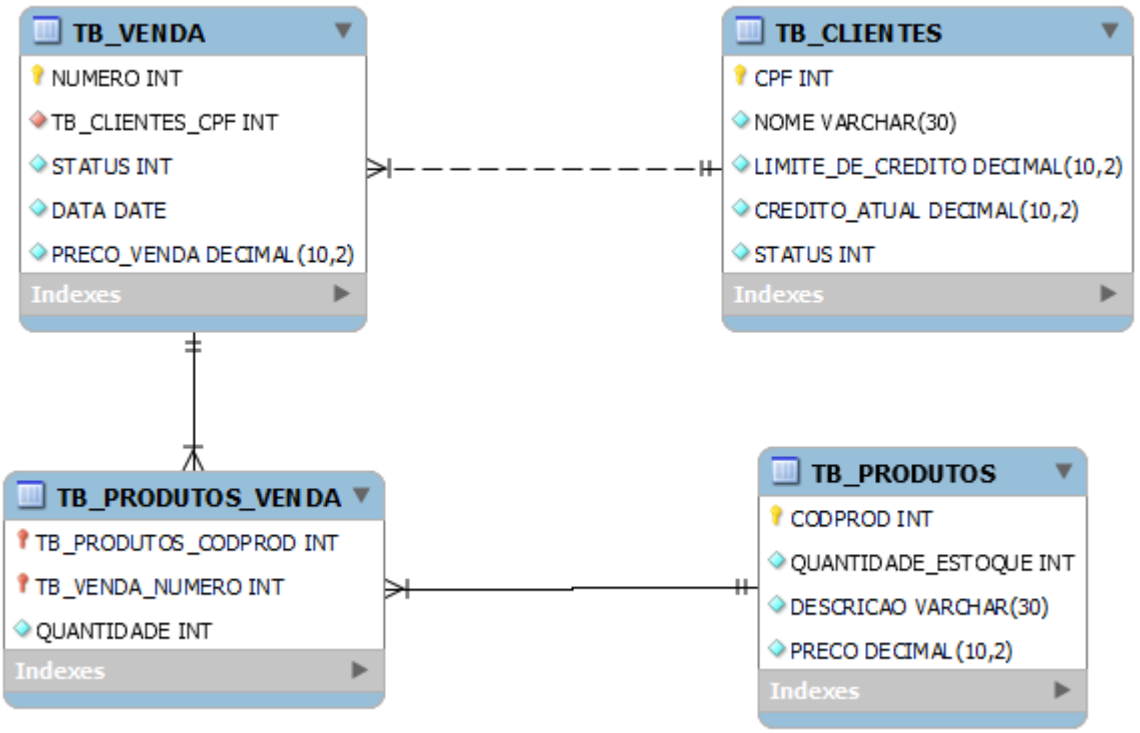
Aluno: Arthur da Silva Bassani
Turma: COM1009/CC2

MER:



OBS: Nas tabelas o cpf está como INT, porém na prática e no código está como Unsigned Long Long.

Definições para facilitar no entendimento do código:

```
#define CONT_PROD "ContProd.xyz"
#define CONT_VEND "ContVend.xyz"
#define TB_CLIENTE "Clientes.xyz"
#define TB_PRODUTOS "Produtos.xyz"
#define TB_VENDA "Vendas.xyz"
#define TB_PRODUTO_VENDA "Produto_Venda.xyz"
#define FILTROS "Filtros.xyz"
#define QUITADO 0
#define DEVENDO 1
#define EM_ATRASO 2
#define DESATIVADO 3
```

As funções que eram pra ser feitas na biblioteca:

Retrieve:

```
void* retrieve  (pDFile arq, void* chave, FuncaoComparacao pfc){
    int resultado = 0;
    void* dados = malloc(sizeof(arq->tamanhoRegistro));
    rewind(arq->arquivo);
    do{
        resultado = fread(dados, arq->tamanhoRegistro,1,arq->arquivo);
        if(pfc(dados,chave) == 0){
            close(arq);
            return dados;
        }
    }while(resultado != 0);
    close(arq);
    return NULL;
}
```

Update:

```
void update(pDFile arq, void* chave, void* dados, FuncaoComparacao pfc){
    if(arq->arquivo == NULL){
        printf("Erro ao abrir o arquivo (update)\n");
        return;
    }
    int resultado = 0;
    rewind(arq->arquivo);
    void* info = malloc(sizeof(arq->tamanhoRegistro));
    do{
        resultado = fread(info, arq->tamanhoRegistro,1,arq->arquivo);
        if(pfc(info,chave) == 0){
        fseek(arq->arquivo, -(sizeof(arq->tamanhoRegistro)), SEEK_CUR);
        fwrite(dados, sizeof(arq->tamanhoRegistro), 1, arq->arquivo);
        printf("Cadastro atualizado com sucesso\n");
        close(arq);
        return;
        }
    }while(resultado != 0);
    close(arq);
}
```

Delete:

```
void deletee(pDFile arq,char nomeArq[30], void* chave, FuncaoComparacao pfc){
    if(arq->arquivo == NULL){
        printf("Erro ao abrir arquivo (delete)\n");
        return;
    }
    FILE* arqTemp = fopen("Temp.xyz","wb+");
    rewind(arq->arquivo);
    int resultado = 0;
    void* dados = malloc(sizeof(arq->tamanhoRegistro));
    do{
        resultado = fread(dados,sizeof(arq->tamanhoRegistro),1,arq->arquivo);
        if(pfc(dados,chave) != 0){
            fwrite(dados,sizeof(arq->tamanhoRegistro),1,arqTemp);
        }
    }while(resultado != 0);

    close(arq);
    fclose(arqTemp);

    if (remove(nomeArq) == 0 && rename("Temp.xyz",nomeArq) == 0){
        printf("Cadastro deletado com sucesso\n");
    }else{
        printf("Erro ao deletar cadastro(delete)\n");
        return;
    }
}
```

