

Projeto de Laboratório de Programação

Licenciatura em Segurança Informática em Redes de Computadores

2020/2021

**Grupo XXX**

8200341, Maria Sofia Cunha

8200335, Gonçalo Pedro Gil

## Índice

[1. Introdução 3](#_Toc60935844)

[2. Funcionalidades requeridas 3](#_Toc60935845)

[3. Funcionalidades propostas 6](#_Toc60935846)

[4. Estrutura analítica do projeto 6](#_Toc60935847)

[5. Funcionalidades implementadas 7](#_Toc60935848)

[6. Conclusão 9](#_Toc60935849)

## Introdução

Descrição do trabalho efetuado e das decisões tomadas no decorrer do desenvolvimento do projeto.

* Uso do código como identificador
* Repartir o código pelos doc

## Funcionalidades requeridas

Este grupo de funcionalidades é dedicado a todas aquelas que são diretamente essenciais para o cumprimento do objetivo do trabalho.

Para tal, viu-se a necessidade de criar estruturas que permitissem guardar de forma organizada e correta a informação indispensável. Tais como:

* **Data** – Contém as variáveis necessárias para todas as datas guardadas, dia (*int*), mês (*int*) e ano (*int*);
* **Funcionario** – Tem como objetivo guardar os dados do funcionário. É composta por variáveis como: o “ativo”, este só toma valor de 1 se o funcionário se encontrar ativo ou 0 caso tenha sido removido; o “nascimento”, “entrada\_emp” e “saida\_emp”, todas estas variáveis são do tipo “Data” (acima descrita) e guardam os valores da data de nascimento, de entrada na empresa e de saída (se o funcionário ainda se encontrar na empresa esta toma valores de 0-0-0), respetivamente. São também guardados valores para o estado civil, “Est\_Civil”, e para o cargo na empresa, “Cargo”, estas são enumerações, que contêm todos os tipos de estados civis e de cargos nesta empresa, o nome, “nome”, número de filhos, “numero\_filhos”, número de titulares, “titulares”, (caso não se aplique é atribuído o valor de 0), o valor à hora que o funcionário recebe, “valor\_hora”, o valor por dia do subsídio de alimentação, “valor\_sub\_ali”, e o código do funcionário;
* **Empresa** – É composta por um contador e por um apontador (“funcionarios\_array”) do tipo “Funcionario” e tem como objetivo guardar toda a informação dos funcionários introduzidos no programa;
* **Conta** – Foi criada para receber os dados para o processamento salarial. Recebe valores para os dias completos trabalhados, “dias\_compl”, meios dias, “dias\_meios”, fins de semana, “dias\_fds”, dias faltados, “dias\_faltas”, o mês e o ano desse salário, o “estado” que toma valores de 0 quando o salário está por processar, 1 quando já foi processado e -1 quando os dados recebidos são inválidos e por fim, o código do funcionário;
* **Lista\_calc** - É composta por um contador e por um apontador (“dados\_calculo\_array”) do tipo “Conta” e tem como objetivo guardar toda informação dos dados para o cálculo salarial introduzidos no programa;
* **Calculo** – Recebe todos os dados dos salários depois de processados, tais como, o vencimento líquido, “venc\_liquido”, o vencimento ilíquido , “venc\_iliquido”, o encargo total da empresa, “encargo\_total\_emp”, os descontos para o IRS, “irs”, o subsidío de alimentação, “sub\_ali”, o bónus, “bónus”, o código e por fim, os descontos para a segurança social entidade patronal e do trabalhador, “ss\_ent\_patronal” e “ss\_ent\_pessoal”, respetivamente;
* **Lista\_salarios** - É composta por um contador e por um apontador (“calculo\_array”) do tipo “Calculo” e tem como objetivo guardar toda informação dos dados para o cálculo salarial introduzidos no programa;
* **IRS** – Recebe os dados das linhas das tabelas de IRS, estes são, os vencimentos, “vencimento ”, os valores para quem não tem filhos, “filho\_zero”, para um filho, “filho\_um”, dois filhos, “filho\_dois”, três filhos, “filho\_tres”, quatro filhos, “filho\_quatro” e mais de cinco filhos, “filho\_cinco”;
* **DoisTitulares, UnicoTitular, NaoCasado** – Estas estruturas têm como objetivo guardar a informação das tabelas e são constituídas por um contador e um apontador (“Dois\_titulares\_array”, “Unico\_titular\_array”,” Nao\_casado\_array”, respetivamente) do tipo IRS;
* **Taxas** – É composta pelas taxas para a segurança social, a taxa para os funcionários em geral, entidade empregadora, “geral\_empregadora”, e para o trabalhador, “geral\_trabalhador”, e para membros da chefia, entidade empregadora, “admin\_empregadora”, e trabalhador, “admin\_trabalhador”.

