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

JOÃO VITOR MENDES CAMPOS

TRABALHO PRÁTICO I

SÃO JOÃO EVANGELISTA
2022

JOÃO VITOR MENDES CAMPOS

LISTA ENCADEADA E LISTA COM ARRANJO

SÃO JOÃO EVANGELISTA
2022

SUMÁRIO

1. INTRODUÇÃO

O presente trabalho é sobre Lista encadeada e lista com arranjo, mais concretamente sobre como são utilizados em programas como: cadastros de funcionários e projetos. São objetivos deste trabalho o entendimento e aperfeiçoamento das matérias que foram apresentadas em sala.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral a construção de um programa de cadastros de funcionários e projetos, onde há o cadastro do funcionário, criação de um projeto, a exclusão de funcionário e a apresentação do contra - cheque.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Objetivo 1. Prática de listas(encadeada e com arranjo).
- Objetivo 2. Construção de um programa(Cadastro de funcionários).
- Objetivo 3. Entendimento de como funciona um Cadastro de funcionário(código).

1.3. Justificativa

Esse trabalho é justificado devido aos assuntos abordados em sala de aula, como uma fixação do conteúdo aprendido durante as aulas de AEDS I. E com o objetivo da distribuição de pontos para que o estudante passe na disciplina.

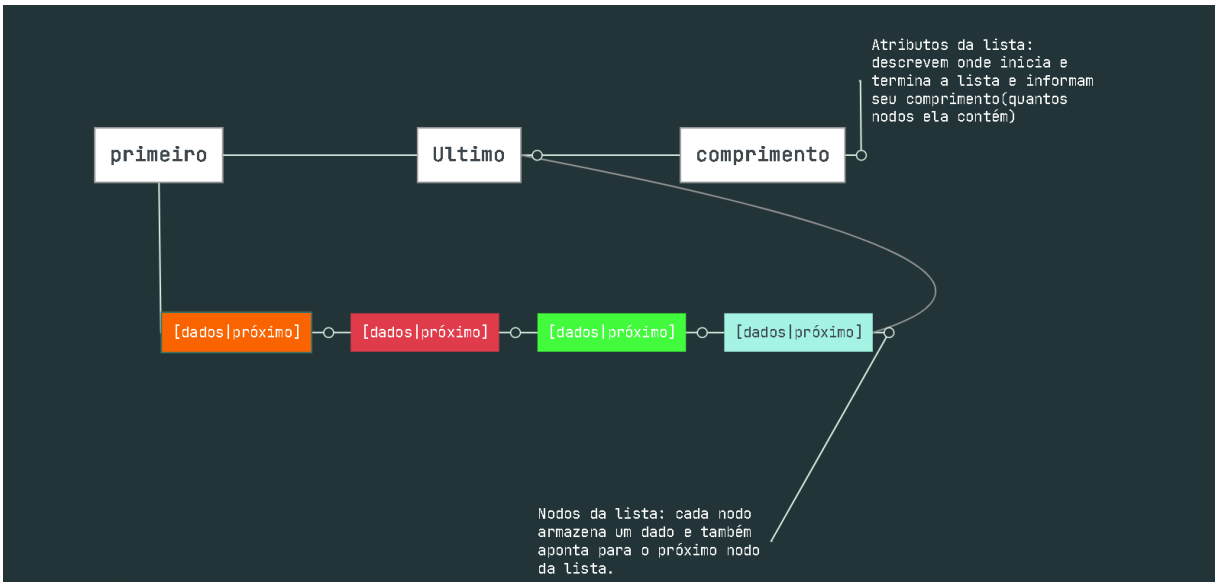
2. DESENVOLVIMENTO

Neste capítulo é apresentado o desenvolvimento do trabalho sobre listas. Lista encadeada é uma lista uma representação de uma sequência de objetivos, todos do mesmo tipo, na memória RAM(=random access memory) do computador. Cada elemento de sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, segundo na segunda, e assim por diante.