QuerryBy:

```
pDLista queryBy (pDFile arq,FuncaoPredicado pred){
   if (arq->arquivo == NULL){
       printf("Arquivo não foi aberto!(querryBy)\n");
       return NULL;
   }

    pDLista pLista = criarLista();

    if (pLista == NULL){
       printf("Erro ao criar lista(querryBy)\n");
       return NULL;
    }

   rewind(arq->arquivo);
   int result;

   do{
      void *dados = malloc(sizeof(arq->tamanhoRegistro));
      result = fread (dados, arq->tamanhoRegistro, 1, arq->arquivo);
      if(pred(dados) == 0){
        incluirInfo(pLista, dados);
      }
   }while(result != 0);
    close(arq);
    if(pLista->quantidade <= 0){return NULL;}
    else                       {return pLista;}
}
```

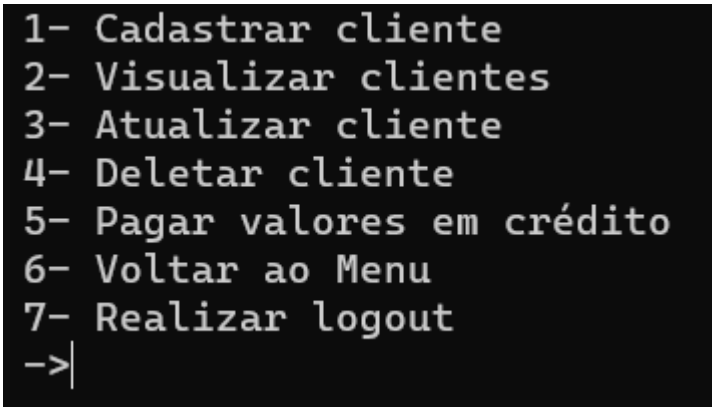
PersistAll:

```c
void persistAll(pDFile arq, pDLista pLista){
    if(arq->arquivo == NULL){
        printf("Erro ao abrir o arquivo (persistAll)\n");
    }
    if(pLista == NULL){
        printf("Erro ao abrir lista (persistAll)\n");
    }
    pNoh Aux = pLista->primeiro;
    while(Aux != NULL){
        createe(arq,Aux->info);
        Aux = Aux->prox;
    }
    printf("Lista Cadastrada No Banco\n");
    close(arq);
}
```

O programa inicializa exibindo o menu inicial:

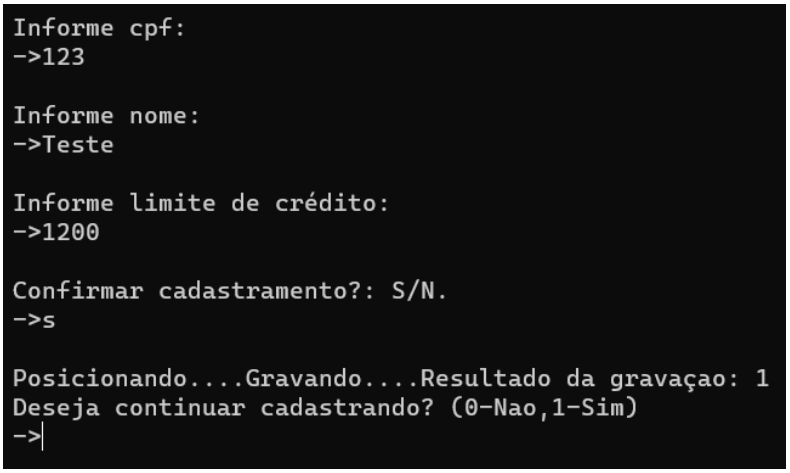
```
Escolha opçao
1- Cliente
2- Produto
3- Venda
4- Realizar logout
->
```

Caso 1 - Menu Cliente:



O sistema roda a função int menuCliente(); Que retorna 1 ou 2;
1-Para voltar ao menu inicial, 2- Para fazer logout do sistema;

Caso 1 - Cadastrar Cliente:



O sistema roda o procedimento void leCliente();
que lê as informações do cliente e verifica se o Cpf informado já existe no banco de dados (caso sim, é pedido para informar novamente o cpf), o status é automaticamente dado como QUITADO, e seu valor de crédito atual é dado como valor de limite de crédito;

```c
do  //VERIFICA SE JÁ EXISTE CLIENTE CADASTRADO
{
    printf("Informe cpf:\n");
    printf("->");
    scanf("%llu",&cliente->cpf);
    printf("\n");
    fflush(stdin);

    pessoa = (dCliente*) retrieve(open(TB_CLIENTE,sizeof(dCliente)),alocaInt(cliente->cpf),comparaCliente);
    if(pessoa != NULL)
    {
        printf("Cpf já existente no banco de dados\n");
        imprimeCliente(pessoa);
        printf("Informe novamente os dados, ou delete cadastro repetido\n");
        exist = 1;
    }
    else
    {
        exist = 0;
    }
    fflush(stdin);

}
while(exist == 1);  //VERIFICA SE JÁ EXISTE CLIENTE CADASTRADO
```

```c
printf("Informe nome:\n");
printf("->");
fgets(cliente->nome,sizeof(cliente->nome),stdin);
cliente->nome[strcspn(cliente->nome,"\n")] = '\0';
printf("\n");
fflush(stdin);

printf("Informe limite de crédito:\n");
printf("->");
scanf("%f",&cliente->limiteCredito);
printf("\n");
fflush(stdin);

printf("Confirmar cadastramento?: S/N.\n");
printf("->");
scanf("%c",&conf);
printf("\n");
fflush(stdin);

if(conf == 'S' || conf == 's'){
    cliente->status = QUITADO;
    cliente->creditoAtual = cliente->limiteCredito;
    createe(open(TB_CLIENTE,sizeof(dCliente)),cliente);
}else{
    printf("Cliente não cadastrado\n");
}
```

