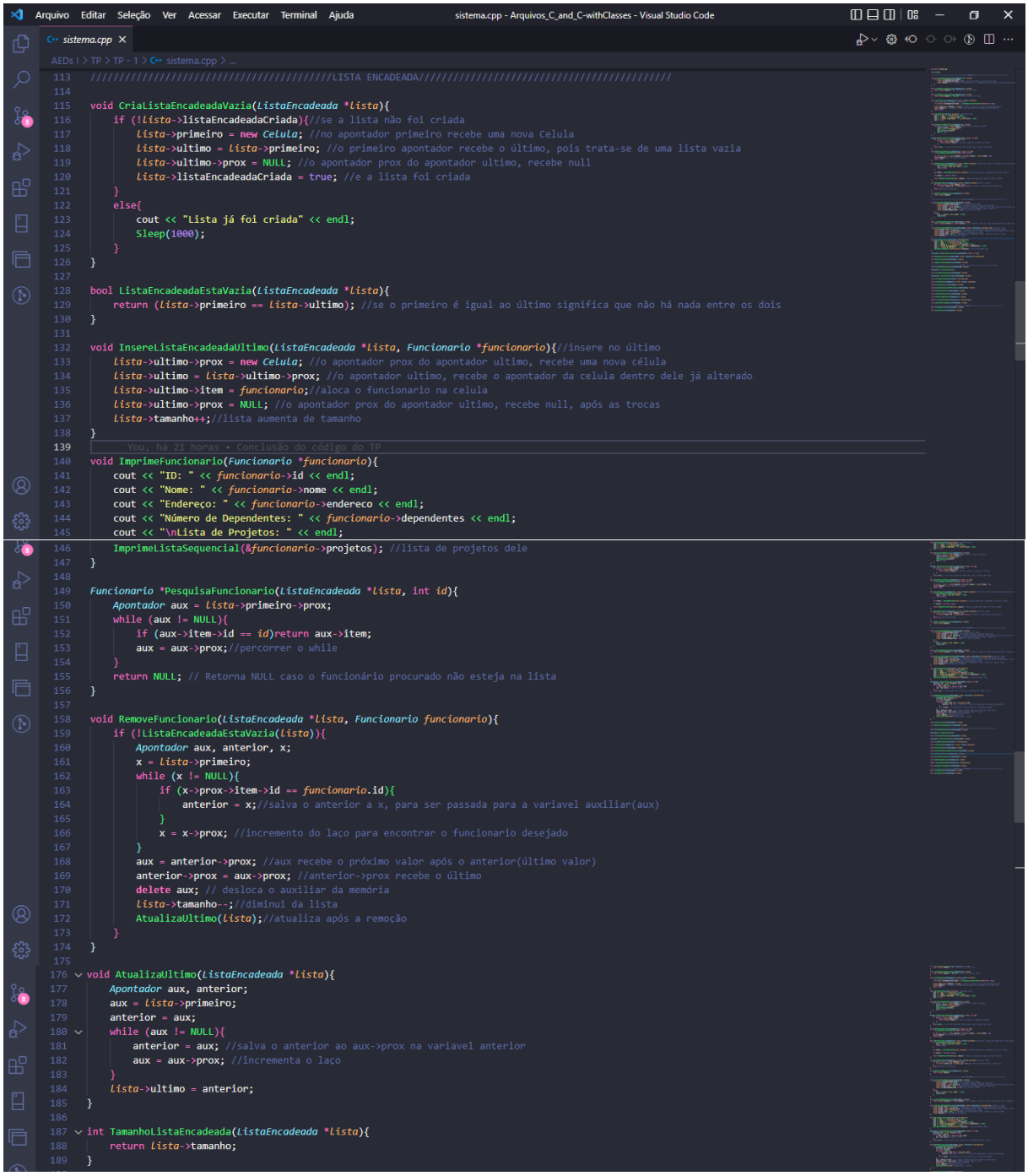
COSTA TRINDADE. Eduardo Augusto. Algoritmos e Estrutura de Dados – Ponteiros. 2022. Apresentação PDF. Disponível em: . Acesso em : 28 de setembro de 2022.

