UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

Documentação de Projeto de Software

Sistema de Gerenciamento de Folha de Pagamento

Grupo: Ludmila Gomes, Wesley Alves

João Pessoa, PB

2022

SUMÁRIO

1. INTRODUÇÃO

Este documento apresenta o projeto do Sistema de Gerenciamento de Folha de Pagamento. O presente projeto foi sendo implementado e, no decorrer deste processo, foram vistos detalhes quanto à estrutura necessária do código e classes. A seção 2 destina-se a explicar brevemente algumas características do projeto e a plataforma de implementação utilizada; a seção 3 apresenta o projeto de forma mais detalhada; a seção 4 apresenta os requisitos do sistema necessários para execução plena do programa em outras máquinas; a seção 5 apresenta os testes realizados; a seção 6 mostra algumas reflexões acerca do projeto; por fim, a seção 7 possui algumas informações complementares sobre o projeto.

2. PLATAFORMA DE IMPLEMENTAÇÃO

O programa do Sistema de Gerenciamento de Folha de Pagamento possui as características:

- Busca automática na internet de informações de endereço de um funcionário a partir do CEP digitado no programa;
- O programa armazena os dados dos funcionários em arquivos, bem como as informações sobre as Folhas Salariais mensais e anuais calculadas;

Então, para o presente projeto, foi utilizada a linguagem de programação C++, e o programa *wget* (*Wget for Windows*, adaptação para utilização no sistema operacional do Windows), que permite o download de dados da web.

3. PROJETO

3.1 VISÃO GERAL

O programa tem o objetivo de gerenciar funcionários e as folhas salariais, mensais e anuais, de uma empresa. À medida que o projeto foi desenvolvido, de acordo com as necessidades, foram implementadas classes e funcionalidades.

Foram, por exemplo, adicionadas as classes *'Data'* e *'Endereco'*, para armazenar os dados dos funcionários e evitar repetições desnecessárias no código. Além disso, nas classes *'Pessoa', 'Funcionario', 'Operador', 'Gerente', 'Diretor' e 'Presidente'*, foi feito uso de alguns recursos da Programação Orientada a Objetos, como herança e polimorfismo. Foram utilizadas bibliotecas básicas da linguagem *C++*, como *<fstream>*, para manipulação de arquivos pelo programa, assim como, para o armazenamento de informações usando um vetor de elementos, foi utilizada a biblioteca <*vector*>. Ademais, para a verificação dos dados digitados pelo usuário, criamos as funções *'IsDigit()'*, que apura a presença de apenas números numa *string*, e *'IsLetter()*, que analisa se a entrada digitada pelo usuário é composta apenas de letras, incluindo sua acentuação, se possuir. Por fim, com o intuito de automatizar o cadastro do endereço dos funcionários, usamos a ferramenta *wget*. Então, é solicitado o CEP e, a partir deste, o programa envia uma linha de comando para o terminal e, finalmente, o *wget* faz o download de um arquivo *.json* da internet, copiando as informações do endereço e salvando diretamente no registro do funcionário. Alguns problemas no funcionamento do software foram encontrados, como o mau funcionamento na passagem de strings com acentuação do programa para o arquivo.