Lista com arranjo é uma estrutura de dados na qual elementos de um mesmo tipo de dado estão organizados de forma sequencial.

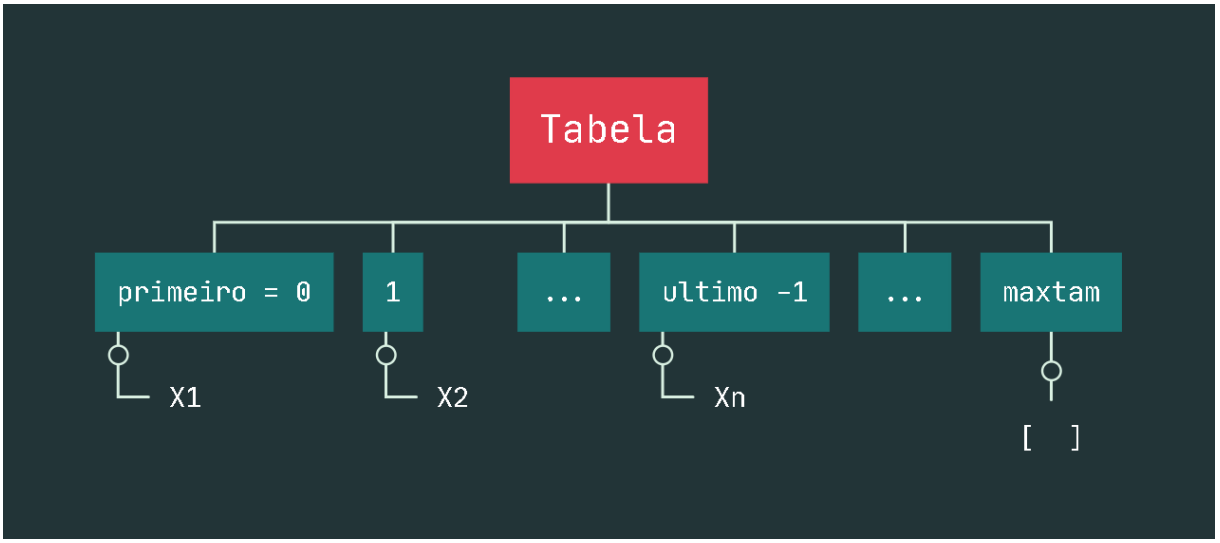
2.1. Lista Encadeada e Lista com arranjo

Figura 1 – Lista Encadeada



A Figura 1 apresenta uma estrutura de lista encadeada, pela imagem pode-se notar que os atributos Primeiro e Último mostram respectivamente o início e o término da lista, já o atributo Comprimento recebe o tamanho da lista que é baseado em quantos elementos a lista tem ao final de uma inserção.

Figura 2 - Lista com Arranjo



A figura 2 apresenta uma estrutura de lista com arranjo, pode-se perceber que é constituída por um vetor onde o primeiro item X1 da lista estará na posição 0 e os demais itens ao longo dessa lista até o último espaço a ser ocupado que é determinado pelo MAXTAM, nessa lista o atributo Último Xn representa o item que foi adicionado por último. Por meio dessa figura fica fácil notar como são as estruturas das listas. O diagrama da TAD trabalhada no trabalho está localizado no Apêndice A.

2.2. Implementação

Na parte de implementação foi utilizada a TAD disponibilizada pelo professor, que continha as funções básicas que foram estudadas em sala, como: Criar Lista Vazia, Verifica Lista Vazia, Inserir , Pesquisa. Essas são as funções básicas da TAD.

As novas funções que foram implementadas foram: Cadastrar Funcionário, Consulta Funcionário, Exclui Funcionário, Deleta Funcionário, Cadastra Projeto, Exclui Projetos, Apaga Projetos, essa são as funções usada para a manipulação dos funcionários e dos projetos, agora a funções para o cálculo do contracheque foram: Calcula Horas Semanais, Calcula Salário Bruto, Dados Funcionário, Imprime Projetos, Quant Projetos e Imprime ContraCheque.

