

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
IFSP - Guarulhos

Lista de contatos

Lista de contatos de clientes da empresa ACME

Christopher Willians - GU3054047
Gabriel Vitor - GU3054446
Guarulhos
2024

Christopher Willians - GU3054047
Gabriel Vitor - GU3054446

Lista de contatos

Lista de contatos de clientes da empresa ACME

Trabalho de Conclusão da disciplina Estrutura de Dados 1 do curso Tecnólogo em
Análise e Desenvolvimento de Sistemas do IFSP como requisito parcial para a
aprovação da disciplina.

Professor: Antonio Angelo de Souza Tartaglia
Guarulhos
2024

Dedicatória

Dedicamos este trabalho aos nossos colegas de turma e professores, que sempre nos incentivaram a superar desafios e alcançar novos horizontes.

Sumário

1	INTRODUÇÃO	5
2	DESCRIÇÃO DO PROJETO	5
3	VISÃO GERAL DO PROJETO	6
4	DECISÕES DE IMPLEMENTAÇÃO	14
4.1	Função de abertura do arquivo binário (abrirArquivo)	14
4.2	Função de coleta de dados (coletadados)	15
4.3	Função para verificação do identificador (confereDuplicidade)	17
4.4	Função para exibição de todos os clientes (listarTodos)	18
4.5	Função de consultar pelo nome (consultaNome)	19
4.6	Função de formatação (converteNome)	21
4.7	Função de exibição (relatorio)	22
4.8	Função de edição de dados do cliente (editarCliente)	22
4.9	Função para salvar o arquivo com as novas informações (salvarArquivo)	24
5	DIAGRAMA	25

1 INTRODUÇÃO

Este trabalho é desenvolvido como requisito da disciplina Estrutura de Dados 1, com o objetivo de consolidar os conceitos aprendidos ao longo do semestre. A proposta consiste na criação de um programa para gerenciar uma lista de clientes da empresa fictícia ACME, utilizando a estrutura de lista dinâmica implementada na linguagem C.

A lista dinâmica foi escolhida por sua flexibilidade, permitindo a alocação de memória durante a execução do programa e facilitando operações como inserção, remoção e busca de clientes. Este projeto não apenas reforça a aplicação prática das estruturas de dados, mas também explora boas práticas de programação, como modularidade e organização do código.

O programa irá oferecer funcionalidades básicas, como:

- Resgatar informações de sessões anteriores salvas em um arquivo binário;
- Adicionar novos clientes à lista;
- Remover clientes existentes;
- Pesquisar informações de um cliente;
- Exibir todos os clientes cadastrados;
- Salvar as informações em arquivo binário.

Com isso, busca-se demonstrar o domínio sobre o uso de ponteiros, alocação dinâmica de memória e manipulação de dados em estruturas não lineares, essenciais para a construção de sistemas eficientes e adaptáveis.

2 DESCRIÇÃO DO PROJETO

Este projeto é um sistema de gerenciamento de contatos, que permite ao usuário adicionar, visualizar, buscar, editar e excluir informações para cada um. Temos um menu de opções e uma lógica de salvamento dos dados em um arquivo binário no fechamento do arquivo, para que possamos restaurá-los na próxima vez que alguém for utilizar.

3 VISÃO GERAL DO PROJETO

O projeto se dá início no main.c com inicialização das variáveis, criação das estruturas e da abertura do arquivo binário.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "acme.h"

int main()
{
    Lista *li = NULL;
    int x, identificador, quantidade, escolha = 0;
    char nome[80];

    CLIENTE al_consulta, al;

    if ((li = criaLista()) == NULL)
    {
        abortaPrograma();
    }

    FILE *arquivo = NULL;
    arquivo = abriArquivo(li);
```

Após isso temos a apresentação dos desenvolvedores:

