Curso de Graduação em Ciência da Computação
Disciplina: Algoritmos 2
Professor: Manoel Pereira Junior

Documentação Técnica
Sistema de Controle Gerencial Para Hotel

Autores:
1 – Ana Luiza Guimarães Ramos N° 0035347
2 – Jean Claudio Teixeira N° 0040067
3 – Thomas Santos Pollarini N° 0064232

Formiga, 24 de novembro de 2023.

1. Introdução

O objetivo desta documentação é explicar como o sistema de controle gerencial para um hotel foi desenvolvido, quais são os seus módulos e funcionalidades, assim como as decisões tomadas mediante aos requisitos apresentados pelo enunciado.

Foi requisitado a criação de um sistema gerencial de um Hotel completo com os seguintes módulos:

- Módulo de Cadastros e Gestão de dados
- Módulo de Reservas
- Módulo de Transações
- Módulo de Feedback
- Módulo de importação/exportação de dados

2. Estruturas

Este módulo comporta todas as funções de entrada, edição e exclusão de dados. Para implementar este módulo foi necessário a criação de algumas structs para o armazenamento dos dados que posteriormente viriam dos arquivos. Todas as *structs* geradas durante a implementação estão no cabeçalho "bancoDados.h".

2.1. Hotel

Esta *struct* contém os dados do hotel: nome fantasia, razão social, inscrição estadual, CNPJ, endereço, telefone, e-mail, responsável (nome e telefone), horários de check-in e check-out, margem de lucro.

```
//struct com variáveis sobre o cadastro do hotel
typedef struct{
    int  inscEstadual,horaCheckin,horaCheckout;
    float margem;
    char nome[50],razaoSocial[30],email[50],nomeResp[50],
         cnpj[15],telefone[12],telefoneResp[12];
    endereco end;
}hotel;
```

O campo de endereço foi abstraído para a seguinte *struct*:

```
typedef struct{
    int num;
    char rua[50],comp[50],bairro[50],cidade[30],estado[30];
}endereco;
```

2.2 Hóspedes

Esta *struct* contém os dados dos clientes do hotel: código (único), nome, endereço, CPF, telefone, e-mail, sexo, estado civil, data de nascimento.

```
typedef struct{
    int cod;
    char nome[50],email[50],sexo[30],civil[30],nasc[30],cpf[12],telefone[12];
    endereco end;
}hospede;
```

2.3 Acomodações

Esta *struct* contém os dados das acomodações oferecidas pelo Hotel: código (único), descrição, facilidades (ex.: televisão, ar-condicionado, etc.), categoria (pré-cadastrada).

```
typedef struct{
        int cod,cat;
        char desc[100],fac[50];
    }acomodacao;
```

2.4 Categoria de Acomodação

Esta *struct* contém as categorias de acomodação. A categoria indica o estilo da acomodação (standard, luxo, presidencial, etc.), bem como o preço de sua diária: código (único), descrição, valor da diária, quantidade de pessoas que comporta (adultos e crianças).

```
typedef struct{
            int cod,crianca,adulto;
            float vlr;
            char desc[100];

        }categoria;
```

2.5 Produto

Esta *struct* contém os dados dos produtos (refrigerantes, frios, balas, chocolates, etc) que podem ser consumidos (vendidos aos hóspedes): código (único), descrição, estoque, estoque mínimo, preço de custo, preço de venda.

```
typedef struct{
    int cod,est,estmin;
    float custo,venda;
    char desc[50];
}produto;
```

2.6 Fornecedores

Esta *struct* contém os dados dos fornecedores do hotel. Os fornecedores vendem seus produtos ao hotel, que por sua vez os revende aos hóspedes: código (único), nome fantasia, razão social, inscrição estadual, CNPJ, endereço completo, telefone, e-mail.

```
typedef struct{
    int cod,inscEstadual;
    char nome[50],razaoSocial[30],email[50],cnpj[15],telefone[12];
    endereco end;
}fornecedor;
```

2.7 Operadores do Sistema

Esta *struct* contém os dados dos funcionários do hotel que irão operar o sistema. a partir dessa estrutura é concedida ou negada permissões em algumas funcionalidades do sistema: código (único), nome, usuário, senha, permissões de acesso.

```
typedef struct{
    int cod,acesso[5];
    char usuario[50],nome[50],senha[50];
}operador;
```