Caso já exista o CPF informado,é impresso o cliente que o possui e então é exibido uma mensagem de erro e reiniciado o processo de cadastro:

```
Cpf já existente no banco de dados
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
STATUS CLIENTE: QUITADO

Informe novamente os dados, ou delete cadastro repetido
Informe cpf:
->
```

Caso 2 - Vizualizar clientes

```
1 - Vizualizar todos os clientes cadastrados
2 - Vizualizar cliente específico
3 - Voltar ao menu
4 - Realizar logout
->
```

3 - Faz a função menuCliente retornar 1 | 4 - Faz a função menuCliente retornar 2;

```c
do //MENU VIZUALIZAR
{
    printf("1 - Visualizar todos os clientes cadastrados\n");
    printf("2 - Visualizar cliente específico\n");
    printf("3 - Voltar ao menu\n");
    printf("4 - Realizar logout\n");
    printf("->");
    scanf("%d",&op);
    printf("\n");
    fflush(stdin);
    if(op < 1 || op > 4) printf("Opção Inválida\t|\tInforme novamente\n");
}
while(op < 1 || op > 4);
```

Caso 1 - Visualizar todos os clientes cadastrados

É impresso todos os clientes cadastrados

```
Lista de Clientes atuais:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
STATUS CLIENTE: QUITADO

CPF: 456
NOME: Teste2
LIMITE DE CRÉDITO: 2100,00
STATUS CLIENTE: QUITADO
```

```c
if(op == 1) //IMPRIMIR TODOS OS CLIENTES
{
    pDLista pListaClientes = queryAll(open(TB_CLIENTE,sizeof(dCliente)));
    if(pListaClientes != NULL)
    {
        printf("Lista de Clientes atuais:\n");
        imprimirLista(pListaClientes,imprimeCliente);
        printf("\n");
    }
    else
    {
        printf("Clientes não encontrados no banco de dados\n");
        printf("\n");
    }
}
```

Caso 2 - Visualizar cliente específico

É pedido o método de busca (cpf/nome), e é impresso o cliente caso exista, além da opção de continuar buscando clientes

```
1 - Buscar por cpf
2 - Buscar por nome
->1

Informe cpf para busca:
->123

Cliente encontrado:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
STATUS CLIENTE: QUITADO


Continuar buscando clientes? (0 - Nâo | 1 - Sim)
->0
```

Menu de opção:

```c
else if(op == 2) //VIZUALIZAR CLIENTE ESPECIFICO
{

    do  //CONTINUAR BUSCANDO CLIENTE
    {

        do  //VERIFICA OPÇÃO ESCOLHIDA
        {
            printf("1 - Buscar por cpf\n");
            printf("2 - Buscar por nome\n->");
            scanf("%d",&op);
            fflush(stdin);
            printf("\n");
            if(op < 1 || op > 2) printf("Opção inválida\t|\tInforme novamente\n");
        }
        while(op < 1 || op > 2); //VERIFICA OPÇÃO ESCOLHIDA
```

Busca por cpf:

```c
if(op == 1)
{

    printf("Informe cpf para busca: \n");
    printf("->");
    scanf("%llu",&codigo);
    printf("\n");
    fflush(stdin);

    cliente = retrieve(open(TB_CLIENTE,sizeof(dCliente)),alocaLLU(codigo),comparaCliente);

    if(cliente == NULL)
    {
        printf("Cliente não encontrado\n");
    }
    else
    {
        printf("Cliente encontrado:\n");
        imprimeCliente(cliente);
        printf("\n");
    }

}
```

Busca por nome:

```c
else if(op == 2)
{

    char nomeBuscaCliente[30];

    printf("Informe nome para busca\n->");
    fgets(nomeBuscaCliente,sizeof(nomeBuscaCliente),stdin);
    nomeBuscaCliente[strcspn(nomeBuscaCliente,"\n")] = '\0';
    printf("\n");
    fflush(stdin);

    cliente = (dCliente*) retrieve(open(TB_CLIENTE,sizeof(dCliente)),alocaChar(nomeBuscaCliente),comparaNomeCliente);

    if(cliente == NULL)
    {
        printf("Cliente não encontrado\n");
    }
    else
    {
        printf("Cliente Encontrado:\n");
        imprimeCliente(cliente);
    }
}
```

Caso 3 - Atualizar cliente

```
1 - Atualizar Nome
2 - Atualizar Limite de Crédito
3 - Atualizar Status
4 - Voltar ao Menu
5 - Realizar Logout
->
```

4 - Faz a função menuCliente retornar 1 | 5 - Faz a função menuCliente retornar 2;

Caso 1, 2, 3 - Métodos de atualização

Em si, os casos 1,2,3 possuem a mesma lógica, mudando praticamente só o dado a ser atualizado.

```
Informe cpf do cliente a ser atualizado
->123

Cliente encontrado:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
STATUS CLIENTE: QUITADO

Informe novo nome:
->TesteNovo

Confirmar atualizaçao? (0 - Nao | 1 - Sim)
->1

Cadastro atualizado com sucesso
Nome atualizado
```

```c
printf("Informe novo nome:\n");
printf("->");
fgets(cliente->nome,sizeof(cliente->nome),stdin);
cliente->nome[strcspn(cliente->nome,"\n")] = '\0';
fflush(stdin);
printf("\n");
do
{
    printf("Confirmar atualização? (0 - Não | 1 - Sim)\n");
    printf("->");
    scanf("%d",&op);
    printf("\n");
    fflush(stdin);
    if (op < 0 || op > 1) printf("Opção Inválida\t|\tInforme novamente\n");
}
while(op < 0 || op > 1);
if(op == 1)
{
    arq = open(TB_CLIENTE,sizeof(dCliente));
    update(arq,alocaLLU(codigo),cliente,comparaCliente);
    close(arq);
    printf("Nome atualizado\n");
}
else
{
    printf("Operação cancelada\n");
}
```