Cadastrar_Funcionário: o parâmetro usado é a Lista encadeada, fica assim void CadatraFuncionário(TListaencadeada *listaE), quando essa opção é selecionada é pedido para entrar com o código do funcionário, após isso há uma verificação para saber se esse código já não foi utilizado por outro funcionário, se o código foi utilizado por outro funcionário o programa pedirá para que entre com um código diferente, passado por essa verificação será pedido o nome, endereço e o número de dependentes do funcionário. Mas se o usuário entra com algum caracter diferente de um número na parte de dependentes, o programa pedirá para que o usuário digite somente números, somente depois que o usuário entrar com números o programa irá inserir o cadastro na lista, por meio da função Inserir. Apêndice B

Consultar_Funcionário: o parâmetro usado é a Lista encadeada, ficando assim void ConsultaFuncionário(TListaEncadeada *listaE), essa função tem como utilidade a consulta, é pedido o código do funcionário, logo após a inserção do

código é feito uma pesquisa, utilizando a função Pesquisa. Se o resultado da pesquisa for 1 será imprimido os dados do funcionário e se ele tiver algum projeto será exibido juntos dos dados, esse projetos serão impressos pela função Imprime projetos, porém se o resultado da pesquisa for qualquer outro valor será exibido que o funcionário não foi encontrado. Apêndice C

Exclui_Funcionário: parametro usado Lista encadeada, função void ExcluiFuncionário(TListaEncadeada *listaE), essa função tem o objetivo de excluir qualquer funcionário que não tenha nenhum projeto. Essa função faz a verificação de cada funcionário, enquanto não verificar todos o funcionários ela não para de ser executada, após a verificação ela deleta os funcionários que não tem nenhum projeto, a deleção é feita por meio da função Deleta Funcionário. Apêndice D

Deletar_Funcionário: parâmetro usados Lista encadeada, Apontador X, void Funcionário,DeletaFuncionário(TApontador x, TListaEncadeada *listaE, TFuncionário *fun), tem como objetivo deletar o funcionário, mas antes há um verificação pra ver se a Lista em que os funcionário são inseridos não está vazia, para isso usa-se a função VerificaListaVazia. Após a verificação um apontador Q recebe a posição do funcionário, recebe também os dados dele. Depois essa posição do funcionário será deletada. Apêndice E

Cadastra_Projeto: parâmetro usado é a Lista encadeada, void CadastraProjeto(TListaEncadeada *listaE), objetivo cadastrar os projetos, entra com o código do funcionário, ocorre uma pesquisa para encontrar o funcionário se encontrado começa o cadastro do projeto, é pedido código, nome, e quantidade de horas do projeto, se inserir caracteres inválidos na quantidade de horas será pedido para que reinsira novamente só que um valor numérico. Após isso o projeto é inserido na lista com arranjo do funcionário, assim o projeto é cadastrado. Apêndice F

Exclui_Projeto: parametro usado é a Lista encadeada, void ExcluiProjeto(TListaEncadeada *listaE), objetivo excluir um projeto, entra com código do funcionário, realização da pesquisa, impressão dos dados, verificando quantidade de projetos,entra com o código do projeto que deseja excluir, função

Apaga Projeto, apaga o projeto do funcionário e pergunta se deseja excluir mais algum. Apêndice G

Apaga_Projeto: parametro usado Apontador prt, Lista com arranjo *listaS, Chave cod, Projeto *proj, void ApagaProjeto(TApontador prt, TListaSequencial *listaS, TChave cod, TProjeto *proj), objetivo apagar um projeto, realiza uma pesquisa na lista dos projetos(lista com arranjo), o *proj vai receber os dados do projeto a ser apagado e depois por meio de um laço de repetição vai havendo uma troca de dados na lista de projetos. Ao final será removido um ou mais projeto. Apêndice H

Calcula_Horas_Semanais: parametro usado é a Lista com arranjo, void CalculaHorasSemanais(TListaSequencial *listaS), objetivo calcular a horas trabalhadas, por meio de um laço de repetição vai se somando as horas. Apêndice I

