



UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Organização e Recuperação da Informação
Trabalho Prático 1

Professor: Ricardo Cerri

Integrantes do grupo

Bruno Leandro Pereira RA: 791067
Carlos Eduardo Fontaneli RA: 769949

São Carlos, 20 de outubro de 2021

1. Estrutura de dados:

Decidiu-se adotar *struct* para definir o banco de dados, pois tal estrutura atende bem o objetivo de armazenar todos os dados que o grupo considerou necessário para descrever os dados de uma pessoa.

Os campos da estrutura são vetores de tamanho fixo, sendo o tamanho deles um valor igual para todos os campos, tal tamanho é definido pela variável *TAM_MAX_CAMPO*.

```
typedef struct endereco
{
    char rua[TAM_MAX_CAMPO];          // tamanho do campo rua
    char num[TAM_MAX_CAMPO];          // tamanho do campo num
    char complemento[TAM_MAX_CAMPO];  // tamanho do campo complemento
} endereco;

typedef struct pessoa
{
    char chave[TAM_MAX_CAMPO];        // tamanho do campo chave
    char primeiro[TAM_MAX_CAMPO];     // tamanho do campo primeiro nome
    char ultimo[TAM_MAX_CAMPO];       // tamanho do campo último nome
    endereco endereco;                // tamanho do campo endereço
    char cidade[TAM_MAX_CAMPO];       // tamanho do campo cidade
    char estado[TAM_MAX_CAMPO];       // tamanho do campo estado
    char cep[TAM_MAX_CAMPO];          // tamanho do campo cep
    char telefone[TAM_MAX_CAMPO];     // tamanho do campo telefone
} pessoa;
```

Imagem 1: Registros que definem a estrutura de dados escolhida.

2. Funções e Procedimentos:

2.1. Insere Registro Sequencialmente:

Insere um registro sequencialmente, ou seja, sempre no final do arquivo, essa função fica em *loop* durante a inserção do registro, terminando a execução apenas quando o usuário decidir.

Esse procedimento recebe um arquivo, o qual será editado com os registros, e abre um segundo arquivo responsável por armazenar os índices secundários. Dentro da função é chamada um procedimento para ler um registro do usuário o qual será arquivado. Na sequência, é armazenado o *offset*(posição em que o registro vai ser

escrito) do registro, escreve-se daí cada campo do registro no arquivo e escreve-se no arquivo de índices o nome presente no registro e seu *offset*, isto é feito para assegurar o funcionamento do método de pesquisa por nome.

```
void insereRegistroSequencialmente(FILE *arquivo)
{
    int escrevendo = 0; // Variavel para
    controle de escrita de registro
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa

    FILE *arquivoIndices;
    int offset;
    arquivoIndices = fopen("indices.bin", "ab");
    while (NULL == arquivo)
    {
        char caminhoArquivo[100], modoAbertura[3];

        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
        printf("\nDigite o modo de abertura: ");
        scanf("%s", modoAbertura);

        arquivoIndices = fopen(caminhoArquivo, modoAbertura);
    }

    printf("1 - Escrever registro;\n0 - Sair;\n");
    scanf("%d", &escrevendo);
    // Escrever registro no arquivo
    while (escrevendo)
    {
        registro = obterRegistro();
        offset = ftell(arquivo);
        fwrite(&registro->chave, 16, 1, arquivo);
        fwrite(&registro->primeiro, 16, 1, arquivo);
        fwrite(&registro->ultimo, 16, 1, arquivo);
        fwrite(&registro->endereco.rua, 16, 1, arquivo);
        fwrite(&registro->endereco.num, 16, 1, arquivo);
        fwrite(&registro->endereco.complemento, 16, 1, arquivo);
    }
}
```

```
fwrite(&registro->cidade, 16, 1, arquivo);  
fwrite(&registro->estado, 16, 1, arquivo);  
fwrite(&registro->cep, 16, 1, arquivo);  
fwrite(&registro->telefone, 16, 1, arquivo);  
  
// Salvar o indice  
fwrite(&registro->primeiro, 16, 1, arquivoIndices);  
fwrite(&offset, 32, 1, arquivoIndices);  
  
printf("1 - Escrever registro;\n0 - Sair;\n");  
scanf("%d", &escrevendo);  
}  
  
fclose(arquivo);  
fclose(arquivoIndices);  
}
```

Imagem 2: Procedimento que insere os registros sequencialmente.

2.2. Insere Registro Reaproveitamento:

Insere um registro reaproveitando o espaço de outro registro marcado como apagado, ou seja, o método lê o arquivo que armazena os registros em busca de um espaço demarcado com remoção lógica, se ele encontra o registro é armazenado nesse espaço, se ele não encontra o registro é armazenado ao final do arquivo.

Esse procedimento recebe um arquivo, o qual será editado com os registros, e abre um segundo arquivo responsável por armazenar os índices secundários. O arquivo de registro é percorrido em busca de uma remoção lógica, então o procedimento lê cada registro e se não encontra um apagado ele pula um *offset* referente a um registro, caso encontre ele volta um *offset* referente ao tamanho do campo *chave* do registro, isso faz com que a posição corrente de escrita fique no início do registro, e depois escreve o novo registro.

Dentro da função é chamada um procedimento para ler um registro do usuário o qual será arquivado. Na sequência, é armazenado o *offset*(posição em que o registro vai ser escrito) do registro, escreve-se daí cada campo do registro no arquivo e escreve-se

no arquivo de índices o nome presente no registro e seu *offset*, isto é feito para assegurar o funcionamento do método de pesquisa por nome.

```
void insereRegistroReaproveitamento(FILE *arquivo)
{
    char chave[16] = "apagado\0"; // Variável para encontrar o
    registro apagado

    pessoa *registro = malloc(sizeof(pessoa)); // Alocação de
    estrutura pessoa

    FILE *arquivoIndices;
    int offset;
    arquivoIndices = fopen("indices.bin", "ab+");
    while (NULL == arquivo)
    {
        char caminhoArquivo[100], modoAbertura[3];

        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
        printf("\nDigite o modo de abertura: ");
        scanf("%s", modoAbertura);

        arquivoIndices = fopen(caminhoArquivo, modoAbertura);
    }

    fseek(arquivo, 0, SEEK_SET); // Posiciona a posição de leitura no
    inicio do arquivo
    // Lendo e comparando o campo
    while (fread(chave, 16, 1, arquivo) && strcmp(registro->chave,
    chave))
    {
        fseek(arquivo, 144, SEEK_CUR);
        fread(registro->chave, 16, 1, arquivo);
    };
    fseek(arquivo, -16, SEEK_CUR);
    registro = obterRegistro();
    offset = ftell(arquivo);
    fwrite(registro->chave, 16, 1, arquivo);
    fwrite(registro->primeiro, 16, 1, arquivo);
}
```

```
fwrite(registro->ultimo, 16, 1, arquivo);
fwrite(registro->endereco.rua, 16, 1, arquivo);
fwrite(registro->endereco.num, 16, 1, arquivo);
fwrite(registro->endereco.complemento, 16, 1, arquivo);
fwrite(registro->cidade, 16, 1, arquivo);
fwrite(registro->estado, 16, 1, arquivo);
fwrite(registro->cep, 16, 1, arquivo);
fwrite(registro->telefone, 16, 1, arquivo);

fseek(arquivoIndices, 0, SEEK_SET);
fwrite(registro->primeiro, 16, 1, arquivoIndices);
fwrite(&offset, 32, 1, arquivoIndices);

fclose(arquivo);
fclose(arquivoIndices);
}
```

Imagem 3: Procedimento que insere os registros com reaproveitamento dos espaços.

2.3. Apagar Registro:

Pesquisa pela chave e faz a exclusão lógica do registro, marcando o campo chave como “apagado”.

Esse procedimento percorre cada registro do arquivo passado para ele, procurando pela chave a ser apagada. Caso encontre, ele retorna um *offset* relativo ao tamanho do campo chave do registro e escreve a mensagem padrão de registro logicamente apagado.

```
void apagaRegistro(FILE *arquivo)
{
    int i;
    char chave[16], *espacoAdic;
    char apagado[16] = "apagado      \0";
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    printf("Digite a chave a ser pesquisada: ");
    scanf("%s", chave);
    espacoAdic = (char *)malloc(16 - strlen(chave) + 1);
    for (i = 0; i < 15 - strlen(chave); i++)
    {
        espacoAdic[i] = ' ';
    }
}
```

```
} // for
espacoAdic[i] = '\\0';
strcat(chave, espacoAdic);

fseek(arquivo, 0, SEEK_SET); // Posiciona a posicao de leitura no
inicio do arquivo

while (fread(registro->chave, 16, 1, arquivo))
{
    if (!strcmp(registro->chave, chave))
    {
        fseek(arquivo, -16, SEEK_CUR);
        fwrite(&apagado, 16, 1, arquivo);
        return;
    }
    fseek(arquivo, 144, SEEK_CUR);
};

fclose(arquivo);
}
```

Imagem 4: Procedimento que faz a exclusão lógica dos registros.