Caso 4 - Deletar cliente

```
Informe cpf do cliente a ser deletado:
->123
Cliente encontrado:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
STATUS CLIENTE: QUITADO

Deletar Cliente? (0 - Nao | 1 - Sim)
Obs: Essa informaçao nao poderá ser recuperada e provavelmente deixará falhas no banco de dados
->1

Cadastro deletado com sucesso
```

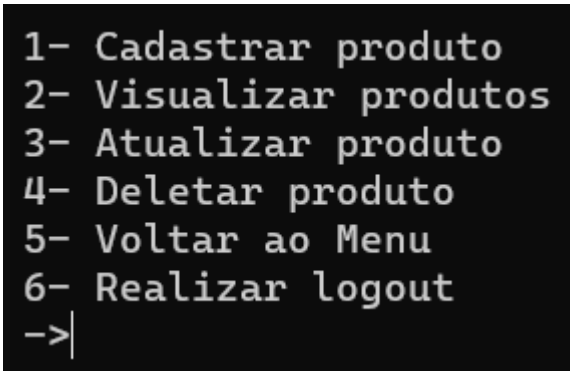
```c
printf("Cliente encontrado:\n");
imprimeCliente(cliente);

do
{
    printf("Deletar Cliente? (0 - Não | 1 - Sim)\n");
    printf("Obs: Essa informação não poderá ser recuperada e provavelmente deixará falhas no banco de dados\n");
    printf("->");
    scanf("%d",&op);
    printf("\n");
    fflush(stdin);
    if (op < 0 || op > 1) printf("Opção Inválida\t|\tInforme novamente\n");
}
while(op < 0 || op > 1);

if(op == 1)
{
    arq = open(TB_CLIENTE,sizeof(dCliente));
    deletee(arq,TB_CLIENTE,alocaLLU(codigo),comparaCliente);
    close(arq);
    printf("Cliente deletado com sucesso\n");
}
else if(op == 0)
{
    printf("Operação cancelada\n");
}
```

Caso 5 - Pagar valores em crédito

```
Informe cpf do cliente a pagar
->123

Cliente encontrado:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
CRÉDITO ATUAL: 1200,00
STATUS CLIENTE: QUITADO

Informe valor em haver a ser pago:
->100

Confirmar atualizaçao do crédito atual? (0 - Nao | 1 - Sim)
->1

Status do cliente atualizado para QUITADO
Cadastro atualizado com sucesso
Crédito atualizado
```

```c
if(op == 1)
{
    cliente->creditoAtual += valor;
    if(cliente->creditoAtual == cliente->limiteCredito){

        printf("Status do cliente atualizado para QUITADO\n");
        cliente->status = QUITADO;

    }else if(cliente->creditoAtual > cliente->limiteCredito){
        cliente->creditoAtual = cliente->limiteCredito;
        printf("Status do cliente atualizado para QUITADO\n");
        cliente->status = QUITADO;

    }else if(cliente->creditoAtual < cliente->limiteCredito){
        printf("Cliente ainda se encontra devendo %.2f\n",cliente->limiteCredito - cliente->creditoAtual);
        cliente->status = DEVENDO;

    }
    arq = open(TB_CLIENTE,sizeof(dCliente));
    update(arq,alocaLLU(codigo),cliente,comparaCliente);
    close(arq);
    printf("Crédito atualizado\n");
}
else
{
    printf("Operação cancelada\n");
}
```

Caso 2 - Menu Produto:

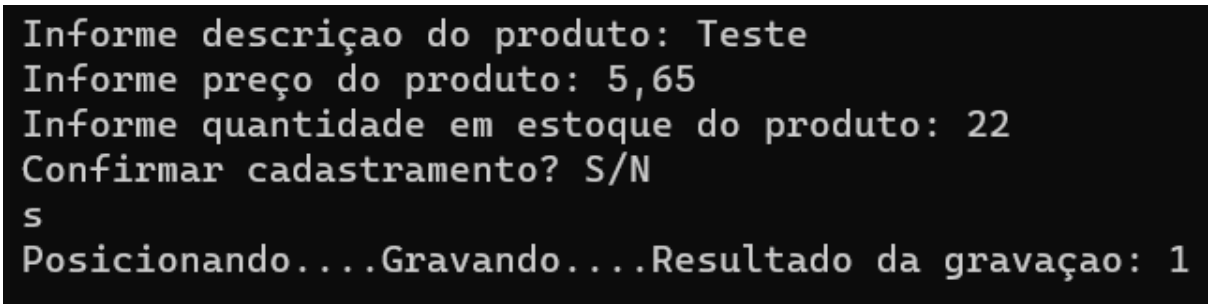


O sistema roda a função int menuCliente(); Que retorna 1 ou 2;
5-Para voltar ao menu inicial, 6- Para fazer logout do sistema;

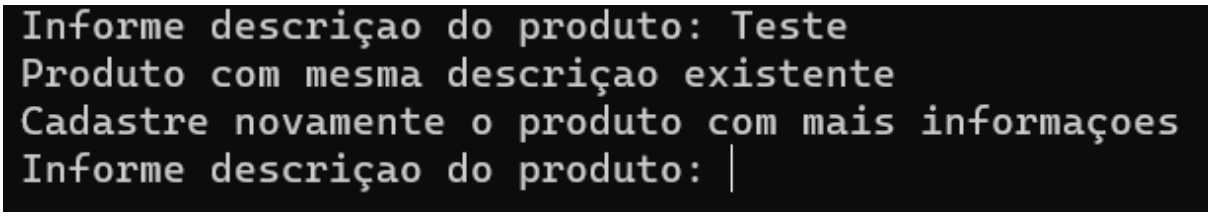
Caso 1 - Cadastrar produto

O sistema roda o procedimento void leProduto();
que lê as informações do produto e verifica se a descrição/nome informado já existe no banco de dados (caso sim, é pedido para informar novamente), o código do produto é gerado automaticamente pela função int geraCodProd(); que salva o último código gerado no arquivo CONT_PROD

```c
int geraCodProd(){
    int aux;

    FILE* arq = fopen(CONT_PROD,"rb+");
    fscanf(arq,"%d",&aux);
    fclose(arq);

    arq = fopen(CONT_PROD,"wb+");
    fprintf(arq, "%d", aux+1);
    fclose(arq);

    return aux;
}
```