Calcula_Salario_Bruto: parametro usados Inteiro Total de horas, Apontador x, void CalculaSalarioBruto(int TotaldeHoras, TApontador x), objetivo calcular o salário bruto. Salário Bruto = (((45 * TotaldeHoras) * 4) + (35 * numero de dependentes)). Apêndice J

Dados_Funcionário: parametro usados Chave cod, Lista encadeada, void DadosFuncionário(TChave cod, TListaEncadeada listaE), objetivo mostra os dados dos funcionários, realiza uma pesquisa na lista de funcionários, se encontra exibi os dados. Apêndice K

Imprime_Projetos: parametros usado é a Lista com arranjo, void ImprimeProjetos(TListaSenquencial *listaS), objetivo imprime os projetos, por meio de laço de repetição os dados são imprimidos. Apêndice L

Quant_Projetos: paramentro usado é a Lista com arranjo, void QuantProjetos(TListaSenquencial *listaS), objetivo fazer a contagem dos projetos, por meio de um laço de repetição que verifica se o nome, o código e as horas foraam inseridas, se foram ele dá cont ++ assim fazendo com que a quantidade de projetos aumente. Apêndice M

Imprime_ContraCheque: parametros usado é a Lista encadeada, void ImprimeContraCheque(TListaEncadeada *listaE), objetivo imprimir o contracheque dos funcionários, passa por uma verificação para ver se a lista está vazia, se não estiver vazia, por meio de um laço de repetição vão ser imprimidos os contracheques até que o prox seja igual a null. A partir disso serão impressos o código, nome, total de horas, salário bruto, o desconto de IR, desconto do INSS e o salário líquido. Apêndice N

A modularização foi feita por meio da main.cpp, lista.hpp e funcoes.cpp, os dados deste trabalho são salvo em um arquivo chamado funcionarios.bin, a parte de arquivo é realizada na main onde ele é criado, aberto, escrito e salvo.

3. CONCLUSÃO

Dessa forma podemos finalizar esse trabalho, o intuito era criar um programa de gestão de funcionários, praticar os métodos de listas e entender como é feito o código de um Cadastro de Funcionário. Portanto concluo que todos os objetivos foram alcançados.

REFERÊNCIAS

Definição_de_lista_encadeadahttps://www.ime.usp.br/~pf/algoritmos/aulas/lista.html#:~:text=Uma%20lista%20encadeada%20%C3%A9%20uma,segunda%2C%20e%20assim%20por%20diante.