Todos os *arrays* acima mencionados, são feitos com memória dinâmica. São inicializados no “main” e é lhes criado um espaço de memória, os contadores, são também inicializados, a zero. A abordagem adotada relativamente à adição de memória nos *arrays*, consiste em, sempre que é adicionado conteúdo aos *arrays* é também criada memória para a adição seguinte.

Tiveram de ser criadas funcionalidades para adicionar, editar e remover a informação guardada. Tais como:

* **addFuncionario** – Esta função é responsável por pedir toda a informação referente aos utilizadores (dados da estrutura “Funcionario”) e guardá-la no *array* “funcionarios\_array”;
* **removerFuncionario** – Tem como objetivo remover funcionários, passando a variável “ativo” para 0;
* **editarFuncionario** – Permite editar a informação do funcionário e para tal mostra a informação guardada em memória do mesmo (com uma outra função “infoFuncionario”);
* **addSalarios** - Esta função é responsável por pedir toda a informação referente aos dados salariais (dados da estrutura “Conta”) e guardá-la no *array* “dados\_calculo\_array”;
* **processamento** – É a função responsável pelo processamento dos dados salariais, por guardar os mesmos no *array* “calculo\_array” e adicionar um relatório relativo a cada salário processado ao documento dos relatórios, disponível para o utilizador;
* **saberBonus** – Permite fazer o cálculo do bónus para casa salário e utilizador. A percentagem do bónus é conseguida ao somar as três parcelas e a dividi-las por 100. A percentagem de bónus foi decidida para que pudesse ser no máximo 50% do vencimento ilíquido. Esses 50% seriam divididos em 3, sendo que então, no máximo, 25 seriam em relação à antiguidade do funcionário na empresa, 12.5 da idade possível do funcionário na empresa, e outros 12.5 seriam dos dias trabalhados a mais dos obrigatórios por mês (decididos como 22).

O cálculo dos 25% da **antiguidade** baseiam-se no máximo de anos que um funcionário pode trabalhar na empresa, ou seja, dos 18 anos (idade mínima de entrada na empresa) aos 65 anos (idade de reforma), que são 47 anos. Dividindo então os 25 por 47 anos, equivale a 0.53% por cada ano a trabalhar na empresa. Para se obter a parte em relação à antiguidade do funcionário basta assim multiplicar os anos trabalhados por 0.53.

O cálculo dos 12.5% da **idade** do funcionário baseiam-se na idade que um funcionário pode ter na empresa, ou seja, como só se pode começar a trabalhar na empresa aos 18 e tem de se sair aos 65, esta percentagem vai também ser dividida por 47 anos, dando 0.26 por ano de idade acima dos 18.

O cálculo dos 12.5% dos **dias extra** trabalhados para além dos 22 obrigatórios por mês vai dependendo do número de dias do mês em questão. Sabendo o número máximo de dias extra que podem ser trabalhados nesse mês, retirando 22 aos dias totais do mês, divide-se os 12.5 por esses dias. Esse número é então multiplicado pelo número de dias que o funcionário trabalhou extra. Esses dias são calculados retirando-se 22 aos dias que trabalhou nesse mês, que são a soma dos dias completos, meios completos e de fim de semana trabalhados.

