

Lista 4 – Lab. De Estrutura de Dados 1 e Estrutura de Dados 1

Aluno: Anderson Carlos da Silva Moraes

Matrícula: 2024011327

Repositório: [algorithm-and-data-structure/laboratory/lista-4](https://github.com/AndersonCSM/algorithm-and-data-structure) at [main](#) · [AndersonCSM/algorithm-and-data-structure](#)

Questões

1º Considerando o arquivo texto apresentado, o que o trecho de código abaixo faz?

```
#include <stdio.h> #include <stdlib.h>

int main() {

FILE *fin;

fin = fopen("intro.txt", "r");


char ch;

fscanf(fin, "%c", &ch); fclose(fin);

printf("%c\n", ch); return 0;

}
```

O código irá abrir o arquivo intro.txt, irá ler o primeiro caractere do arquivo, armazenando na variável ch e irá mostrá-lo no terminal.

Modifique o programa para ler do arquivo texto:

a) O oitavo caractere

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
    setlocale(LC_ALL, "Portuguese");
```

```

FILE *fin;
fin = fopen("intro.txt", "r");

char ch;

fseek(fin, 7, SEEK_SET);
fscanf(fin, "%c", &ch);
fclose(fin);
printf("%c\n", ch);
return 0;
}

```

b) Uma palavra

```

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
    setlocale(LC_ALL, "Portuguese");

    FILE *fin;
    fin = fopen("intro.txt", "r");

    char palavra[100];

    fscanf(fin, "%99s", palavra);
    fclose(fin);
    printf("%s\n", palavra);
    return 0;
}

```

c) A quinta palavra

```

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
    setlocale(LC_ALL, "Portuguese");

    FILE *fin;
    fin = fopen("intro.txt", "r");

    char palavra[100];
    for (int i = 0; i < 5; i++)
    {

```

```

        fscanf(fin, "%99s", palavra);
    }
    fclose(fin);
    printf("%s\n", palavra);
    return 0;
}

```

d) Uma linha

```

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
    setlocale(LC_ALL, "Portuguese");

    FILE *fin;
    fin = fopen("intro.txt", "r");

    char linha[256];

    if (fgets(linha, sizeof(linha), fin) != NULL)
    {
        printf("%s\n", linha);
    }
    fclose(fin);
    return 0;
}

```

2º Considere o arquivo texto apresentado abaixo:

```

Ontem, às 12 horas, faziam 40 graus ao sol e 38 à sombra.
Eu andei 100 metros antes de alcançar um lugar à sombra.
Foram 5 minutos de sofrimento.

```

Escreva um programa para ler todos os números do texto e gravá-los em outro arquivo.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");

```

```

FILE *fin, *fout;
char palavra[256];
char *endptr; // Para strtol, aponta para o primeiro caractere não
convertido
long num;

fin = fopen("q2_input.txt", "r");
fout = fopen("q2_output.txt", "w");

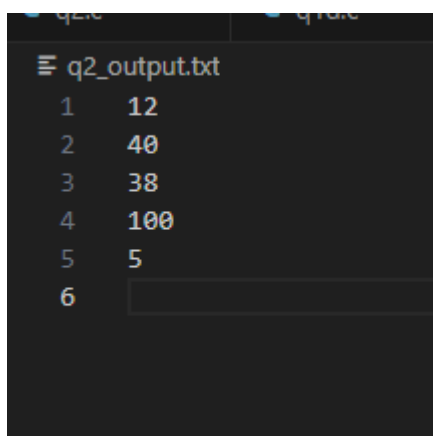
// Loop para ler as palavras(sequências de caracteres separadas por
espaço/nova linha)
while (fscanf(fin, "%255s", palavra) == 1)
{
    num = strtol(palavra, &endptr, 10); // Tenta converter a
'palavra' para long int strtol(palavra, ponteiro para o 1 caractere que
não converte, base decimal)

    // 1. endptr != palavra: Garante que pelo menos um caractere foi
processado por strtol. Se nenhum caractere pudesse formar um número,
endptr seria igual a palavra.
    // 2. *endptr == '\0': Garante que strtol processou a string
'palavra' inteira, ou seja, não havia caracteres não numéricos após a
parte numérica.
    if (endptr != palavra && *endptr == '\0')
    {
        fprintf(fout, "%ld\n", num);
    }
}

fclose(fin);
fclose(fout);

return 0;
}

```



```

q2_output.txt
1 12
2 40
3 38
4 100
5 5
6

```

3º Construa um programa que leia um número desconhecido de peixes de um arquivo texto, chamado “pescado.txt”, e exiba a quantidade total de quilos de peixe pescado. O arquivo é formatado como no exemplo abaixo: o nome do peixe, o peso em gramas e o comprimento em centímetros.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");

    FILE *arquivo = fopen("pescado.txt", "r");
    if (arquivo == NULL)
    {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    char nome[100];
    int peso = 0, comprimento = 0;
    int peso_tot = 0;

    // Cabeçalho da tabela formatado
    printf("%-20s | %-10s | %-18s\n", "Nome do Peixe", "Peso (g)",
"Comprimento (cm)");
    printf("=====|=====|=====\\n");
    while (fscanf(arquivo, "%99s %d %d", nome, &peso, &comprimento) == 3)
    {
        peso_tot += peso;
        printf("%-20s | %-10d | %-18d\\n", nome, peso, comprimento);
    }

    fclose(arquivo);

    printf("=====|=====|=====\\n");
    printf("\\n");

    fclose(arquivo);

    printf("Peso total: %.2f kg\\n", peso_tot / 1000.00);
    return 0;
}
```

4º Construa um programa que leia um arquivo texto contendo o nome e as três notas de vários alunos (uma quantidade indefinida de alunos). Escreva em outro arquivo texto o nome e a situação do aluno (aprovado, quarta prova ou reprovado).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

int main()
{
    // setlocale(LC_ALL, "Portuguese"); // o uso do setlocale muda o
    // caractere marcado do tipo float ao ler um arquivo, isso interfere no
    // código

    FILE *arq_inp = fopen("alunos.txt", "r");
    FILE *arq_out = fopen("alunos_status.txt", "w");

    if (arq_inp == NULL)
    {
        perror("Erro ao abrir o arquivo de entrada (alunos.txt)");
        return 1;
    }
    if (arq_out == NULL)
    {
        perror("Erro ao abrir o arquivo de saída (alunos_situacao.txt)");
        return 1;
    }

    char nome[100];
    float n1 = 0, n2 = 0, n3 = 0;
    float media = 0;
    char situacao[20];

    while (fscanf(arq_inp, "%99s %f %f %f", nome, &n1, &n2, &n3) == 4)
    {
        media = (n1 + n2 + n3) / 3.0;

        if (media >= 7.0)
        {
            strcpy(situacao, "Aprovado"); // copiar string
        }
        else if (media >= 3.0 && media < 7.0)
        {
            strcpy(situacao, "Quarta prova"); // copiar string
        }
        else
        {
            // reprovado
        }
    }
}
```

```

        strcpy(situacao, "Reprovado"); // copiar string
    }

    // Escreve os dados lidos e a situação calculada no arquivo de
SAÍDA
    fprintf(arq_out, "%-20s %-15s\n", nome, situacao);
}

fclose(arq_inp);
fclose(arq_out);

return 0;
}

```

5º Escreva um programa que abra um arquivo texto, leia caractere a caractere até o fim do arquivo e exiba na tela o número total de caracteres, o número de vogais, o número de consoantes, e a quantidade de outros caracteres presentes no texto.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");

    FILE *arq = fopen("texto.txt", "r");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo alunos.txt");
        return 1;
    }

    long tot_char = 0;
    long vogais = 0;
    long consoantes = 0;
    long outros = 0;
    int ch_int; // fgetc retorna int para poder representar EOF

    // Loop para ler caractere a caractere até o fim do arquivo (EOF)
    while ((ch_int = fgetc(arq)) != EOF)
    {
        tot_char++; // Incrementa o total de caracteres lidos

        char ch = (char)ch_int; // Converte o int lido para char
    }
}

```

```

        char ch_lower = tolower(ch); // Converte para minúsculo

        if (isalpha(ch_lower)) // Verifica se é uma letra do alfabeto
        {
            if (ch_lower == 'a' || ch_lower == 'e' || ch_lower == 'i' ||
ch_lower == 'o' || ch_lower == 'u')
            {
                vogais++;
            }
            else
            {
                consoantes++;
            }
        }
        else if (ch_lower != ' ')
        {
            outros++;
        }
    }

    fclose(arq);

    printf("Total de caracteres: %ld\n", tot_char);
    printf("Vogais: %ld\n", vogais);
    printf("Consoantes: %ld\n", consoantes);
    printf("Outros caracteres: %ld\n", outros);

    return 0;
}

```

6º Escreva um programa que peça ao usuário para digitar um nome de arquivo texto e uma palavra que ele deseja procurar neste arquivo. O programa deve dizer se a palavra está ou não presente no arquivo.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");
    char nome_arquivo[100];
    char palavra[100];
    char target[100];
    int encontrada = 0;