2.4. Leitura Completa:

Faz a leitura sequencial de todos os registros que não estão marcados como apagados e chama outra função para imprimí-los.

Esse procedimento percorre o arquivo lendo todos os registros e, se o mesmo não estiver marcado como apagado, depois o registro lido é passado como parâmetro para um procedimento responsável por imprimir cada campo do registro. Caso for encontrado um registro apagado, o procedimento pula um *offset* referente ao resto do registro levando a posição de leitura para o início do próximo registro.

```
void leituraCompleta(FILE *arquivo)
{
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    fseek(arquivo, 0, SEEK_SET); // Posiciona a posição
    de leitura no inicio do arquivo

    // Lê todos os registros que não estão como apagados
    while (fread(registro->chave, 16, 1, arquivo))
```

```
{
    if (strcmp(registro->chave, "apagado \0"))
    {
        fread(registro->primeiro, 16, 1, arquivo);
        fread(registro->ultimo, 16, 1, arquivo);
        fread(registro->endereco.rua, 16, 1, arquivo);
        fread(registro->endereco.num, 16, 1, arquivo);
        fread(registro->endereco.complemento, 16, 1, arquivo);
        fread(registro->cidade, 16, 1, arquivo);
        fread(registro->estado, 16, 1, arquivo);
        fread(registro->cep, 16, 1, arquivo);
        fread(registro->telefone, 16, 1, arquivo);
        imprimeRegistro(registro);
    }
}

fclose(arquivo);
}
```

Imagem 5: Procedimento que faz a leitura de todos os registros.

2.5. Imprimir Registro:

Imprime/exibe na tela os registros lidos. O procedimento recebe um registro como parâmetro e imprime na tela cada campo do registro junto com uma determinada formatação da interface.

```
void imprimeRegistro(pessoa *registro)
{
    if (registro != NULL)
    {
        printf("-----\n");
        printf("Nome: %s\n", registro->primeiro);
        printf("Sobrenome: %s\n", registro->ultimo);
        printf("Endereço: \n");
        printf("Estado: %s\n", registro->estado);
        printf("Cidade: %s\n", registro->cidade);
        printf("Rua: %s Número: %s Complemento: %s\n",
registro->endereco.rua,
registro->endereco.num,
registro->endereco.complemento);
        printf("CEP: %d\n", atoi(registro->cep));
        printf("Telefone: %d\n", atoi(registro->telefone));
        printf("-----\n");
    }
}
```




Imagem 6: Procedimento que exibe os registros na tela/monitor.

2.6. Obter campo:

Obtém um campo para os *struct's*, o procedimento recebe um *buffer* (que é o campo do registro), limpa a entrada e recebe o campo especificado do usuário em um vetor auxiliar. Depois, o método vai receber o espaço que falta para completar o tamanho fixo do campo e vai preencher ele com caracteres de espaço, depois esse espaço extra é concatenado ao campo final.

```
void obterCampo(char *buffer)
{
    char bufferAux[16]; // buffer utilizado para ler um campo do
teclado
    char *espacoAdic;    // espaco restante de um campo que deve
armazenado no buffer
    int i;              // variavel utilizada para acessar o array
espacoAdic

    fflush(stdin);      // limpar o buffer de entrada e saida
    scanf("%s", bufferAux); // obtendo dados de um campo da
estrutura pessoa
    strcat(buffer, bufferAux); // colocando os dados do campo no fim
do buffer

    espacoAdic = (char *)malloc(16 - strlen(bufferAux) + 1);

    for (i = 0; i < 15 - strlen(bufferAux); i++)
    {
        espacoAdic[i] = ' ';
    } // for
    espacoAdic[i] = '\0';

    strcat(buffer, espacoAdic);
}
```

Imagem 7: Procedimento que obtém um campo para a struct.

2.7. Realiza a operação:

Possibilita escolher qual operação executar. O método cria um ponteiro para o arquivo e para um registro que serão usados pelos demais procedimentos. Depois, imprime uma interface gráfica para o usuário poder escolher qual função ele deseja realizar. A execução da escolha do usuário é feita depois da leitura da opção e por meio de um *switch case*, onde para caso uma sequência de passos relativos a escolha é executada.

```
void realizaOperacao()
{
    int operacao;
    FILE *arquivo;
    pessoa *registro = malloc(sizeof(pessoa));
    int offset;

    do
    {
        printf("1 - Ler todos os registros;\n2 - Pesquisar por um\nregistro por chave;\n3 - Pesquisar registro por numero;\n4 -\nPesquisar registro por nome;\n5 - Escrever registro(s)\nsequencialmente;\n6 - Escrever registro com reaproveitamento de\nespaco\n7 - Apagar um registro;\n0 - Finalizar programa;\n");
        scanf("%d", &operacao);
        switch (operacao)
        {
            case 1: // Ler todos os registros
                arquivo = escolheArquivo("rb");
                leituraCompleta(arquivo);
                fclose(arquivo);
                break;
            case 2: // Pesquisar por um registro por chave
                arquivo = escolheArquivo("rb");
                registro = pesquisaRegistroChave(arquivo);
                imprimeRegistro(registro);
                fclose(arquivo);
                break;
            case 3: // Pesquisar registro por numero
                arquivo = escolheArquivo("rb");
                registro = pesquisaRegistroNumero(arquivo);
```

```
    imprimeRegistro(registro);  
    fclose(arquivo);  
    break;  
case 4: // Pesquisar registro por nome  
    arquivo = escolheArquivo("rb");  
    offset = pesquisaRegistroNome(1);  
    if (offset != -1)  
    {  
        leituraRegistroNome(offset, arquivo);  
    }  
    fclose(arquivo);  
    break;  
case 5: // Escrever registro(s) sequencialmente  
    arquivo = escolheArquivo("ab");  
    insereRegistroSequencialmente(arquivo);  
    fclose(arquivo);  
    break;  
case 6: // Escrever registro com reaproveitamento de espaço  
    arquivo = escolheArquivo("rb+");  
    insereRegistroReaproveitamento(arquivo);  
    fclose(arquivo);  
    break;  
case 7: // Apagar um registro  
    arquivo = escolheArquivo("rb+");  
    apagaRegistro(arquivo);  
    fclose(arquivo);  
    break;  
case 0: // Finalizar Programa  
    break;  
default:  
    printf("Valor digitado inválido!\n");  
    break;  
}  
} while (operacao != 0);
```

Imagem 8: Procedimento que possibilita escolher qual operação executar

2.8. Escolher Arquivo:

Encontra o arquivo através do caminho/nome e abre de acordo com o modo de abertura passado como parâmetro. O procedimento recebe um modo de abertura como parâmetro e solicita ao usuário o caminho relativo do arquivo dos registros. Na

sequência, abre-se o arquivo e testa-se se o arquivo foi aberto corretamente, em caso de erro uma mensagem é exibida para o usuário e um novo caminho é solicitado, em caso de sucesso o arquivo aberto é retornado.

```
FILE *escolheArquivo(char *modoAbertura)
{
    char caminhoArquivo[100]; // Variável que armazena o caminho até
o arquivo
    FILE *arquivo;

    printf("Digite o caminho(path) até o arquivo: ");
    scanf("%s", caminhoArquivo);

    arquivo = fopen(caminhoArquivo, modoAbertura);

    while (NULL == arquivo)
    {
        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);

        arquivo = fopen(caminhoArquivo, modoAbertura);
    }

    return arquivo;
}
```

Imagem 9: Função para abertura do arquivo.

2.9. Pesquisa registro pela CHAVE:

Faz a pesquisa do registro através do campo chave, o procedimento percorre todo o arquivo de registro lendo todos os registros em busca da chave selecionada, caso encontre ele retorna a chave, caso não encontre uma mensagem de aviso é imprimida na tela.

O procedimento lê uma entrada do usuário e completa o espaço adicional com espaços vazios (de forma semelhante ao método *apagaRegistro*).

```
pessoa *pesquisaRegistroChave(FILE *arquivo)
{
    int i;
    char chave[16], *espacoAdic;
```

```
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
estruturas pessoa
printf("Digite a chave a ser pesquisada: ");
scanf("%s", chave);

// Completa o espaço que sobrou no vetor da chave
espacoAdic = (char *)malloc(16 - strlen(chave) + 1);
for (i = 0; i < 15 - strlen(chave); i++)
{
    espacoAdic[i] = ' ';
} // for
espacoAdic[i] = '\0';
strcat(chave, espacoAdic);

fseek(arquivo, 0, SEEK_SET); // Posiciona a posicao de leitura no
inicio do arquivo
// Lendo o campo
while (fread(registro->chave, 16, 1, arquivo))
{
    // Comparando a chave a ser procurada com as que estão no
registro
    if (!strcmp(registro->chave, chave))
    {
        printf("Registro encontrado: \n");
        fread(registro->primeiro, 16, 1, arquivo);
        fread(registro->ultimo, 16, 1, arquivo);
        fread(registro->endereco.rua, 16, 1, arquivo);
        fread(registro->endereco.num, 16, 1, arquivo);
        fread(registro->endereco.complemento, 16, 1, arquivo);
        fread(registro->cidade, 16, 1, arquivo);
        fread(registro->estado, 16, 1, arquivo);
        fread(registro->cep, 16, 1, arquivo);
        fread(registro->telefone, 16, 1, arquivo);

        return registro;
    }

    fseek(arquivo, 144, SEEK_CUR); // Posiciona a posição de
leitura no próximo campo chave
};
```

```
printf("Registro não encontrado\n");  
return NULL;  
  
fclose(arquivo);  
}
```

Imagem 10: Função que possibilita a procura de um registro pela chave.