Por fim, são somadas essas três partes, dando a percentagem de bónus final, que com 65 anos de idade a trabalhar na empresa há 45 anos e trabalhando os dias todos de um mês, seria 50%.

Esta recebe o máximo de dias possíveis trabalhados naquele mês, “dia\_max”, o “num\_salarios” e um “x” que definem a posição no *array,* “conta.calculo\_array”,do salário a ser processado. Primeiramente, é feito o somatório de dias trabalhados pelo funcionário e, de seguida, são calculadas as percentagens de antiguidade e idade, estas variam com a antiguidade do funcionário na empresa e com a idade. Para a antiguidade, multiplica-se os 0.53 pelo número de anos trabalhados na empresa (recorrendo ao defineData para saber o ano atual) e para a idade, multiplica-se os 0.26 pelo número de anos de idade acima de dezoito anos (recorrendo mais uma vez ao defineData para saber o ano atual). Por último é calculada a percentagem do bónus aos dias trabalhos para além dos vinte e dois obrigatórios. Para este cálculo, verifica-se se o número de dias trabalhados, “dias\_trab”, é menor ou igual a vinte e dois e, caso se verifique, essa parcela do bónus é nula, caso contrário, é feito o cálculo de quantos dias o funcionário trabalhou a mais e esse valor é multiplicado por 12.5 a dividir pelo máximo de dias “extra” que podem ser trabalhados nesse mês.

Tal como previamente mencionado, o programa tem a capacidade de guardar as informações entre utilizações. Para tal, tem de ser importada toda a informação dos documentos que guardam essas informações (estes estão todos guardados em subpastas dentro da pasta “PROGRAMA”. Com esse objetivo, foram feitas as seguintes funções:

* **importar\_users\_sys** – Importa para a memória todos os utilizadores que se encontram no ficheiro “users.bin” na pasta “USERS”;
* **importar\_salarios\_sys** - Importa para a memória todos os dados salariais que se encontram no ficheiro “salarios.bin” na pasta “SALARIOS”;
* **importar\_salarios\_proc** - Importa para a memória todos os salários já processados que se encontram no ficheiro “SALARIOS\_PROCESSADOS\_bin” na pasta “CALCULO”;
* **importarDoisTitulares, importarUnicoTitular, importarNaoCasado** - Importa para a memória as tabelas de descontos para o IRS que se encontram nos ficheiros “DEPENDENTE\_CASADO\_DOIS\_TITULARES\_bin”, “DEPENDENTE\_CASADO\_UNICO\_TITULAR\_bin”, “DEPENDENTE\_NAO\_CASADO\_bin”, respetivamente, na pasta “TABELAS\_RETENCAO\_IRS”;
* **importarTablelaSS** – Importa para a memória as taxas todas de descontos para a segurança social, que se encontram no ficheiro “TAXAS.txt”, na pasta “SEGURANCA\_SOCIAL”.

Criaram-se, também, funções para alterar a informação das tabelas acima mencionadas (IRS e segurança social):

* **AlterarCriterioSS** – Tem como objetivo alterar os valores de descontos para a segurança social. Recebe a estrutura “Taxas”;
* **alterarCriterioIRS** - Tem como objetivo alterar os valores de descontos para o IRS de um vencimento especificado perguntando ao utilizador em que vencimento deseja fazer as alterações e os novos valores. Recebe as estruturas “DoisTitulares”, “UnicoTitular” e “NaoCasado”;
* **addCriterioIRS** - Tem como objetivo adicionar uma linha de valores de descontos para o IRS pedindo os novos valores e adicionando-os no devido *array* (“Dois\_titulares\_array”, “Unico\_titular\_array” ou “Nao\_casado\_array). Recebe as estruturas “DoisTitulares”, “UnicoTitular” e “NaoCasado”.