```



```

printf("Nome do arquivo: ");
fgets(nome_arquivo, sizeof(nome_arquivo), stdin); // Remove o
caractere de nova linha, se presente
nome_arquivo[strcspn(nome_arquivo, "\n")] = 0;

FILE *arq = fopen(nome_arquivo, "r");
if (arq == NULL)
{
    perror("Erro ao abrir o arquivo");
    return 1;
}
else
{
    printf("Palavra: ");
    fgets(palavra, sizeof(palavra), stdin);
    palavra[strcspn(palavra, "\n")] = 0;

    while (fscanf(arq, "%99s", target) != EOF)
    {
        if (strcspn(palavra, target) == 0)
        {
            printf("A palavra \"%s\" está presente no texto",
palavra);

            encontrada = 1;
            break;
        }
    }

    if (!encontrada) // Se a flag ainda for falsa após o loop
    {
        printf("A palavra \"%s\" NÃO está presente no texto.\n",
palavra);
    }

    fclose(arq);
    return 0;
}

```

7º Uma palavra é um palíndromo se a sequência de letras que a forma é a mesma quando lida da esquerda para a direita ou da direita para a esquerda (ex: raiar). Escreva uma função que seja capaz de descobrir se uma palavra é um palíndromo. Nesta verificação desconsidere maiúsculas e minúsculas (i.e. Ana é um palíndromo).

O programa deve ler um número indefinido de palavras de um arquivo texto e, para cada palavra, verificar se ela é ou não um palíndromo. Exiba a quantidade de palíndromos encontrados no texto, bem como as palavras que são palíndromos, como mostrado abaixo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

#define TAMANHO_MAX_PALAVRA 100

// Mantendo o nome da sua função
int check_palindromo(const char *word)
{
    if (word == NULL)
    {
        return 0;
    }
    int ini = 0;
    int fim = strlen(word) - 1;

    while (ini < fim)
    {
        char char_i = tolower((unsigned char)word[ini]);
        char char_f = tolower((unsigned char)word[fim]);

        if (char_i != char_f)
        {
            return 0; // Não é palíndromo
        }
        ini++;
        fim--;
    }
    return 1; // É palíndromo
}

int main()
{
    setlocale(LC_ALL, "Portuguese");

    int qtd = 0;
    char nome_arquivo[100] = "texto3.txt";
    char palavra[TAMANHO_MAX_PALAVRA]; // Buffer para ler cada palavra

    // Array dinâmico para armazenar as palavras palíndromo encontradas
    char (*lista_palindromos)[TAMANHO_MAX_PALAVRA] = NULL;
```

```

    int capacidade_lista = 0; // Capacidade atual do array
    lista_palindromos

    FILE *arq = fopen(nome_arquivo, "r");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo");
        fprintf(stderr, "Arquivo: %s\n", nome_arquivo);
        return 1;
    }

    // Lê palavras do arquivo uma por uma
    while (fscanf(arq, "%99s", palavra) == 1)
    {
        if (check_palindromo(palavra))
        {
            qtd++;

            // Realoca o array lista_palindromos se necessário
            if (qtd > capacidade_lista)
            {
                int n = (capacidade_lista == 0) ? 10 : capacidade_lista *
2;

                // Ajusta o tipo do ponteiro temporário e o sizeof para
realloc
                char (*temp)[TAMANHO_MAX_PALAVRA] =
realloc(lista_palindromos, n * sizeof(char[TAMANHO_MAX_PALAVRA]));

                if (temp == NULL)
                {
                    perror("Erro ao realocar memória para lista de
palíndromos");
                    free(lista_palindromos); // Libera o bloco anterior,
se houver
                    fclose(arq);
                    return 1;
                }
                lista_palindromos = temp;
                capacidade_lista = n;
            }

            // Copia a palavra palíndromo para a lista
            strcpy(lista_palindromos[qtd - 1], palavra);
        }
    }
    fclose(arq);

    printf("Arquivo: %s\n", nome_arquivo);

```

```

if (qtd > 0)
{
    printf("Palíndromos encontrados:\n");
    for (int i = 0; i < qtd; i++)
    {
        printf("- %s\n", lista_palindromos[i]);
    }
}
printf("Foram encontrados %d palíndromos neste arquivo.\n", qtd);

free(lista_palindromos);

return 0;
}

```

8º Uma escola deseja fazer uma competição interclasses com seus alunos. A secretaria da escola montou um arquivo texto com a lista dos alunos interessados em participar da competição. A listagem contém o nome do aluno e um código que indica o turno (manhã ou tarde) e a série (6ª, 7ª ou 8ª), como no exemplo abaixo:

Crie um registro para representar um aluno, leia as informações do arquivo e guarde em um vetor de alunos. Em seguida use laços e testes condicionais para

separar e exibir os alunos agrupados por turno e série, como no exemplo abaixo.

Sugestão: tente também listar os alunos separados apenas por turno.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

typedef struct
{
    char nome[100];
    int turma;
    char turno;
} aluno;

void exibir_alunos(const aluno *lista_alunos, int num_alunos)
{
    char turnos[] = {'M', 'T'};
    int series[] = {6, 7, 8};

    for (int i = 0; i < sizeof(turnos) / sizeof(char); i++)

```

```

{
    char turno_att = turnos[i];
    const char *nome_turno = (turno_att == 'M' ? "Matutino" :
"Vespertino");

    for (int j = 0; j < sizeof(series) / sizeof(int); j++)
    {
        int serie_att = series[j];
        int contador = 0;
        for (int k = 0; k < num_alunos; k++)
        {
            if (lista_alunos[k].turno == turno_att &&
lista_alunos[k].turma == serie_att)
            {
                contador++;
                break;
            }
        }

        if (contador > 0)
        {
            printf("%s %da Serie\n", nome_turno, serie_att);
            printf("-----\n");

            for (int k = 0; k < num_alunos; k++)
            {
                if (lista_alunos[k].turno == turno_att &&
lista_alunos[k].turma == serie_att)
                {
                    printf("%-30s %c%d\n", lista_alunos[k].nome,
lista_alunos[k].turno, lista_alunos[k].turma);
                }
            }
            printf("\n");
        }
    }
}

int main()
{
    char nome_arquivo[100] = "inter.txt";
    aluno *lista_alunos = NULL;
    int n = 0;
    int capacidade_alunos = 0;

    char pri_nome[50];
    char seg_nome[50];

```

```

char codigo_str[10];
char turno_aluno;
int serie_aluno;

FILE *arq = fopen(nome_arquivo, "r");
if (arq == NULL)
{
    perror("Erro ao abrir o arquivo");
    fprintf(stderr, "Arquivo: %s\n", nome_arquivo);
    return 1;
}

while (fscanf(arq, "%49s %49s %9s", pri_nome, seg_nome, codigo_str)
== 3)
{
    // Aumentar a capacidade do vetor de alunos se necessario
    if (n >= capacidade_alunos)
    {
        capacidade_alunos = (capacidade_alunos == 0) ? 10 :
capacidade_alunos * 2;
        aluno *temp = realloc(lista_alunos, capacidade_alunos *
sizeof(aluno));
        if (temp == NULL)
        {
            perror("Erro ao realocar memoria para lista_alunos");
            free(lista_alunos);
            fclose(arq);
            return 1;
        }
        lista_alunos = temp;
    }

    // Montar o nome completo
    snprintf(lista_alunos[n].nome, sizeof(lista_alunos[n].nome), "%s
%s", pri_nome, seg_nome);
    if (strlen(codigo_str) >= 2 && (codigo_str[0] == 'M' ||
codigo_str[0] == 'T'))
    {
        turno_aluno = codigo_str[0];
        serie_aluno = atoi(&codigo_str[1]); // Converte o numero da
serie para int
    }
    else
    {
        fprintf(stderr, "Codigo de turma/turno invalido '%s' para o
aluno '%s'. Pulando.\n", codigo_str, lista_alunos[n].nome);
        continue; // Nao incrementa n
    }
}

```

```

        lista_alunos[n].turno = turno_aluno;
        lista_alunos[n].turma = serie_aluno;
        n++;
    }
    fclose(arq);

    exibir_alunos(lista_alunos, n);

    free(lista_alunos);

    return 0;
}

```

- a) Seria possível separar os alunos por sexo usando os dados fornecidos no programa? Descreva sua solução.

Não é possível com os dados atuais do programa, pois é informado apenas o nome e série, e nomes femininos não qualificam o sexo de um indivíduo.

- b) Se fosse possível adicionar novas informações aos dados, qual seria a forma mais fácil de conseguir fazer essa separação por sexo?