3.2 ESTRUTURA DO PROGRAMA

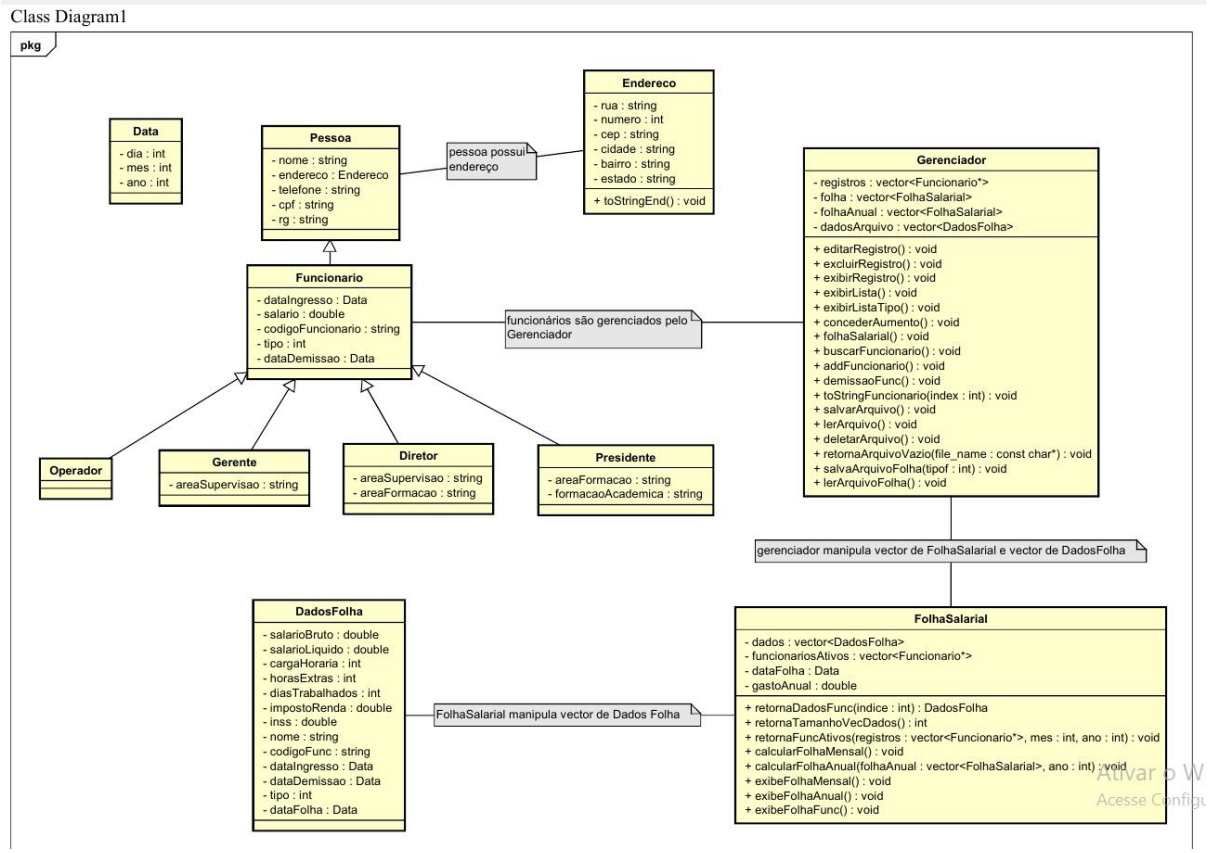


Imagem: Diagrama de Classes do projeto.

As classes *‘Data’*, *‘Pessoa’* e *‘Endereco’* foram necessárias para evitar a repetição de código, uma vez que podemos criar objetos dos mesmos e usá-los em diversas situações que precisamos. Por exemplo, todos os funcionários têm uma data de ingresso, e apenas a declaração de um objeto de *‘Data’* nos possibilita usar todos os atributos e métodos pertencentes a esta classe, diminuindo assim a recorrência de linhas de código e consequentemente, deixando o código mais organizado e limpo. Passando para as classes *‘Operador’, ‘Gerente’, ‘Diretor’ e ‘Presidente’,* que herdam da superclasse *‘Funcionario’* , elas representam os tipos de colaboradores da empresa. Mesmo sendo de tipos diferentes, todos eles são funcionários, portanto, compartilham esta característica. Por isso, seguindo o conceito de herança, todas elas herdam os seus atributos e métodos, não sendo mais preciso declarar tais atributos e métodos separadamente em cada classe. Fazendo-se fundamental declarar apenas os que são específicos de cada categoria. No *‘Gerenciador’*, temos a condução de todas as classes, e é nela que implementamos as funcionalidades gerais do programa, além de, quando preciso, os tratamentos de erros devidos, com o intuito de reduzir as linhas de código da função principal (*main*) localizada no arquivo *‘main.cpp’*. Assim, a classe *‘Gerenciador’* se relaciona com as anteriores, pois é ela quem administra todas as funções e dados relativos aos funcionários e à empresa. No caso das folhas salariais, o Gerenciador armazena-as por meio da utilização do vector ‘folha’ e *‘folhaAnual’*, após serem calculadas por meio de seus próprios métodos. Já o vector *‘dadosArquivo’* guarda os atributos recebidos do arquivo de folha salarial, o que torna a manipulação destes dados mais fácil. A criação da classe ‘FolhaSalarial’ foi importante no sentido de melhorar a lógica e manutenibilidade do programa, pois possibilita o cálculo e armazenamento das folhas salariais no Gerenciador.