2.10. Pesquisa registro pelo NOME:

Faz a pesquisa do registro, através do campo nome, dentro do arquivo de índices, o procedimento percorre todo o arquivo lendo todos os índices em busca do nome selecionado, caso encontre o procedimento lê o *offset* do registro relativo ao nome, caso não encontre uma mensagem de aviso é imprimida na tela.

O procedimento lê uma entrada do usuário e completa o espaço adicional com espaços vazios (de forma semelhante ao método *apagaRegistro*). Além disso, quando um registro é apagado este método é chamado para apagar se o índice do registro que foi deletado.

```
int pesquisaRegistroNome(int pesquisa)  
{  
    int i;  
    char nomePesquisado[16], *espacoAdic;  
    char nomeArquivo[16];  
  
    FILE *arquivo;  
    int offset;  
    arquivo = fopen("indices.bin", "rb+");  
  
    while (NULL == arquivo)  
    {  
        char caminhoArquivo[100], modoAbertura[3];  
  
        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o  
caminho(path) até o arquivo: ");  
        scanf("%s", caminhoArquivo);  
        printf("\nDigite o modo de abertura: ");  
        scanf("%s", modoAbertura);  
  
        arquivo = fopen(caminhoArquivo, modoAbertura);  
    }
```

```
}  
printf("Digite o nome do registro a ser pesquisado: ");  
scanf("%s", nomePesquisado);  
  
// Completa o espaço que sobrou no vetor do primeiro nome  
espacoAdic = (char *)malloc(16 - strlen(nomePesquisado) + 1);  
for (i = 0; i < 15 - strlen(nomePesquisado); i++)  
{  
    espacoAdic[i] = ' ';  
} // for  
espacoAdic[i] = '\\0';  
strcat(nomePesquisado, espacoAdic);  
  
// Lendo o campo  
while (fread(nomeArquivo, 16, 1, arquivo))  
{  
    // Comparando o nome a ser procurado com os que estão no  
registro  
    if (!strcmp(nomeArquivo, nomePesquisado) && pesquisa)  
    {  
        printf("Registro encontrado: \\n");  
        fread(&offset, 32, 1, arquivo);  
  
        return offset;  
    }  
    else if (!pesquisa)  
    {  
        char apagado[16] = "apagado        \\0";  
        fseek(arquivo, -16, SEEK_CUR);  
        fwrite(&apagado, 16, 1, arquivo);  
  
        return 0;  
    }  
    fseek(arquivo, 32, SEEK_CUR);  
};  
  
printf("Registro não encontrado\\n");  
fclose(arquivo);  
  
return -1;
```

```
}
```

Imagem 11: Função que possibilita a procura de um registro pelo nome.

2.11. Pesquisa registro pelo NÚMERO:

Faz a pesquisa do registro através do campo número, o procedimento percorre todo o arquivo de registro lendo todos os registros em busca do número selecionada, caso encontre ele retorna o número, caso não encontre uma mensagem de aviso é imprimida na tela.

O procedimento lê uma entrada do usuário e completa o espaço adicional com espaços vazios(de forma semelhante ao método *apagaRegistro*).

```

pessoa *pesquisaRegistroNumero(FILE *arquivo)
{
    int numero, i = 0;
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    printf("Digite o numero do registro a ser pesquisado: ");
    scanf("%d", &numero);
    fseek(arquivo, 0, SEEK_SET); // Posiciona a posicao de leitura no
    inicio do arquivo
    fread(registro->chave, 16, 1, arquivo);
    while (i < numero || !strcmp(registro->chave, "apagado
    \0"))
    {
        if (strcmp(registro->chave, "apagado \0"))
        {
            i++;
            fseek(arquivo, 144, SEEK_CUR);
        }
        else
        {
            fseek(arquivo, 144, SEEK_CUR);
        }
        fread(registro->chave, 16, 1, arquivo);
    }
    fseek(arquivo, -16, SEEK_CUR);
    // Lendo o campo
    if (fread(registro->chave, 16, 1, arquivo))
    {
        printf("Registro encontrado: \n");
    }
}

```



```
fread(registro->primeiro, 16, 1, arquivo);
fread(registro->ultimo, 16, 1, arquivo);
fread(registro->endereco.rua, 16, 1, arquivo);
fread(registro->endereco.num, 16, 1, arquivo);
fread(registro->endereco.complemento, 16, 1, arquivo);
fread(registro->cidade, 16, 1, arquivo);
fread(registro->estado, 16, 1, arquivo);
fread(registro->cep, 16, 1, arquivo);
fread(registro->telefone, 16, 1, arquivo);

return registro;
}
else
{
    printf("Registro não encontrado\n");
    return NULL;
}

fclose(arquivo);
}
```

Imagem 12: Função que possibilita a procura de um registro pelo número.

2.12. Leitura do registro encontrado pelo nome:

Colocando a posição de leitura no byte offset do registro no arquivo principal, é feita a leitura do mesmo e chamada outra função para imprimí-los. Este procedimento recebe um *offset* (retornado pelo método `pesquisaRegistroNome`) relativo a um determinado registro e define a posição de leitura para o início do registro relativo. Na sequência o registro é lido e passado ao método responsável por imprimi-lo.

```
void leituraRegistroNome(int offset, FILE *arquivo)
{
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa

    // Posiciona a posição de leitura no byte offset indicado pela
    pesquisa por nome
    fseek(arquivo, offset, SEEK_SET);
    fread(registro->chave, 16, 1, arquivo);
    fread(registro->primeiro, 16, 1, arquivo);
    fread(registro->ultimo, 16, 1, arquivo);
}
```

```
fread(registro->endereco.rua, 16, 1, arquivo);  
fread(registro->endereco.num, 16, 1, arquivo);  
fread(registro->endereco.complemento, 16, 1, arquivo);  
fread(registro->cidade, 16, 1, arquivo);  
fread(registro->estado, 16, 1, arquivo);  
fread(registro->cep, 16, 1, arquivo);  
fread(registro->telefone, 16, 1, arquivo);  
imprimeRegistro(registro);  
fclose(arquivo);  
}
```

Imagem 13: Procedimento que lê o registro encontrado através do nome.

2.13. Obter Registro:

Requisita os dados do campo do registro ao usuário. Esta função imprime uma mensagem relativa a cada campo do registro e chama o método responsável por obter e formatar tal campo. Por fim, após preencher o registro o método o retorna.

```
pessoa *obterRegistro()  
{  
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da  
    estrutura pessoa  
  
    printf("Digite a chave do registro: ");  
    obterCampo(registro->chave);  
  
    printf("Digite o primeiro nome: ");  
    obterCampo(registro->primeiro);  
  
    printf("Digite o ultimo nome: ");  
    obterCampo(registro->ultimo);  
  
    printf("Digite a rua: ");  
    obterCampo(registro->endereco.rua);  
  
    printf("Digite o número da casa: ");  
    obterCampo(registro->endereco.num);  
  
    printf("Digite o complemento do endereço: ");  
    obterCampo(registro->endereco.complemento);  
  
    printf("Digite a cidade: ");
```

```
obterCampo(registro->cidade);  
  
printf("Digite o estado: ");  
obterCampo(registro->estado);  
  
printf("Digite o CEP: ");  
obterCampo(registro->cep);  
  
printf("Digite o telefone: ");  
obterCampo(registro->telefone);  
  
return registro;  
}
```

Imagem 14: Função que obtém os dados do registro.

2.14. Main:

```
int main()  
{  
    system("clear");  
    printf("Bem vindo ao programa que trata de registros com tamanho  
fixo!\n\n");  
    realizaOperacao();  
    system("clear");  
    printf("\n\nExecução Finalizada!\n\n");  
    return 0;  
}
```

Imagem 15: Função “main” do programa de registros de tamanhos fixos.

TAMANHO VARIÁVEL:

3. Estrutura de dados:

A estrutura de dados adotada foi a mesma do programa que trata de arquivos de tamanho fixo.

3.1. Inicia registro nulo:

A estrutura de dados adotada foi a mesma do programa que trata de arquivos de tamanho fixo.

