ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Projeto de Laboratório de Programação

Licenciatura em Engenharia Informática

2021/2022

Grupo 14

Gil Leão – 8210414

Pedro Nunes – 8210390

Rui Ferreira – 8210399

1. Introdução

Neste projeto foi-nos proposta a criação de uma loja de calçado online, tal como já tínhamos vindo a desenvolver durante as aulas de Laboratório de Programação. No entanto, desta vez o objetivo foi construir um programa ajustado o mais possível à realidade, de forma a ficar mais completo e funcional.

Assim sendo, introduzimos várias funcionalidades solicitadas no enunciado, tais como a gestão de clientes, de artigos e de tabelas de custo por parte do administrador, sendo que o mesmo pode criar, editar e remover clientes e artigos e ainda alterar as tabelas de custo. Para além disso, outra das funcionalidades foi o registo das encomendas, por parte do cliente, e posterior resumo da encomenda, gerado num ficheiro.

Naturalmente, devido à exigência e complexidade do projeto, não negamos que usufruímos da ajuda da internet e de opinião de terceiros em algumas ocasiões, mas tudo isso contribuiu para a melhor perceção do trabalho e preparação do mesmo.

2. Funcionalidades requeridas

No enunciado deste trabalho foram-nos propostas várias funcionalidades a implementar, sendo possível para o administrador fazer a gestão de clientes, de artigos e da tabela de custos e para o cliente fazer o registo de uma encomenda.

```
case 2:
    do {
        printf("\nAdministrador----------------------------------------------------");
        printf("\n1 - C.R.U.D. de clientes.");
        printf("\n2 - C.R.U.D. de artigos.");
        printf("\n3 - C.R.U.D. das tabelas de custo.");
        printf("\n0 - Sair");
        printf("\n-------------------------------------------------------------");

        printf("\nOpcão: ");
        scanf("%d", &opcao2);
```

Gestão de Clientes

Quanto à gestão de clientes, é pedido que seja possível inserir, editar/atualizar e remover.

```
    case 1:
        do {
            printf("\nC.R.U.D. de clientes---------------------------------------");
            printf("\n1 - Inserir");
            printf("\n2 - Procurar");
            printf("\n3 - Atualizar");
            printf("\n4 - Remover");
            printf("\n5 - Listar");
            printf("\n0 - Sair");
            printf("\n-------------------------------------------------------------");
            printf("\nClientes: %d/%d", clientes.contador, clientes.tamanho);

            printf("\nOpcão: ");
```

INSERIR CLIENTES

Para inserir clientes, utilizamos a função abaixo representada:

```
void inserirClientes(Clientes *clientes, char *ficheiro) {
    if (clientes->contador == clientes->tamanho) {
        expandirClientes(clientes);
    }

        if (inserirCliente(clientes) == -1) {
            puts(ERRO_CLIENTE_EXISTE);
        } else {
            inserirClienteFX(*clientes, ficheiro);
        }
}
```

Nesta função, o primeiro "if" verifica se já nos encontramos no limite máximo de clientes. Se isto de facto se verificar irá ativar a função "expandirClientes", que contém memória dinâmica e irá permitir a inserção de mais clientes.

No segundo "if", caso a função "inserirCliente" retorne -1, significa que esse cliente já se encontra atribuído ("ERRO_CLIENTE_EXISTE") e, caso contrário ("else"), irá guardar os clientes num ficheiro binário, através da atribuição da função "inserirClienteFX".