Programar em C++/Entrada e saída de dados 2. **WIKI LIVROS**, 16/04/2020.Disponível em: https://pt.wikibooks.org/wiki/Programar_em_C%2B%2B/Entrada_e_saída_de_dados_2#:~:text=biblioteca%20padrão%20fstreamEditar&text=Esta%20biblioteca%20define%203%20novos,"in%20from%20a%20file"%20. Acesso em: 25 de setembro de 2022.

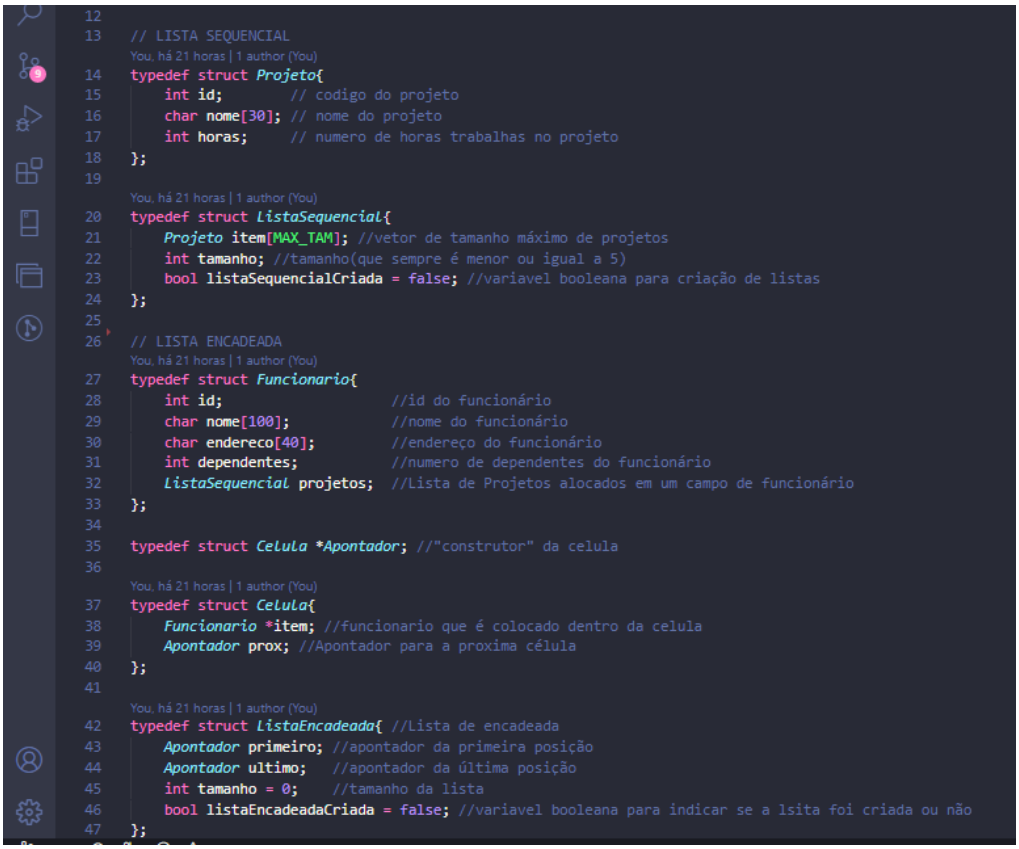
5. APÊNDICES

5.1. APÊNDICE A – Funções das TADs no arquivo.cpp




Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda
sistema.cpp - Arquivos_C_and_C-withClasses - Visual Studio Code
sistema.cpp
AEDs I > TP > TP - 1 > sistema.cpp > ...
//LISTA ENCADEADA//
void CriaListaEncadeadaVazia(ListaEncadeada *lista){
if (!lista->listaEncadeadaCriada){//se a lista não foi criada
lista->primeiro = new Celula; //no apontador primeiro recebe uma nova Celula
lista->ultimo = lista->primeiro; //o primeiro apontador recebe o último, pois trata-se de uma lista vazia
lista->ultimo->prox = NULL; //o apontador prox do apontador ultimo, recebe null
lista->listaEncadeadaCriada = true; //e a lista foi criada
}
else{
cout << "Lista já foi criada" << endl;
Sleep(1000);
}
}
bool ListaEncadeadaEstaVazia(ListaEncadeada *lista){
return (lista->primeiro == lista->ultimo); //se o primeiro é igual ao último significa que não há nada entre os dois
}
void InsereListaEncadeadaUltimo(ListaEncadeada *lista, Funcionario *funcionario){//insere no último
lista->ultimo->prox = new Celula; //o apontador prox do apontador ultimo, recebe uma nova célula
lista->ultimo = lista->ultimo->prox; //o apontador ultimo, recebe o apontador da celula dentro dele já alterado
lista->ultimo->item = funcionario;//aloca o funcionario na celula
lista->ultimo->prox = NULL; //o apontador prox do apontador ultimo, recebe null, após as trocas