A seguir, explicamos com mais detalhes os métodos que julgamos de mais complexa compreensão.

```cpp
// Função para verificar os funcioários ativos de um determinado mês
void FolhaSalarial::retornaFuncAtivos(std::vector<Funcionario *> registros, int mes, int ano)
{
    int cont = 0;
    for (int i = 0; i < registros.size(); i++)
    {
        cont = 0;
        // se o funcionário não foi demitido ou se o funcionário está na empresa até o presente mês e ano
        if (registros[i]->getDataDemissao().getMes() != 0 && registros[i]->getDataDemissao().getMes() <= mes && registros[i]->getDataDemissao().getAno() <= ano)
        {
            cont++;
        }

        if (registros[i]->getDataIngresso().getAno() < ano)
        {
            cont++;
        }

        if (registros[i]->getDataIngresso().getAno() == ano)
        {
            if (registros[i]->getDataIngresso().getMes() > mes)
            {
                cont++;
            }
        }

        // if para garantir que não teremos funcionários repetidos em FuncAtivos
        if (cont == 0)
        {
            if (registros[i]->getTipo() == 1)
            {
                // Adiciona o funcionário - operador
                funcionariosAtivos.push_back((Operador *)registros[i]);
            }
        }

        if (cont == 0)
            if (registros[i]->getTipo() == 2)
            {
                // Adiciona o funcionário - Gerente
                funcionariosAtivos.push_back((Gerente *)registros[i]);
            }

        if (cont == 0)
        {
            if (registros[i]->getTipo() == 3)
            {
                // Adiciona o funcionário - Diretor
                funcionariosAtivos.push_back((Diretor *)registros[i]);
            }
        }

        if (cont == 0)
        {
            if (registros[i]->getTipo() == 4)
            {
                // Adiciona o funcionário - presidente
                funcionariosAtivos.push_back((Presidente *)registros[i]);
            }
        }
    }
    setDataFolha(mes, ano);
}
```

O método ‘retornaFuncAtivos()’, implementado na classe ‘FolhaSalarial’, tem o objetivo de salvar no vector ‘funcionariosAtivos’ os funcionários que estão presentes e trabalhando na empresa num determinado mês de um ano (mês e ano são parâmetros deste método). Assim, no primeiro desvio condicional, é verificado se o funcionário foi demitido e se sua data de demissão é anterior ao mês solicitado. Então, se o funcionário foi demitido antes do mês pedido, não será adicionado ao vector de funcionários ativos. Após isso, o segundo desvio condicional verifica se o