ATUALIZAR CLIENTES

Para editar clientes, utilizamos a função abaixo representada:

```
void atualizarClientes(Clientes *clientes, char *ficheiro) {
    int cod_cli = procurarCliente(*clientes, obterInt(VAL_MIN,
            clientes->contador - 1, MSG_OBTER_NUM_CLIENTE));

    if (cod_cli != -1) {
        atualizarCliente(&(*clientes).clientes[cod_cli]);

        atualizarClienteFX(*clientes, cod_cli, ficheiro);
    } else {
        puts(ERRO_CLIENTE_NAO_EXISTE);
    }
}
```

Nesta função a função "procurarCliente" vai pedir um valor e procurar o mesmo, retornando -1 caso não encontre e vai retornar o código do cliente se existir. Se for -1 significa que o cliente não se encontra atribuído ("ERRO_CLIENTE_NAO_EXISTE"). Caso contrário vai pedir os novos dados do cliente e atualizar o mesmo através da função "atualizarCliente". A função "atualizarClienteFX" vai atualizar o ficheiro binário.

REMOVER CLIENTES

Para remover clientes, utilizamos a função abaixo representada:

```
*/
void removerClientes(Clientes *clientes, char *ficheiro) {
    int i, cod_cli = procurarCliente(*clientes, obterInt(VAL_MIN, clientes->contador - 1, MSG_OBTER_NUM_CLIENTE));

    if (cod_cli != -1) {
        for (i = cod_cli; i < clientes->contador - 1; i++) {
            clientes->clientes[i] = clientes->clientes[i + 1];
        }

        apagarDadosCliente(&clientes->clientes[i]);
        clientes->contador--;

        removerClienteFX(*clientes, ficheiro);
    } else {
        puts(ERRO_CLIENTE_NAO_EXISTE);
    }
}
```

Nesta função a função “procurarCliente” vai pedir um valor e procurar o mesmo, retornando -1 caso não encontre e vai retornar o código do cliente se existir. Se for -1 significa que o cliente não se encontra atribuído (“ERRO_CLIENTE_NAO_EXISTE”). Caso contrário vai apagar os dados do cliente através da função (“apagarDadosCliente”). A função (“removerClienteFX”) remove os clientes do ficheiro binário.

Gestão de Artigos

```
case 2:

    do {
        printf("\nC.R.U.D. de artigos--------------------------------------");
        printf("\n1 - Inserir");
        printf("\n2 - Procurar");
        printf("\n3 - Atualizar");
        printf("\n4 - Remover");
        printf("\n5 - Listar");
        printf("\n0 - Sair");
        printf("\n-------------------------------------------------------------");
        printf("\nArtigos: %d/%d", artigos.contador, artigos.tamanho);

        printf("\nOpcão: ");
        scanf("%d", &opcao3);
```

3. Funcionalidades propostas

1ª: Uma das funcionalidades implementadas pelo grupo foi a criação de um histórico, que indica as últimas atividades realizadas pelo utilizador. Para isso usamos a função a seguir apresentada:

```c
void logMsg(char *msg, char *filename) {
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);

    FILE *fp = fopen(filename, "a");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fprintf(fp, "%d-%02d-%02d %02d:%02d:%02d - %s\n", tm->tm_year + 1900,
            tm->tm_mon + 1, tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec, msg);

    fclose(fp);
}
```

```c
case 1:
    inserirArtigos(&artigos, ARTIGOS_DB_FILE);
    logMsg("Inserir_art", LOGS_FILE);
    break;
case 2:
    procurarArtigos(artigos);
    logMsg("Procurar_art", LOGS_FILE);
    break;
```

No caso acima apresentado, depois de o utilizador selecionar a opção desejada, por exemplo "procurar artigos", a função "logMsg" é chamada e vai adicionar ao ficheiro "LOGS_FILE" a mais recente opção escolhida pelo utilizador, acompanhado da hora e data em que foi selecionada.