Adicionar um caractere no arquivo txt que permita identificar o gênero do indivíduo e, assim, seja possível fazer o tratamento adequado quanto a gênero:

Anita Torcato	F	M8
Bárbara Borja	F	T6
Estevan processado	M	T7

9º Use o registro peixe definido abaixo.

- a) Construa um programa que leia do usuário os dados de um único peixe e salve-os em um arquivo binário. Cada execução do programa deve acrescentar um peixe ao arquivo.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

struct peixe
{

```

```

    char nome[20];
    unsigned peso;
    float comp;
};

// CREATE: Adiciona um peixe ao final do arquivo binário.
// Retorna 0 em sucesso, -1 em erro.
int adicionar_peixe_arquivo(const char *filename, struct peixe p)
{
    FILE *arq = fopen(filename, "ab");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo para escrita");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    size_t escritos = fwrite(&p, sizeof(struct peixe), 1, arq);
    fclose(arq);

    if (escritos < 1)
    {
        perror("Erro ao escrever peixe no arquivo");
        return -1;
    }
    return 0;
}

// READ: Lê e exibe todos os peixes do arquivo binário.
// Retorna o número de peixes lidos, ou -1 se o arquivo não puder ser
aberto.
int exibir_todos_os_peixes_do_arquivo(const char *filename)
{
    FILE *arq = fopen(filename, "rb");
    if (arq == NULL)
    {
        return -1; // Considera 0 peixes se o arquivo não existir ou não
puder ser aberto.
    }

    struct peixe p_temp;
    int contador = 0;
    printf("\n--- Peixes Cadastrados ---\n");
    while (fread(&p_temp, sizeof(struct peixe), 1, arq) == 1)
    {
        printf("Nome: %-20s | Peso: %5u g | Comprimento: %5.2f cm\n",
            p_temp.nome, p_temp.peso, p_temp.comp);
        contador++;
    }
}

```



```

fclose(arq);

if (contador == 0)
{
    printf("Nenhum peixe cadastrado no arquivo.\n");
}
printf("-----\n");
return contador;
}

// UPDATE: Atualiza os dados de um peixe no arquivo (identificado pelo
nome).
// Retorna 0 se atualizado, 1 se não encontrado, -1 se erro de arquivo.
int atualizar_peixe_no_arquivo(const char *filename, const char
*nome_chave, struct peixe dados_atualizados)
{
    FILE *arq = fopen(filename, "rb+");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo para atualização");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    struct peixe p_temp;
    int encontrado = 0;
    while (fread(&p_temp, sizeof(struct peixe), 1, arq) == 1)
    {
        if (strcmp(p_temp.nome, nome_chave) == 0)
        {
            // Volta para o início do registro atual para sobrescrevê-lo
            if (fseek(arq, -(long)sizeof(struct peixe), SEEK_CUR) != 0)
            {
                perror("Erro ao posicionar para atualização (fseek)");
                fclose(arq);
                return -1;
            }
            // Sobrescreve o registro com os novos dados
            if (fwrite(&dados_atualizados, sizeof(struct peixe), 1, arq)
< 1)
            {
                perror("Erro ao escrever dados atualizados do peixe");
                fclose(arq);
                return -1;
            }
            encontrado = 1;
            break; // Peixe encontrado e atualizado
        }
    }
}

```

```

fclose(arq);
return encontrado ? 0 : 1; // 0 se sucesso, 1 se não encontrado
}

// DELETE: Exclui um peixe do arquivo (identificado pelo nome).
// Retorna 0 se excluído, 1 se não encontrado, -1 se erro.
int excluir_peixe_do_arquivo(const char *filename, const char
*nome_chave)
{
    FILE *arq_original = fopen(filename, "rb");
    if (arq_original == NULL)
    {
        perror("Erro ao abrir o arquivo original para exclusão");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    char temp_filename[256];
    sprintf(temp_filename, "%s_temp.bin", filename);

    FILE *arq_temp = fopen(temp_filename, "wb");
    if (arq_temp == NULL)
    {
        perror("Erro ao criar arquivo temporário para exclusão");
        fprintf(stderr, "Arquivo: %s\n", temp_filename);
        fclose(arq_original);
        return -1;
    }

    struct peixe p_temp;
    int encontrado_e_excluido = 0;
    while (fread(&p_temp, sizeof(struct peixe), 1, arq_original) == 1)
    {
        if (strcmp(p_temp.nome, nome_chave) == 0)
        {
            encontrado_e_excluido = 1; // Marca que o peixe foi
encontrado (e não será copiado)
        }
        else
        {
            if (fwrite(&p_temp, sizeof(struct peixe), 1, arq_temp) < 1)
            {
                perror("Erro ao escrever no arquivo temporário");
                fclose(arq_original);
                fclose(arq_temp);
                remove(temp_filename); // Tenta remover o temporário
                return -1;
            }
        }
    }
}

```

```

    }

    fclose(arq_original);
    fclose(arq_temp);

    if (remove(filename) != 0)
    {
        perror("Erro ao remover o arquivo original");
        fprintf(stderr, "Arquivo: %s\n", filename);
        remove(temp_filename); // Tenta remover o temporário
        return -1;
    }

    if (rename(temp_filename, filename) != 0)
    {
        perror("Erro ao renomear o arquivo temporário");
        fprintf(stderr, "Arquivo: %s para %s\n", temp_filename,
filename);
        remove(temp_filename);
        return -1;
    }

    return encontrado_e_excluido ? 0 : 1; // 0 se excluído com sucesso, 1
se não foi encontrado
}

int main()
{
    setlocale(LC_ALL, "Portuguese");
    char nome_arquivo_bin[100] = "peixes.bin";
    struct peixe novo_peixe;

    printf("\n--- Cadastro de Novo Peixe ---\n");
    printf("Digite o nome do peixe (max 19 caracteres): ");
    if (fgets(novo_peixe.nome, sizeof(novo_peixe.nome), stdin) != NULL)
    {
        novo_peixe.nome[strcspn(novo_peixe.nome, "\n")] = 0;
    }
    else
    {
        fprintf(stderr, "Erro ao ler nome do peixe.\n");
        return 1;
    }

    printf("Digite o peso do peixe (em gramas): ");
    if (scanf("%u", &novo_peixe.peso) != 1)
    {
        fprintf(stderr, "Entrada inválida para peso.\n");
        // Limpar buffer de entrada em caso de erro de scanf
    }
}

```

```

        while (getchar() != '\n')
            ;
        return 1;
    }
    // Limpar o buffer após scanf para o próximo fgets/scanf
    while (getchar() != '\n')
        ;

    printf("Digite o comprimento do peixe (em cm): ");
    if (scanf("%f", &novo_peixe.comp) != 1)
    {
        fprintf(stderr, "Entrada inválida para comprimento.\n");
        while (getchar() != '\n')
            ;
        return 1;
    }
    // Limpar o buffer após scanf
    while (getchar() != '\n')
        ;

    if (adicionar_peixe_arquivo(nome_arquivo_bin, novo_peixe) == 0)
    {
        printf("Peixe '%s' adicionado com sucesso!\n", novo_peixe.nome);
    }
    else
    {
        printf("Falha ao adicionar o peixe '%s'.\n", novo_peixe.nome);
    }

    return 0;
}

```

- b) Modifique o programa obtido no item anterior para ler e exibir os peixes cadastrados no arquivo binário antes de permitir que o usuário faça um novo cadastro.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

struct peixe
{
    char nome[20];
    unsigned peso;
    float comp;
};

```

```

// CREATE: Adiciona um peixe ao final do arquivo binário.
// Retorna 0 em sucesso, -1 em erro.
int adicionar_peixe_arquivo(const char *filename, struct peixe p)
{
    FILE *arq = fopen(filename, "ab");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo para escrita");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    size_t escritos = fwrite(&p, sizeof(struct peixe), 1, arq);
    fclose(arq);

    if (escritos < 1)
    {
        perror("Erro ao escrever peixe no arquivo");
        return -1;
    }
    return 0;
}

// READ: Lê e exibe todos os peixes do arquivo binário.
// Retorna o número de peixes lidos, ou 0 se o arquivo não puder ser
aberto.
int exibir_todos_os_peixes_do_arquivo(const char *filename)
{
    FILE *arq = fopen(filename, "rb");
    if (arq == NULL)
    {
        printf("Nenhum peixe cadastrado no arquivo (ou arquivo
inacessível).\n");
        printf("-----\n");
        return 0; // Retorna 0 peixes lidos
    }

    struct peixe p_temp;
    int contador = 0;
    printf("\n--- Peixes Cadastrados ---\n");
    while (fread(&p_temp, sizeof(struct peixe), 1, arq) == 1)
    {
        printf("Nome: %-20s | Peso: %5u g | Comprimento: %5.2f cm\n",
            p_temp.nome, p_temp.peso, p_temp.comp);
        contador++;
    }
    fclose(arq);
}