funcionário entrou na empresa antes do ano solicitado. O terceiro desvio condicional verifica se o ano de ingresso do funcionário é o mesmo do ano solicitado, para que, após isso, seja feita uma verificação mais específica, que apura se o empregado entrou antes ou depois do mês pedido. Por fim, o colaborador será (ou não) adicionado ao vector de funcionários ativos, caso os requisitos dos desvios condicionais sejam atendidos. Este método é útil para os cálculos das folhas mensais, pois é necessário analisar quais funcionários estão operantes na data especificada pelo usuário.

```cpp
void FolhaSalarial::calculaFolhaAnual(std::vector<FolhaSalarial> folhaMensal, int ano) // Calcula a folha anual solicitada pelo usuário
{
    DadosFolha dado;

    double somaSalarioBruto = 0;
    double somaSalarioLiquido = 0;
    double somaInss = 0;
    double somaImpostoRenda = 0;
    int somaCargaHoraria = 0;
    int somaHorasExtras = 0;
    int somaDiasTrabalhados = 0;

    int mes = 1;

    while (1)
    {
        for (int i = 0; i < folhaMensal.size(); i++)
        {
            if (folhaMensal[i].getDataFolha().getMes() == mes && folhaMensal[i].getDataFolha().getAno() == ano) // Verifica se a folha encontrada é a correta
            {
                for (int l = 0; l < folhaMensal[i].retornaTamanhoVecDados(); l++) // Percorre todo o vector de dados que guarda as informações dos funcioários
                {
                    DadosFolha dado;
                    // Adiciona os funcionários dependendo se ainda estão ativos na empresa, ou seja, não foram demitidos
                    if (mes == 1 || folhaMensal[i].retornaDadosFunc(l).getDataIngresso().getMes() == mes && folhaMensal[i].retornaDadosFunc(l).getDataIngresso().getAno() == ano)
                    {
                        dado.setSalarioBruto(folhaMensal[i].retornaDadosFunc(l).getSalarioBruto());
                        dado.setSalarioLiquido(folhaMensal[i].retornaDadosFunc(l).getSalarioLiquido());
                        dado.setCargaHoraria(folhaMensal[i].retornaDadosFunc(l).getCargaHoraria());
                        dado.setHorasExtras(folhaMensal[i].retornaDadosFunc(l).getHorasExtras());
                        dado.setDiasTrabalhados(folhaMensal[i].retornaDadosFunc(l).getDiasTrabalhados());
                        dado.setImpostoRenda(folhaMensal[i].retornaDadosFunc(l).getImpostoRenda());
                        dado.setInss(folhaMensal[i].retornaDadosFunc(l).getInss());
                        dado.setNome(folhaMensal[i].retornaDadosFunc(l).getNome());
                        dado.setCodigoFunc(folhaMensal[i].retornaDadosFunc(l).getCodigoFunc());
                        dado.setDataIngresso(folhaMensal[i].retornaDadosFunc(l).getDataIngresso());
                        dado.setDataDemissao(folhaMensal[i].retornaDadosFunc(l).getDataDemissao());
                        dado.setTipo(folhaMensal[i].retornaDadosFunc(l).getTipo());

                        dados.push_back(dado);
                    }
                    else
                    {
                        if (folhaMensal[i].retornaDadosFunc(l).getCodigoFunc() == dados[l].getCodigoFunc())
                        {
                            // Faz as somas de todos os meses que o funcionário trabalhou
                            somaSalarioBruto = dados[l].getSalarioBruto() + folhaMensal[i].retornaDadosFunc(l).getSalarioBruto();
                            somaSalarioLiquido = dados[l].getSalarioLiquido() + folhaMensal[i].retornaDadosFunc(l).getSalarioLiquido();
                            somaInss = dados[l].getInss() + folhaMensal[i].retornaDadosFunc(l).getInss();
                            somaImpostoRenda = dados[l].getImpostoRenda() + folhaMensal[i].retornaDadosFunc(l).getImpostoRenda();
                            somaCargaHoraria = dados[l].getCargaHoraria() + folhaMensal[i].retornaDadosFunc(l).getCargaHoraria();
                            somaHorasExtras = dados[l].getHorasExtras() + folhaMensal[i].retornaDadosFunc(l).getHorasExtras();
                            somaDiasTrabalhados = dados[l].getDiasTrabalhados() + folhaMensal[i].retornaDadosFunc(l).getDiasTrabalhados();

                            dados[l].setSalarioBruto(somaSalarioBruto);
                            dados[l].setSalarioLiquido(somaSalarioLiquido);
                            dados[l].setCargaHoraria(somaCargaHoraria);
                            dados[l].setHorasExtras(somaHorasExtras);
                            dados[l].setDiasTrabalhados(somaDiasTrabalhados);
                            dados[l].setImpostoRenda(somaImpostoRenda);
                            dados[l].setInss(somaInss);
                        }
                    }
                }

                break;
            }
        }
        if (mes == 12)
        {
            break;
        }
        mes++;
    }
    cout << "Folha Anual calculada com sucesso!" << endl;
    setDataFolha(0, ano);
}
```

