Nome: **Estevão Luiz Duarte da Costa** Matrícula: **GU3011771**

ED1D2 – Projeto Final – Relatório

Explicação das decisões de implementação tomadas

Para os menus foi preferível utilizar a função `switch()`, por ser visualmente mais limpa que uma cadeia de `ifs`. Para entrada de cadeias de caracteres foi usada a função fgets() por ser mais segura que a função `gets()`, pois essa pode ocasionar um buffer overflow. Quanto às funções `system("cls")` e `system("pause")`, existem profissionais que afirmam que esses comandos podem causar vulnerabilidades, podendo expor o sistema à possíveis ataques, não é uma certeza, mas por via das dúvidas substituí essas funções por `limpa()` e `getchar()`, respectivamente, as quais tem os mesmos efeitos, sendo `limpa()` uma API para limpar a tela em sistemas de base DOS.

Na Lista Sequencial Estática optei por preencher dois vetores com os nomes e códigos dos cargos, respectivamente, e apenas depois juntá-los na lista usando uma função, isso facilitou a escrita e implementação, além de facilitar a correção de um possível erro. Para a exibição de mensagens foi utilizada uma sub-rotina `mensagem()`, as funções de ação eram do tipo inteiro e retornavam um valor para cada problema, ou 0 caso fosse tudo bem. Esse valor é recebido diretamente pela função `mensagem()`, que imediatamente exibe a mensagem com o erro ocorrido. Foi implementada uma variável `cod` do tipo inteiro, que inicia com 0 ao abrir o programa, ele serve como uma espécie de chave primária para cada registro da lista, quando o registro é apagado o número também é perdido, assim, as pesquisas por id se tornam mais precisas e fáceis de implementar. Em teste de confirmação, que dependem de uma entrada 's' ou 'S' para continuar, aceitam também o valor 1 como confirmação, isso tornou o processo de teste mais rápido.

Visão geral do funcionamento do programa

O programa em questão gerencia o registro de funcionários utilizando algoritmos de Lista Sequencial Dinâmica e Lista Sequencial Estática. Nele, é possível adicionar novos registros de funcionários, editá-los, removê-los, realizar pesquisa por id e por nome, e por fim, exibir todos eles pelo id ou por um intervalo de salário informado pelo usuário.

Ao adicionar ou editar o registro de um funcionário, é permitida a inserção de informações gerais como nome, endereço, idade e salário à critério do usuário, respeitando o tipo de cada campo. Exceções são aplicadas ao campo id, que é gerado automaticamente ao adicionar um novo funcionário, não pode ser alterado ou repetido, e ao campo cargo, que é do tipo inteiro e guarda o código do cargo selecionado segundo o Código Brasileiro de Ocupações. Não é possível inserir manualmente qualquer valor nesse campo, o acesso a ele se dá apenas através de um menu, que apresenta os cargos disponíveis e que automaticamente guardará e apresentará o cargo desejado pelo usuário para o registro.

Comentários sobre os testes

Foram realizados testes do programa na completa extensão da sua funcionalidade. Adicionar, excluir, editar, buscar e apresentar registros, todas essas operações formas completadas e abortadas no meio de sua execução. Todos esses testes foram realizados além de testes com entrada inválida de dados.

Manual básico

Ao iniciar o programa o menu principal será exibido, mostrando as opções disponíveis ao usuário.

Para **adicionar um funcionário** deve-se entrar o valor ‘1’ e teclar Enter. O menu adicionar se abrirá esperando uma entrada. Entre o nome, endereço, idade e salário, sempre separando as entradas com Enter. Após entrar o salário e teclar Enter, uma tela com os cargos será exibida, entre o número correspondente ao cargo escolhido e tecle Enter. Uma tela de confirmação se

abrirá exibindo o registro, caso deseje continuar digite ‘S’ e tecle Enter, caso contrário entre ‘N’ e a operação será abortada. Após qualquer uma das opções anteriores o menu principal será exibido.

Para **exibir o funcionário** cadastrado a partir do menu principal entre o valor ‘5’ e tecle Enter. Um submenu será apresentado com duas opções. Para **exibir os funcionários por id** entre o valor ‘1’ e tecle Enter, todos os cadastros serão apresentados, juntamente com seus ids. Para **exibir os funcionários por faixa salarial** entre ‘2’ e tecle Enter. Um novo menu será aberto, entre o valor mínimo e o valor máximo conforme indicado. O programa exibirá os registros caso haja alguma correspondência, caso contrário ele informará. Depois disso retornará para o menu principal.

Para **excluir um funcionário**, a partir do menu principal entre o valor ‘2’ e tecle Enter. Um novo menu se abrirá aguardando uma entrada. Entre o id do funcionário a ser excluído e tecle Enter. Caso seja encontrado, o funcionário será exibido e uma opção de exclusão aparecerá. Entre ‘S’ para confirmar a exclusão ou ‘N’ para abortar a operação. Ao final da execução o programa retornará ao menu principal.