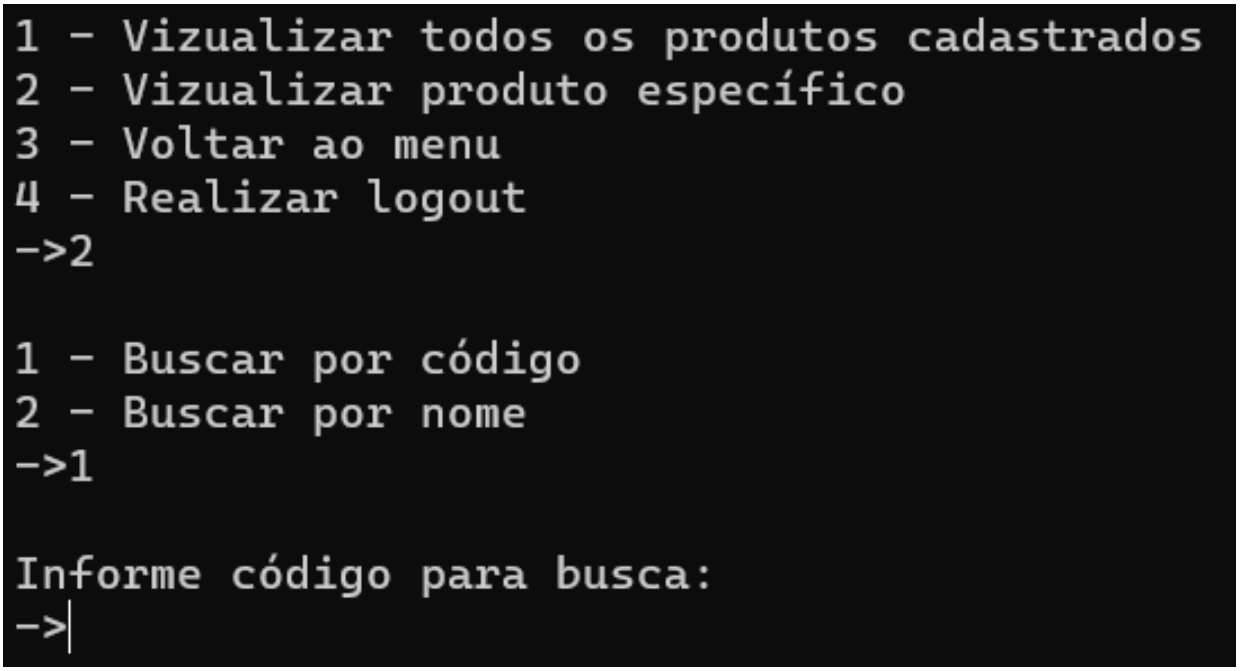


Caso já exista a descrição/nome informado, é reiniciado o processo de cadastro:



Caso 2 - Visualizar produtos

Funciona basicamente igual o visualizar clientes, porém com produtos



Caso 3 - Atualizar produto

Funciona basicamente igual o atualizar clientes, porém tendo como alterar apenas nome, estoque, preço, ou todo o produto de uma vez, tendo a mesma lógica aplicada para cada opção

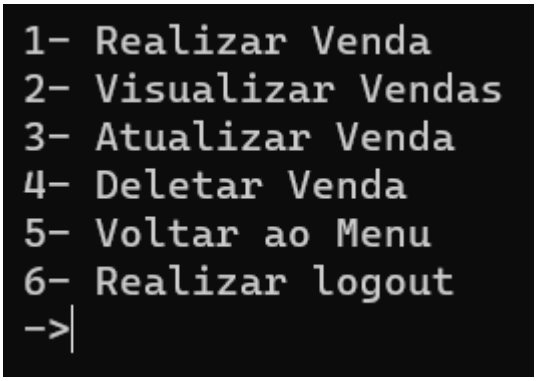
```
1 - Atualizar Nome/Descriçao
2 - Atualizar Preço
3 - Atualizar Estoque
4 - Atualizar Tudo
5 - Voltar ao Menu
6 - Realizar Logout
->1

Informe código do produto a ser atualizado
->0

Produto encontrado
Código do produto: 0
Descriçao do produto: Teste
Preço do produto: 5,65
Quantidade disponível do produto: 22

Informe novo nome/descriçao do produto
->Teste2

Confirmar atualizaçao? (0 - Nao | 1 - Sim)
->1

Cadastro atualizado com sucesso
Nome/Descriçao atualizados
```