lista->tamanho++;//lista aumenta de tamanho
}
You, há 21 horas • Conclusão do código do TP
void ImprimeFuncionario(Funcionario *funcionario){
cout << "ID: " << funcionario->id << endl;
cout << "Nome: " << funcionario->nome << endl;
cout << "Endereço: " << funcionario->endereco << endl;
cout << "Número de Dependentes: " << funcionario->dependentes << endl;
cout << "\nLista de Projetos: " << endl;
ImprimeListaSequencial(&funcionario->projetos); //lista de projetos dele
}
Funcionario *PesquisaFuncionario(ListaEncadeada *lista, int id){
Apontador aux = lista->primeiro->prox;
while (aux != NULL){
if (aux->item->id == id)return aux->item;
aux = aux->prox;//percorrer o while
}
return NULL; // Retorna NULL caso o funcionário procurado não esteja na lista
}
void RemoveFuncionario(ListaEncadeada *lista, Funcionario funcionario){
if (!ListaEncadeadaEstaVazia(lista)){
Apontador aux, anterior, x;
x = lista->primeiro;
while (x != NULL){
if (x->prox->item->id == funcionario.id){
anterior = x;//salva o anterior a x, para ser passada para a variavel auxiliar(aux)
}
x = x->prox; //incremento do laço para encontrar o funcionario desejado
}
aux = anterior->prox; //aux recebe o próximo valor após o anterior(último valor)
anterior->prox = aux->prox; //anterior->prox recebe o último
delete aux; // desloca o auxiliar da memória
lista->tamanho--;//diminui da lista
AtualizaUltimo(lista);//atualiza após a remoção
}
}
void AtualizaUltimo(ListaEncadeada *lista){
Apontador aux, anterior;
aux = lista->primeiro;
anterior = aux;
while (aux != NULL){
anterior = aux; //salva o anterior ao aux->prox na variavel anterior
aux = aux->prox; //incrementa o laço
}
lista->ultimo = anterior;
}
int TamanhoListaEncadeada(ListaEncadeada *lista){
return lista->tamanho;
}