Para **editar um funcionário**, a partir do menu principal entre o valor ‘3’ e tecle Enter. Na nova página que se abrirá entre o id do funcionário a ser editado e tecle Enter. Se o registro for encontrado o programa o exibirá e uma opção de confirmação será aberta. Caso não queira editar o funcionário entre ‘N’ e o programa retornará ao menu principal, caso contrário entre ‘S’ e tecle Enter. Entre os novos dados para o funcionário, da mesma forma que na opção de adicionar funcionário. Após entrar o cargo, o programa perguntará se deseja confirmar a mudança, entre ‘S’ para confirmar e voltar ao menu principal, ou ‘N’ para abortar a operação e voltar ao menu principal.

Para **buscar um funcionário**, a partir do menu principal entre o valor ‘4’ e tecle Enter. Um submenu será exibido, caso queira **buscar por id** entre o valor ‘1’ e depois o id do funcionário. Caso queira **buscar por nome** entre o valor ‘2’ e depois o nome do funcionário exatamente da mesma forma como foi cadastrado. Em ambos os casos caso seja encontrado um registro correspondente, esse será apresentado, caso contrário uma mensagem será exibida. O programa retornará ao menu principal.

Caso haja alguma dificuldade em encontrar um funcionário pela busca, recomenda-se exibir todos os funcionários por id, dessa forma não será necessário se lembrar de nenhum id e nenhum registro será ocultado.

Resultado dos testes

No processo de desenvolvimento do programa muitos testes foram realizados a fim de que indicassem possíveis irregularidades no funcionamento dele. Ao final da implementação, todos os testes realizados apresentaram resultados satisfatórios quanto ao funcionamento do software. O programa é capaz de realizar todas as funções as quais eram propostas ao mesmo.

A única observação que pode ser feita é que nem todos os tratamentos de erros foram realizados, isso deve-se ao grande número de entradas possíveis em um número considerável de entradas que o programa permite. Assim, realizar a entrada de valores alfanuméricos em campos numéricos podem gerar mal funcionamento do programa. Entrar mais de 100 caracteres em campos alfanuméricos, como nome e endereço do funcionário, pode comprometer os campos seguintes. Sugere-se que o programa seja utilizado da forma que foi desenvolvido para sê-lo.

Problemas e como foram resolvidos

A maior dificuldade, com toda a certeza, foi entender como realmente funcionava a proteção de dados no `arquivo.c`, juntamente à `biblioteca.h`. Como à princípio a ideia de que cada vez que mudasse uma função tivesse também que mudar seus protótipos na biblioteca não parecia muito convidativa, e certamente exigiria um esforço maior, escrevi o programa inteiro diretamente pela biblioteca e posteriormente colocá-lo-ia no arquivo .c, o que me deu bastante dor de cabeça.

Inicialmente estava tendo dificuldades em alocar memória para os registros, pois não havia me atentado que no material de apoio, material passado em aula, os `typedefs` "`Lista`" e "`ELEM`" eram diferentes ("`ELEM`" era um "`struct ELEMENTO`" enquanto "`Lista`" um ponteiro de "`struct ELEMENTO`"), e estava tentando alocar memória com o tamanho de um ponteiro, o que

certamente não caberia uma `struct`. Dessa forma o programa sempre dava erro ao fechar, ou mesmo era encerrado no meio da execução.

Para resolver esse problema removi o ponteiro do `typedef` e na alocação de memória para a lista ligada coloquei diretamente um ponteiro para um ponteiro (`cad**`), inclusive na criação da variável no `main()` e nas funções relacionadas à lista, e tudo parecia funcionar. Fiz o mesmo para a lista estática e os tipos de dados.

Tudo ocorreu bem até que o código fosse portado da `biblioteca.h` para o `arquivo.c`, então incontáveis erros surgiram e nada mais funcionava. Depois de ler e reler o material de apoio, pude perceber meu erro: a biblioteca não consegue acessar estruturas em um `arquivo.c` a não ser por um ponteiro, então os `typedefs` deveriam todos ser ponteiros para que pudessem ser usados pela `biblioteca.h` e `main.c`, o que me levou a outro erro: tornando os `typedefs` ponteiros para estruturas, todas as minhas variáveis com ponteiros de dois níveis (`**`) se tornaram ponteiros de três níveis (`***`), e a função de alocação de memória dinâmica novamente tinha como referência o tamanho de um ponteiro, o que novamente gerou inúmeros erros e inviabilizou a execução do programa.

Graças à padronização na declaração de funções e variáveis, pude substituir as declarações com certa facilidade utilizando um recurso do próprio CodeBlocks. Em relação à alocação de memória, já que as funções de alocação estavam todas encapsuladas, passei a estrutura dos dados diretamente como parâmetro para o `sizeof()` do `malloc()`. Fiz isso para todas as estruturas e o programa funcionou sem maiores problemas com todos os dados encapsulados, apenas utilizando ponteiros para estruturas na biblioteca.