Caso 4 - Deletar produto

Segue a mesma lógica do deletar cliente

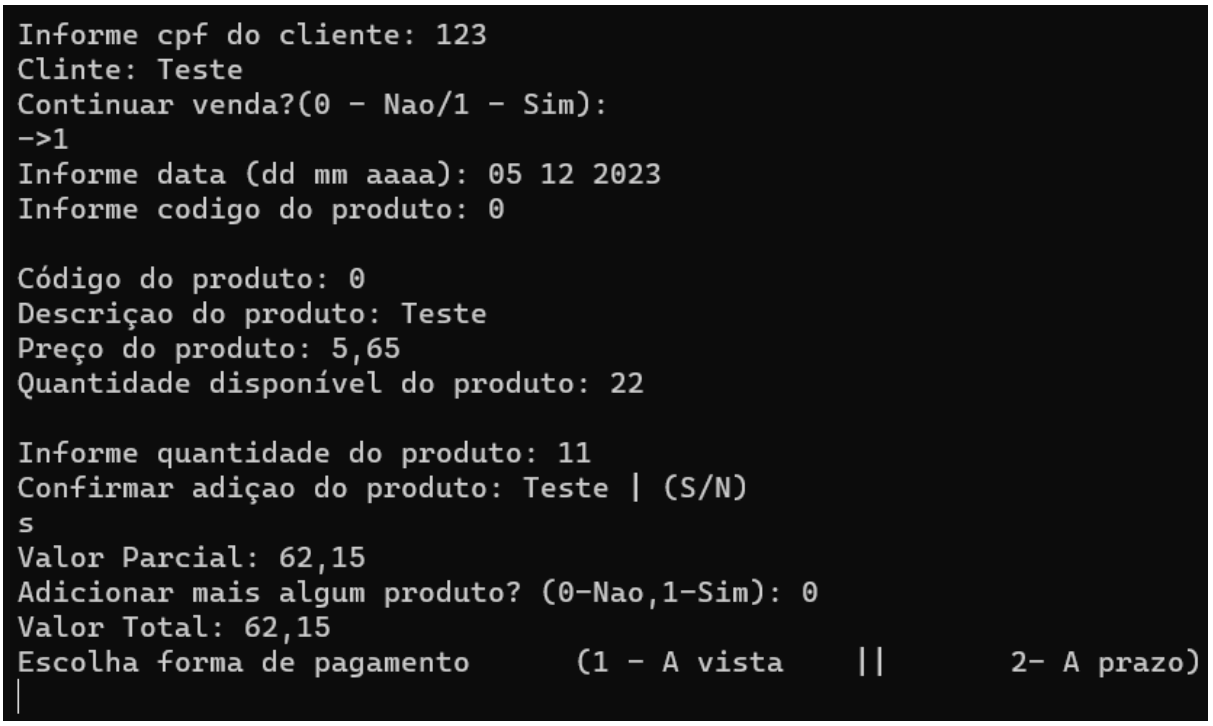
```
Informe código do produto a ser deletado:
->0
Produto encontrado:
Código do produto: 0
Descriçao do produto: Teste
Preço do produto: 5,65
Quantidade disponível do produto: 22

Deletar Produto? (0 - Nao | 1 - Sim)
Obs: Essa informaçao nao poderá ser recuperada e provavelmente deixará falhas no banco de dados
->1
```

Caso 3 - Menu Venda:



O sistema roda a função int menuVenda(); Que retorna 1 ou 2;
5-Para voltar ao menu inicial, 6- Para fazer logout do sistema;

Caso 1 - Realizar Venda



->É informado o cpf
->O sistema busca o cliente e pede confirmação para continuar a venda
->É pedido a data da venda
->O código do produto a ser vendido
->O sistema busca pelo produto e imprime caso exista
->Pede a quantidade
->O sistema calcula o valor da venda até o momento

->Confirma se deve ser adicionado mais algum produto, caso sim, repete o processo, caso não pede para informar a forma de pagamento

->Se a forma de pagamento for à vista, a venda é finalizada com Status:QUITADO e cadastrada no banco de dados

->Se a forma de pagamento for a prazo, o sistema verifica se o cliente possui crédito suficiente para realizar a compra, se houver, o sistema finaliza a venda, atualiza o status do cliente e da venda para DEVENDO, desconta o valor da venda do crédito atual do cliente e grava a venda, e atualiza a quantidade de cada produto da venda chamando a função void atualizaEstoque(pDLista pLista);

```
Limite suficiente
Finalizar Venda? S/N
s
Saldo limite atualizado
Cadastro atualizado com sucesso
Posicionando....Gravando....Resultado da gravaçao: 1
```

```c
printf("Informe codigo do produto: "); //PEDE CODIGO DO PRODUTO PARA BUSCA
scanf("%d",&prodVenda->codProd);
fflush(stdin);

printf("\n");
produto = (dProduto*) retrieve(open(TB_PRODUTOS,sizeof(dProduto)),alocaInt(prodVenda->codProd),comparaCodProduto);
imprimeProdutos(produto);

do  //VERIFICA SE A QUANTIDADE SE TEM EM ESTOQUE A QUANTIDADE SUFICIENTE PARA VENDA
{

    printf("Informe quantidade do produto: ");
    scanf("%d",&prodVenda->quantidade);
    fflush(stdin);

    if(prodVenda->quantidade > produto->quantidadeEstoque)
    {
        printf("Saldo insuficiente em estoque\n");
        printf("Informe nova quantidade\n");
    }
}
while(prodVenda->quantidade > produto->quantidadeEstoque);  //VERIFICA SE A QUANTIDADE SE TEM EM ESTOQUE A QUANTID

printf("Confirmar adição do produto: %s | (S/N)\n",produto->descricao);
scanf("%c",&conf);
fflush(stdin);

if(conf == 's' || conf == 'S')
{
    valorTotal += (produto->preco * prodVenda->quantidade); //CALCULA O VALOR DA VENDA
    incluirInfo(pListaProdutos,prodVenda); //INCLUI O PRODUTO ESCOLHIDO NA LISTA DE PRODUTOS_VENDA
}
```

```c
printf("Finalizar Venda? S/N\n");
scanf("%c",&conf);
fflush(stdin);
if(conf == 'S' || conf == 's')
{
    if(forma == 2)
    {
        printf("Limite de crédito atualizado\n");
        cliente->creditoAtual -= valorTotal; //DESCONTA O VALOR DA VENDA DO CRÉDITO DO CLIENTE
        cliente->status = DEVENDO; //ATUALIZA O STATUS DO CLIENTE PARA DEVENDO
        update(open(TB_CLIENTE,sizeof(dCliente)),alocaLLU(cliente->cpf),cliente,comparaCliente);
    }
    venda->PrecoVenda = valorTotal;
    pDFile arq = open(TB_VENDA,sizeof(dVenda));
    createe(arq,venda); //CADASTRA A VENDA NO BANCO DE DADOS
    close(arq);
    persistAll(open(TB_PRODUTO_VENDA,sizeof(dProdVenda)),pListaProdutos); //CADASTRA TODOS OS PRO
    printf("Venda realizada com sucesso\n");
    atualizaEstoque(pListaProdutos); //ATUALIZA O ESTOQUE DOS PRODUTOS VENDIDOS
    imprimeVenda(venda);
}
else
{
    printf("Venda Cancelada\n");
    retornaCodVenda();
}

printf("Deseja continuar cadastrando? (0-Não,1-Sim)\n");
scanf("%d",&op);
fflush(stdin);

valorTotal = 0;
```