Por fim, para que toda a informação em memória seja guardada nos ficheiros acima mencionados, foi criada uma função **guardar** que substitui ou adiciona toda a informação já existente nos ficheiros com a informação que se encontra em memória.

## Funcionalidades propostas

Neste trabalho foram pedidas cinco funcionalidades adicionais, cuja finalidade ficou ao encargue do grupo.

* **Mostrar ex-funcionários** – Mostra todos os funcionários que já não se encontram na empresa, ou seja, todos aqueles em que a data de saída tem valores diferentes de 0-0-0, para isto utiliza a estrutura “Empresa”;
* **Total gasto pela empresa** – Mostra o somatório de todos os encargos da empresa, nos salários até então processados e em memória. Com esse intuito é utilizada a estrutura “Lista\_salarios”;
* **Total gasto pela empresa em impostos** – Mostra o somatório de todos os descontos para a segurança social que a empresa pagou até à data em todos os salários em memória, usando a estrutura “Lista\_salarios”;
* **Mostrar funcionários removidos** – Mostra todos os funcionários removidos, ou seja, todos aqueles em que a variável “ativo” se encontra a 0. Para tal, utiliza a estrutura “Empresa”;
* **criarFicheirosUser** – Esta função permite ao utilizador imprimir para um documento (.txt) todos os dados em memória dos utilizadores, dados salariais e salários já processados. Com base no documento que o utilizador pretende imprimir podem ser utilizadas as estruturas “Empresa”, “Lista\_calc” ou “Lista\_salarios”.

## Estrutura analítica do projeto

Para um correto funcionamento do programa tiveram de ser criadas algumas funções que, apesar de não obrigatórias, sem estas não seria possível alcançar o objetivo com sucesso e precisão. Assim, foram criadas as seguintes funções:

* **saberIRS** – Tem como objetivo retornar o valor correto da taxa de IRS para aplicar no salário em questão. Recebe como argumentos o salário ilíquido do funcionario, o código e as estruturas da “Empresa”, “DoisTitulares”, “UnicoTitular” e “NaoCasado”. Desta forma é capaz de saber em que tabela deve ir procurar os valores (pois na estrutura da “Empresa” consegue obter a informação do número de titulares do funcionário) a que linha (vencimento) e coluna (através, mais uma vez, da estrutura “Empresa” consegue obter o número de filhos do funcionário);
* **saberSS** – Tem como objetivo retornar o valor correto das deduções para a segurança social no caso em questão. Recebe a entidade (patronal ou pessoal), o código e as estruturas “Empresa” e “Taxa”, onde irá obter a informação do cargo e dos valores a retornar, respetivamente;
* **procurarFuncionario** – Retorna o valor (i) da posição no *array* “funcionarios\_array” do código que lhe é dado, de forma a localizar-se em que posição no *array* se encontra o funcionário em questão. Recebe o código do funcionário e a estrutura “Empresa”;
* **mostrarTaxas** – Foi concebida com o objetivo de mostrar os valores das taxas para descontos para a segurança social que se encontram em memória. Recebe a estrutura “Taxas”;
* **mostrarTabelas** - Foi concebida com o objetivo de perguntar ao utilizador que tabela deseja ver e, mostrar as tabelas para os descontos do IRS que se encontram em memória. Recebe as estruturas “DoisTitulares”, “UnicoTitular” e “NaoCasado”;
* **posicaoVencimento** - Retorna o valor (i) da posição nos *arrays* “Dois\_titulares\_array”, “Unico\_titular\_array” e “Nao\_casado\_array” do vencimento que lhe é dado, de forma a localizar-se em que posição no *array* se encontra o vencimento em questão. Recebe o a tabela onde se quer procurar, o vencimento e as estruturas, “DoisTitulares”, “UnicoTitular” e “NaoCasado”. Desta forma é possível da função “addCriterioIRS” saber em que linha deve ser adicionado os novos critérios;

## Funcionalidades implementadas

Para que todos os dados, cálculos, procedimentos e até mesmo a experiência de utilização do utilizador fossem corretamente concretizados foi necessário criar funcionalidades que visassem esse fim.