5.2. APÊNDICE B – Sistema.hpp



```
// FUNÇÕES DA LISTA SEQUENCIAL
void CriaListaSequencialVazia(ListaSequencial *lista);
bool ListaSequencialEstaVazia(ListaSequencial *lista);
bool ListaSequencialEstaCheia(ListaSequencial *lista);
bool InsereProjeto(ListaSequencial *lista, Projeto projeto);
void ImprimeProjeto(Projeto projeto);
void ImprimeListaSequencial(ListaSequencial *lista);
Projeto *PesquisaProjetoPorID(ListaSequencial *lista, int id);
bool RemoveProjetoPorId(ListaSequencial *lista, int id);
bool RemoveProjeto(ListaSequencial *lista, Projeto projeto);
int IndiceDeProjeto(ListaSequencial *lista, Projeto projeto);
int TamanhoListaSequencial(ListaSequencial *lista);

// FUNÇÕES DA LISTA ENCADEADA
void CriaListaEncadeadaVazia(ListaEncadeada *lista);
bool ListaEncadeadaEstaVazia(ListaEncadeada *lista);
void InsereListaEncadeadaUltimo(ListaEncadeada *lista, Funcionario *funcionario);
void ImprimeFuncionario(Funcionario *funcionario);
Funcionario *PesquisaFuncionario(ListaEncadeada *lista, int id);
void RemoveFuncionario(ListaEncadeada *lista, Funcionario funcionario);
void AtualizaUltimo(ListaEncadeada *lista);
int TamanhoListaEncadeada(ListaEncadeada *lista);

// FUNÇÕES DO SISTEMA
void menu();
void incluiNovoFuncionario(ListaEncadeada *lista);
Funcionario *criaFuncionario();
void incluiNovosProjetos(ListaEncadeada *lista);
Funcionario *FuncionarioPorID(ListaEncadeada *lista);
void incluiNovoProjeto(Funcionario *funcionario);
void criaProjeto(ListaSequencial *lista, Projeto *projeto);
void excluiProjetos(ListaEncadeada *lista);
void excluiFuncionariosSemProjetos(ListaEncadeada *lista);
void ImprimeListaFunc(ListaEncadeada *lista);
void consultaFuncionario(ListaEncadeada *lista);
int totalHorasSemanais(ListaSequencial lista);
double calculaSalarioBruto(Funcionario *funcionario);
void imprimeContraCheque(ListaEncadeada *lista);
    You, há 21 horas • Conclusão do código do TP
```

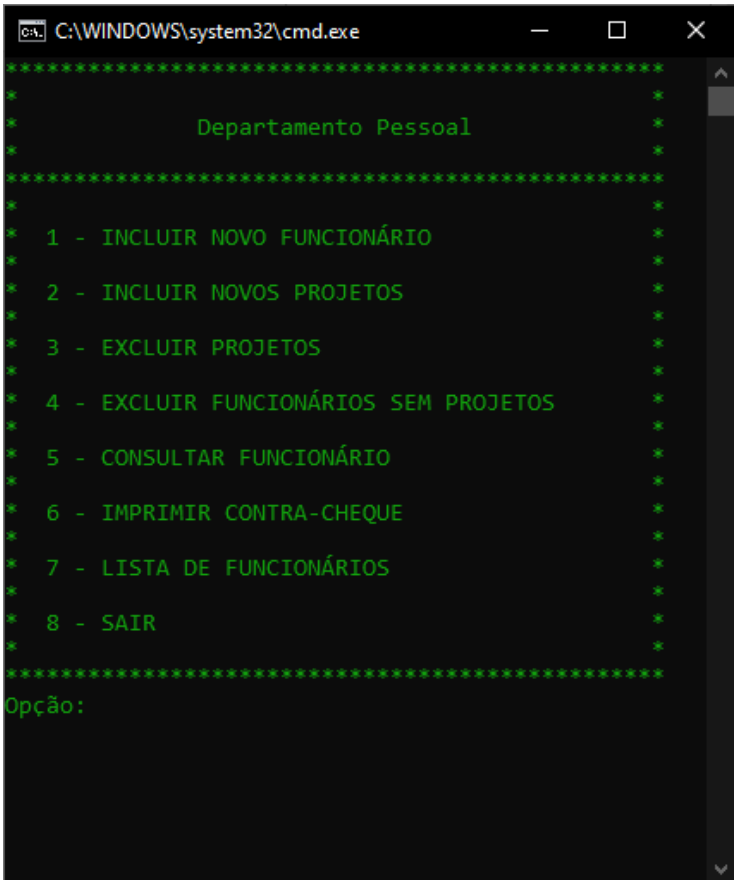
```
// FUNÇÕES DE MANIPULAÇÃO DE ARQUIVOS
bool carregaArquivo(ListaEncadeada *lista);
bool salvaArquivo(ListaEncadeada *lista);
```