```

```

    if (contador == 0)
    {
        printf("Nenhum peixe cadastrado no arquivo.\n");
    }
    printf("-----\n");
    return contador;
}

// UPDATE: Atualiza os dados de um peixe no arquivo (identificado pelo
nome).
// Retorna 0 se atualizado, 1 se não encontrado, -1 se erro de arquivo.
int atualizar_peixe_no_arquivo(const char *filename, const char
*nome_chave, struct peixe dados_atualizados)
{
    FILE *arq = fopen(filename, "rb+");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo para atualização");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    struct peixe p_temp;
    int encontrado = 0;
    while (fread(&p_temp, sizeof(struct peixe), 1, arq) == 1)
    {
        if (strcmp(p_temp.nome, nome_chave) == 0)
        {
            // Volta para o início do registro atual para sobrescrevê-lo
            if (fseek(arq, -(long)sizeof(struct peixe), SEEK_CUR) != 0)
            {
                perror("Erro ao posicionar para atualização (fseek)");
                fclose(arq);
                return -1;
            }
            // Sobrescreve o registro com os novos dados
            if (fwrite(&dados_atualizados, sizeof(struct peixe), 1, arq)
< 1)
            {
                perror("Erro ao escrever dados atualizados do peixe");
                fclose(arq);
                return -1;
            }
            encontrado = 1;
            break; // Peixe encontrado e atualizado
        }
    }
    fclose(arq);
    return encontrado ? 0 : 1; // 0 se sucesso, 1 se não encontrado
}

```

```

}

// DELETE: Exclui um peixe do arquivo (identificado pelo nome).
// Retorna 0 se excluído, 1 se não encontrado, -1 se erro.
int excluir_peixe_do_arquivo(const char *filename, const char
*nome_chave)
{
    FILE *arq_original = fopen(filename, "rb");
    if (arq_original == NULL)
    {
        perror("Erro ao abrir o arquivo original para exclusão");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    char temp_filename[256];
    sprintf(temp_filename, "%s_temp.bin", filename);

    FILE *arq_temp = fopen(temp_filename, "wb");
    if (arq_temp == NULL)
    {
        perror("Erro ao criar arquivo temporário para exclusão");
        fprintf(stderr, "Arquivo: %s\n", temp_filename);
        fclose(arq_original);
        return -1;
    }

    struct peixe p_temp;
    int encontrado_e_excluido = 0;
    while (fread(&p_temp, sizeof(struct peixe), 1, arq_original) == 1)
    {
        if (strcmp(p_temp.nome, nome_chave) == 0)
        {
            encontrado_e_excluido = 1; // Marca que o peixe foi
encontrado (e não será copiado)
        }
        else
        {
            if (fwrite(&p_temp, sizeof(struct peixe), 1, arq_temp) < 1)
            {
                perror("Erro ao escrever no arquivo temporário");
                fclose(arq_original);
                fclose(arq_temp);
                remove(temp_filename); // Tenta remover o temporário
                return -1;
            }
        }
    }
}

```

```

fclose(arq_original);
fclose(arq_temp);

if (remove(filename) != 0)
{
    perror("Erro ao remover o arquivo original");
    fprintf(stderr, "Arquivo: %s\n", filename);
    remove(temp_filename); // Tenta remover o temporário
    return -1;
}

if (rename(temp_filename, filename) != 0)
{
    perror("Erro ao renomear o arquivo temporário");
    fprintf(stderr, "Arquivo: %s para %s\n", temp_filename,
filename);
    remove(temp_filename);
    return -1;
}

return encontrado_e_excluido ? 0 : 1; // 0 se excluído com sucesso, 1
se não foi encontrado
}

int main()
{
    setlocale(LC_ALL, "Portuguese");
    char nome_arquivo_bin[100] = "peixes.bin";
    struct peixe novo_peixe;

    printf("Exibindo peixes ja cadastrados\n");
    exibir_todos_os_peixes_do_arquivo(nome_arquivo_bin);

    printf("\n--- Cadastro de Novo Peixe ---\n");
    printf("Digite o nome do peixe (max 19 caracteres): ");
    if (fgets(novo_peixe.nome, sizeof(novo_peixe.nome), stdin) != NULL)
    {
        novo_peixe.nome[strcspn(novo_peixe.nome, "\n")] = 0;
    }
    else
    {
        fprintf(stderr, "Erro ao ler nome do peixe.\n");
        return 1;
    }

    printf("Digite o peso do peixe (em gramas): ");
    if (scanf("%u", &novo_peixe.peso) != 1)
    {
        fprintf(stderr, "Entrada inválida para peso.\n");
    }
}

```



```

        // Limpar buffer de entrada em caso de erro de scanf
        while (getchar() != '\n')
            ;
        return 1;
    }
    // Limpar o buffer após scanf para o próximo fgets/scanf
    while (getchar() != '\n')
        ;

    printf("Digite o comprimento do peixe (em cm): ");
    if (scanf("%f", &novo_peixe.comp) != 1)
    {
        fprintf(stderr, "Entrada inválida para comprimento.\n");
        while (getchar() != '\n')
            ;
        return 1;
    }
    // Limpar o buffer após scanf
    while (getchar() != '\n')
        ;

    if (adicionar_peixe_arquivo(nome_arquivo_bin, novo_peixe) == 0)
    {
        printf("Peixe '%s' adicionado com sucesso!\n", novo_peixe.nome);
    }
    else
    {
        printf("Falha ao adicionar o peixe '%s'.\n", novo_peixe.nome);
    }

    return 0;
}

```

10º Construa um programa que leia um número desconhecido de peixes de um arquivo texto (semelhante ao arquivo utilizado na primeira questão) e grave estas informações em um arquivo binário que possa ser lido pelo programa da segunda questão. Compare o tamanho dos arquivos texto e binário.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct peixe
{
    char nome[20];

```

```

    unsigned peso;
    float comp;
};

// CREATE: Adiciona um peixe ao final do arquivo binário.
int adicionar_peixe_arquivo(const char *filename, struct peixe p)
{
    FILE *arq = fopen(filename, "ab");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arquivo binário para escrita
(adicionar_peixe_arquivo)");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }

    size_t escritos = fwrite(&p, sizeof(struct peixe), 1, arq);
    fclose(arq);

    if (escritos < 1)
    {
        perror("Erro ao escrever peixe no arquivo binário");
        return -1;
    }
    return 0;
}

// READ: Lê e exibe todos os peixes do arquivo binário.
int exibir_todos_os_peixes_do_arquivo(const char *filename)
{
    FILE *arq = fopen(filename, "rb");
    if (arq == NULL)
    {
        printf("\n--- Peixes Cadastrados (Arquivo Binário) ---\n");
        printf("Nenhum peixe cadastrado no arquivo binário (ou arquivo
inacessível).\n");
        printf("-----\n");
        return 0;
    }

    struct peixe p_temp;
    int contador = 0;
    printf("\n--- Peixes Cadastrados (Arquivo Binário) ---\n");
    while (fread(&p_temp, sizeof(struct peixe), 1, arq) == 1)
    {
        printf("Nome: %-20s | Peso: %5u g | Comprimento: %5.2f cm\n",
            p_temp.nome, p_temp.peso, p_temp.comp);
        contador++;
    }
}

```

```

fclose(arq);

if (contador == 0)
{
    printf("Nenhum peixe cadastrado no arquivo binário.\n");
}
printf("-----\n");
return contador;
}

int write_txt_to_bin(const char *filetxt, const char *filebin)
{
    struct peixe p_temp;
    int peixes_lidos_txt = 0;

    FILE *arq_txt = fopen(filetxt, "r");
    if (arq_txt == NULL)
    {
        perror("Erro ao abrir o arquivo texto para leitura");
        fprintf(stderr, "Arquivo: %s\n", filetxt);
        return -1;
    }

    while (fscanf(arq_txt, " %19s %u %f", p_temp.nome, &p_temp.peso,
&p_temp.comp) == 3)
    {
        if (adicionar_peixe_arquivo(filebin, p_temp) != 0)
        {
            fprintf(stderr, "Erro ao adicionar o peixe '%s' ao arquivo
binário.\n", p_temp.nome);
            fclose(arq_txt);
            return -2;
        }
        peixes_lidos_txt++;
    }

    fclose(arq_txt);

    if (peixes_lidos_txt > 0)
    {
        printf("%d peixe(s) repassado(s) do arquivo texto para o arquivo
binario!\n", peixes_lidos_txt);
    }

    return peixes_lidos_txt;
}

long size_file(const char *filename)
{

```

```

    FILE *arq = fopen(filename, "rb");
    if (arq == NULL)
    {
        return -1;
    }
    fseek(arq, 0, SEEK_END);
    long tamanho = ftell(arq);
    fclose(arq);
    return tamanho;
}

int main()
{
    // REMOVIDO: setlocale(LC_ALL, "Portuguese");
    // O programa agora usará o locale padrão "C".
    // Para acentuação correta no console Windows, execute 'chcp 65001'
    antes de rodar o programa.