Foi criada uma função, “cleanInputBuffer”, que tem como objetivo limpar todo o buffer do sistema sempre que forem pedidos valores ao utilizador, diminuindo assim a probabilidade de erros provenientes do buffer e possibilitando que estes sejam lidos de forma correta.

Para pedir e validar se a informação, guardada em memória (*arrays*), do utilizador foi corretamente inserida, foram criadas as seguintes funções auxiliares:

* **obterInt** - Recebe um valor de mínimo, um valor de máximo, e uma mensagem de erro (*string*). Esta pede um valor inteiro (*int*) e, caso este se encontre entre o mínimo e o máximo retorna esse valor;
* **obterFloat** – Funciona exatamente da mesma forma que a função anteriormente descrita, porém esta retorna um número decimal (*float)*;
* **obterString** - recebe um *pointer* para o local onde o valor retornado será guardado, o tamanho máximo desse valor e uma mensagem de erro (*string*). Com a biblioteca **<ctype.h>**, é pedida o input de uma *string* até valida. Essa string, uma vez válida, é então guardada no espaço do *pointer* recebido.
* **obterLong** - Funciona da mesma forma que a função “obterFloat”, porém esta retorna um número longo (*long)*;
* **saberDia** – Tem como objetivo pedir valores para o dia. Usando a função “obterInt” para pedir o valor e passa o valor da função “saberDiaMax” como o argumento de valor máximo;
* **saberMes** -Tem como objetivo pedir valores para o mês. Usando a função “obterInt” para pedir o valor e dá à mesma o valor máximo para o mês, usando a função “defineData” como o argumento de valor máximo caso a data a tratar seja do ano atual, caso contrário, dá doze como valor máximo;
* **saberAno** - Tem como objetivo pedir valores para o ano. Usando a função “obterInt” para pedir o valor e passa o valor da função “defineData” como o argumento do valor máximo;

A exatidão dos valores relativos as datas foram uma área de grande foco do grupo logo, para além das funções acima mencionadas (“saberDia”, “saberMes” e “saberAno”) foram, ainda, criadas as seguintes funções:

* **defineData –** Esta função foi concebida com o objetivo de retornar os valores do dia, mês, ano, hora, minuto e segundo atuais. Para tal, recebe o “pedido”, que é a variável que indica qual das variantes anteriores se quer retornar e usa a biblioteca <time.h>;
* **saberDiaMax** - Esta função foi concebida com o objetivo de retornar o número máximo de dias de um mês. Para isso, recebe o ano e o mês. Para concretizar o objetivo, primeiramente, o ano é dividido por quatro, caso esta divisão dê resto zero sabemos que o ano em questão é bissexto, de seguida é verificado se o mês e o ano, são diferentes do ano e mês atuais (usando a função defineData), caso essa condição se confirme retorna o valor máximo desse mês (se o mês for igual a fevereiro e o ano bissexto incrementa um dia no número máximo de dias), caso a condição não se confirme, ou seja, o mês e o ano são os atuais, é retornado o dia atual como número máximo de dias.