5.3. APÊNDICE C – Manipulação de arquivos

```
//////////////////////////// MANIPULAÇÃO DE ARQUIVOS ////////////////////////////////////
bool carregaArquivo(ListaEncadeada *lista){
    ifstream arquivo;
    arquivo.open(fileName, fstream::binary); // Abre o arquivo em modo binario, ou cria se não existir

    if (arquivo.fail()){ // Se falhar
        arquivo.clear(); // Fecha o arquivo
        return false;
    }
    while (arquivo.peek() != ifstream::traits_type::eof()){ // Enquanto o arquivo nao for vazio
        Funcionario *funcionario = new Funcionario;
        arquivo.read((char *)funcionario, sizeof(Funcionario)); // Le a Struct dos funcionarios completos
        InsereListaEncadeadaUltimo(lista, funcionario); // Insere o projeto que pegou do arquivo na lista encadeada
    }
    arquivo.clear();
    return true;
}

bool salvaArquivo(ListaEncadeada *lista){
    ofstream arquivo;
    arquivo.open(fileName, ofstream::trunc | fstream::binary); // Abre arquivo em modo trunc pra deixa-lo vazio e em modo binario

    if (arquivo.fail()){
        arquivo.clear();
        return false;
    }

    Apontador aux = lista->primeiro->prox;
    while (aux != NULL){
        Funcionario *funcionario = aux->item;
        arquivo.write((char *)funcionario, sizeof(Funcionario)); // Escreve no arquivo a Struct do funcionario completa
        delete funcionario;
        aux = aux->prox;
    }
    return true;
}
```

5.4. APÊNDICE D – Menu

```
You, há 21 horas | 1 author (You)
#include "sistema.hpp"

void menu(){
    system("cls");
    cout << "************************************************" << endl;
    cout << "*                                              *" << endl;
    cout << "*            Departamento Pessoal              *" << endl;
    cout << "*                                              *" << endl;
    cout << "************************************************" << endl;
    cout << "*                                              *" << endl;
    cout << "*  1 - INCLUIR NOVO FUNCIONÁRIO                *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  2 - INCLUIR NOVOS PROJETOS                  *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  3 - EXCLUIR PROJETOS                        *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  4 - EXCLUIR FUNCIONÁRIOS SEM PROJETOS       *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  5 - CONSULTAR FUNCIONÁRIO                   *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  6 - IMPRIMIR CONTRA-CHEQUE                  *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  7 - LISTA DE FUNCIONÁRIOS                   *" << endl;
    cout << "*                                              *" << endl;
    cout << "*  8 - SAIR                                    *" << endl;
    cout << "*                                              *" << endl;
    cout << "************************************************" << endl;
}
```



5.5. APÊNDICE E – Função de Inserção de funcionário

```
192 void incluiNovoFuncionario(ListaEncadeada *lista){
193     Funcionario *funcionario = criaFuncionario();
194 
195     if (PesquisaFuncionario(lista, funcionario->id) == NULL){
196         InsereListaEncadeadaUltimo(lista, funcionario); //insere no fim da lista, isso vem lá da listaEncadeada.cpp
197         cout << "\nFuncionário incluído com Sucesso!\n" << endl;
198         int opcao;
199         do{
200             cout << "Deseja adicionar projetos ao funcionário?" << endl;
201             cout << "1 - Sim" << endl;
202             cout << "2 - Não" << endl;
203             cin >> opcao;
204             if (opcao == 1){
205                 system("cls");
206                 int quantidade;
207                 cout << "Quantidade de projetos que deseja adicionar: ";
208                 cin >> quantidade;
209                 for (int i = 0; i < quantidade; i++) incluiNovoProjeto(funcionario);
210             }
211             system("cls");
212         } while (opcao != 1 && opcao != 2);
213     }
214     else{ //caso exista um funcionario com o mesmo ID, que quero inserir
215         cout << "\nJá existe funcionário com ID: " << funcionario->id << endl;
216         cout << "Adicione novamente com outro ID" << endl;
217         delete funcionario; // desloca o funcionario da memória, já que ele não será usado
218         Sleep(1000);
219     }
220 }
```