```
printf("\n\n");
```

```

printf("#####\n");
printf("Bem vindo ao sistema de gerenciamento de contatos\n\n");
printf("Desenvolvido por: \n");
printf("Christopher Willians - Gu3054047\n");
printf("Gabriel Vitor - Gu3054446\n");

printf("#####\n");

```

Depois disso, temos a parte principal que é o menu de escolhas do usuário, onde ele escolherá a função que deseja utilizar, esse menu fica dentro de um while que manterá o loop enquanto a opção escolhida for diferente de 7 (função de saída do programa) onde ao término de cada função limpará o terminal e retornará ao menu de escolha. Dentro do while temos um switch case que capta a escolha do usuário e chama a função correspondente, sendo elas:

1. Função para inserir um novo contato (sendo obrigatório passar a quantidade de clientes a ser inseridos);
2. Função para exibir todos os contatos já salvos na lista;
3. Função para buscar um contato específico pelo o seu identificador;
4. Função para buscar contatos pelo nome ou trecho dele (trazendo todos os clientes com nomes correspondentes);
5. Função para editar informações de algum contato existente (sendo obrigatório passar o valor do identificador do cliente a ser editado);
6. Função para excluir um contato específico com base em um identificador passado;
7. Encerramento do programa.

```

while (escolha != 7)
{
    printf("\nEscolha uma das opcoes:\n\n");
    printf("1 - Incluir um novo contato\n");
    printf("2 - Relatorio de todos os contatos\n");
    printf("3 - Buscar contato pelo identificador\n");
    printf("4 - Buscar contatos pelo nome\n");
    printf("5 - Editar as informacoes de um contato\n");
}

```

```

printf("6 - Excluir um contato com base no identificador\n");
printf("7 - Sair\n\n");
printf("Digite a sua escolha: ");
scanf("%d", &escolha);
getchar();

switch (escolha)
{

case 1:
    printf("\nDigite a quantidade de contatos que deseja inserir:
");

    scanf(" %d", &quantidade);
    getchar();

    for (int i = 0; i < quantidade; i++)
    {
        printf("\n");
        al = coletadados(li, 0);
        x = insereOrdenado(li, al);
        if (x)
        {
            printf("\nAluno %d inserido ordenado com sucesso!",
x);

            printf("\n\n");
            system("pause");
            system("cls");
        }
        else
        {
            printf("\nnao foi possivel inserir ordenado");
            printf("\n\n");
            system("pause");
            system("cls");
        }
    }
    break;

case 2:
    listarTodos(li);

```



```

        printf("\n\n");
        system("pause");
        system("cls");
        break;

case 3:
    printf("\n\n");

    printf("Digite a identificador do aluno que deseja buscar: ");
    scanf("%d", &identificador);
    getchar();

    x = consultaIdentificador(li, identificador, &al_consulta);
    if (x)
    {
        printf("\n\n");
        relatorio(&al_consulta);
        printf("\n\n");
        system("pause");
        system("cls");
    }
    else
    {
        printf("\nidentificador %d nao existe.", 110);
        printf("\n\n");
        system("pause");
        system("cls");
    }
    break;

case 4:
    printf("\n\n");

    printf("Digite o nome do aluno que deseja buscar: ");
    fgets(nome, sizeof(nome), stdin);

    if (strlen(nome) > 0 && nome[strlen(nome) - 1] == '\n')
    {
        nome[strlen(nome) - 1] = '\0';
    }

```

```

        consultaNome(li, nome);

        printf("\n\n");
        system("pause");
        system("cls");
        break;

    case 5:
        printf("\nDigite a identificador do aluno que deseja editar:
");

        scanf("%d", &identificador);
        getchar();

        editarCliente(li, identificador);
        printf("\n\n");
        system("pause");
        system("cls");
        break;

    case 6:
        printf("\nDigite a identificador do aluno que deseja excluir:
");

        scanf("%d", &identificador);
        getchar();