Para mostrar ao utilizador a informação guardada, em algumas situações, foram criadas funções para passar a informação guardada em memória para texto, possibilitando, assim, a compreensão dos dados. Tais como:

* **est\_civilToString** – Recebe o valor do estado civil do funcionário e retorna esse valor no formato de texto (Ex: recebe: 2, retorna: “Divorciado”);
* **cargoToString -** Recebe o valor do cargo do funcionário e retorna esse valor no formato de texto (Ex: recebe: 2, retorna: “Administrador”);
* **estadoToString -** Recebe o valor do estado do salário e retorna esse valor no formato de texto (Ex: recebe: -1, retorna: “Dados inválidos”);

Tal como fora dito na introdução, o programa em questão possibilita a introdução de dados no sistema a partir de ficheiros texto criados pelo utilizador. Para atingir esse fim, foram desenvolvidas as seguintes funcionalidades:

* **importar\_users\_doc –** Lê toda informação que o utilizador guardou no documento “funcionarios.txt”, na pasta “UTILIZADOR” e guarda no *array* “funcionarios\_array”, informa quantos funcionários foram adicionados e se o utilizador pretende ver a informação guardada;
* **importar\_salarios\_doc -** Lê toda informação que o utilizador guardou no documento “salarios.txt”, na pasta “UTILIZADOR” e guarda no *array* “dados\_calculo\_array”, informa quantos dados para o processo salarial foram adicionados, se o utilizador pretende ver a informação guardada e se foram adicionados salários e funcionários não pertencentes ao *database* ou com valores não válidos.

A precisão dos resultados foi outro grande tópico que o grupo dedicou muita atenção e trabalho, para que todos os dados e pedidos pelo utilizador fossem sempre validados. Tiveram de ser criadas funções de validação de *inputs.* Tais como:

* **verificacaoDias** – Faz a soma de todos os dias introduzidos para o cálculo salarial e verifica se este não é superior a máximo de dias desse mês (usando a função saberDiaMax);
* **verificacaoAddFuncionarios –** Verifica se o código inserido existe/está em uso. Recebe a estrutura “Empresa” e o código em questão;
* **VerificacaoEditarFuncionarios –** Verifica se o código inserido se encontra no *database* e ativo (usando a variável “ativo” guardada na estrutura “Empesa”), caso se verifique mostra a informação (com a função infoFuncionario). Caso contrário, é perguntado ao utilizador se deseja adicionar um funcionário novo;
* **VerificacaoFuncionariosCalculo –** Verifica se o código inserido se encontra no *database* e ativo (usando a variável “ativo” guardada na estrutura “Empesa”), caso se verifique deixa o utilizador continuar o processo de adicionar salários. Caso contrário, é perguntado ao utilizador se deseja adicionar um funcionário novo com esse código;
* **VerificarVencimento –** Verifica se o vencimento indicado pelo utilizador para a alteração de critérios nas tabelas de IRS é válido ou não. Para isso, percorre o *array* da tabela em questão até encontrar um valor igual, caso se verifique, retorna um como sucesso, caso contrário mostra uma mensagem de erro. Para esse fim, a função recebe como argumentos a tabela que vai ser usada, o vencimento e as estruturas “DoisTitulares”, “UnicoTitular” e “NaoCasado”;
* **verificacaoDiasDoc –** Faz o somatório dos dias para o processamento salarial importados a partir do documento de salários do utilizador, retorna um (ou seja, sucesso) se esse valor for igual o número máximo de dias (para isso usa a função saberDiasMax) e não retorna caso o valor seja diferente. Para isto, recebe como argumentos os dias completos trabalhados, meios dias, fins de semana e dias faltados.

Tendo em vista a comodidade do utilizador, foi criada uma função de limpeza de memória “limpar\_memoria”, que permite ao utilizador limpar todos os dados dos documentos do sistema relativos aos dados dos funcionários, dados para o processamento salarial e salários já processados. Foi também concebida uma função, “Logs”, que guarda todos as interações dos utilizadores com o sistema.

Por fim, devido ao uso de memórias dinâmicas foi necessário criar uma funcionalidade, “freeMemoria”, que permite ao sistema apagar toda a memória e libertar os apontadores de memória usados.

## Conclusão

Quaisquer considerações finais que o grupo ache pertinentes para avaliação.