    char nome_arquivo_bin[100] = "peixes.bin";
    char nome_arquivo_txt[100] = "peixes.txt";

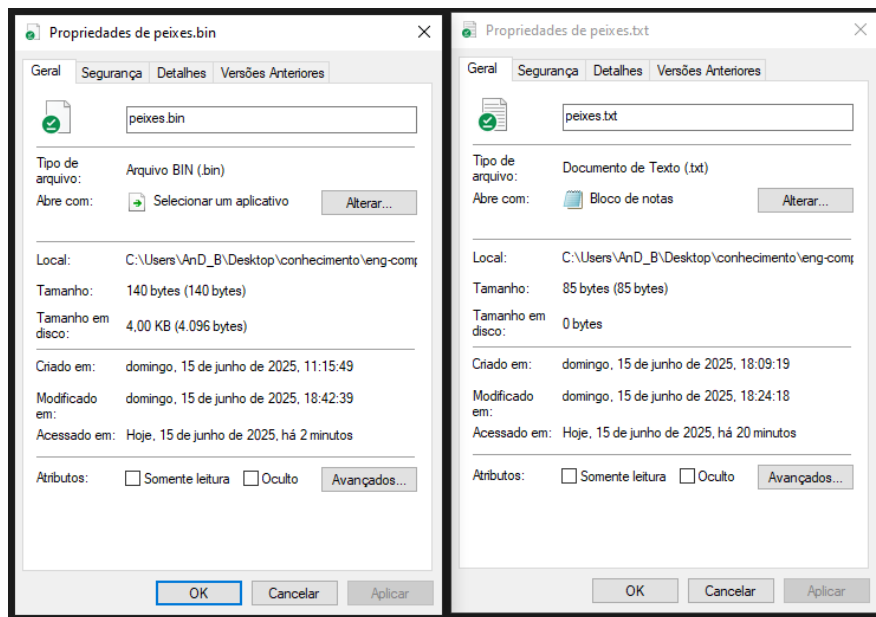
    exibir_todos_os_peixes_do_arquivo(nome_arquivo_bin);
    int num_processados = write_txt_to_bin(nome_arquivo_txt,
nome_arquivo_bin);
    exibir_todos_os_peixes_do_arquivo(nome_arquivo_bin);

    long size_txt = size_file(nome_arquivo_txt);
    long size_bin = size_file(nome_arquivo_bin);

    printf("Tamanho do arquivo texto ('%s'): %ld bytes\n",
nome_arquivo_txt, size_txt);
    printf("Tamanho do arquivo binário ('%s'): %ld bytes\n",
nome_arquivo_bin, size_bin);

    return 0;
}

```



O arquivo binário ocupa mais espaço em disco, pois ao utilizar a struct peixe, ele sempre irá usar a mesma quantidade de bytes para armazenar as variáveis, enquanto o arquivo txt irá armazenar somente a quantidade de caracteres realmente utilizadas.

Se o nome de um peixe é "Pacu" (4 caracteres), ele ainda ocupará 20 bytes no arquivo binário. Os 16 bytes restantes são preenchidos (geralmente com o caractere nulo \0),

No arquivo peixes.txt, cada nome ocupa apenas o número de caracteres que ele realmente tem, mais um espaço delimitador: "Pacu" (4) + espaço (1) = 5 bytes.

De maneira similar ocorrer para os números que são armazenados como sequências de caracteres.

11º Escreva um programa que leia um valor inteiro do usuário e armazene-o em um arquivo binário chamado "integer.bin". Ao entrar, o programa deve verificar se o arquivo "integer.bin" existe, e em caso positivo, deve ler e exibir o número na tela.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");

    const char *filename = "integer.bin";
```

```

int n_arq;
int num;
FILE *arquivo;

arquivo = fopen(filename, "rb");
if (arquivo != NULL)
{
    // fread retorna o número de itens lidos com sucesso.
    if (fread(&n_arq, sizeof(int), 1, arquivo) == 1)
    {
        printf("Arquivo integer.bin encontrado!\n");
        printf("Ele contém o número %d.\n\n", n_arq);
    }
    fclose(arquivo);
}

printf("Digite um inteiro: ");
scanf("%d", &num);

arquivo = fopen(filename, "wb"); // Abre para escrita binária (cria o
arquivo se não existir, ou trunca se existir)
if (arquivo == NULL)
{
    perror("Erro ao abrir o arquivo integer.bin para escrita");
    return 1;
}

// fwrite retorna o número de itens escritos com sucesso.
if (fwrite(&num, sizeof(int), 1, arquivo) == 1)
{
    printf("Número armazenado no arquivo integer.bin.\n");
}

fclose(arquivo);

return 0;
}

```

12º Construa um programa que leia do usuário uma sequência de números inteiros, até que o número zero seja lido. Os números devem ser armazenados

em um arquivo binário chamado “vet.dat”. O primeiro número do arquivo deve ser a quantidade de números digitados. Na entrada do programa verifique se o arquivo existe e, caso positivo, exiba os valores do vetor.

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL, "Portuguese");

    const char *filename = "vet.dat";
    FILE *arq;
    int numeros[100];
    int cont = 0;
    int num;

    arq = fopen(filename, "rb");
    if (arq != NULL)
    {
        int qtd;
        if (fread(&qtd, sizeof(int), 1, arq) == 1) // já avança o
ponteiro do arq binário
        {
            if (qtd > 0)
            {
                printf("O arq vet.dat contém:\n");

                int numero_lido;
                for (int i = 0; i < qtd; i++)
                {
                    if (fread(&numero_lido, sizeof(int), 1, arq) == 1) //
ponteiro já leu o inteiro responsável pela quantidade de elementos
                    {
                        printf("%d ", numero_lido);
                    }
                    else
                    {
                        fprintf(stderr, "Erro ao ler dado do arq %s
existente.\n", filename);
                        break;
                    }
                }
                printf("\n");
            }
        }
        fclose(arq);
    }

    // Se arq == NULL
    printf("Digite números (zero para encerrar):\n");
    while (cont < 100)
    {

```

```

        if (scanf("%d", &num) != 1) // concatenado para tratamento de
entrada -> mais otimizado que as versões anteriores
        {
            printf("Entrada inválida. Por favor, digite um número
inteiro.\n");
            while (getchar() != '\n')
                ;
            continue;
        }

        if (num == 0)
        {
            break;
        }

        numeros[cont] = num;
        cont++;
    }

    arq = fopen(filename, "wb");
    if (arq == NULL)
    {
        perror("Erro ao abrir o arq vet.dat para escrita");
        return 1;
    }

    // Escreve a quantidade de números (pode ser 0 se nenhum número
válido foi digitado)
    if (fwrite(&cont, sizeof(int), 1, arq) != 1) // já avança o ponteiro
do arq binário
    {
        fprintf(stderr, "Erro ao escrever a quantidade no arq %s.\n",
filename);
        fclose(arq);
        return 1;
    }

    for (int i = 0; i < cont; i++)
    {
        if (fwrite(&numeros[i], sizeof(int), 1, arq) != 1)
        {
            fprintf(stderr, "Erro ao escrever o número %d no arq %s.\n",
i + 1, filename);
            fclose(arq);
            return 1;
        }
    }

    printf("%d números foram armazenados em vet.dat.\n", cont);

```



```
fclose(arq);

return 0;
}
```

13º O Exercício de Revisão 1 do Laboratório 6 (Tipos Inteiros) mostra como usar códigos de Escape para mudar a cor do texto exibido em um terminal. A sequência de caracteres "\033[38;5;000;48;5;154m" define a cor 000 para o texto e 154 para o fundo. Considerando que as cores são valores de 3 dígitos de 000 a 255, construa um programa para ler do usuário um texto com no máximo 80 caracteres, um código para a cor do texto e um código para a cor do fundo. Guarde essas informações em um arquivo binário. O usuário deve ter a opção de guardar uma nova frase ou exibir a frase já armazenada através de um menu, como mostrado no exemplo abaixo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

#define MAX_TEXTO 81
#define NOME_ARQUIVO "mensagem_colorida.dat"

struct MensagemColorida
{
    char texto[MAX_TEXTO];
    int cor_texto;
    int cor_fundo;
};

void limpar_buffer()
{
    int c;
    while ((c = getchar()) != '\n' && c != EOF)
        ;
}

void armazenar_mensagem()
{
    struct MensagemColorida msg;
    FILE *arquivo;

    printf("Sua mensagem: ");
```

```

if (fgets(msg.texto, MAX_TEXTO, stdin) == NULL)
{
    fprintf(stderr, "Erro ao ler a mensagem.\n");
    return;
}
msg.texto[strcspn(msg.texto, "\n")] = 0;

printf("Cor do texto: ");
if (scanf("%d", &msg.cor_texto) != 1)
{
    fprintf(stderr, "Cor do texto inválida. Deve ser entre 000 e
255.\n");
    limpar_buffer(); // Limpa o buffer em caso de entrada inválida
    return;
}
limpar_buffer(); // Limpa o '\n' deixado por scanf

printf("Cor do fundo: ");
if (scanf("%d", &msg.cor_fundo) != 1)
{
    fprintf(stderr, "Cor do fundo inválida. Deve ser entre 000 e
255.\n");
    limpar_buffer();
    return;
}
limpar_buffer();

arquivo = fopen(NOME_ARQUIVO, "wb"); // Abre para escrita binária
(sobrescreve)
if (arquivo == NULL)
{
    perror("Erro ao abrir o arquivo para armazenamento");
    return;
}

if (fwrite(&msg, sizeof(struct MensagemColorida), 1, arquivo) != 1)
{
    fprintf(stderr, "Erro ao escrever a mensagem no arquivo.\n");
}
else
{
    printf("Texto colorido foi armazenado.\n");
}

fclose(arquivo);
}

// Função para exibir a mensagem do arquivo
void exibir_mensagem()