Ou seja, esta função vai gerar em ficheiro as últimas ações do utilizador no programa, bem como a data e hora da realização das mesmas.

logs.txt - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

```
2022-01-10 14:31:46 - Inserir_cli_cli
2022-01-10 14:31:56 - Procurar_cli_cli
2022-01-11 23:10:19 - Procurar_cli_cli
2022-01-11 23:10:37 - Inserir_cli_cli
2022-01-11 23:10:42 - Procurar_cli_cli
2022-01-11 23:11:15 - Inserir_cli_cli
2022-01-11 23:11:34 - Listar_cli
2022-01-11 23:11:36 - Listar_cli
2022-01-11 23:11:43 - Procurar_cli
2022-01-11 23:11:46 - Procurar_cli
2022-01-11 23:11:49 - Procurar_cli
2022-01-11 23:11:56 - Remover_cli
2022-01-11 23:12:00 - Remover_cli
2022-01-11 23:12:04 - Remover_cli
2022-01-11 23:12:09 - Remover_cli
2022-01-11 23:12:11 - Remover_cli
2022-01-11 23:12:24 - Inserir_cli
2022-01-11 23:12:27 - Listar_cli
2022-01-11 23:12:51 - Actualizar_cli
```

2ª: Para além disso também implementamos uma função que permita ao administrador procurar os clientes que já se encontrem com conta criada.

Caso o utilizador já tenha criado conta, ou seja, se ao procurar cliente o código de cliente que for inserido já existir é porque o cliente realmente existe e o cliente é imprimido, através da função "imprimirCliente". Caso contrário, ou seja, se o código introduzido quando se está a procurar cliente não estiver atribuído, aparece no ecrã a mensagem "O cliente não existe na lista", mensagem esta apresentada pela constante "ERRO_CLIENTE_NAO_EXISTE".

```c
void procurarClientes(Clientes clientes) {
    int cod_cli = procurarCliente(clientes, obterInt(VAL_MIN, clientes.contador - 1, MSG_OBTER_NUM_CLIENTE));

    if (cod_cli != -1) {
        imprimirCliente(clientes.clientes[cod_cli]);
    } else {
        puts(ERRO_CLIENTE_NAO_EXISTE);
    }
}
```

```c
int procurarCliente(Clientes clientes, int cod_cli) {
    int i;
    for (i = 0; i < clientes.contador; i++) {
        if (clientes.clientes[i].cod_cli == cod_cli) {
            return i;
        }
    }
    return -1;
}
```

3ª: Outra das funcionalidades que decidimos implementar, tal como fizemos para os clientes, foi criar uma função que permita ao administrador procurar os artigos já inseridos.

Caso já tenham sido inseridos artigos, ou seja, se ao procurar artigos o código de artigo que for inserido já existir é porque o artigo já foi inserido e é imprimido, através da função "imprimirArtigo", o seu código, nome, tipo de calçado, numero máximo e número mínimo. Caso contrário, ou seja, se o código de artigo introduzido quando se está a procurar artigo não estiver atribuído, aparece no ecrã a mensagem "O artigo não existe na lista", mensagem esta apresentada pela constante "ERRO_ARTIGO_NAO_EXISTE".

```
void procurarArtigos(Artigos artigos) {
    int cod_art = procurarArtigo(artigos, obterInt(MIN_NUM_ALUNO, MAX_NUM_ALUNO, MSG_OBTER_NUM_ALUNO));

    if (cod_art != -1) {
        imprimirArtigo(artigos.artigos[cod_art]);
    } else {
        puts(ERRO_ARTIGO_NAO_EXISTE);
    }
}
```

```
int procurarArtigo(Artigos artigos, int cod_art) {
    int i;
    for (i = 0; i < artigos.contador; i++) {
        if (artigos.artigos[i].cod_art == cod_art) {
            return i;
        }
    }
    return -1;
}
```

4ª : Decidimos também, entendendo que torna o programa mais completo e ajustado à realidade, criar uma função que permita ao utilizador demonstrar o seu grau de satisfação relativamente ao serviço prestado. Esta vai gerar em ficheiro a classificação que o utilizador atribuir.

```c
void grau(char *ficheiro){
    int grau;
    do {
        printf("\n-------------------------------------------------"
               "------------");
        printf("\n1 - ★");
        printf("\n2 - ★★");
        printf("\n3 - ★★★");
        printf("\n4 - ★★★★");
        printf("\n5 - ★★★★★");
        printf("\n0 - Sair");
        printf("\n-------------------------------------------------"
               "------------");

        printf("\nOpcão: ");
        scanf("%d", &grau);
    }while(grau < 0 || grau > 5);
    if(grau == 0){
    }else{
        FILE *fpp = fopen(ficheiro, "a");
        fclose(fpp);
        FILE *fp = fopen(ficheiro, "w");
        if (fp == NULL) {
            exit(EXIT_FAILURE);
        }
        fprintf(fp, "grau de satisfaçao: %d ★\n", grau);
        fclose(fp);
    }
}
```