Caso 2 - Vizualizar venda

```
1- Vizualizar todas as vendas
2- Vizualizar venda específica
->1

Vendas cadastradas no banco de dados:
Número da venda: 0
Cliente: Teste          Data da venda: 5/12/2023
Produtos:
11 un - Teste
Valor total da venda: 62,15

Pressione qualquer tecla para continuar. . .
```

Também possui um menu de visualizar todas as vendas ou apenas vendas específicas

```
1- Buscar por código da venda
2- Filtrar vendas
->2

Filtrar por:
1- Cpf Cliente
2- Código produto
3- Valor maior que (>)
4- Valor menor que (<)
->1

Informe CPF do cliente:
->123

Cliente:
CPF: 123
NOME: Teste
LIMITE DE CRÉDITO: 1200,00
CRÉDITO ATUAL: 1200,00
STATUS CLIENTE: QUITADO


Vendas:
Número da venda: 0
Cliente: Teste          Data da venda: 5/12/2023
```

A busca por código da venda possui a mesma lógica da busca por CPF ou buscar por código do produto, já a filtrar por utiliza de um novo procedimento void openFiltro(void* dados ,int tam);

```
printf("Informe valor para filtrar:\n");
printf("->");
scanf("%f",&valor);
printf("\n");
fflush(stdin);

openFiltro(alocaFloat(valor),sizeof(float));
pListaVenda = queryBy(open(TB_VENDA,sizeof(dVenda)),filtraMaiorQue);

if(pListaVenda == NULL)
{
    printf("Não há vendas com valor total maior que %.2f (menuVenda/ switch 3/ switch 3)\n",valor);
}
else
{
    printf("Vendas: \n");
    imprimirLista(pListaVenda,imprimeVenda);
    printf("\n");
}
break;
```

Esse procedimento abre o arquivo FILTRO, definido no cabeçalho, grava um dado void de tamanho int no arquivo

```c
void openFiltro(void* dados,int tam)
{
    FILE* arq = fopen(FILTROS,"wb+");
    if(arq == NULL)
    {
        printf("Erro ao abrir o arquivo (openFiltro)\n");
        return;
    }
    fwrite(dados,tam,1,arq);
    fclose(arq);
}
```

Após isso a função queryBy, cria uma lista com todas as vendas que se encaixam nesse “filtro”, pois a função int filtraMaiorQue(void* dados); abre o arquivo FILTROS, e compara o item em FILTROS com a chave passada pela função queryBy

```c
int filtraMaiorQue(void* dados)
{
    pDFile arq = open(FILTROS,sizeof(float));
    if(arq->arquivo == NULL)
    {
        printf("Erro ao abrir o arquivo (maiorQue)\n");
        return;
    }
    rewind(arq->arquivo);
    float* valorComp;
    fread(valorComp,sizeof(float),1,arq->arquivo);
    close(arq);
    dVenda* venda = (dVenda*) dados;
    if(venda->PrecoVenda >= *valorComp)
    {
        return 0;
    }
    return 1;
}
```

Assim podendo utilizar a função queryBy para comparar predicados diferentes dependendo do que for ser filtrado. Todos os outros filtros possuem a mesma lógica, porém cada um adaptado para ler um tipo de dados diferente no arquivo FILTROS.

Em todos os casos a função void imprimirVenda(dVenda*); é chamada

```
void imprimeVenda(dVenda* venda){

    dCliente* cliente = retrieve(open(TB_CLIENTE,sizeof(dCliente)),alocaInt(venda->cpf),comparaCliente);
    dProdVenda* prodVenda;
    dProduto* produto;

    printf("Número da venda: %d\n",venda->numero);
    printf("Cliente: %s\t\t",cliente->nome);
    printf("Data da venda: %d/%d/%d\n",venda->Data.dia,venda->Data.mes,venda->Data.ano);
    printf("Produtos: \n");
```

Nessa primeira parte, alguns ponteiros importantes são declarados, e os dados “fáceis” de extrair da venda são impressos. A seguir:

```
openFiltro(alocaInt(venda->numero),sizeof(int));

pDLista pListaProdVenda = queryBy(open(TB_PRODUTO_VENDA,sizeof(dProdVenda)),comparaCodVendaProd);
pDLista pListaProdutos = criarLista();
pNoh pAux = pListaProdVenda->primeiro;
void* dados;
while(pAux != NULL){

    prodVenda = (dProdVenda*) pAux->info;
    dados = retrieve(open(TB_PRODUTOS,sizeof(dProduto)),alocaInt(prodVenda->codProd),comparaCodProduto);
    incluirInfo(pListaProdutos,dados);
    pAux = pAux->prox;
}
pAux = pListaProdVenda->primeiro;
pNoh pProd = pListaProdutos->primeiro;

while(pProd != NULL){
    prodVenda = (dProdVenda*) pAux->info;
    produto = (dProduto*) pProd->info;
    printf("%d un - %s\n",prodVenda->quantidade, produto->descricao);
    pAux = pAux->prox;
    pProd = pProd->prox;
}
printf("Valor total da venda: %.2f\n",venda->PrecoVenda);
```

A função openFiltro escreve o número/código da venda no arquivo FILTROS.

Após isso a função queryBy gera uma lista com todas as estruturas que possuem o mesmo número da venda (escrito no arquivo de FILTROS anteriormente) dentro do arquivo relacional TB_PRODUTO_VENDA.