Este procedimento recebe um registro e inicia todos os seus campos para ‘\0’. Isso é necessário, porque como os registros possuem tamanho variado, ao se ler um registro do arquivo e armazenar na estrutura *registro* o mesmo pode conter lixo de memória.

```
void iniciaRegistroNulo(pessoa *registro)
{
    registro->chave[15] = '\0';
    registro->primeiro[15] = '\0';
    registro->ultimo[15] = '\0';
    registro->endereco.rua[15] = '\0';
    registro->endereco.num[15] = '\0';
    registro->endereco.complemento[15] = '\0';
    registro->cidade[15] = '\0';
    registro->telefone[15] = '\0';
    registro->estado[15] = '\0';
    registro->cep[15] = '\0';
}
```

Imagem 16: Procedimento que inicia um registro como nulo.

3.2. Limpa campos do registro:

Este procedimento recebe um registro e preenche todos os espaços de todos os campos para '\0'. Isto é necessário, porque como o programa trata de ler arquivos com registros de tamanho variado, ao se ler um registro de tamanho X e na sequência ler um arquivo de tamanho X - Y, a variável que contém o registro estará armazenando caracteres relativos ao registro anteriormente lido, causando erros na execução.

```
void limpaCamposRegistro(pessoa *registro)
{
    memset(registro->chave, '\0', 16);
    memset(registro->primeiro, '\0', 16);
    memset(registro->ultimo, '\0', 16);
    memset(registro->endereco.rua, '\0', 16);
    memset(registro->endereco.num, '\0', 16);
    memset(registro->endereco.complemento, '\0', 16);
    memset(registro->cidade, '\0', 16);
    memset(registro->telefone, '\0', 16);
    memset(registro->estado, '\0', 16);
    memset(registro->cep, '\0', 16);
}
```

Imagem 17: Função para limpar campos de um registro.

3.3. Insere registro sequencialmente:

Insere um registro sequencialmente, ou seja, sempre no final do arquivo, essa função fica em *loop* durante a inserção do registro, terminando a execução apenas quando o usuário decidir.

Esse procedimento recebe um arquivo, o qual será editado com os registros, e abre um segundo arquivo responsável por armazenar os índices secundários. Dentro da função é chamada um procedimento para ler um registro do usuário, o qual será arquivado, depois a função define o tamanho total do registro e de cada um de seus campos, armazenando tais valores sempre nessa ordem: tamanho do campo e depois o campo em si. Na sequência, é armazenado o *offset*(posição em que o registro vai ser escrito) do registro e escreve-se no arquivo de índices o nome presente no registro e seu *offset*, isto é feito para assegurar o funcionamento do método de pesquisa por nome.

```
void insereRegistroSequencialmente(FILE *arquivo)
{
    int escrevendo = 0, tamCampo, offset;           // Variavel para
    controle de escrita de registro
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    iniciaRegistroNulo(registro);

    FILE *arquivoIndices;
    arquivoIndices = fopen("indices.bin", "ab");
    while (NULL == arquivo)
    {
        char caminhoArquivo[100], modoAbertura[3];

        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
        printf("\nDigite o modo de abertura: ");
        scanf("%s", modoAbertura);

        arquivoIndices = fopen(caminhoArquivo, modoAbertura);
    }

    printf("1 - Escrever registro;\n0 - Sair;\n");
    scanf("%d", &escrevendo);
```

```
// Escrever registro no arquivo
while (escrevendo)
{
    registro = obterRegistro();
    offset = ftell(arquivo);
    registro->tamRegistro += sizeof(int) * 10;
    fwrite(&registro->tamRegistro, sizeof(int), 1, arquivo);

    tamCampo = strlen(registro->chave);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->chave, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->primeiro);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->primeiro, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->ultimo);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->ultimo, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->endereco.rua);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->endereco.rua, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->endereco.num);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->endereco.num, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->endereco.complemento);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->endereco.complemento, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->cidade);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->cidade, tamCampo, 1, arquivo);

    tamCampo = strlen(registro->estado);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&registro->estado, tamCampo, 1, arquivo);
}
```

```
tamCampo = strlen(registro->cep);
fwrite(&tamCampo, sizeof(int), 1, arquivo);
fwrite(&registro->cep, tamCampo, 1, arquivo);

tamCampo = strlen(registro->telefone);
fwrite(&tamCampo, sizeof(int), 1, arquivo);
fwrite(&registro->telefone, tamCampo, 1, arquivo);

// Salvar o indice
tamCampo = strlen(registro->primeiro);
fwrite(&tamCampo, sizeof(int), 1, arquivoIndices);
fwrite(&registro->primeiro, tamCampo, 1, arquivoIndices);
fwrite(&offset, 32, 1, arquivoIndices);

limpaCamposRegistro(registro);
printf("1 - Escrever registro;\n0 - Sair;\n");
scanf("%d", &escrevendo);
}

fclose(arquivo);
fclose(arquivoIndices);
}
```

Imagem 18: Procedimento para inserção sequencial de registros.

3.4. Insere registro com reaproveitamento:

Insere um registro reaproveitando o espaço de outro registro marcado como apagado, ou seja, o método lê o arquivo que armazena os registros em busca de um espaço demarcado com remoção lógica, se ele encontra o registro é armazenado nesse espaço, se ele não encontra o registro é armazenado ao final do arquivo.

Esse procedimento recebe um arquivo, o qual será editado com os registros, e abre um segundo arquivo responsável por armazenar os índices secundários. Dentro da função é chamada um procedimento para ler um registro do usuário, o qual será arquivado, depois a função define o tamanho total do registro e de cada um de seus campos, armazenando tais valores sempre nessa ordem: tamanho do campo e depois o campo em si. Na sequência, é armazenado o *offset* (posição em que o registro vai ser

escrito) do registro e escreve-se no arquivo de índices o nome presente no registro e seu *offset*, isto é feito para assegurar o funcionamento do método de pesquisa por nome.

```
void insereRegistroReaproveitamento(FILE *arquivo)
{
    char chave[16] = "apagado\0"; // Variável para encontrar o
registro apagado
    int offset, tamCampo, tamRegistro;

    pessoa *registro = malloc(sizeof(pessoa)); // Alocação de
estrutura pessoa
    iniciaRegistroNulo(registro);

    FILE *arquivoIndices;

    arquivoIndices = fopen("indices.bin", "ab+");
    while (NULL == arquivo)
    {
        char caminhoArquivo[100], modoAbertura[3];

        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
        printf("\nDigite o modo de abertura: ");
        scanf("%s", modoAbertura);

        arquivoIndices = fopen(caminhoArquivo, modoAbertura);
    }

    fseek(arquivo, 0, SEEK_SET); // Posiciona a posição de leitura no
início do arquivo
    fread(&tamRegistro, sizeof(int), 1, arquivo);
    fread(&tamCampo, sizeof(int), 1, arquivo);
    // Lendo e comparando o campo
    while (fread(registro->chave, tamCampo, 1, arquivo) &&
strcmp(registro->chave, chave))
    {
        fseek(arquivo, tamRegistro - (tamCampo + sizeof(int)),
SEEK_CUR);
        fread(&tamRegistro, sizeof(int), 1, arquivo);
        fread(&tamCampo, sizeof(int), 1, arquivo);
    }
}
```



```
};  
fseek(arquivo, -(tamCampo + (2 * sizeof(int))), SEEK_CUR);  
registro = obterRegistro();  
offset = ftell(arquivo);  
registro->tamRegistro += sizeof(int) * 10;  
fwrite(&registro->tamRegistro, sizeof(int), 1, arquivo);  
  
tamCampo = strlen(registro->chave);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->chave, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->primeiro);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->primeiro, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->ultimo);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->ultimo, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->endereco.rua);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->endereco.rua, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->endereco.num);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->endereco.num, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->endereco.complemento);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->endereco.complemento, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->cidade);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->cidade, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->estado);  
fwrite(&tamCampo, sizeof(int), 1, arquivo);  
fwrite(&registro->estado, tamCampo, 1, arquivo);  
  
tamCampo = strlen(registro->cep);
```

```
fwrite(&tamCampo, sizeof(int), 1, arquivo);
fwrite(&registro->cep, tamCampo, 1, arquivo);

tamCampo = strlen(registro->telefone);
fwrite(&tamCampo, sizeof(int), 1, arquivo);
fwrite(&registro->telefone, tamCampo, 1, arquivo);

fseek(arquivoIndices, 0, SEEK_SET);
tamCampo = strlen(registro->primeiro);
fwrite(&tamCampo, sizeof(int), 1, arquivoIndices);
fwrite(&registro->primeiro, tamCampo, 1, arquivoIndices);
fwrite(&offset, 32, 1, arquivoIndices);

fclose(arquivo);
fclose(arquivoIndices);
}
```

Imagem 19: Procedimento para inserção com reaproveitamento de espaço de registros.

3.5. Apagar registro:

Pesquisa pela chave e faz a exclusão lógica do registro, marcando o campo chave como “apagado”.