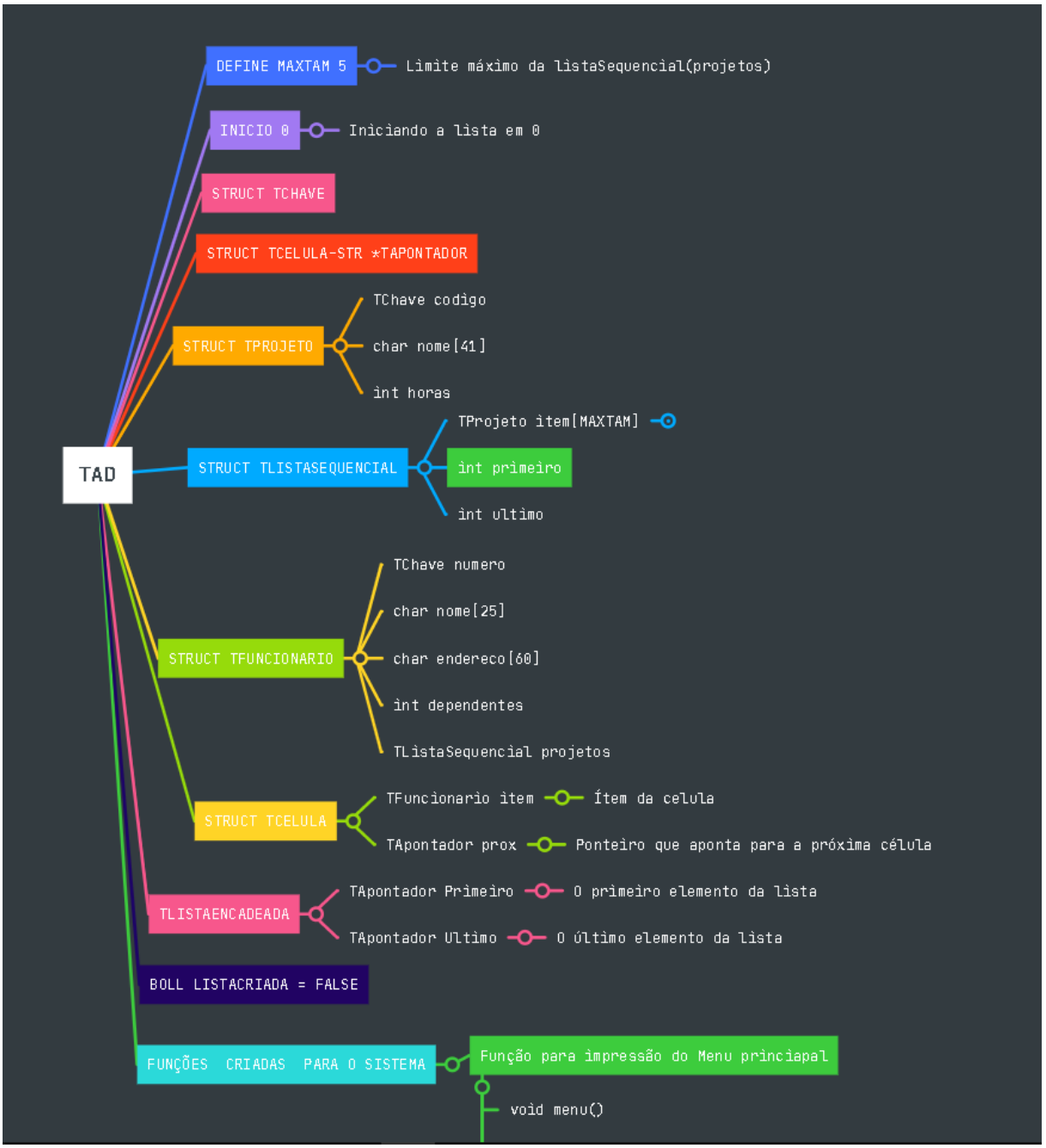
Definição_de_lista_com_arranjohttps://www.cos.ufrj.br/~rfarias/cos121/aula_10.html#:~:text=Lista%20linear%20%C3%A9%20uma%20estrutura,uma%20ordem%20l%C3%B3gica%20entre%20eles.

Figura 1 https://www.mindmeister.com/map/2433334115

Figura 2 https://www.mindmeister.com/map/2433387032

Apêndice A https://www.mindmeister.com/map/2433269134

APÊNDICE A – (Diagrama TAD)




void CriaListaVaziaEncadeada(TListaEncadeada *listaE)
void InserirEncadeada(TFuncionario fun, TListaEncadeada *listaE)
int PesquisaListaEncadeada(TChave cod, TListaEncadeada listaE, TApontador *ptr)
void CriaListaVaziaSequencial(TListaSequencial *listaS)
void InsereSequencial(TProjeto proj, TListaSequencial *listaS)
bool VerificaListaVaziaSequencial(TListaSequencial listaS)
int PesquisaListaSequencial(TListaSequencial *listaS, TChave cod)
Funções para a manipulação do dados dos funcionários e seu projetos
void CadastraFuncionario(TListaEncadeada *listaE)
void ConsultaFuncionario(TListaEncadeada *listaE)
void ExcluiFuncionario(TListaEncadeada *listaE)
void DeletaFuncionario(TApontador x, TListaEncadeada *listaE, TFuncionario *fun)
void CadastraProjetos(TListaEncadeada *listaE)
void ExcluiProjetos(TListaEncadeada *listaE)
void ApagaProjeto(TApontador ptr, TListaSequencial *listaS, TChave cod, TProjeto *proj)
Funções para a realização de cálculo e impressão do contra-cheque
int CalculaHorasSemanais(TListaSequencial listaS)
float CalculaSalarioBruto(int TotaldeHoras, TApontador x)
void DadosFuncionario(TChave cod, TListaEncadeada *listaE)
void ImprimeProjetos(TListaSequencial listaS)
int QuantProjetos(TListaSequencial listaS)
void ImprimeContraCheque(TListaEncadeada *listaE)