O método ‘calculaFolhaAnual()’, implementado na classe ‘folhaSalarial’, calcula a folha de pagamento de um determinado ano que é passado como parâmetro da função. Inicialmente, declaramos algumas variáveis para ajudar na manipulação dos dados. Um laço de repetição while foi necessário para que todos os meses fossem calculados, sendo quebrado quando a variável ‘mes’ atinge o valor 12. O segundo laço de repetição percorre todo o vector de ‘folhaMensal’, contendo um if que é ativado quando a data da folha é igual à data passada como parâmetro e, dentro desse if, um for irá percorrer as informações dos funcionários (vector de dados) da folha selecionada, adicionando-os na folha anual se a condição for atendida, e, posteriormente, fazendo o somatório dos atributos que devem constar na folha anual individualmente. Após isso, é verificado se o mês já atingiu o número 12. Caso o valor da variável ‘mes’ não seja 12, ela é incrementada em uma unidade e volta para o início do loop, realizando todas as etapas novamente.

4. REQUISITO

4.1 ESPECIFICAÇÃO REVISADA

O presente projeto não apresenta grandes requisitos ou restrições para o sistema. Mas uma pequena alteração é necessária para o funcionamento pleno do programa em outros computadores. A alteração devida seria o download do programa *wget*, cuja presença é necessária para a funcionalidade da busca automática dos dados do endereço do funcionário. Assim, é preciso fazer o download e incluir o arquivo *‘wget.exe’* na pasta em que está guardado o programa. Além disso, o computador também deve possuir acesso à Internet.

4.2 MANUAL DO USUÁRIO

Ao iniciar o software, o usuário se depara com um menu onde deverá escolher a funcionalidade que deseja executar. Cada opção digitada leva-o até a parte desejada na qual o programa realizará a opção escolhida. Para operar o sistema, as entradas são sempre através de campos de digitação. No caso de buscar o endereço através do CEP, o computador deve ter conexão com a internet. É preferível que o usuário digite corretamente os dados, mas, se por algum motivo isto não for feito, o programa possui um conjunto de tratamento de possíveis erros que eventualmente não permitirá que o dado seja armazenado, solicitando que o

usuário digite novamente. É necessário que seja cadastrado ao menos um funcionário (opção (1) do primeiro menu) para que o programa exiba algum dado ou faça alguma alteração no mesmo. Em relação a folha de pagamento, a folha anual deve ser calculada após as as folhas mensais do ano que se quer calcular tenham sido previamente geradas, entretanto, se isso não for feito, o programa irá gerar aleatoriamente as que estiverem ausentes. Para que o programa armazene os dados cadastrados e folhas calculadas nos arquivos, o programa deve ser encerrado corretamente, ou seja, deve-se teclar a opção 0 (zero), essa opção salvará os dados corretamente e encerrará o programa.

4.3 DESEMPENHO

Como dito anteriormente, o programa não possui requisitos ou restrições para o sistema. Então, na execução normal do programa, as funcionalidades são acessadas e espera-se que não haja tempo de espera para nenhuma aplicação do software. Em relação à memória consumida pelo programa, são necessários pelo menos 6 MB do sistema. Deve-se levar em consideração que os arquivos também consumirão memória.