Esse procedimento percorre cada registro do arquivo passado para ele, procurando pela chave a ser apagada. Caso encontre, ele retorna um *offset* relativo ao tamanho do campo chave do registro e escreve a mensagem padrão de registro logicamente apagado.

```
void apagaRegistro(FILE *arquivo)
{
    int tamCampo, tamRegistro;
    char chave[16];
    char apagado[16] = "apagado\0";
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    iniciaRegistroNulo(registro);

    printf("Digite a chave a ser pesquisada: ");
    scanf("%s", chave);

    fseek(arquivo, 0, SEEK_SET); // Posiciona a posicao de leitura no
    inicio do arquivo
```

```
fread(&tamRegistro, sizeof(int), 1, arquivo);
fread(&tamCampo, sizeof(int), 1, arquivo);
while (fread(registro->chave, tamCampo, 1, arquivo))
{
    if (!strcmp(registro->chave, chave))
    {
        printf("Registrado encontrado!\n");
        fseek(arquivo, -(tamCampo + sizeof(int)), SEEK_CUR);
        tamCampo = strlen(apagado);
        fwrite(&tamCampo, sizeof(int), 1, arquivo);
        fwrite(apagado, tamCampo, 1, arquivo);
        printf("Registrado apagado!\n");
        fclose(arquivo);
        return;
    }

    fseek(arquivo, tamRegistro - (tamCampo + sizeof(int)),
SEEK_CUR);
    for (int i = 0; i < tamCampo; i++)
    {
        registro->chave[i] = '\0';
    }
    fread(&tamRegistro, sizeof(int), 1, arquivo);
    fread(&tamCampo, sizeof(int), 1, arquivo);
};

fclose(arquivo);
}
```

Imagem 20: Função para apagar um registro.

3.6. Leitura completa:

Pesquisa pela chave e faz a exclusão lógica do registro, marcando o campo chave como “apagado”.

Esse procedimento percorre cada registro do arquivo passado para ele, procurando pela chave a ser apagada. Caso encontre, ele retorna um *offset* relativo ao tamanho do campo chave do registro e escreve a mensagem padrão de registro logicamente apagado.

```
void leituraCompleta(FILE *arquivo)
{
```

```
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
estrutur
a pessoa
    iniciaRegistroNulo(registro);

    fseek(arquivo, 0, SEEK_SET); // Posiciona a posição de leitura no
início do arquivo
    int tamRegistro, tamCampo;

    // Lê todos os registros que não estão como apagados

    while (fread(&tamRegistro, sizeof(int), 1, arquivo))
    {
        fread(&tamCampo, sizeof(int), 1, arquivo);
        fread(registro->chave, tamCampo, 1, arquivo);
        if (strcmp(registro->chave, "apagado\0"))
        {
            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->primeiro, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->ultimo, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->endereco.rua, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->endereco.num, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->endereco.complemento, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->cidade, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->estado, tamCampo, 1, arquivo);

            fread(&tamCampo, sizeof(int), 1, arquivo);
            fread(registro->cep, tamCampo, 1, arquivo);
        }
    }
}
```

```
fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->telefone, tamCampo, 1, arquivo);

imprimeRegistro(registro);
limpaCamposRegistro(registro);
}
else
{
    fseek(arquivo, tamRegistro - (tamCampo + sizeof(int)),
SEEK_CUR);
    for (int i = 0; i < tamCampo; i++)
    {
        registro->chave[i] = '\\0';
    }
}
}
```

Imagem 21: Procedimento para ler todos os registros de um arquivo.

3.7. Imprime registro:

Imprime/exibe na tela os registros lidos. O procedimento recebe um registro como parâmetro e imprime na tela cada campo do registro junto com uma determinada formatação da interface.

```
void imprimeRegistro(pessoa *registro)
{
    if (registro != NULL)
    {
        printf("-----\\n");
        printf("Nome: %s\\n", registro->primeiro);
        printf("Sobrenome: %s\\n", registro->ultimo);
        printf("Endereço: \\n");
        printf("Estado: %s\\n", registro->estado);
        printf("Cidade: %s\\n", registro->cidade);
        printf("Rua: %s Número: %s Complemento: %s\\n",
registro->endereco.rua,
registro->endereco.num,
registro->endereco.complemento);
        printf("CEP: %d\\n", atoi(registro->cep));
        printf("Telefone: %d\\n", atoi(registro->telefone));
        printf("-----\\n");
    }
}
```

```
}
```

Imagem 22:

3.8. Obter campo: procedimento que imprime um registro.

Obtém um campo para os *struct's*, o procedimento recebe um *buffer* (que é o campo do registro), limpa a entrada e recebe o campo especificado do usuário em um vetor auxiliar.

```
void obterCampo(char *buffer)
{
    fflush(stdin); // limpar o buffer de entrada e saída
    scanf("%s", buffer); // obtendo dados de um campo da estrutura
    pessoa
}
```

Imagem 23: Procedimento para obter o campo de um registro.

3.9. Realiza operação:

Possibilita escolher qual operação executar. O método cria um ponteiro para o arquivo e para um registro que serão usados pelos demais procedimentos. Depois, imprime uma interface gráfica para o usuário poder escolher qual função ele deseja realizar. A execução da escolha do usuário é feita depois da leitura da opção e por meio de um *switch case*, onde para caso uma sequência de passos relativos a escolha é executada.

```
void realizaOperacao()
{
    int operacao;
    FILE *arquivo;
    pessoa *registro = malloc(sizeof(pessoa));
    iniciaRegistroNulo(registro);

    int offset;

    do
    {
        printf("1 - Ler todos os registros;\n2 - Pesquisar por um\nregistro por chave;\n3 - Pesquisar registro por nome;\n4 - Escrever\nregistro(s) sequencialmente;\n5 - Escrever registro com\nreaproveitamento de espaço;\n6 - Apagar um registro;\n0 - Finalizar\nprograma;\n");
        printf("Opção escolhida: ");
```

```
scanf("%d", &operacao);
switch (operacao)
{
case 1: // Ler todos os registros
    system("clear");
    printf("Realizando leitura completa:  \n\n");
    arquivo = escolheArquivo("rb");
    leituraCompleta(arquivo);
    break;
case 2: // Pesquisar por um registro por chave
    system("clear");
    printf("Realizando pesquisa por chave:  \n\n");
    arquivo = escolheArquivo("rb");
    registro = pesquisaRegistroChave(arquivo);
    imprimeRegistro(registro);
    break;
case 3: // Pesquisar registro por nome
    system("clear");
    printf("Realizando pesquisa por nome:  \n\n");
    arquivo = escolheArquivo("rb");
    offset = pesquisaRegistroNome(1);
    if (offset != -1)
    {
        leituraRegistroNome(offset, arquivo);
    }
    break;
case 4: // Escrever registro(s) sequencialmente
    system("clear");
    printf("Realizando escrita sequencial de registros:  \n\n");
    arquivo = escolheArquivo("ab");
    insereRegistroSequencialmente(arquivo);
    break;
case 5: // Escrever registro com reaproveitamento de espaço
    system("clear");
    printf("Realizando escrita de registros com reaproveitamento:
\n\n");
    arquivo = escolheArquivo("rb+");
    insereRegistroReaproveitamento(arquivo);
    break;
case 6: // Apagar um registro
```

```

    system("clear");
    printf("Realizando exclusão de registro:  \n\n");
    arquivo = escolheArquivo("rb+");
    apagaRegistro(arquivo);
    break;
case 0: // Finalizar Programa
    break;
default:
    printf("Valor digitado inválido!\n");
    break;
}
} while (operacao != 0);
}

```

Imagem 24: Procedimento que realiza a operação escolhida pelo usuário.

3.10. Escolher arquivo:

Encontra o arquivo através do caminho/nome e abre de acordo com o modo de abertura passado como parâmetro. O procedimento recebe um modo de abertura como parâmetro e solicita ao usuário o caminho relativo do arquivo dos registros. Na sequência, abre-se o arquivo e testa-se se o arquivo foi aberto corretamente, em caso de erro uma mensagem é exibida para o usuário e um novo caminho é solicitado, em caso de sucesso o arquivo aberto é retornado.

```

FILE *escolheArquivo(char *modoAbertura)
{
    char caminhoArquivo[100]; // Variável que armazena o caminho até
o arquivo
    FILE *arquivo;

    printf("Digite o caminho(path) até o arquivo: ");
    scanf("%s", caminhoArquivo);

    arquivo = fopen(caminhoArquivo, modoAbertura);

    while (NULL == arquivo)
    {
        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o
caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
    }
}

```



```
    arquivo = fopen(caminhoArquivo, modoAbertura);  
}  
  
return arquivo;  
}
```

Imagem 25: Função que abre um arquivo.

3.11. Pesquisa registro pela CHAVE:

Faz a pesquisa do registro através do campo chave, o procedimento percorre todo o arquivo de registro lendo todos os registros em busca da chave selecionada, caso encontre ele retorna a chave, caso não encontre uma mensagem de aviso é imprimida na tela.