APÊNDICE B – (Cadastro de funcionários)

```
void CadastraFuncionario(TListaEncadeada *listaE) // Função para cadastrar os funcionários
{
    TFuncionario fun;
    TApontador x = listaE->Primeiro;

    cout << "*****************************************" << endl;
    cout << "**    TELA DE  CADASTRO DE FUNCIONÁRIO   **" << endl;
    cout << "*****************************************" << endl;
    cout << "Informe um Código para o Funcionário: ";
    cin >> fun.numero;

    // Verifica se já existe algum funcionário com o mesmo código
    while (x->prox != NULL)
    {
        if (x->prox->item.numero == fun.numero)
        {
            cout << "O Código já existe, favor digitar outro: ";
            cin >> fun.numero;
            x = listaE->Primeiro;
        }
        else
        {
            x = x->prox;
        }
    }
    cin.ignore();
    fflush(stdin);
    cout << "Digite o nome do funcionário: ";
    cin >> fun.nome;
    cout << "Digite o endereço do funcionário: ";
    cin >> fun.endereco;
    while ((cout << "Infome o número de dependentes: ") && !(cin >> fun.dependentes)) //Verifica se o usúario digitou algo sem ser número
    {
        cout << "Você inseriu um valor não numérico." << endl;
        cin.clear();
        cin.ignore();
    }
```

APÊNDICE C – (Consulta de funcionários)

```
void ConsultaFuncionario(TListaEncadeada *listaE) //Função que informa a codigo, nome, endereço, número de dependentes e os projetos do funcion
{
    TChave cod;
    TApontador ptr;
    system("cls");
    int ret;

    cout << "*****************************************" << endl;
    cout << "**    TELA DE CONSULTA DE FUNCIONÁRIO    **" << endl;
    cout << "*****************************************" << endl << endl;
    cout << "Informe o código do funcionário: ";
    cin >> cod;
    system("cls");

    ret = PesquisaListaEncadeada(cod, *listaE, &ptr);

    if (ret == 1)
    {
        cout << "Código: " << ptr->prox->item.numero << endl;
        cout << "Nome: " << ptr->prox->item.nome << endl;
        cout << "Endereço: " << ptr->prox->item.endereco << endl;
        cout << "Dependentes: " << ptr->prox->item.dependentes << endl;

        cout << endl << "Projetos do Funcionário: " << endl;

        ImprimeProjetos(ptr->prox->item.projetos);

        system("pause");
        system("cls");

        cout << endl;
    }
    else
    {
        cout << endl << "Funcionário não encontrado!" << endl << endl;
        system("pause");
        system("cls");
    }
```

APÊNDICE D – (Exclui funcionários)

```
void ExcluiFuncionario(TListaEncadeada *listaE) // Função para excluir funcionário
{
    TFuncionario fun;
    TApontador x = listaE->Primeiro;
    int cont = 0, ret_proj;

    while (x->prox != NULL)
    {
        ret_proj = QuantProjetos(x->prox->item.projetos);

        if (ret_proj == 0)
        {
            DeletaFuncionario(x, listaE, &fun);
            cont++;
        }
        else
        {
            x = x->prox;
        }
    }
    system("cls");
    cout << "****************************************" << endl;
    cout << "*    TELA DE EXCLUSÃO DE FUNCIONÁRIOS     *" << endl;
    cout << "****************************************" << endl << endl;
    cout << endl << cont << " Funcionário(s) excluído(s) com sucesso!" << endl << endl;
    system("cls");
}
```