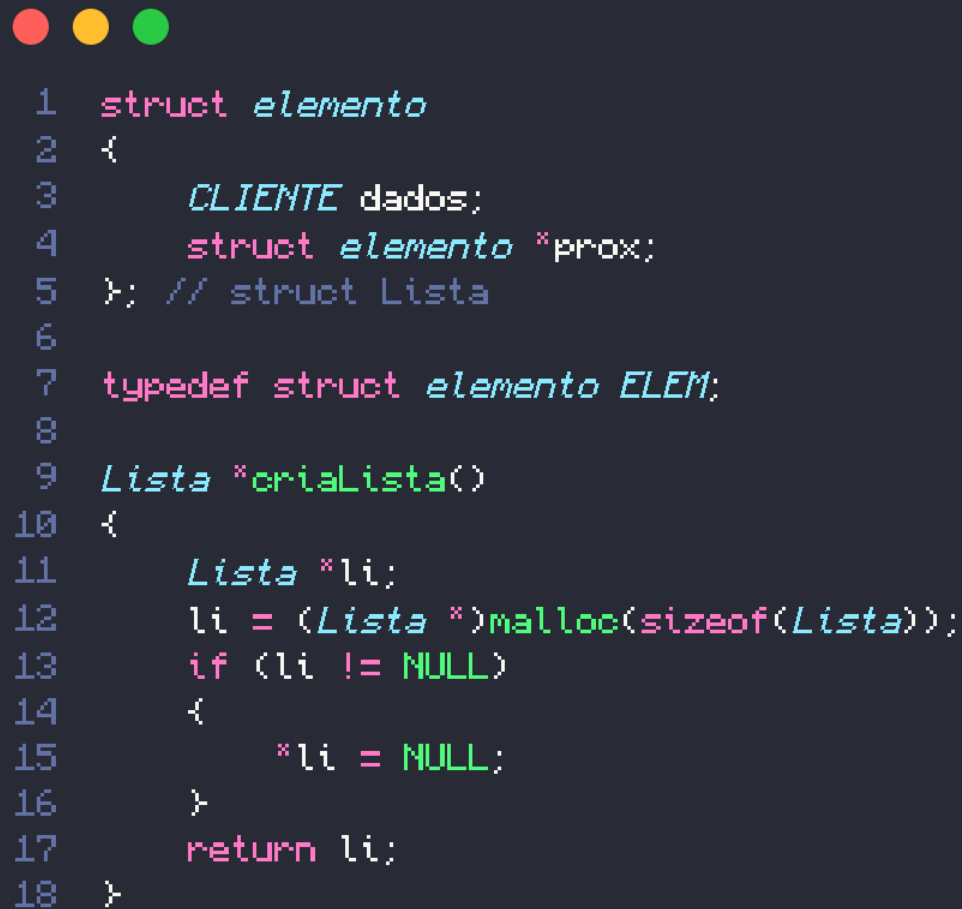
        x = removeOrdenado(li, identificador);
        if (x)
        {
            printf("\nAluno %d removido ordenado com sucesso!", x);
            printf("\n\n");
            system("pause");
            system("cls");
        }
        else
        {
            printf("\nnao foi possivel remover ordenado");
            printf("\n\n");
            system("pause");
            system("cls");
        }
    }
}

```

```
    }  
    break;  
  
    case 7:  
        printf("\n\n");  
        printf("Fim do programa");  
        printf("\n\n");  
        break;  
  
    default:  
        printf("\n\n");  
        printf("Opcao invalida");  
        printf("\n\n");  
        system("pause");  
        system("cls");  
        break;  
    }  
}
```

Caso seja pedido o encerramento do programa, será chamada função para salvar o arquivo (gravar todas as informações no arquivo binário) e a função para apagar a lista (liberar a memória do programa).

No Arquivo acme.c temos a criação da estrutura da lista dinâmica e a criação da lista (com a função criaLista que retorna um ponteiro do tipo Lista), ambas são ativadas no momento em que o programa é inicializado.



```

1  struct elemento
2  {
3      CLIENTE dados;
4      struct elemento *prox;
5  }; // struct Lista
6
7  typedef struct elemento ELEM;
8
9  Lista *criaLista()
10 {
11     Lista *li;
12     li = (Lista *)malloc(sizeof(Lista));
13     if (li != NULL)
14     {
15         *li = NULL;
16     }
17     return li;
18 }

```

Já no arquivo acme.h, temos as declarações de todas as variáveis, criação da struct CLIENTE e a declaração da estrutura da lista dinâmica (aquela criada no acme.c).

```

typedef struct cliente{
    int identificador;
    char nome[80];
    char empresa[80];
    char departamento[80];
    char telefone[15];
    char celular[15];
}

```

```
    char email[80];
} CLIENTE;

typedef struct elemento* Lista;

Lista *criaLista();

void abortaPrograma();

void apagaLista(Lista *li);