O procedimento lê uma entrada do usuário e completa o espaço adicional com espaços vazios (de forma semelhante ao método *apagaRegistro*).

```
pessoa *pesquisaRegistroChave(FILE *arquivo)  
{  
    char chave[16];  
    int tamCampo, tamRegistro;  
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da  
    estrutura pessoa  
    iniciaRegistroNulo(registro);  
  
    printf("Digite a chave a ser pesquisada: ");  
    scanf("%s", chave);  
  
    fseek(arquivo, 0, SEEK_SET); // Posiciona a posicao de leitura no  
    inicio do arquivo  
    // Lendo o campo  
    while (fread(&tamRegistro, sizeof(int), 1, arquivo))  
    {  
        fread(&tamCampo, sizeof(int), 1, arquivo);  
        fread(registro->chave, tamCampo, 1, arquivo);  
        if (!strcmp(registro->chave, chave))  
        {  
            fread(&tamCampo, sizeof(int), 1, arquivo);  
            fread(registro->primeiro, tamCampo, 1, arquivo);  
  
            fread(&tamCampo, sizeof(int), 1, arquivo);  
            fread(registro->ultimo, tamCampo, 1, arquivo);  
        }  
    }
```

```
fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.rua, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.num, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.complemento, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->cidade, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->estado, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->cep, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->telefone, tamCampo, 1, arquivo);

return registro;
}
else
{
    fseek(arquivo, tamRegistro - tamCampo - sizeof(int),
SEEK_CUR);
    for (int i = 0; i < tamCampo; i++)
    {
        registro->chave[i] = '\\0';
    }
}
printf("Registro não encontrado\\n");
return NULL;

fclose(arquivo);
}
```

Imagem 26: Função que pesquisa um registro dado sua chave.

3.12. Pesquisa registro pelo NOME:

Faz a pesquisa do registro, através do campo nome, dentro do arquivo de índices, o procedimento percorre todo o arquivo lendo todos os índices em busca do nome selecionado, caso encontre o procedimento lê o *offset* do registro relativo ao nome, caso não encontre uma mensagem de aviso é imprimida na tela.

O procedimento lê uma entrada do usuário e completa o espaço adicional com espaços vazios (de forma semelhante ao método *apagaRegistro*). Além disso, quando um registro é apagado este método é chamado para apagar se o índice do registro que foi deletado.

```
int pesquisaRegistroNome(int pesquisa)
{
    int tamCampo;
    char nomePesquisado[16] = {'\0'};
    char nomeArquivo[16] = {'\0'};

    FILE *arquivo;
    int offset;
    arquivo = fopen("indices.bin", "rb+");

    while (NULL == arquivo)
    {
        char caminhoArquivo[100], modoAbertura[3];

        printf("ERRO AO ABRIR ARQUIVO!\nDigite novamente o\n"
            caminho(path) até o arquivo: ");
        scanf("%s", caminhoArquivo);
        printf("\nDigite o modo de abertura: ");
        scanf("%s", modoAbertura);

        arquivo = fopen(caminhoArquivo, modoAbertura);
    }
    printf("Digite o nome do registro a ser pesquisado: ");
    scanf("%s", nomePesquisado);

    // Lendo o campo
    while (fread(&tamCampo, sizeof(int), 1, arquivo))
    {
        fread(nomeArquivo, tamCampo, 1, arquivo);
    }
}
```

```
// Comparando o nome a ser procurado com os que estão no
registro
if (!strcmp(nomeArquivo, nomePesquisado) && pesquisa)
{
    printf("Registro encontrado: \n");
    fread(&offset, 32, 1, arquivo);

    return offset;
}
else if (!pesquisa)
{
    char apagado[16] = "apagado\0";
    fseek(arquivo, -(tamCampo + sizeof(int)), SEEK_CUR);
    tamCampo = strlen(apagado);
    fwrite(&tamCampo, sizeof(int), 1, arquivo);
    fwrite(&apagado, tamCampo, 1, arquivo);

    return 0;
}
fseek(arquivo, 32, SEEK_CUR);
};

printf("Registro não encontrado\n");
fclose(arquivo);

return -1;
}
```

Imagem 27: Função que pesquisa um registro dado seu nome.

3.13. Leitura registro nome:

Colocando a posição de leitura no byte offset do registro no arquivo principal, é feita a leitura do mesmo e chamada outra função para imprimí-los. Este procedimento recebe um *offset* (retornado pelo método `pesquisaRegistroNome`) relativo a um determinado registro e defini a posição de leitura para o início do registro relativo. Na sequência o registro é lido e passado ao método responsável por imprimi-lo.

```
void leituraRegistroNome(int offset, FILE *arquivo)
{
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
}
```

```
iniciaRegistroNulo(registro);

int tamCampo, tamRegistro;
// Posiciona a posição de leitura no byte offset indicado pela
pesquisa por nome
fseek(arquivo, offset, SEEK_SET);
fread(&tamRegistro, sizeof(int), 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->chave, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->primeiro, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->ultimo, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.rua, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.num, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->endereco.complemento, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->cidade, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->estado, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->cep, tamCampo, 1, arquivo);

fread(&tamCampo, sizeof(int), 1, arquivo);
fread(registro->telefone, tamCampo, 1, arquivo);

imprimeRegistro(registro);
}
```

Imagem 28: Função que faz a leitura de um registro dado sua chave.

3.14. Obter registro:

Requisita os dados do campo do registro ao usuário. Esta função imprime uma mensagem relativa a cada campo do registro e chama o método responsável por obter e formatar tal campo. Por fim, após preencher o registro o método o retorna.

```

pessoa *obterRegistro()
{
    pessoa *registro = malloc(sizeof(pessoa)); // Alocação da
    estrutura pessoa
    iniciaRegistroNulo(registro);

    registro->tamRegistro = 0;

    printf("Digite a chave do registro: ");
    obterCampo(registro->chave);
    registro->tamRegistro += strlen(registro->chave);

    printf("Digite o primeiro nome: ");
    obterCampo(registro->primeiro);
    registro->tamRegistro += strlen(registro->primeiro);

    printf("Digite o ultimo nome: ");
    obterCampo(registro->ultimo);
    registro->tamRegistro += strlen(registro->ultimo);

    printf("Digite a rua: ");
    obterCampo(registro->endereco.rua);
    registro->tamRegistro += strlen(registro->endereco.rua);

    printf("Digite o número da casa: ");
    obterCampo(registro->endereco.num);
    registro->tamRegistro += strlen(registro->endereco.num);

    printf("Digite o complemento do endereço: ");
    obterCampo(registro->endereco.complemento);
    registro->tamRegistro += strlen(registro->endereco.complemento);

    printf("Digite a cidade: ");
    obterCampo(registro->cidade);

```

```
registro->tamRegistro += strlen(registro->cidade);

printf("Digite o estado: ");
obterCampo(registro->estado);
registro->tamRegistro += strlen(registro->estado);

printf("Digite o CEP: ");
obterCampo(registro->cep);
registro->tamRegistro += strlen(registro->cep);

printf("Digite o telefone: ");
obterCampo(registro->telefone);
registro->tamRegistro += strlen(registro->telefone);

return registro;
}
```

Imagem 29: Função que obtém os campos de um registro.

3.13. Main:

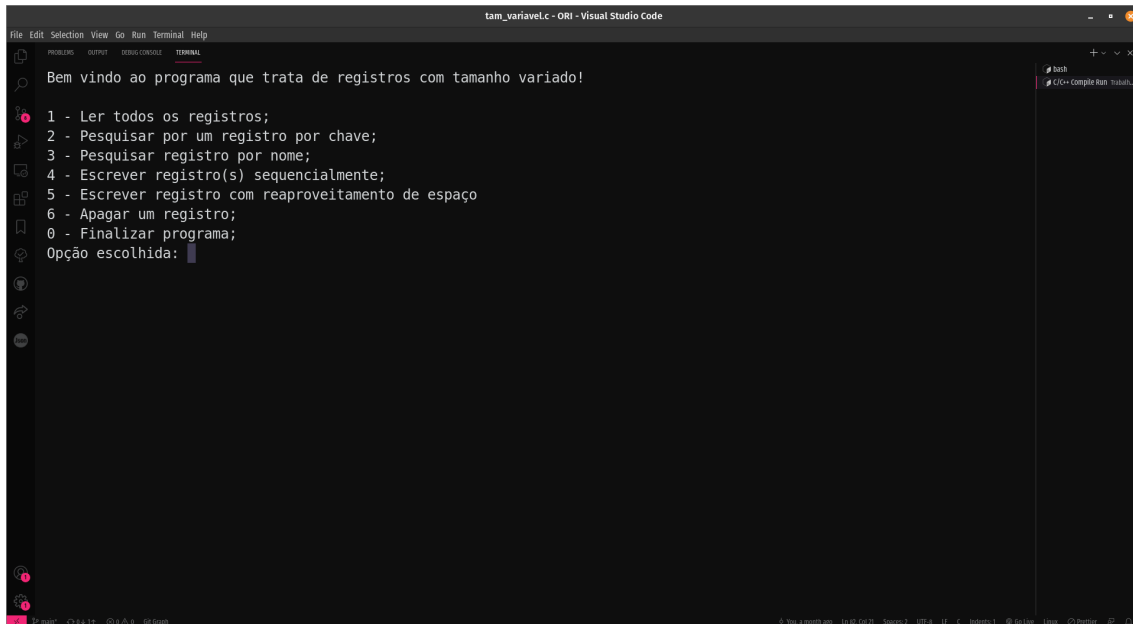
```
int main()
{
    system("clear");
    printf("Bem vindo ao programa que trata de registros com tamanho  
variado!\n\n");
    realizaOperacao();
    system("clear");
    printf("\n\nExecução Finalizada!\n\n");
    return 0;
}
```

Imagem 30: Função “main” do programa de registros de tamanho variáveis.