5.6. APÊNDICE F – Função de retorno de funcionário, para inserção

```
222 Funcionario *criaFuncionario(){
223         Funcionario *funcionario = new Funcionario;
224         cout << "Cadastro Funcionário" << endl;
225         cout << "ID: ";
226         cin >> funcionario->id;
227         cin.ignore();
228         cout << "Nome: ";
229         cin.getline(funcionario->nome, 100);
230         cout << "Endereço: ";
231         cin.getline(funcionario->endereco, 40);
232         cout << "Número de Dependentes: ";
233         cin >> funcionario->dependentes;
234         CriaListaSequencialVazia(&funcionario->projetos);
235         return funcionario;
236 }
```

5.7. APÊNDICE G – Inclusão de Projeto

```
237
238  void incluiNovosProjetos(ListaEncadeada *lista){
239      if (!ListaEncadeadaEstaVazia(lista)){
240          Funcionario *funcionario = FuncionarioPorID(lista);
241
242          if (funcionario != NULL){ // Só adiciona se o funcionario existir
243              int opcao = 1;
244              do{
245                  if (opcao == 1)incluiNovoProjeto(funcionario);
246                  system("cls");
247                  cout << "Deseja adicionar outro projeto para esse funcionário?\n";
248                  cout << "1-Sim" << endl;
249                  cout << "2-Não" << endl;
250                  cin >> opcao;
251                  system("cls");
252              }while (opcao != 2);
253          }
254          else{
255              cout << "\nNão existe funcionário com esse id";
256              Sleep(1000);
257          }
258      }
259      else{
260          cout << "Erro: Funcionário não cadastrado!" << endl;
261          Sleep(1000);
262      }
263  }
264
265  void criaProjeto(ListaSequencial *lista, Projeto *projeto){
266      cout << "Cadastro de Novo Projeto" << endl;
267      cout << "ID: ";
268      cin >> projeto->id;
269      cin.ignore();
270      cout << "Nome: ";
271      cin.getline(projeto->nome, 30);
272      cout << "Horas Trabalhadas: ";
273      cin >> projeto->horas;
274      cin.ignore();
275  }
```

```
277  void incluiNovoProjeto(Funcionario *funcionario){
278      Projeto projeto;
279      criaProjeto(&funcionario->projetos, &projeto);
280      if (PesquisaProjetoPorID(&funcionario->projetos, projeto.id) == NULL){ // TRUE se o projeto não está na lista, FALSE do contrário
281          bool inseridoComSucesso = InsereProjeto(&funcionario->projetos, projeto);
282          if (inseridoComSucesso) cout << "\nProjeto inserido com sucesso" << endl;
283          else cout << "\nEsse funcionário atingiu o máximo de projetos possíveis";
284      }
285      else cout << "\nJá existe Projeto com ID: " << projeto.id << " nesse funcionário.\nAdicione novamente com outro ID" << endl;
286      Sleep(1000);
287  }
```

5.8. APÊNDICE H – Pesquisa funcionário por ID

```
289  Funcionario *FuncionarioPorID(ListaEncadeada *lista){
290      int idFuncionario;
291      cout << "ID do Funcionário: ";
292      cin >> idFuncionario;
293      Funcionario *funcionario = PesquisaFuncionario(lista, idFuncionario);
294      return funcionario;
295  }
```

5.9. APÊNDICE I – Exclui Projetos

```
297 void excluiProjetos(ListaEncadeada *lista){
298     if (!ListaEncadeadaEstaVazia(lista)){
299         Funcionario *funcionario = FuncionarioPorID(lista); // Retorna ou o funcionário ou NULL
300 
301         if (funcionario != NULL){
302             int idProjeto;
303             cout << "ID do Projeto: ";
304             cin >> idProjeto;
305 
306             Projeto *projeto = PesquisaProjetoPorID(&funcionario->projetos, idProjeto);
307             if (projeto == NULL) cout << "\nProjeto com id igual à " << idProjeto << " não encontrado" << endl;
308             else{
309                 bool removeuComSucesso = RemoveProjeto(&funcionario->projetos, *projeto);
310                 if (removeuComSucesso) cout << "\nProjeto com id: " << idProjeto << " foi removido com sucesso" << endl;
311                 else  cout << "\nErro: Não foi possível excluir projeto com id: " << idProjeto << endl;
312             }
313         }
314         else cout << "\nNão existe funcionário com esse Id" << endl;
315     }
316     else  cout << "Erro: Nenhum Funcionário cadastrado na lista" << endl;
317     Sleep(2000);
318 }
```