2.8 Pesquisa

Esta *struct* contém informações sobre uma pesquisa, que inclui o número de crianças, adultos, categoria, facilidades oferecidas e a data da pesquisa. Os campos incluem o número de crianças, o número de adultos, a categoria, as facilidades oferecidas, e a data da pesquisa.

```
typedef struct{
        int crianca,adulto,cat;
        char fac[50];
        data data_pes;
}pesquisa;
```

2.9 Comanda

Esta *struct* representa uma comanda, contendo um código único e um valor associado.

```
typedef struct{
    int cod;
    float valor;
}comanda;
```

2.10 CheckIn

Esta *struct* representa um check-in, contendo um código, código de reserva, indicador de pagamento e uma lista de produtos associados.

```
typedef struct{
    int cod,reserva,pagou;
    comanda *prod;
}checkin;
```

2.11 Caixa

Esta struct representa uma movimentação de caixa, contendo um código, valor , tipo, método de pagamento e a data da transação .

```
typedef struct{
    int cod;
    float valor;
    char tipo[15],pag[10];
    data dataTrans;
}movCaixa;
```

2.12 Movimentação de Cartão

Esta *struct* representa uma movimentação de cartão, contendo um código (cod), número de parcelas (parc), código de chave, valor e data de vencimento.

```
typedef struct{
    int cod,parc,codFor;
    float valor;
    data vencimento;
}movCartao;
```

2.13 Compra

Esta *struct* representa uma compra, contendo um código (cod), quantidade (qtd) e valor unitário (vUnid).

```
typedef struct{
    int cod,qtd;
    float vUnid;
}compra;
```

2.14 Nota

Esta *struct* representa uma nota de entrada, contendo um código, código do fornecedor, valor do frete, valor de impostos, valor total e a lista de produtos comprados.

```
typedef struct{
    int cod,codFor;
    float vFrete,vImp,vTotal;
    compra *prod;
}notaEntrada;
```

2.15 Movimentações de Acomodação

Esta *struct* representa uma movimentação de acomodação, contendo um código, código de acomodação, número de dias e valor.

```
typedef struct{
    int cod,codAco,numDia;
    float valor;
}movAcomodacao;
```

2.16 Venda

Esta *struct* representa uma venda, contendo um código, código do produto , valor e método de pagamento.

```
typedef struct{
    int cod,codPro;
    float valor;
    char pag[10];
}venda;
```

As estruturas acima serão utilizadas ao longo da implementação, para armazenar os dados criados durante o programa para garantir a persistência dos mesmos, é utilizado arquivos .txt ou .bin (texto ou binário) a escolha do usuário do sistema. Para manipular isso temos um arquivo separado chamado “bancoDados.c”.

3. Arquivos e implementação

3.1 “bancoDados.c”

Ao começo do programa é verificado a existência de dois arquivos, “Texto.txt” ou “Binario.bin”, se nenhum dos dois existir significa que é a primeira vez sendo executado o sistema e assim é chamada a função “**inicializa_Arquivos**” sem nenhum parâmetro. A função “**inicializa_Arquivos**” é responsável por criar e inicializar arquivos de dados necessários para o funcionamento do sistema. Ela verifica se o sistema está sendo executado pela primeira vez e, se for o caso, cria os arquivos necessários e define o formato dos dados como texto por padrão. Se não for a primeira execução, ela verifica a existência dos arquivos binários correspondentes e os cria se necessário. A função lida com vários arquivos, incluindo dados sobre o hotel, hóspedes, acomodações, categorias, produtos, fornecedores, reservas, check-ins, comandas, caixa, movimentação do caixa, movimentação de acomodações, contas a receber, vendas, contas a pagar, notas de entrada e compras. A função também é projetada para lidar com possíveis erros ao criar ou escrever nos arquivos.

Há muitas funções no arquivo “**bancoDados.c**” porém todas elas seguem um padrão de implementação, para cada *struct* principal existem 3 funções, “**dados**”, “**cadastrar**” e “**atualizar**”, além disso há funções de relatório que também são padronizadas “**relatorio**”.

As funções de prefixo **dados**, são responsáveis por carregar o banco de dados da entidade específica na memória. Ela verifica o tipo de arquivo utilizado pelo sistema e, dependendo disso, lê os dados da entidade de um arquivo de texto ou binário. A função utiliza fgets e strtok para ler e tokenizar o arquivo de texto, e fread para ler o arquivo binário. Ela armazena os dados lidos no vetor passado como parâmetro, realocando o vetor conforme necessário para acomodar novos registros. A função também trata possíveis erros durante a abertura ou manipulação dos arquivos, exibindo mensagens de erro se necessário. Após a leitura bem-sucedida dos dados, a função retorna o vetor contendo cada entidade e seus respectivos dados.