4. Bateria de testes:

4.1. Interface do programa:

4.1.1. Programa com registros de tamanho variável:



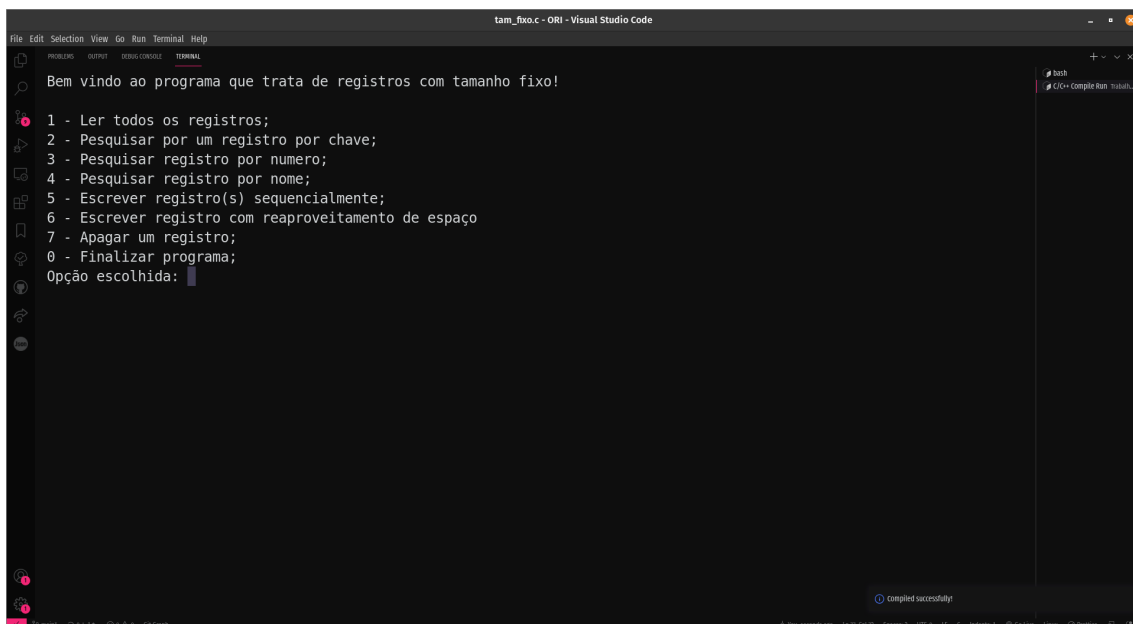
```
tam_variavel.c - ORI - Visual Studio Code

File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Bem vindo ao programa que trata de registros com tamanho variado!

1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por nome;
4 - Escrever registro(s) sequencialmente;
5 - Escrever registro com reaproveitamento de espaço
6 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida: 
```

4.1.2. Programa com registro de tamanho fixo:



```
tam_fixo.c - ORI - Visual Studio Code

File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

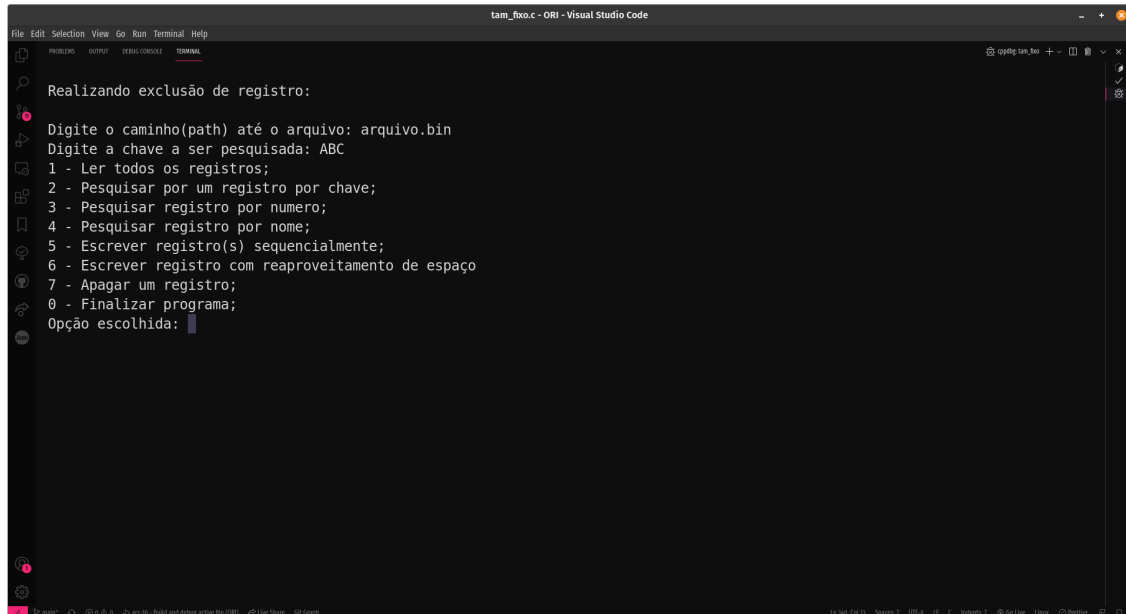
Bem vindo ao programa que trata de registros com tamanho fixo!

1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida: 

compiled successfully
```

4.2. Apagar registro:

Teste 1: Exclusão com sucesso.

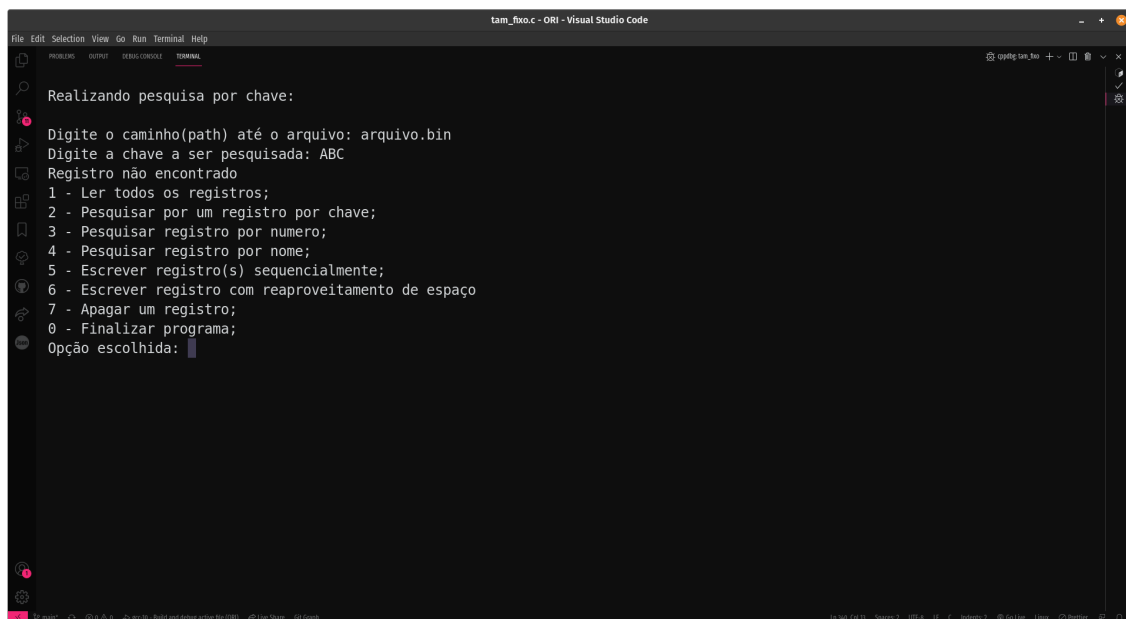


```
tam_fixo.c - ORI - Visual Studio Code

Realizando exclusão de registro:

Digite o caminho(path) até o arquivo: arquivo.bin
Digite a chave a ser pesquisada: ABC
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida:
```

Teste 2: Exclusão com fracasso.



```
tam_fixo.c - ORI - Visual Studio Code

Realizando pesquisa por chave:

Digite o caminho(path) até o arquivo: arquivo.bin
Digite a chave a ser pesquisada: ABC
Registro não encontrado
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida:
```