5.10. APÊNDICE J – Exclui funcionários sem Projeto

```
320 void excluiFuncionariosSemProjetos(ListaEncadeada *lista){
321     if (!ListaEncadeadaEstaVazia(lista)){
322         Apontador aux = lista->primeiro->prox;
323         Apontador anterior = aux;
324         while (aux != NULL && PesquisaFuncionario(lista, aux->item->id) != NULL){
325             if (TamanhoListaSequencial(&aux->item->projetos) == 0){
326                 RemoveFuncionario(lista, *aux->item);
327                 aux = anterior;
328             }
329             anterior = aux;
330             aux = anterior->prox;
331         }
332         cout << "Todos os funcionário sem projetos foram apagados." << endl;
333         Sleep(2000);
334         }
335     else  cout << "Erro: Nenhum Funcionário cadastrado" << endl;
336     Sleep(2000);
337 }
```

5.11. APÊNDICE K – Impressão de Funcionário

```
void ImprimeListaFunc(ListaEncadeada *lista){
    if(ListaEncadeadaEstaVazia(lista)){ //verifica se o endereço da lista é true
        cout << "Lista Vazia";
        Sleep(1000);
        return ;
    }
    Apontador aux;
    aux = lista->primeiro->prox; //recebe o apontador de próximo, depois do primeiro
    while(aux != NULL){ //enquanto não for nulo, ele imprime
        ImprimeFuncionario(aux->item);
        aux = aux->prox;//aux sendo 'incrementado'
    }
    Sleep(5000);
}

void consultaFuncionario(ListaEncadeada *lista){
    if (!ListaEncadeadaEstaVazia(lista)){
        Funcionario *funcionario = FuncionarioPorID(lista);

        if (funcionario != NULL) ImprimeFuncionario(funcionario);
        else cout << "\nNão existe funcionário com esse Id";
    }
    else cout << "Erro: Nenhum Funcionário cadastrado" << endl;
    Sleep(5000);
}
```

5.12. APÊNDICE L – Cálculo de Salário Bruto

```
int totalHorasSemanais(ListaSequencial lista){
    int totalHoras = 0;
    for (int i = 0; i < lista.tamanho; i++){
        Projeto projeto = lista.item[i];
        totalHoras += projeto.horas;
    }
    return totalHoras;
}

double calculaSalarioBruto(Funcionario *funcionario){
    double salarioBruto = 0;
    int totalHoras = totalHorasSemanais(funcionario->projetos);
    salarioBruto = (totalHoras * 45) + (35 * funcionario->dependentes);
    return salarioBruto;
}
```

```
void imprimeContraCheque(ListaEncadeada *lista){
    if (!ListaEncadeadaEstaVazia(lista)){
        Apontador aux;
        aux = lista->primeiro->prox;

        while (aux != NULL){
            Funcionario *funcionario = aux->item;
            int totalHoras;
            double descontoINSS, salarioBruto, descontoIR, salarioLiquido;
            salarioBruto = calculaSalarioBruto(funcionario);
            descontoINSS = (8.5 / 100.0) * salarioBruto;
            descontoIR = (15.0 / 100.0) * salarioBruto;
            salarioLiquido = salarioBruto - descontoINSS - descontoIR;
            totalHoras = totalHorasSemanais(funcionario->projetos);
            //impressão
            cout << "Id: " << funcionario->id;
            cout << "\nNome: " << funcionario->nome;
            cout << "\nTotal de Horas Semanais: " << totalHoras;
            cout << "\nSalário Bruto: " << salarioBruto;
            cout << "\nDesconto do INSS: " << descontoINSS;
            cout << "\nDesconto de Imposto de renda: " << descontoIR;
            cout << "\nSalário líquido: " << salarioLiquido;
            cout << "\n";
            aux = aux->prox;
        }
    }
    else  cout << "Erro: Nenhum Funcionário cadastrado" << endl;
    Sleep(2000);
}
```