FILE* abreArquivo(Lista *li);

int listaCheia(Lista *li);

int listaVazia(Lista *li);

int insereOrdenado(Lista *li, CLIENTE al);

int removeOrdenado(Lista *li, int id);

int consultaIdentificador(Lista *li, int id, CLIENTE *al);

int confereDuplicidade(Lista *li, int id);

void consultaNome(Lista *li, char nome[]);

void converteNome(char nome[]);

void relatorio(CLIENTE *al);

struct cliente coletadados(Lista *li, int id);

void listarTodos(Lista *li);

void editarCliente(Lista *li, int id);

void salvarArquivo(Lista *li);
```

4 DECISÕES DE IMPLEMENTAÇÃO

4.1 Função de abertura do arquivo binário (abrirArquivo)

Essa função é responsável pela abertura do arquivo binário, ela recebe o endereço de memória da lista como argumento, após isso ela cria um ponteiro FILE e abre o arquivo em modo de leitura e verifica se o arquivo existe, caso não exista ele comunica o usuário que o arquivo é inexistente e o programa será aberto sem dados pré existentes (retornando o ponteiro FILE), se a resposta for positiva ela cria uma estrutura do tipo CLIENTE e percorre o arquivo com o while copiando cada estrutura existente para a estrutura criada (usando a função fread, passando como argumento a struct que receberá a estrutura do arquivo, o tamanho padrão dela, quantas struct serão copiadas e o arquivo que fornecerá os dados) e para cada estrutura chama a função insereOrdenado (passando a lista e estrutura copiada como argumento) para inserir os clientes na lista, ao término do loop ela retorna o ponteiro FILE.

```
1  FILE *abrirArquivo(Lista *li)
2  {
3      FILE *arquivo = fopen("clientes.bin", "rb");
4
5      if (arquivo == NULL)
6      {
7          printf("Arquivo inexistente, abrindo sem dados\n");
8
9          return arquivo;
10     }
11
12     CLIENTE cliente;
13     while (fread(&cliente, sizeof(CLIENTE), 1, arquivo) == 1)
14     {
15         insereOrdenado(li, cliente);
16     }
17
18     return arquivo;
19 }
```

4.2 Função de coleta de dados (coletaDados)

Essa função foi modificada (já que a sua base era do código fornecido em aula) para ser reutilizada na função de edição de dados, ela recebe o valor de um identificador e o endereço de memória da lista para poder criar uma estrutura para armazenar as informações que serão passadas e depois retorna essa estrutura.

```
struct cliente coletaDados(Lista *li, int id)
{

    struct cliente all;
    int verificador = 0;

    if (id != 0)
    {
        all.identificador = id;
        verificador = 1;
    }
    while (verificador == 0)
    {

        printf("\nDigite o identificador (id) do cliente:");
        scanf("%d", &all.identificador);
        getchar();

        verificador = confereDupllicidade(li, all.identificador);
        if (verificador == 0)
        {
            printf("\n\nIdentificador ja existente, digite outro
identificador.\n\n");
        }
    }

    printf("\nDigite o nome do cliente:");
    fgets(all.nome, sizeof(all.nome), stdin);
    all.nome[strcspn(all.nome, "\n")] = '\0';

    printf("\nDigite a empresa do cliente:");
    fgets(all.empresa, sizeof(all.empresa), stdin);
    all.empresa[strcspn(all.empresa, "\n")] = '\0';
```

```

printf("\nDigite o departamento do cliente:");
fgets(all.departamento, sizeof(all.departamento), stdin);
all.departamento[strcspn(all.departamento, "\n")] = '\0';

printf("\nDigite o telefone do cliente:");
fgets(all.telefone, sizeof(all.telefone), stdin);
all.telefone[strcspn(all.telefone, "\n")] = '\0';

printf("\nDigite o celular do cliente:");
fgets(all.celular, sizeof(all.celular), stdin);
all.celular[strcspn(all.celular, "\n")] = '\0';

printf("\nDigite o email do cliente:");
fgets(all.email, sizeof(all.email), stdin);
all.email[strcspn(all.email, "\n")] = '\0';