4.3. Insere registro com reaproveitamento:

```
tam_fixo.c - ORI - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Realizando escrita de registros com reaproveitamento:
Digite o caminho(path) até o arquivo: arquivo.bin
Digite a chave do registro: DEF
Digite o primeiro nome: Fernando
Digite o último nome: Oliveira
Digite a rua: DEF
Digite o número da casa: 456
Digite o complemento do endereço: DEF
Digite a cidade: DEF
Digite o estado: DEF
Digite o CEP: 123456789
Digite o telefone: 123456789
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida: 
```

```
tam_fixo.c - ORI - Visual Studio Code
Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Realizando leitura completa:
Digite o caminho(path) até o arquivo: arquivo.bin
-----
Nome: Fernando
Sobrenome: Oliveira
Endereço:
Estado: DEF
Cidade: DEF
Rua: DEF          Número: 123          Complemento: DEF
CEP: 123456789
Telefone: 123456789
-----
Nome: Bruno
Sobrenome: Pereira
Endereço:
Estado: CBA
Cidade: CBA
Rua: CBA          Número: 321          Complemento: CBA
CEP: 987654321
Telefone: 987654321
-----
Nome: Fernando
Sobrenome: Oliveira
Endereço:
Estado: DEF
```

4.4. Insere registro sequencialmente:

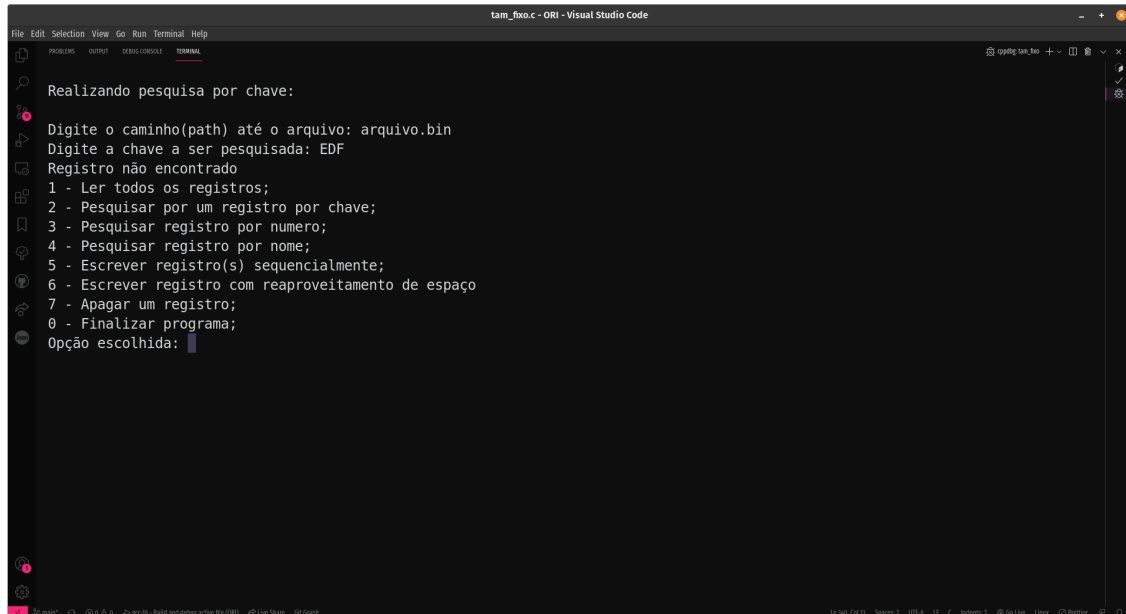
```
tam_fixo.c - ORI - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Digite o caminho(path) até o arquivo: arquivo.bin
1 - Escrever registro;
0 - Sair;
1
Digite a chave do registro: ABC
Digite o primeiro nome: Carlos
Digite o ultimo nome: Eduardo
Digite a rua: ABC
Digite o número da casa: 123
Digite o complemento do endereço: ABC
Digite a cidade: ABC
Digite o estado: ABC
Digite o CEP: 123456789
Digite o telefone: 123456789
1 - Escrever registro;
0 - Sair;
1
Digite a chave do registro: CBA
Digite o primeiro nome: Bruno
Digite o ultimo nome: Pereira
Digite a rua: CBA
Digite o número da casa: 321
Digite o complemento do endereço: CBA
Digite a cidade: CBA
Digite o estado: CBA
Digite o CEP: 987654321
Digite o telefone: 987654321
1 - Escrever registro;
```

4.5. Leitura Completa:

```
tam_fixo.c - ORI - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Realizando leitura completa:
Digite o caminho(path) até o arquivo: arquivo.bin
-----
Nome: Carlos
Sobrenome: Eduardo
Endereço:
Estado: ABC
Cidade: ABC
Rua: ABC          Número: 123          Complemento: ABC
CEP: 123456789
Telefone: 123456789
-----
Nome: Bruno
Sobrenome: Pereira
Endereço:
Estado: CBA
Cidade: CBA
Rua: CBA          Número: 321          Complemento: CBA
CEP: 987654321
Telefone: 987654321
-----
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
```

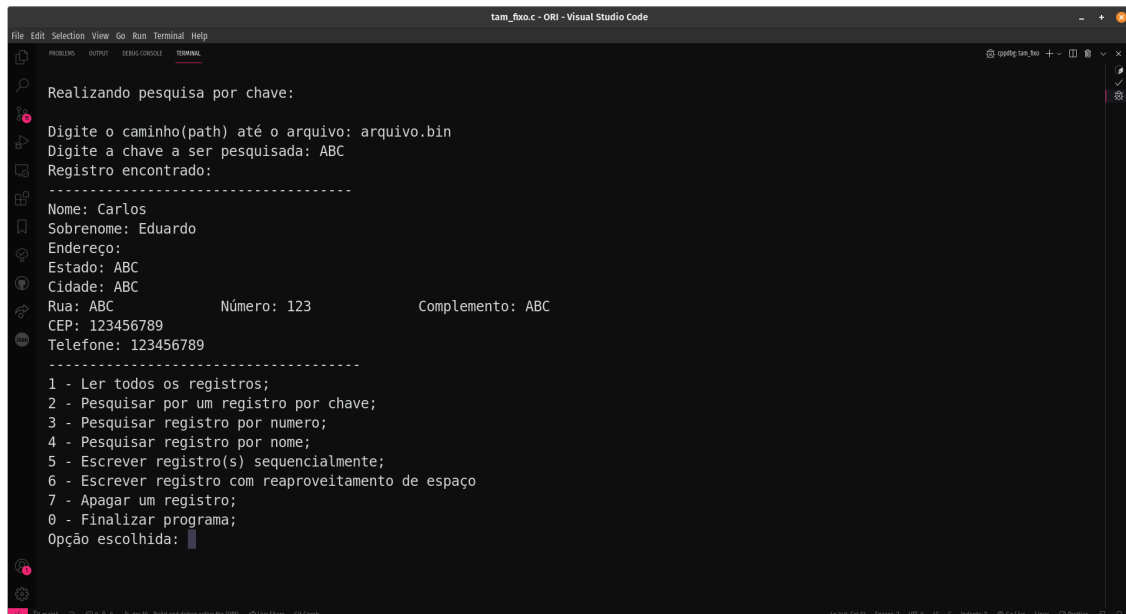
4.6. Pesquisa pela CHAVE:

Teste 1: Pesquisa com fracasso.



```
tam_fixo.c - ORI - Visual Studio Code
Realizando pesquisa por chave:
Digite o caminho(path) até o arquivo: arquivo.bin
Digite a chave a ser pesquisada: EDF
Registro não encontrado
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida:
```

Teste 2: Pesquisa com sucesso.



```
tam_fixo.c - ORI - Visual Studio Code
Realizando pesquisa por chave:
Digite o caminho(path) até o arquivo: arquivo.bin
Digite a chave a ser pesquisada: ABC
Registro encontrado:
-----
Nome: Carlos
Sobrenome: Eduardo
Endereço:
Estado: ABC
Cidade: ABC
Rua: ABC          Número: 123          Complemento: ABC
CEP: 123456789
Telefone: 123456789
-----
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida:
```

4.7. Pesquisa pelo NOME:

```
tam_fixo.c - ORI - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Realizando pesquisa por nome:
Digite o caminho(path) até o arquivo: arquivo.bin
Digite o nome do registro a ser pesquisado: Bruno
Registro encontrado:
-----
Nome: Bruno
Sobrenome: Pereira
Endereço:
Estado: CBA
Cidade: CBA
Rua: CBA          Número: 321          Complemento: CBA
CEP: 987654321
Telefone: 987654321
-----
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida: 
```

4.8. Pesquisa pelo NÚMERO:

```
tam_fixo.c - ORI - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Realizando pesquisa por número do registro:
Digite o caminho(path) até o arquivo: arquivo.bin
Digite o numero do registro a ser pesquisado: 0
Registro encontrado:
-----
Nome: Carlos
Sobrenome: Eduardo
Endereço:
Estado: ABC
Cidade: ABC
Rua: ABC          Número: 123          Complemento: ABC
CEP: 123456789
Telefone: 123456789
-----
1 - Ler todos os registros;
2 - Pesquisar por um registro por chave;
3 - Pesquisar registro por numero;
4 - Pesquisar registro por nome;
5 - Escrever registro(s) sequencialmente;
6 - Escrever registro com reaproveitamento de espaço
7 - Apagar um registro;
0 - Finalizar programa;
Opção escolhida: 
```