5ª: Também criamos uma funcionalidade que consiste em gerar automaticamente o id do utilizador, ou seja, tendo 2 clientes, o id do primeiro cliente será 0 e o id do segundo cliente será 1. Neste sentido, os próximos clientes a criar conta ficarão quando o id sucessivamente maior.

```
0 pedro 123456789 pt
1 rui 123412341 pt
```

4. Estrutura analítica do projeto

Para desenvolvermos o trabalho de forma mais prática e eficaz, primeiramente começamos por ler atentamente a estrutura do relatório e pensar no código que iríamos utilizar para cada funcionalidade a implementar.

Iniciamos o trabalho através de uma chamada de grupo no discord (aplicativo de voz sobre IP proprietário e gratuito, projetado inicialmente para comunidades de jogos, mas também vulgarmente utilizado para realização de projetos) , apontamos todos os pormenores em que tínhamos que ter atenção para a elaboração do projeto e de seguida realizamos todos os menus para ficarmos com uma ideia mais clara de como abordar o restante código e, a partir daí, fomos a cada opção dos menus e introduzimos as funções necessárias.

Depois definimos o que cada um deveria fazer. Sendo que o Gil ficou com as funcionalidades do perfil administrador, o Pedro ficou com as funcionalidades do perfil cliente e o Rui ficou responsável por toda a memória dinâmica. Quanto à persistência de dados, tivemos de esperar pela aula de FP em que abordamos essa parte da matéria e depois realizamo-la em conjunto.

No entanto, sempre que surgia alguma dúvida ou problema, combinávamos e voltávamos a reunir em grupo no discord, para resolver o problema.

O relatório foi realizado depois de ter o código completo e todos contribuímos para o mesmo.

Quanto à documentação do código, deixamos para o fim e recorremos ao DOXYGEN.

5. Funcionalidades implementadas

Gestão de clientes – Todas as funcionalidades que foram propostas para a gestão de clientes foram implementadas.

Gestão de artigos – Todas as funcionalidades relativas à gestão de artigos foram implementadas.

Persistência de dados – Todos os dados foram corretamente gerados em ficheiro, tais como a lista de clientes, o resumo das encomendas, a lista de artigos, o grau de satisfação e o histórico de pesquisa.

Memória dinâmica – Sempre que necessário foi implementada memória dinâmica.

Listagens de dados – Foram desenvolvidas 5 listagens de dados, listagens estas que foram descritas no ponto 3 deste relatório.

6. Conclusão

Com a realização deste trabalho, para além de desenvolvermos interessantes dinâmicas de grupo, consolidamos de forma mais sólida todo o conteúdo lecionado durante o 1º semestre nas cadeiras de Fundamentos da Programação e Laboratório da Programação!

Ao longo da realização do trabalho foi percetível a extrema importância de adicionar comentários à linha de código. Isto é extremamente essencial uma vez que permite aos elementos entender com mais facilidade o código escrito e, para além disso, facilita a interpretação do código depois de não olhar para o mesmo durante algum tempo!

O relatório em si foi uma grande ajuda, uma vez que à medida que fazíamos a descrição do trabalho efetuado e das funcionalidades implementadas, ficamos com uma ideia melhor

construída acerca daquilo que fizemos, o que permitiu a todos os elementos do grupo entender o código mais facilmente!

Com isto concluímos que este trabalho nos ajudou a adquirir imensas bases que vão, certamente, ser-nos bastante úteis no futuro!