APÊNDICE E – (Deleta funcionários)

```
void DeletaFuncionario(TApontador x, TListaEncadeada *listaE, TFuncionario *fun) //Função que deleta o funcionário
{
    TApontador q;

    if ((VerificaListaVaziaEncadeada(*listaE)) || (x == '\0') || (x->prox == '\0'))
    {
        cout << "Lista vazia!";
    }
    else
    {
        q = x->prox;
        *fun = q->item;
        x->prox = q->prox;
        if (x->prox == NULL)
        {
            listaE->Ultimo = x;
        }
        delete q;
    }
}
```

APÊNDICE F – (Cadastra projetos)

```
void CadastraProjetos(TListaEncadeada *listaE) // Função que realiza o cadastro dos projetos
{
    TProjeto proj;
    TApontador ptr;
    TChave cod;
    int ret;
    cout << "****************************************" << endl;
    cout << "*       TELA DE CADASTRO DE PROJETOS      *" << endl;
    cout << "****************************************" << endl << endl;

    cout << "Informe o código do funcionário: ";
    cin >> cod;

    cout << endl;

    ret = PesquisaListaEncadeada(cod, *listaE, &ptr);

    if (ret == 1)
    {
        DadosFuncionario(cod, listaE);
        cout << endl;
        cout << "Informe o código do projeto: ";
        cin >> proj.codigo;
        cin.ignore();
        fflush(stdin);
        cout << "Informe o nome do projeto: ";
         fgets(proj.nome, MAXTAM, stdin);

        while ((cout << "Horas trabalhadas: ") && !(cin >> proj.horas))
        {
            cout << "Você inseriu um valor não numérico." << endl;
            cin.clear();
            cin.ignore();
        }

        InsereSequencial(proj, &(ptr->prox->item.projetos));

        cout << endl << "Projeto cadastrado com sucesso!" << endl<< endl;
```

APÊNDICE G – (Exclui Projetos)

```
void ExcluiProjetos(TListaEncadeada *listaE) //Função que exclui projeto
{

    TProjeto proj;
    TChave cod;
    TApontador ptr;

    int ret, codigo_projeto, op, numero_projeto;

    cout << "*****************************************" << endl;
    cout << "**      TELA DE EXCLUSÃO DE PROJETO      **" << endl;
    cout << "*****************************************" << endl << endl;

    cout << "Informe o código do funcionário: ";
    cin >> cod;

    cout << endl;

    ret = PesquisaListaEncadeada(cod, *listaE, &ptr);

    if (ret == 1)
    {
        DadosFuncionario(cod, listaE);

        do
        {
            numero_projeto = QuantProjetos(ptr->prox->item.projetos);
            if (numero_projeto > 0)
            {
                cout << endl;
                cout << "Informe o código do projeto que queira excluí-lo: ";
                cin >> codigo_projeto;

                ApagaProjeto(ptr, &ptr->prox->item.projetos, codigo_projeto, &proj);

                cout << "Deseja excluir mais algum projeto? Digite: 1-SIM 2-NÃO: ";
                cin >> op;
                system("cls");
```

APÊNDICE H – (Apaga Projetos)

```
void ApagaProjeto(TApontador ptr, TListaSequencial *listaS, TChave cod, TProjeto *proj) //Função que apaga os p
{
    int indice;

    indice = PesquisaListaSequencial(&ptr->prox->item.projetos, cod);

    if (indice >= 0)
    {
        *proj = listaS->item[indice];
        for (int i = indice; i < listaS->ultimo - 1; i++)
        {
            listaS->item[i] = listaS->item[i + 1];
        }
        listaS->ultimo--;
        cout << "O projeto informado foi removido com sucesso do sistema!" << endl << endl;
    }
    else
    {
        cout << "O projeto informado não consta na lista!" << endl << endl;
    }
```