As funções de prefixo **cadastrar**, são responsáveis por registrar uma nova entidade no sistema, seja em um arquivo de texto ou binário. Elas recebem como parâmetro as informações da nova entidade. As funções verificam o tipo de arquivo utilizado pelo sistema e, dependendo disso, abre o arquivo correspondente no modo de adição para adicionar os novos dados ao final do arquivo. As funções utilizam fprintf para escreverem nos arquivos de texto e fwrite para os arquivos binário. Elas gravam as informações da nova entidade no arquivo e, após o registro bem-sucedido dos dados, as funções limpam a tela do console. As funções também tratam possíveis erros durante a abertura ou manipulação dos arquivos, exibindo mensagens de erro se necessário.

As funções de prefixo **atualizar**, são responsáveis por atualizar os dados das respectivas entidades nos arquivos, sejam eles em formato de texto ou binário. Elas recebem vetores de estruturas do tipo da entidade e o comprimento desses vetores como parâmetros. As funções verificam o tipo de arquivo utilizado pelo sistema e, dependendo disso, abrem os arquivos correspondentes no modo de escrita para recriá-los do zero. As funções utilizam fprintf para escrever nos arquivos de texto e fwrite para os arquivos binários. Elas gravam os dados das entidades nos arquivos e, após a atualização bem-sucedida dos dados, as funções limpam a tela do console. As funções também tratam possíveis erros durante a abertura ou manipulação dos arquivos, exibindo mensagens de erro se necessário.

As funções com prefixo **relatorio**, têm como objetivo escrever relatórios das respectivas entidades em arquivos. Essas funções recebem como parâmetros um ponteiro para o arquivo onde o relatório será escrito e um vetor de estruturas contendo as informações que comporão o relatório. As funções começam obtendo o comprimento do vetor usando a função apropriada. Em seguida, utilizam `fprintf` para escrever o índice do relatório no arquivo, indicando as informações que serão incluídas. Dentro de um loop, as funções percorrem cada item no vetor e escrevem suas informações no arquivo, utilizando vírgulas como delimitadores para criar um formato de CSV (Comma-Separated Values). Cada linha do arquivo representa as informações de um item, e cada campo é separado por vírgulas. As funções tratam possíveis erros ao escrever no arquivo, verificando se o resultado de `fprintf` é menor que zero. Se ocorrer um erro, imprimem uma mensagem de erro e retornam. Finalmente, após a conclusão bem-sucedida da escrita do relatório, as funções fecham o arquivo e imprimem uma mensagem indicando que o relatório foi criado com sucesso. Essas funções são úteis para gerar relatórios de forma organizada e estruturada em um formato CSV, que pode ser facilmente importado e lido por diversas ferramentas e aplicativos de planilhas.

3.2 “Util.c”

As funções `**cria_ADM**`, `**login**`, `**verifica_Acesso**` e `**menu_Acesso**` são responsáveis pelo gerenciamento de contas de operadores e suas permissões de acesso. A função `cria_ADM` é responsável por criar uma nova conta administrativa, recebendo um ponteiro para uma estrutura de operador como parâmetro e coletando informações como código da conta, nome, nome de usuário e senha, definindo permissões de acesso padrão para todos os módulos. A função `login` é usada para autenticação do usuário, solicitando ao usuário que insira um nome de usuário e senha, comparando essas informações com os dados no banco de operadores e concedendo acesso se as credenciais estiverem corretas. A função `verifica_Acesso` verifica se um determinado operador tem acesso a um módulo específico, retornando 1 se o acesso for concedido e 0 caso contrário. Por fim, a função `menu_Acesso` gerencia as permissões de acesso para um determinado operador, exibindo um menu com opções relacionadas a diferentes módulos e permitindo que o administrador conceda ou revogue o acesso a esses módulos para outro operador. Essas funções garantem que as contas de operadores sejam gerenciadas de forma eficaz e segura, protegendo a integridade do sistema.

A função de `**menu_Tabelas**` é responsável por permitir ao usuário escolher quais arquivos/módulos que serão importados ou exportados durante o módulo de importação/exportação, chamando a função respectiva dos arquivos “**importacao.c**” e “**exportacao.c**” e passando como parâmetro as opções que o usuário escolheu.

A função `**coleta_data**` solicita ao usuário que insira as datas de início e término da estadia. As datas são convertidas em um formato numérico para comparação. Se a data de check-out for igual ou anterior à data de check-in, o sistema exibe uma mensagem de erro e solicita novamente as datas. Este processo se repete até que o usuário forneça uma data de check-out posterior à data de check-in.