```

```

{
    struct MensagemColorida msg;
    FILE *arquivo;

    arquivo = fopen(NOME_ARQUIVO, "rb");
    if (arquivo == NULL)
    {
        printf("Nenhuma mensagem armazenada ainda.\n");
        perror("Erro ao abrir o arquivo para exibição");
        return;
    }

    if (fread(&msg, sizeof(struct MensagemColorida), 1, arquivo) != 1)
    {
        printf("Nenhuma mensagem válida encontrada no arquivo ou erro de
leitura.\n");
    }
    else
    {
        // Monta a string de escape ANSI
        // \033[38;5;{COR_TEXTO}m para cor do texto
        // \033[48;5;{COR_FUNDO}m para cor do fundo
        // \033[0m para resetar as cores
        printf("\nExibindo mensagem armazenada:\n");
        printf("\033[38;5;%03dm\033[48;5;%03dm%s\033[0m\n",
msg.cor_texto, msg.cor_fundo, msg.texto);
    }

    fclose(arquivo);
}

int main()
{
    setlocale(LC_ALL, "Portuguese");

    char opc;
    do
    {
        printf("\nMensagens Coloridas\n");
        printf("-----\n");
        printf("[A]rmazenar\n");
        printf("[E]xibir\n");
        printf("[S]air\n");
        printf("-----\n");
        printf("Opção: ");

        if (scanf(" %c", &opc) != 1)
        { // Espaço antes de %c para consumir newlines pendentes
            fprintf(stderr, "Erro ao ler a opção.\n");
        }
    }
    while (opc != 'A' & opc != 'E' & opc != 'S');
}

```

```

        limpar_buffer();
        opc = ' '; // Força a repetição do loop
        continue;
    }
    limpar_buffer();

    opc = toupper(opc);
    switch (opc)
    {
        case 'A':
            armazenar_mensagem();
            break;
        case 'E':
            exibir_mensagem();
            break;
        case 'S':
            break;
        default:
            printf("Opção inválida. Tente novamente.\n");
    }
} while (opc != 'S');

return 0;
}

```

14º No exercício anterior, se o usuário digitar apenas espaços para a mensagem, o resultado será uma faixa colorida com a cor de fundo. Podemos usar isso para criar uma imagem formada apenas por caracteres de espaço coloridos.

Construa um programa que peça a largura e altura da imagem e leia do usuário uma matriz de números. Cada número de 3 dígitos representa a cor de um “bloco” da imagem. Guarde em um arquivo binário os valores de altura, largura e de cada bloco da imagem. Assim como o programa anterior, construa um menu para controlar as opções de armazenamento e exibição da imagem.

```

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <ctype.h>

#define ARQUIVO_IMAGEM "blocos.dat"
#define MAX_DIM 50

struct Imagem
{

```

```

    int largura;
    int altura;
};

void limpar_buffer()
{
    int c;
    while ((c = getchar()) != '\n' && c != EOF)
        ;
}

void armazenar_imagem()
{
    struct Imagem img;
    int matriz_cores[MAX_DIM][MAX_DIM]; // Matriz para armazenar as cores
    temporariamente
    FILE *arquivo;

    printf("Digite a largura da imagem (1-%d): ", MAX_DIM);
    if (scanf("%d", &img.largura) != 1)
    {
        fprintf(stderr, "Largura inválida.\n");
        limpar_buffer();
        return;
    }
    limpar_buffer();

    printf("Digite a altura da imagem (1-%d): ", MAX_DIM);
    if (scanf("%d", &img.altura) != 1)
    {
        fprintf(stderr, "Altura inválida.\n");
        limpar_buffer();
        return;
    }
    limpar_buffer();

    printf("Digite os códigos de cor (0-255) para cada linha:\n");
    for (int i = 0; i < img.altura; i++)
    {
        printf("#%d (%d cores): ", i + 1, img.largura);
        for (int j = 0; j < img.largura; j++)
        {
            if (scanf("%d", &matriz_cores[i][j]) != 1)
            {
                fprintf(stderr, "\nCor inválida na linha %d, coluna
%d.\n", i + 1, j + 1);
                limpar_buffer(); // Limpa o resto da linha inválida
                i--;              // Repete a leitura para a linha inteira
            }
        }
    }
}

```

```

        break; // Sai do loop interno (colunas) para
repetir a linha
    }
}
if (i < img.altura - 1 && i >= 0)
    limpar_buffer(); // Limpa o '\n' no final da linha de scanf,
exceto para a última linha ou se i foi decrementado
}
// Se a última entrada foi válida, o último \n pode precisar ser
limpo antes de futuras operações de menu
// No entanto, o " %" no scanf do menu principal geralmente lida com
isso.

arquivo = fopen(ARQUIVO_IMAGEM, "wb");
if (arquivo == NULL)
{
    perror("Erro ao abrir o arquivo para armazenamento da imagem");
    return;
}

if (fwrite(&img, sizeof(struct Imagem), 1, arquivo) != 1)
{
    fprintf(stderr, "Erro ao escrever o cabeçalho da imagem no
arquivo.\n");
    fclose(arquivo);
    return;
}

for (int i = 0; i < img.altura; i++)
{
    for (int j = 0; j < img.largura; j++)
    {
        if (fwrite(&matriz_cores[i][j], sizeof(int), 1, arquivo) !=
1)
        {
            fprintf(stderr, "Erro ao escrever o bloco de cor (%d,%d)
no arquivo.\n", i, j);
            fclose(arquivo);
            return;
        }
    }
}

printf("Imagem de blocos coloridos foi armazenada.\n");
fclose(arquivo);
}

void exibir_imagem()
{

```

```

    struct Imagem img;
    int cor_bloco;
    FILE *arquivo;

    // A função exibir_imagem processa os dados da imagem "em fluxo"
    (streaming).
    // Ela lê as dimensões (cabeçalho) e, em seguida, lê a cor de cada
    bloco
    // individualmente do arquivo e a exibe imediatamente.
    // Isso evita a necessidade de carregar toda a matriz de cores para a
    memória
    // de uma vez, tornando a função eficiente em termos de uso de
    memória,
    // especialmente para imagens grandes. Apenas a cor do bloco atual é
    // mantida na memória durante a exibição.

    arquivo = fopen(ARQUIVO_IMAGEM, "rb");
    if (arquivo == NULL)
    {
        printf("Nenhuma imagem armazenada ainda ou arquivo não
    encontrado.\n");
        return;
    }

    if (fread(&img, sizeof(struct Imagem), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao ler o cabeçalho da imagem do arquivo ou
    arquivo inválido.\n");
        fclose(arquivo);
        return;
    }

    if (img.largura <= 0)
    {
        fprintf(stderr, "Dimensões da imagem inválidas encontradas no
    arquivo.\n");
        fclose(arquivo);
        return;
    }

    printf("\nExibindo imagem armazenada (%dx%d):\n", img.largura,
    img.altura);
    for (int i = 0; i < img.altura; i++)
    {
        for (int j = 0; j < img.largura; j++)
        {
            if (fread(&cor_bloco, sizeof(int), 1, arquivo) != 1)
            {

```

```

        fprintf(stderr, "\nErro ao ler o bloco de cor (%d,%d) do
arquivo.\n", i + 1, j + 1);
        printf("\033[0m"); // Reseta cores
        fclose(arquivo);
        return;
    }
    // Imprime um espaço com a cor de fundo definida pela
cor_bloco
    printf("\033[48;5;%03dm \033[0m", cor_bloco); // Um espaço
para cada bloco
    }
    printf("\n");
}
printf("\033[0m"); // Garante que as cores sejam resetadas no final

fclose(arquivo);
}

int main()
{
    setlocale(LC_ALL, "Portuguese");

    char opc;
    do
    {
        printf("\nMensagens Coloridas\n");
        printf("-----\n");
        printf("[A]rmazenar\n");
        printf("[E]xibir\n");
        printf("[S]air\n");
        printf("-----\n");
        printf("Opção: ");

        if (scanf(" %c", &opc) != 1)
        { // Espaço antes de %c para consumir newlines pendentes
            fprintf(stderr, "Erro ao ler a opção.\n");
            limpar_buffer();
            opc = ' '; // Força a repetição do loop
            continue;
        }
        limpar_buffer();

        opc = toupper(opc);
        switch (opc)
        {
            case 'A':
                armazenar_imagem();
                break;
            case 'E':