return all;
}

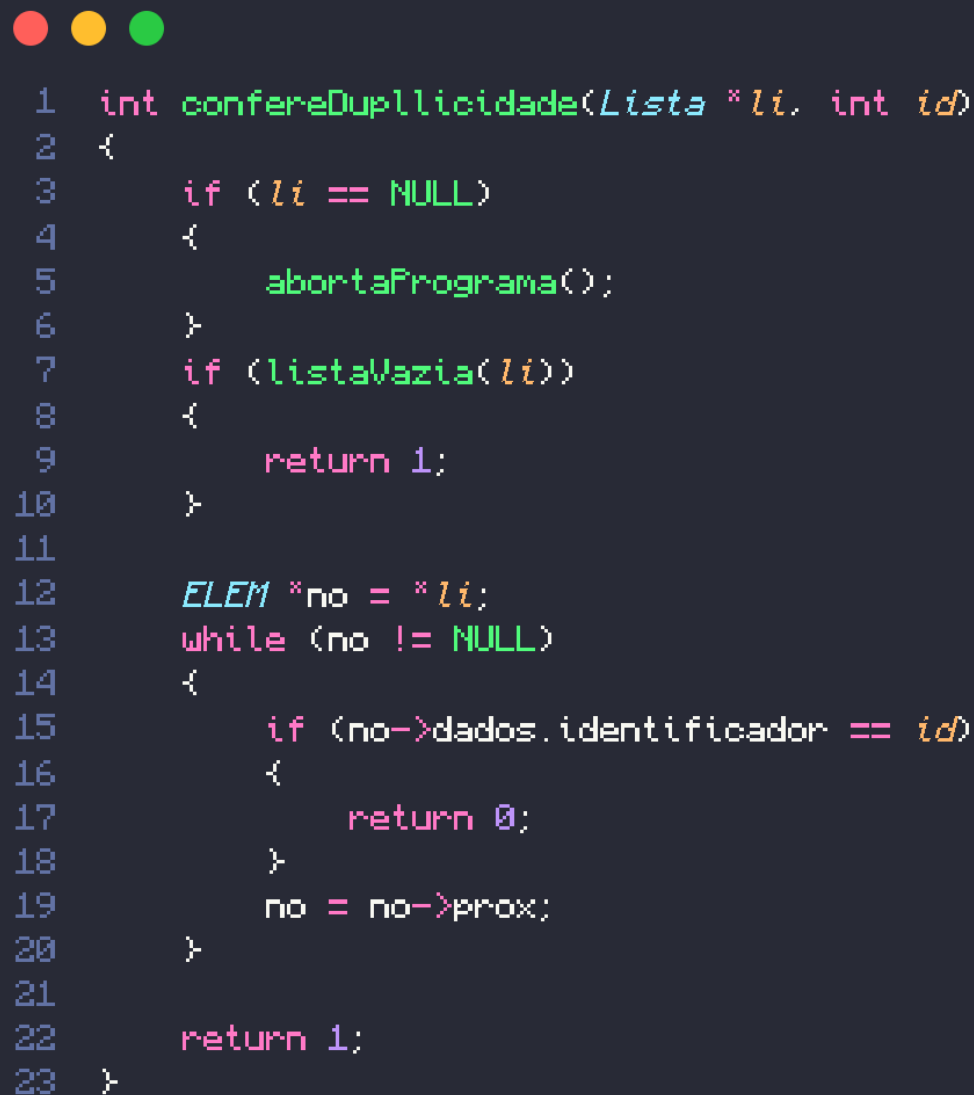
```

A função de coleta dados pega as informações para inserção de um novo cliente e faz uma verificação se o identificador inserido existe na base de dados (passando como argumento a lista e o valor do identificador do novo contato para a função confereDuplicidade que retorna se há ou não o valor na base de dados), ou seja, se já tem algum cliente com o mesmo identificador, caso haja, ele retornará uma mensagem avisando o usuário do programa para digitar outro identificador e só irá passar para o próximo campo de coleta de informações quando for passado um valor inexistente na base de dados (lista).

Já no contexto da função de edição de dados, ela verifica se o identificador passado é diferente de 0 (pois nesse caso for preciso fazer a edição de dados de uma estrutura já existente, logo ela terá um identificador != 0), caso seja positivo a verificação o identificador terá o mesmo valor que o identificador da estrutura que está sendo editada, além disso o código irá pular o trecho de coleta do identificador.

4.3 Função para verificação do identificador (confereDuplicidade)

Essa função é utilizada na função de coleta de dados, ela recebe o endereço de memória da lista e um valor de identificador, verifica se a lista alocada corretamente e se ela não está vazia, faz a inicialização de um ponteiro para percorrer a lista dentro do while, dentro do loop ela passa por cada estrutura e verifica se há algum cliente com o mesmo identificador do que foi passado como argumento, se houver ela retorna 0 (falso), caso contrário ela retorna 1 (verdadeiro).