APÊNDICE I – (Calcula Horas Semanais)

```
int CalculaHorasSemanais(TListaSequencial listaS) //Função que calcula as horas semanais do funcionario
{
    int horas = 0;
    for (int i = 0; i < listaS.ultimo; i++)
    {
        horas += listaS.item[i].horas;
    }
    return horas;
}
```

APÊNDICE J – (Calcula Salário Bruto)

```
float CalculaSalarioBruto(int TotaldeHoras, TApontador x) //Função para calcular o salário bruto do funcionario
{
    float salario = (((45 * TotaldeHoras) * 4) + (35 * x->prox->item.dependentes));
    return salario;
}
```

APÊNDICE K – (Dados funcionários)

```
void DadosFuncionario(TChave cod, TListaEncadeada *listaE) //Função que mostra os dados do funcionário
{
    TApontador ptr;
    int ret;

    ret = PesquisaListaEncadeada(cod, *listaE, &ptr);

    if (ret == 1)
    {
        cout << "Código: " << ptr->prox->item.numero << endl;
        cout << "Nome: " << ptr->prox->item.nome << endl;
        cout << "Endereço: " << ptr->prox->item.endereco << endl;
        cout << "Dependentes: " << ptr->prox->item.dependentes << endl;

        cout << endl << "Projetos atualmente com o funcionário: " << endl;

        ImprimeProjetos(ptr->prox->item.projetos);
    }
    else
    {
        cout << endl << "Funcionário não encontrado." << endl << endl;
    }

}
```

APÊNDICE L – (Imprime Projetos)

```
void ImprimeProjetos(TListaSequencial listaS) //Função que iprime os projetos que foram cadastrados
{
    for (int i = 0; i < listaS.ultimo; i++)
    {
        cout << endl << "Código do Projeto: " << listaS.item[i].codigo;
        cout << endl << "Nome do Projeto: " << listaS.item[i].nome;
        cout << endl << "Horas Trabalhadas: " << listaS.item[i].horas << endl;
    }
}
```

APÊNDICE M – (Quant Projetos)

```
int QuantProjetos(TListaSequencial listaS) //Função que realiza a contagem de projetos
{
    int cont = 0;
    for (int i = 0; i < listaS.ultimo; i++)
    {
        if (listaS.item[i].codigo != '\0' && listaS.item[i].nome != '\0' && listaS.item[i].horas != '\0')
        {
            cont++;
        }
    }
    return cont;
}
```

APÊNDICE N – (Imprime ContraCheque)

```
void ImprimeContraCheque(TListaEncadeada *listaE) // Função para realizar a impressão do contra-cheque
{
    if (VerificaListaVaziaEncadeada(*listaE))
    {
        cout << "Não existe funcionários cadastrados!" << endl << endl;
    }
    else
    {

        TApontador x = listaE->Primeiro;

        int numero_projetos = 0, horasSemanais = 0;
        float salarioBruto = 0, inss = 0, impostoRenda = 0, salarioLiquido = 0;

        while (x->prox != NULL)
        {
            numero_projetos = QuantProjetos(x->prox->item.projetos);

            cout << "*****************************************" << endl;
            cout << "*        CONTRA-CHEQUE DO FUNCIONÁRIO     *" << endl;
            cout << "*****************************************" << endl << endl;

            cout << "Código: " << x->prox->item.numero << endl;
            cout << "Nome do funcionário: " << x->prox->item.nome << endl;

            if (numero_projetos != 0)
            {
                horasSemanais = CalculaHorasSemanais(x->prox->item.projetos);
                cout << "Total de Horas Semanais: " << horasSemanais << endl;

                salarioBruto = CalculaSalarioBruto(horasSemanais, x);
                cout << "Salário Bruto: R$ " << salarioBruto << endl;

                impostoRenda = (salarioBruto * 0.15);
                cout << "Desconto de IR: R$ " << impostoRenda << endl;

                inss = (salarioBruto * 0.085);
                cout << "Desconto de INSS: R$ " << inss << endl;
```