```



```

        exibir_imagem();
        break;
    case 'S':
        break;
    default:
        printf("Opção inválida. Tente novamente.\n");
    }
} while (opc != 'S');

return 0;
}

```

15º Crie um programa que grave os 100 primeiros números naturais ao mesmo tempo em um arquivo texto e em um arquivo binário. Compare os tamanhos dos arquivos e explique como os tipos escolhidos para guardar os números podem tornar o arquivo binário menor ou maior que o arquivo texto.

```

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

// Função para obter o tamanho de um arquivo em bytes
long size_file(const char *filename)
{
    FILE *f = fopen(filename, "rb"); // Abrir em modo binário para obter
    o tamanho corretamente
    if (f == NULL)
    {
        perror("Erro ao abrir arquivo para obter tamanho");
        fprintf(stderr, "Arquivo: %s\n", filename);
        return -1;
    }
    fseek(f, 0, SEEK_END); // Vai para o final do arquivo
    long tamanho = ftell(f); // Pega a posição atual (tamanho)
    fclose(f);
    return tamanho;
}

int main()
{
    setlocale(LC_ALL, "Portuguese");

    const char *arquivo_texto_nome = "naturais.txt";
    const char *arquivo_binario_nome = "naturais.bin";

    FILE *arq_texto;

```

```

FILE *arq_binario;
int i;
int n_bin;

arq_texto = fopen(arquivo_texto_nome, "w");
if (arq_texto == NULL)
{
    perror("Erro ao criar/abrir o arquivo texto para escrita");
    return 1;
}

arq_binario = fopen(arquivo_binario_nome, "wb");
if (arq_binario == NULL)
{
    perror("Erro ao criar/abrir o arquivo binário para escrita");
    fclose(arq_texto);
    return 1;
}

for (i = 1; i <= 100; i++)
{
    if (fprintf(arq_texto, "%d\n", i) < 0)
    { // se for bem sucedido, o num tot de caracteres sempre será
maior que 0 e não entrará no if
        fprintf(stderr, "Erro ao escrever no arquivo texto.\n");
        fclose(arq_texto);
        fclose(arq_binario);
        return 1;
    }

    n_bin = i; // Atribui o valor de i
    if (fwrite(&n_bin, sizeof(int), 1, arq_binario) != 1)
    { // se for bem sucedido retornará 1 e não entrará no if
        fprintf(stderr, "Erro ao escrever no arquivo binário.\n");
        fclose(arq_texto);
        fclose(arq_binario);
        return 1;
    }
}

printf("Gravação concluída.\n\n");

fclose(arq_texto);
fclose(arq_binario);

// Comparar os tamanhos dos arquivos
long tamanho_texto = size_file(arquivo_texto_nome);
long tamanho_binario = size_file(arquivo_binario_nome);

```

```

printf("Tamanho do arquivo texto: %ld bytes\n", tamanho_texto);
printf("Tamanho do arquivo binário: %ld bytes\n", tamanho_binario);

return 0;
}

```

O motivo é que o arquivo de texto irá armazenar os números como sequências de caracteres, em que cada caractere ocupa um byte. Já arquivos binários irão salvar o número com a exata quantidade de bytes a qual o tipo pertence, se o número for salvo por tipos que usem menos bytes, o arquivo irá ser mais leve.

16º Crie um registro Soldado com os campos nome, eliminações, mortes, taxa de eliminação por morte (eliminações/mortes) e número de partidas jogadas. Na função principal leia um soldado de um arquivo binário. Se o arquivo não Para existir, você deve direcionar o usuário para a criação de um novo soldado. Quando o soldado for lido (seja pelo arquivo ou pelo teclado), você deve dar as seguintes opções ao usuário:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <locale.h>

#define MAX_NOME 51
#define ARQUIVO "soldado.dat"

struct Soldado
{
    char nome[MAX_NOME];
    int eliminacoes;
    int mortes;
    float taxa_em;
    int partidas_jogadas;
};

void limpar_buffer()
{
    int c;
    while ((c = getchar()) != '\n' && c != EOF)
        ;
}

// Função para criar um novo soldado (CREATE)
struct Soldado create_s()

```

```

{
    struct Soldado novo_s;

    printf("\n--- Criar Novo Soldado ---\n");
    printf("Nome do Soldado (max %d caracteres): ", MAX_NOME - 1);
    if (fgets(novo_s.nome, MAX_NOME, stdin) == NULL)
    {
        fprintf(stderr, "Erro ao ler o nome. Saindo da criação.\n");
        strcpy(novo_s.nome, "INDEFINIDO");
        novo_s.eliminacoes = 0;
        novo_s.mortes = 0;
        novo_s.partidas_jogadas = 0;
        novo_s.taxa_em = 0;
        return novo_s;
    }

    novo_s.nome[strcspn(novo_s.nome, "\n")] = 0; // Remove newline
    if (strlen(novo_s.nome) == 0)
    {
        fprintf(stderr, "Nome não pode ser vazio. Usando 'Soldado Sem Nome'.\n");
        strcpy(novo_s.nome, "Soldado Sem Nome");
    }

    printf("Eliminações: ");
    if (scanf("%d", &novo_s.eliminacoes) != 1 || novo_s.eliminacoes < 0)
    {
        fprintf(stderr, "Entrada inválida para eliminações. Usando 0.\n");
        novo_s.eliminacoes = 0;
    }
    limpar_buffer();

    printf("Mortes: ");
    if (scanf("%d", &novo_s.mortes) != 1 || novo_s.mortes < 0)
    {
        fprintf(stderr, "Entrada inválida para mortes. Usando 0.\n");
        novo_s.mortes = 0;
    }
    limpar_buffer();

    printf("Partidas Jogadas: ");
    if (scanf("%d", &novo_s.partidas_jogadas) != 1 ||
novo_s.partidas_jogadas < 0)
    {
        fprintf(stderr, "Entrada inválida para partidas jogadas. Usando 0.\n");
        novo_s.partidas_jogadas = 0;
    }
}

```

```

limpar_buffer();

novo_s.taxa_em = (novo_s.mortes > 0) ?
(float)((float)novo_s.eliminacoes / (float)novo_s.mortes) :
(float)novo_s.eliminacoes;
printf("Novo soldado '%s' pronto para ser salvo.\n", novo_s.nome);
return novo_s;
}

void view_s(struct Soldado s)
{
    printf("\n--- Estatísticas do Soldado ---\n");
    printf("Nome: %s\n", s.nome);
    printf("Eliminações: %d\n", s.eliminacoes);
    printf("Mortes: %d\n", s.mortes);
    printf("Taxa E/M: %.2f\n", s.taxa_em);
    printf("Partidas Jogadas: %d\n", s.partidas_jogadas);
    printf("-----\n");
}

// Função para atualizar um soldado existente (UPDATE)
struct Soldado update_s(struct Soldado s_atual)
{
    struct Soldado s_aux = s_atual;
    char buffer_nome[MAX_NOME + 2];

    printf("\n--- Atualizar Soldado: %s ---\n", s_atual.nome);

    printf("Novo Nome (atual: %s, deixe em branco e pressione Enter para manter): ", s_atual.nome);
    if (fgets(buffer_nome, sizeof(buffer_nome), stdin) == NULL)
    {
        fprintf(stderr, "Erro ao ler o novo nome. Nome não alterado.\n");
    }
    else
    {
        buffer_nome[strcspn(buffer_nome, "\n")] = 0; // Remove newline

        if (strlen(buffer_nome) > 0)
        { // Se algo foi digitado
            strcpy(s_aux.nome, buffer_nome);
        }
    }

    printf("Eliminações (atual: %d): ", s_atual.eliminacoes);
    if (scanf("%d", &s_aux.eliminacoes) != 1 || s_aux.eliminacoes < 0)
    {
        fprintf(stderr, "Entrada inválida. Mantendo valor atual.\n");
        s_aux.eliminacoes = s_atual.eliminacoes;
    }
}

```