```

1  int confereDuplicidade(Lista *li, int id)
2  {
3      if (li == NULL)
4      {
5          abortaPrograma();
6      }
7      if (listaVazia(li))
8      {
9          return 1;
10     }
11
12     ELEM *no = *li;
13     while (no != NULL)
14     {
15         if (no->dados.identificador == id)
16         {
17             return 0;
18         }
19         no = no->prox;
20     }
21
22     return 1;
23 }

```

4.4 Função para exibição de todos os clientes (listarTodos)

Essa função recebe o endereço de memória da lista, verifica se a lista alocada corretamente e se ela não está vazia, faz a inicialização de um ponteiro para percorrer a lista dentro do while, dentro do loop ela chama a função de exibição de informações

(função relatório) e passa a próxima estrutura seguindo assim até chegar ao fim da lista.

```
void listarTodos(Lista *li)
{
    if (li == NULL)
    {
        abortaPrograma();
    }
    if (listaVazia(li))
    {
        printf("\n\nLista vazia\n\n");
        return;
    }

    ELEM *no = *li;
    int tamanho = 0;

    while (no != NULL)
    {
        tamanho++;

        printf("\n\n\nCliente %d", tamanho);

        relatorio(&no->dados);

        no = no->prox;
    }

    printf("\n\nTotal de clientes encontrados: %d\n\n", tamanho);
}
```

4.5 Função de consultar pelo nome (consultaNome)

Como solicitado criamos uma função que buscasse na lista de contatos os clientes com um determinado nome ou um trecho de um nome.

Para isso decidimos dividir a função em 3 partes, sendo elas a função de formatação de nomes, a função de exibição e a função de busca. Essa decisão foi tomada visando a modularização do código, aumento da velocidade de desenvolvimento, otimização do programa e na reutilização de funções ao longo do código, uma vez que, a formatação de palavras e exibição de informações será constantemente necessária em diversas funções.

```
void consultaNome(Lista *li, char nome[])
{
    if (li == NULL)
    {
        abortaPrograma();
    }
    if (listaVazia(li))
    {
        return;
    }
    if (nome == NULL)
    {
        printf("\n\nNome invalido\n\n");
        return;
    }

    converteNome(nome);

    ELEM *no = *li;
    int tamanho = 0;
    char nomePesquisa[80];

    while (no != NULL)
    {
        strcpy(nomePesquisa, no->dados.nome);

        converteNome(nomePesquisa);
        if (strstr(nomePesquisa, nome) != NULL)
        {
            tamanho++;
        }
    }
}
```

```

        printf("\n\nCliente %d\n\n", tamanho);

        relatorio(&no->dados);
    }
    no = no->prox;
}

if (tamanho == 0)
{
    printf("\n\nNenhum cliente encontrado com o nome %s\n\n", nome);
}
else
{
    printf("\n\nTotal de clientes encontrados: %d\n\n", tamanho);
}
}

```

Essa função recebe o endereço de memória da lista onde está armazenada todos os clientes e o nome do cliente a ser buscado, verifica se a lista alocada corretamente, se ela não está vazia e se o nome passado é válido (não é nulo), formata o nome a ser buscado, faz a inicialização de um ponteiro para percorrer a lista dentro do while, dentro do loop ela formata o nome da struct que está sendo acessada no momento e verifica se é o mesmo que o do cliente passado ou se o nome passado está incluso na string, se sim ele chama a função relatório para exibir as informações, incrementa a variável **tamanho** e depois passa para a próxima estrutura até chegar ao fim da lista (caso não encontre nenhum cliente ele não incrementa a variável e vai até o término da lista), no fim ela retorna se obteve ou não algum cliente (e o número de resultados encontrados).

4.6 Função de formatação (converteNome)


```

void converteNome(char nome[])
{
    for (int i = 0; i < strlen(nome); i++)
    {
        nome[i] = tolower(nome[i]);
    }
}

```

Essa função recebe um char, ou melhor, uma string que será formatada para lowercase, ou seja, deixará a string em minúscula. Essa função é usada na função de busca por nome para formatar o nome passado pelo usuário para a pesquisa e o nome de cada cliente na lista.

4.7 Função de exibição (relatorio)

A screenshot of a code editor with a dark background and light-colored text. The code is written in C and defines a function named 'relatorio' that takes a pointer to a 'CLIENTE' struct as an argument. The function prints out various fields of the struct, including the identifier, name, company, department, phone, cell phone, and email, each on a new line. The code is numbered from 1 to 10.