O arquivo possui várias funções de **filtro**, elas recebem ponteiros das opções de filtro e exibem ao usuário opções para selecionar quais campos desejam incluir no filtro e seus respectivos valores, assim alterando o que será pesquisado através do ponteiro, como por exemplo, exibir somente as acomodações que possuem código entre 1 e 5, cada função filtro possui seus próprios campos que são pertinentes ao filtro. Após isso a função retorna um *int* onde 1 significa que o relatório foi cancelado e 0 permite que o relatório seja gerado.

Há também várias funções com o prefixo **len** que tem como função retornar a quantidade de entidades que estão no respectivo arquivo do banco, esse valor ajuda a manipular os dados em outras funções.

A função `**tipo_Arquivo**` é responsável por permitir que o usuário escolha o tipo de arquivo a ser utilizado pelo sistema, seja texto ou binário. Através de um menu simples, o usuário seleciona a opção desejada, e a função realiza as devidas verificações e ações conforme a escolha. Se o usuário optar pelo tipo de arquivo texto, a função verifica se os arquivos já são do tipo texto e, caso afirmativo, emite uma mensagem informando que os arquivos já são desse formato. Se os arquivos forem do tipo binário, a função chama a `**converte_Arquivos**` para realizar a conversão para o formato texto. Se não houver nenhum arquivo, a função cria arquivos no formato texto e os inicializa. De maneira análoga, para a escolha do tipo binário, a função realiza verificações semelhantes, emitindo mensagens apropriadas e chamando a `**converte_Arquivos**` quando necessário. Em caso de cancelamento, a função emite uma mensagem indicando que a ação foi cancelada, proporcionando uma interação intuitiva com o usuário durante a configuração do sistema. Essa funcionalidade é crucial para a flexibilidade do sistema, permitindo a fácil adaptação entre os diferentes formatos de arquivos de acordo com as necessidades do usuário.

A função `**converte_Arquivos**` é responsável por converter entre formatos de arquivo texto e binário seguindo a opção desejada pelo usuário. Ela verifica a existência do banco e puxa todas as informações para a memória, remove arquivos antigos, cria novos no formato desejado e atualiza os bancos de dados associados. Em caso de erro, emite uma mensagem indicativa, e em caso de sucesso, informa que a conversão foi bem-sucedida.

A função `**associa_Compras**` é responsável por vincular vetores de compras às respectivas notas de entrada, utilizando estruturas de dados. Ela percorre cada nota de entrada, aloca dinamicamente um vetor temporário para as compras associadas e copia os dados das compras para este vetor. Após percorrer todas as compras, o vetor temporário é associado à nota de entrada correspondente. Esta função facilita a manipulação de informações relacionadas a compras e notas de entrada em operações subsequentes.

3.3 “servicetransacoes.c”

Neste arquivo é feita a interação com o usuário para a requisição de dados para as funções de manipulação de dados no banco, também é feita validações.

As funções `**pagar_Diarias**`, `**pagar_Produtos**`, `**pagar_Fornecedor**` e `**pagar_Nota**` são fundamentais para a realização de pagamentos em um sistema de hospedagem, em uma comanda, para fornecedores e para a baixa de notas, respectivamente. Todas essas funções calculam o valor total devido, seja das diárias, dos produtos, das notas de entrada ou das notas de pagamento, e permitem ao usuário escolher a forma de pagamento. Dependendo da opção de pagamento escolhida, as funções atualizam o caixa do sistema ou registram uma conta a receber ou a pagar. Além disso, um registro de movimentação

é criado e cadastrado no banco de dados, seja ele associado ao pagamento das diárias, à venda de produtos, ao pagamento a fornecedores ou à baixa de notas. Essas funções garantem a atualização adequada do caixa e o registro preciso de todas as transações, contribuindo para a integridade e transparência no controle financeiro do sistema de hospedagem, do estabelecimento, no pagamento a fornecedores ou na gestão de contas a pagar.

A função `**receber_Nota**` é essencial para o processo de dar baixa em notas de contas a receber, permitindo a efetivação do recebimento de valores de contas parceladas. Ela valida o código da nota, apresenta os dados da nota ao usuário e permite a opção de dar baixa. Se a opção for dar baixa, a função atualiza o caixa, lançando o valor da nota, e remove a nota do banco de dados de contas a receber. Essa função contribui para a transparência e controle financeiro do sistema, garantindo a integridade e precisão nas informações financeiras.