4.4 ANÁLISE DE PROBLEMAS

A dificuldade de se passar strings de palavras acentuadas do programa para o arquivo, problema já citado anteriormente, impossibilita a reprodução desses dados para o usuário corretamente. Para resolver este obstáculo, estamos realizando pesquisas de encoding para resolver o impasse da melhor forma possível. Outro problema encontrado foi na passagem de informações para os arquivos do programa quando era necessário fechar o terminal de comando em tempo de execução do software. Quando isto acontecia, as execuções seguintes do programa passavam por erros, e estas falhas afetavam os registros dos funcionários. O defeito anteriormente citado foi resolvido ao restringirmos a passagem de dados do programa para os arquivos apenas ao final da execução do software, antes que o programa se encerrasse.

5. TESTES

O presente projeto foi testado pelos desenvolvedores à medida que as suas funcionalidades foram implementadas. Dessa forma, erros na lógica do programa foram encontrados a partir de testes simples do programa e as devidas correções, feitas. Assim, o programa tem pleno desempenho e é certo que foram minimizados bugs que poderiam comprometer as funcionalidades do software. Alguns dos erros foram facilmente identificados, enquanto outros exigiram testes mais específicos.

6. REFLEXÃO

6.1 AVALIAÇÕES

Um erro cometido pelos desenvolvedores do programa foi a falta de comentários explicativos acerca de métodos e a forma como estes funcionavam. Este equívoco tornou a implementação da classe ‘FolhaSalarial’, por exemplo, mais laboriosa e complexa do que realmente deveria ser.

6.2 LIÇÕES

Aprendemos a importância e necessidade de se comentar o código-fonte, adicionando explicações claras sobre a lógica de programação dos métodos implementados. Isto pois, muito tempo foi gasto com tentativas de se entender códigos feitos anteriormente, o que nos fez despender tempo a mais em pequenos problemas.