```
1 void relatorio(CLIENTE *al)
2 {
3     printf("\n\nIdentificador: %d", al->identificador);
4     printf("\nNome: %s", al->nome);
5     printf("\nEmpresa: %s", al->empresa);
6     printf("\nDepartamento: %s", al->departamento);
7     printf("\nTelefone: %s", al->telefone);
8     printf("\nCelular: %s", al->celular);
9     printf("\nEmail: %s", al->email);
10 }
```

Essa função recebe o endereço de memória da struct do cliente encontrado na lista e faz a exibição das suas informações.

4.8 Função de edição de dados do cliente (editarCliente)

Essa função recebe o endereço de memória da lista e o identificador do cliente que será editado, após verificar se a lista está corretamente alocada e que não está vazia, é criada uma estrutura do tipo CLIENTE e feita a inicialização de um ponteiro para percorrer a lista dentro de while, dentro do loop a função passa por cada estrutura (clientes) até encontrar a estrutura com o mesmo identificador do que foi passado (caso o programa não encontre nenhuma estrutura correspondente, ele retorna um aviso de cliente não encontrado), ao localizar o cliente ele sai do loop deixando o ponteiro apontando para a posição da lista onde está essa estrutura e chama a função de

exibição de informações (relatorio) e passa o endereço de memória da struct contida nessa posição da lista como argumento.

Ela pergunta para o usuário se ele realmente deseja alterar as informações, caso a resposta seja positiva ela chama a função de coleta de dados (coletaDados) e passa o valor do identificador da estrutura a ser editada como argumento e copia a estrutura devolvida por essa função para a estrutura criada no começo da função, após isso chama a função de remoção de clientes (removeOrdenado) passando o identificador da estrutura editada e depois chama a função de inserção de novos clientes passando a estrutura criada (aquele que recebeu as novas informações editadas) como argumento e retorna ao usuário se ela foi inserida ou não.

```
void editarCliente(Lista *li, int id)
{
    if (li == NULL)
    {
        abortaPrograma();
    }
    if (listaVazia(li))
    {
        return;
    }

    CLIENTE al;
    int x;
    ELEM *no = *li;
    while (no != NULL && no->dados.identificador != id)
    {
        no = no->prox;
    }
    if (no == NULL)
    {
        printf("\n\nCliente nao encontrado\n\n");
        return;
    }

    printf("\n\n");
    relatorio(&no->dados);

    printf("\n\nDeseja editar esse cliente? (1 - Sim / 0 - Nao): ");
```

```

scanf("%d", &x);
getchar();

if (x == 1)
{
    al = coletadados(li, no->dados.identificador);
    removeOrdenado(li, id);
    insereOrdenado(li, al);
    printf("\n\nCliente editado com sucesso!\n\n");
}
else
{
    printf("\n\nCliente nao editado\n\n");
}
}

```

4.9 Função para salvar o arquivo com as novas informações (salvarArquivo)

Essa função recebe o endereço de memória da lista, verifica se ela está corretamente alocada e não está vazia, após isso cria um ponteiro do tipo FILE para abrir um novo arquivo binário no modo de gravação (verificando se a abertura foi feita corretamente) e faz a inicialização de um ponteiro para percorrer a lista dentro de while, dentro do loop ela passa por cada estrutura (clientes) e para cada uma chama a função para gravar a estrutura no arquivo binário (fwrite) passando como argumento a estrutura (que está alocada dentro da posição atual em que o ponteiro está apontando), tamanho padrão da struct, quantidade a ser inserida e o arquivo onde será gravado. Ao final da função ela faz o fechamento do arquivo.

```

void salvarArquivo(Lista *li)
{
    if (li == NULL)
    {
        abortaPrograma();
    }
    if (listaVazia(li))
    {

```



```

        return;
    }

    FILE *arquivo = fopen("clientes.bin", "wb");

    if (arquivo == NULL)
    {
        printf("Erro ao abrir o arquivo\n");
        exit(1);
    }

    ELEM *no = *li;

    while (no != NULL)
    {
        fwrite(&no->dados, sizeof(CLIENTE), 1, arquivo);
        no = no->prox;
    }

    fclose(arquivo);
}

```

5 DIAGRAMA

Diagrama dos fluxos possíveis do programa, link para acesso no Mermaid [aqui](#) caso necessário.