```

    }
    limpar_buffer();

    printf("Mortes (atual: %d): ", s_atual.mortes);
    if (scanf("%d", &s_aux.mortes) != 1 || s_aux.mortes < 0)
    {
        fprintf(stderr, "Entrada inválida. Mantendo valor atual.\n");
        s_aux.mortes = s_atual.mortes;
    }
    limpar_buffer();

    printf("Partidas (atual: %d): ", s_atual.partidas_jogadas);
    if (scanf("%d", &s_aux.partidas_jogadas) != 1 ||
s_aux.partidas_jogadas < 0)
    {
        fprintf(stderr, "Entrada inválida. Mantendo valor atual.\n");
        s_aux.partidas_jogadas = s_atual.partidas_jogadas;
    }
    limpar_buffer();

    s_aux.taxa_em = (s_aux.mortes > 0) ? (float)((float)s_aux.eliminacoes
/ (float)s_aux.mortes) : (float)s_aux.eliminacoes;
    printf("Soldado '%s' atualizado.\n", s_aux.nome);
    return s_aux;
}

// Função para gravar os dados de um soldado no arquivo binário
void write_s(struct Soldado s, const char *nome_arquivo)
{
    FILE *arquivo = fopen(nome_arquivo, "wb");
    if (arquivo == NULL)
    {
        perror("Erro ao abrir arquivo para gravação");
        return;
    }
    if (fwrite(&s, sizeof(struct Soldado), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao gravar dados do soldado no arquivo.\n");
    }
    fclose(arquivo);
}

// Retorna 1 se sucesso, 0 se arquivo não existe ou erro de leitura
int read_s(struct Soldado *s, const char *nome_arquivo)
{
    FILE *arquivo = fopen(nome_arquivo, "rb");
    if (arquivo == NULL)
    {
        return 0;
    }

```

```

    }
    if (fread(s, sizeof(struct Soldado), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao ler dados do soldado do arquivo ou
arquivo corrompido.\n");
        fclose(arquivo);
        return 0;
    }
    fclose(arquivo);
    return 1;
}

int main()
{
    struct Soldado soldado_aux;
    int soldado_existe = 0;
    char opc;

    if (read_s(&soldado_aux, ARQUIVO))
    {
        printf("Dados do soldado '%s' carregados do arquivo.\n",
soldado_aux.nome);
        soldado_existe = 1;
    }
    else
    {
        printf("Arquivo '%s' não encontrado ou vazio.\n", ARQUIVO);
        printf("Por favor, crie um novo soldado.\n");
        soldado_aux = create_s();
        write_s(soldado_aux, ARQUIVO);
        soldado_existe = 1;
    }

    do
    {
        printf("\n--- Menu Soldado ---\n");
        printf("[N] Novo Soldado\n");
        printf("[A] Atualizar Soldado\n");
        printf("[E] Exibir Soldado\n");
        printf("[S] Sair\n");
        printf("-----\n");
        printf("Opção: ");

        if (scanf(" %c", &opc) != 1)
        {
            fprintf(stderr, "Erro ao ler a opção.\n");
            limpar_buffer();
            opc = ' ';
            continue;
        }
    } while (opc != 'S');
}

```

```

    }
    limpar_buffer();
    opc = toupper(opc);

    switch (opc)
    {
        case 'N':
            soldado_aux = create_s(soldado_aux, 1);
            write_s(soldado_aux, ARQUIVO);
            soldado_existe = 1;
            break;
        case 'A':
            if (!soldado_existe)
            {
                printf("Nenhum soldado carregado para atualizar. Crie um
novo primeiro (Opção N).\n");
            }
            else
            {
                soldado_aux = update_s(soldado_aux);
                write_s(soldado_aux, ARQUIVO);
            }
            break;
        case 'E':
            if (!soldado_existe)
            {
                printf("Nenhum soldado carregado para exibir. Crie um
novo primeiro (Opção N).\n");
            }
            else
            {
                view_s(soldado_aux);
            }
            break;
        case 'S':
            break; // sai do programa
        default:
            printf("Opção inválida.\n");
    }
} while (opc != 'S');

return 0;
}

```


17º Crie uma união chamada Senha com os campos Alfanumérica e Numérica. Na função principal pergunte ao usuário se ele quer exibir a senha armazenada ou gravar uma nova senha no arquivo. Se o arquivo não tiver sido criado ainda, a opção de exibição não deve fazer nada, a não ser avisar o usuário que o arquivo não existe.

Use um arquivo binário para guardar a senha. Na gravação da senha, grave primeiro um número para representar o tipo da senha. O modo alfanumérico será correspondente ao número 1, e o modo simplesmente numérico será correspondente ao 2. Quando o usuário abrir o programa e selecionar a opção de exibição, o tipo da senha deve ser lido para decidir qual campo da união apresentar na tela (campo alfanumérico ou o campo numérico).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <locale.h>

#define MAX_SENHA 51
#define ALFA 1
#define NUMERICA 2
#define FILE_SENHA "senha_segura.dat"

union Senha
{
    char alfanumerica[MAX_SENHA];
    int numerica;
};

void limpar_buffer()
{
    int c;
    while ((c = getchar()) != '\n' && c != EOF)
        ;
}

void write_senha()
{
    union Senha minha_senha;
    int tipo_senha;
    FILE *arquivo;

    printf("\n--- Gravar Nova Senha ---\n");
    printf("Qual tipo de senha deseja criar?\n");
    printf("1. Alfanumérica\n");
```

```

printf("2. Numérica\n");
printf("Opção: ");

if (scanf("%d", &tipo_senha) != 1 || (tipo_senha != ALFA &&
tipo_senha != NUMERICA)) // versão melhorada garanti opc 1 e 2
{
    fprintf(stderr, "Opção de tipo inválida.\n");
    limpar_buffer();
    return;
}
limpar_buffer();

if (tipo_senha == ALFA)
{
    printf("Digite a senha alfanumérica (max %d caracteres): ",
MAX_SENHA - 1);
    if (fgets(minha_senha.alfanumerica, MAX_SENHA, stdin) == NULL)
    {
        fprintf(stderr, "Erro ao ler a senha alfanumérica.\n");
        return;
    }
    minha_senha.alfanumerica[strcspn(minha_senha.alfanumerica, "\n")]
= 0;
}
else
{ // tipo_senha == NUMERICA
    printf("Digite a senha numérica: ");
    if (scanf("%d", &minha_senha.numerica) != 1)
    {
        fprintf(stderr, "Entrada inválida para senha numérica.\n");
        limpar_buffer();
        return;
    }
    limpar_buffer();
}

arquivo = fopen(FILE_SENHA, "wb");
if (arquivo == NULL)
{
    perror("Erro ao abrir o arquivo para gravação");
    return;
}

if (fwrite(&tipo_senha, sizeof(int), 1, arquivo) != 1)
{
    fprintf(stderr, "Erro ao gravar o tipo da senha no arquivo.\n");
    fclose(arquivo);
    return;
}

```

```

    if (fwrite(&minha_senha, sizeof(union Senha), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao gravar os dados da senha no
arquivo.\n");
        fclose(arquivo);
        return;
    }

    printf("Senha gravada com sucesso no arquivo '%s'.\n", FILE_SENHA);
    fclose(arquivo);
}

void exibir()
{
    union Senha senha_lida;
    int tipo_senha;
    FILE *arquivo;

    arquivo = fopen(FILE_SENHA, "rb");
    if (arquivo == NULL)
    {
        printf("Arquivo de senha não encontrado ou não pôde ser
aberto.\n");
        return;
    }

    if (fread(&tipo_senha, sizeof(int), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao ler o tipo da senha do arquivo ou
arquivo corrompido/vazio.\n");
        fclose(arquivo);
        return;
    }

    if (fread(&senha_lida, sizeof(union Senha), 1, arquivo) != 1)
    {
        fprintf(stderr, "Erro ao ler os dados da senha do arquivo ou
arquivo corrompido.\n");
        fclose(arquivo);
        return;
    }

    printf("Senha armazenada:\n");
    if (tipo_senha == ALFA)
    {
        printf("Tipo: Alfanumérica\n");
        printf("Senha: %s\n", senha_lida.alfanumerica);
    }
}

```

```

else if (tipo_senha == NUMERICA)
{
    printf("Tipo: Numérica\n");
    printf("Senha: %d\n", senha_lida.numerica);
}

fclose(arquivo);
}

int main()
{
    setlocale(LC_ALL, "Portuguese");

    char opc;

    do
    {
        printf("\n--- Gerenciador de Senhas ---\n");
        printf("[G]ravar nova senha\n");
        printf("[E]xibir senha armazenada\n");
        printf("[S]air\n");
        printf("-----\n");
        printf("Opção: ");

        if (scanf(" %c", &opc) != 1)
        { // Espaço antes de %c para consumir newlines pendentes
            fprintf(stderr, "Erro ao ler a opção do menu.\n");
            limpar_buffer();
            opc = ' ';
            continue;
        }
        limpar_buffer();

        opc = toupper(opc);

        switch (opc)
        {
            case 'G':
                write_senha();
                break;
            case 'E':
                exibir();
                break;
            case 'S':
                printf("Saindo do programa...\n");
                break;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }
    }

```

```
} while (opc != 'S');  
  
return 0;  
}
```