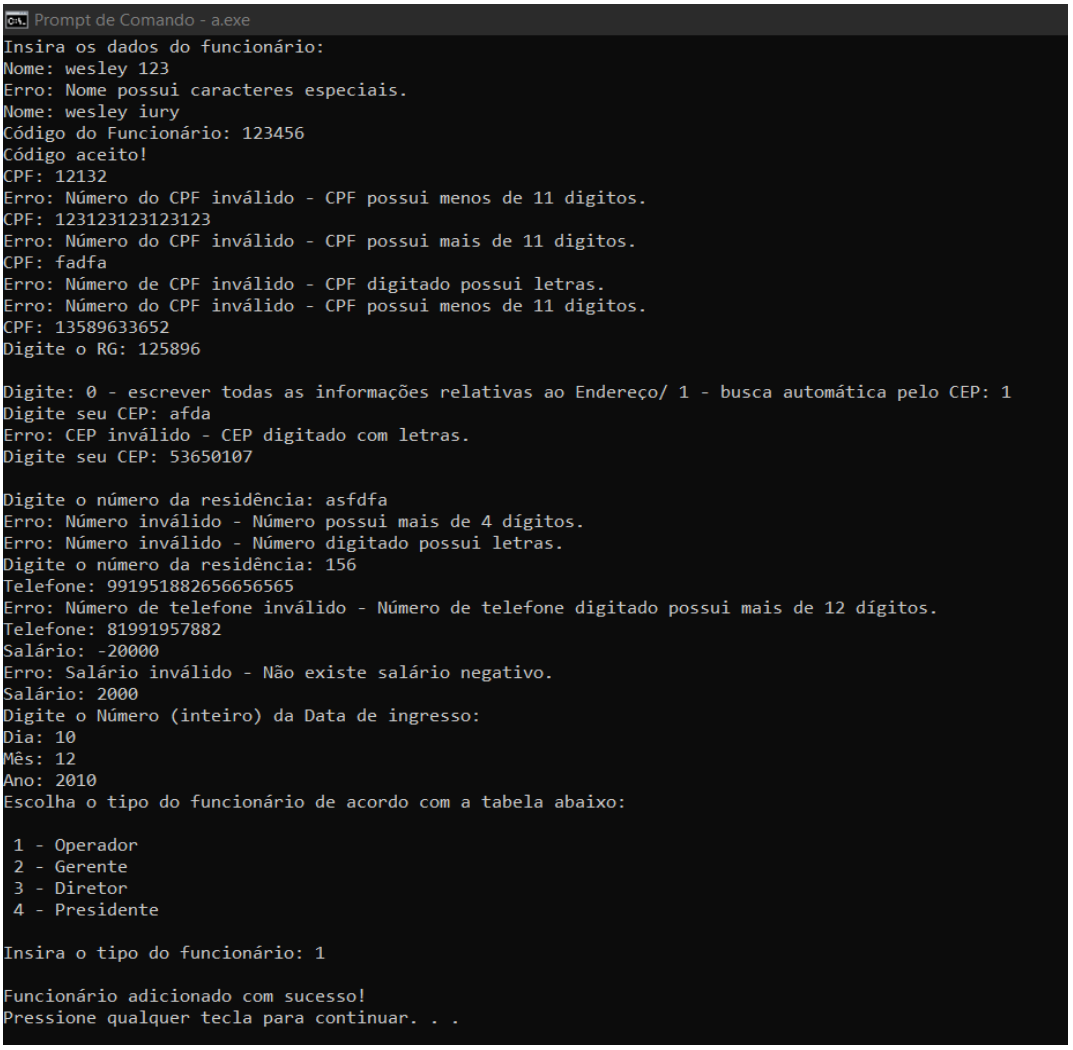
6.3 BUGS E LIMITAÇÕES CONHECIDAS

Existe uma limitação na funcionalidade do cálculo da folha anual, em que necessariamente o usuário precisa ter calculado pelo menos uma folha mensal antes de calcular a folha anual para o ano escolhido. Por exemplo, antes de se calcular a folha salarial anual do ano de 2021, é necessário que o usuário tenha calculado pelo menos uma folha mensal, de qualquer mês, de 2021. Caso contrário, o cálculo da folha anual causará bug no programa. Outro problema conhecido é relativo à biblioteca *‘VerificaCaractere.h’*, que tinha como objetivo permitir a

utilização de caracteres especiais e acentos nas strings de entrada (recebidas do usuário). Vale ressaltar que essa biblioteca possui uma gama limitada de verificações de caracteres especiais, incluindo apenas os mais recorrentes da língua portuguesa.

7. APÊNDICES

7.1 CASOS DE TESTE



Na imagem acima, foram testadas diferentes entradas inválidas na funcionalidade do cadastro de um novo funcionário nos registros do programa.

7.2 DOCUMENTAÇÃO DO CÓDIGO

```
/*
============== DOCUMENTAÇÃO ==============
*
*   AUTORES: Wesley Alves, Ludmila Gomes
*   DATA: desde 30/04/2022
*
*   OBJETIVO: Criar um programa capaz de armazenar as informações de funcionários de uma empresa,
*             além de manipular estes. Ademais, o software também é capaz de armazenar os dados relativos
*             às folhas salariais dos empregados da empresa, tanto mensais, quanto anuais.
*
*   DESCRIÇÃO: O programa gerencia os dados de uma empresa em relação aos seus funcionários e às folhas salariais.
*
*   COMO OPERAR O PROGRAMA: A operação do sistema por parte do usuário ocorre através dos campos de digitação. Assim,
*                           surgem menus na tela, e o usuário deverá digitar a entrada adequada indicada pelo próprio
*                           programa.
*
*   SITUAÇÕES DE ERRO (SE EXISTIREM): Passagem de dados com acentuação para arquivos.
*
============== ============== ==============
*/
```

Imagem: documentação do projeto presente no programa.