3.4 “serviceCadastros.c”

Neste arquivo é feita a interação com o usuário para a requisição de dados para as funções de manipulação de dados no banco, também é feita validações.

As funções `**cadastrar**`, `**atualizar**` e `**excluir**` neste arquivo desempenham um papel crucial na manipulação de dados no sistema. Elas são responsáveis por interagir com o usuário para coletar os dados necessários, carregar o banco de dados existente e realizar uma verificação de validade dos dados inseridos pelo usuário. Se os dados passarem na verificação, as funções correspondentes no arquivo `**bancoDados.c**` são chamadas para executar a operação desejada. Essas funções garantem que os dados sejam manipulados de maneira correta e segura, mantendo a integridade do banco de dados.

Por outro lado, as funções `**ver**` neste arquivo são responsáveis por exibir os dados ao usuário. Elas chamam as funções correspondentes no arquivo `**bancoDados.c**` para recuperar os dados necessários e, em seguida, imprimem esses dados na tela. Isso permite que o usuário visualize as informações de maneira clara e organizada, facilitando a compreensão dos dados. Essas funções desempenham um papel importante na apresentação de informações ao usuário, contribuindo para a usabilidade e eficiência do sistema.

3.5 “relatorio.c”

Neste arquivo foi implementado o módulo de *Feedback*, que é responsável por gerar os relatórios e exibir ao usuário.

As funções de prefixo `**lista**` geram relatórios detalhados sobre as respectivas entidades com base em critérios específicos. Elas definem os filtros chamando as funções de **filtro** que estão no **Util.c**, carregam os dados dos hóspedes chamando do banco de dados e inicializam um novo conjunto de dados para armazenar temporariamente os registros que atendem aos critérios de filtragem.

As funções exibem as informações detalhadas das entidades que atendem aos critérios. Se nenhum registro é retornado por não atender aos critérios de filtragem, a função notifica o usuário. Caso contrário, elas armazenam esses dados em um arquivo. As funções dão ao usuário a opção de escolher um arquivo para armazenar o relatório gerado. Essas funções contribuem significativamente para a gestão eficiente do estabelecimento, permitindo um controle mais preciso sobre as informações relacionadas aos hóspedes.

3.6 “importacao.c” e “exportacao.c”

Esses dois arquivos juntos são os responsáveis pela implementação do módulo de importação/exportação de dados do sistema.

A função `**importa_Arquivos**` funciona permitindo a importação de informações de tabelas chamando as funções específicas de prefixo **importa** a partir de um arquivo especificado pelo usuário,é possível importar somente os arquivos escolhidos pelo usuário. Ela solicita ao usuário o caminho do arquivo a ser importado, tenta abrir o arquivo em modo de leitura e lê o conteúdo do arquivo linha por linha. A função utiliza a função `strtok` para dividir a linha em tokens, permitindo a análise e interpretação estruturada do conteúdo do arquivo. Se encontrar uma tag `tabela`, verifica o tipo de tabela indicado e, se a tabela foi solicitada, invoca a função de importação correspondente. Caso contrário, pula para o final da tabela. Se encontrar uma tag desconhecida ou se o arquivo não estiver formatado corretamente, exibe uma mensagem de erro. Ao finalizar a leitura do arquivo, exibe uma mensagem indicando que a importação foi concluída com sucesso e fecha o arquivo. Essa função contribui significativamente para a integridade e precisão dos dados do sistema.

A parte de exportação funciona similarmente através da função `**exporta_Arquivos**` é fundamental para a exportação de dados do sistema para um arquivo externo. Ela solicita ao usuário o caminho do arquivo de destino, tenta abrir o arquivo em modo de escrita e começa a escrever no arquivo com a marca <dados>. Para cada tabela requisitada pelo usuário, a função chama a função de prefixo **exporta** correspondente. Após a exportação, a função fecha o arquivo com a marca </dados>. Se houver algum erro, a função exibe uma mensagem de erro. Caso contrário, exibe uma mensagem de sucesso. Esta função é crucial para backups, transferência de dados e outras operações que requerem a persistência dos dados do sistema em um arquivo externo.

3.7 “main.c”

O arquivo `**main.c**` atua como a interface inicial do usuário para a aplicação, apresentando menus amigáveis e coordenando as chamadas de funções para os diversos módulos do programa. Ele redireciona as requisições do usuário para as funcionalidades específicas, promovendo a modularidade, e reusabilidade de componentes e um desenvolvimento mais eficiente e escalável.