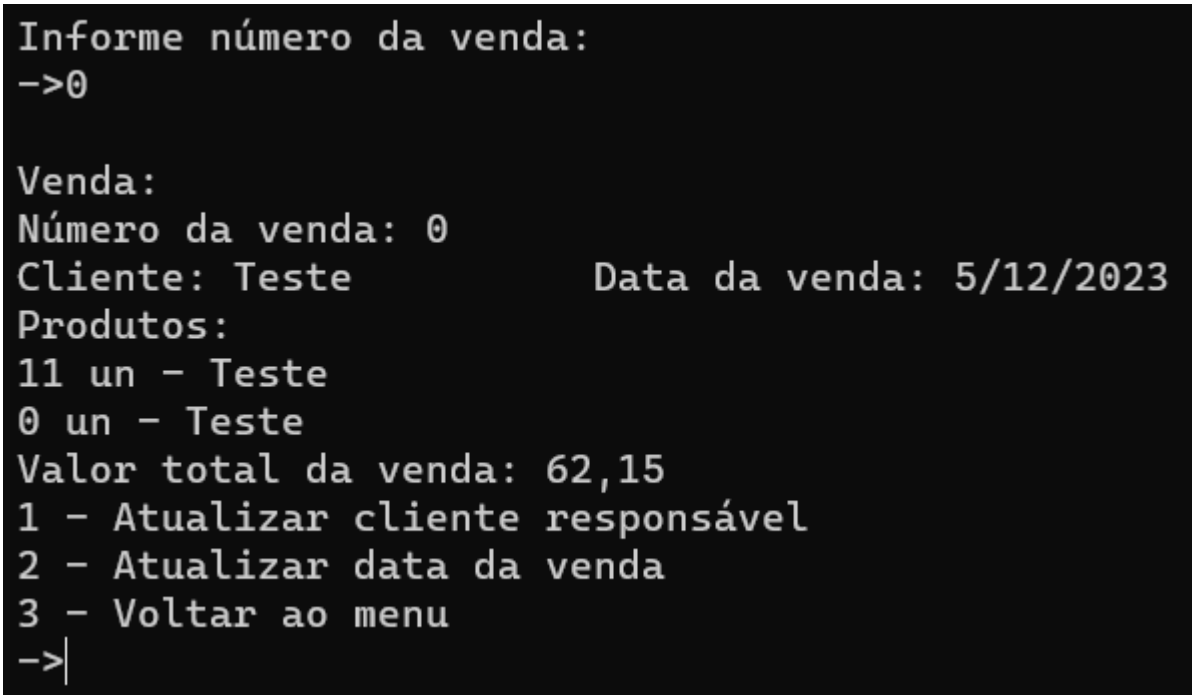
pListaProdVenda a lista gerada pela queryBy, e a pListaProdutos é criada logo abaixo.

O laço de repetição é iniciado no intuito de percorrer a pListaProdVenda, e incluir todos os produtos na pListaProdutos.

O segundo loop é iniciado para imprimir a quantidade do produto “x” vendida e o nome do mesmo (aqui poderiam ser impressos até mesmo todos os dados do produto caso queira).

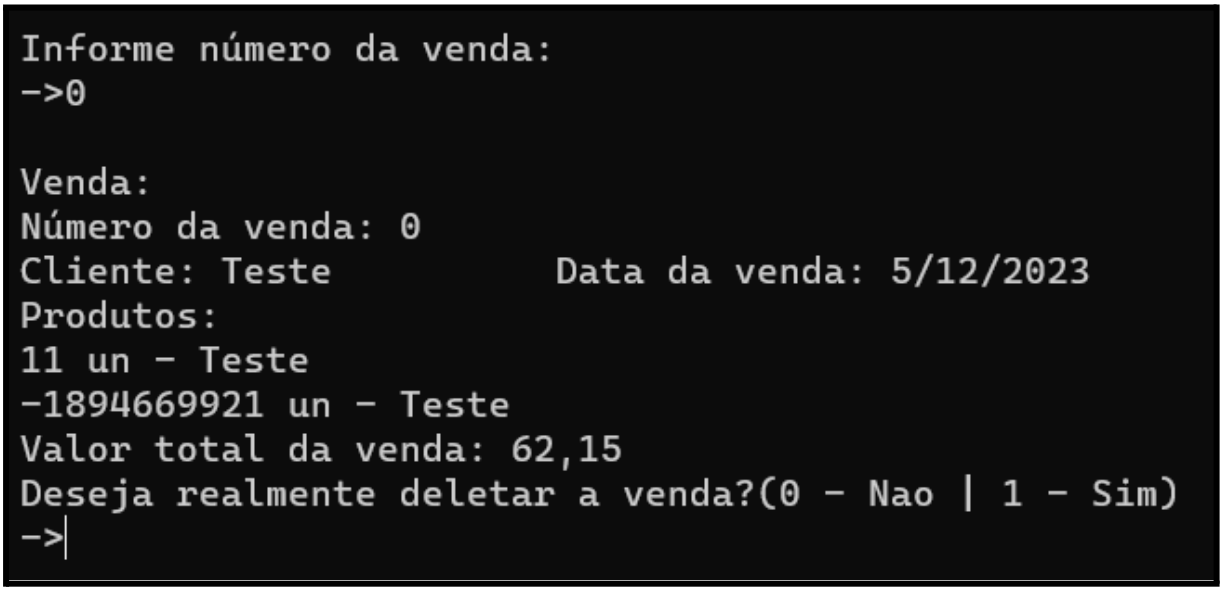
E por último é impresso o valor final da venda.

Caso 3 - Atualizar venda



Só é possível atualizar o cliente para qual a venda foi feita e sua data de emissão, o status da venda é atualizado pelo sistema em outros procedimentos automáticos (não implementados ainda)

Caso 4 - Deletar venda



Possui a mesma lógica do deletar produto e deletar cliente