



UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO

Fernanda Vieira Campos Andrade,
Winnie Barros,
Ana Clara Sesana Moreira

Trabalho Prático - Estrutura De Dados I

SÃO MATEUS – ES
2025

Sumário

1. Introdução.....	3
2. Chocolate.....	3
2.1- Inclusão de bibliotecas no .h:.....	3
2.2. Estrutura.....	3
2.3. Função chocoNode* alocaChoco.....	3
2.4. chocoNode* leChoco();.....	4
2.5. int comparaChoco.....	5
2.6. void imprimeChoco.....	5
2.7. void freeChoco.....	6
3. Filme.....	6
3.1. Inclusão de bibliotecas:.....	6
3.2. Estrutura:.....	6
3.3 FilmeNode* alocaFilme.....	7
3.4 FilmeNode* leFilme.....	8
3.5. int comparaFilme.....	8
3.6. void imprimeFilme.....	9
3.7 void freeFilme.....	9
4 . Livro.....	9
4.1 Inclusão de bibliotecas:.....	9
4.2 Estrutura:.....	10
4.3 LivroNode* alocaLivro:.....	10
4.4 LivroNode* LeLivro:.....	11
4.5 int comparaLivro:.....	11
4.6 void imprimeLivro:.....	12
4.7 freeLivro:.....	12
5. Vinho.....	13
5.1 Inclusão de bibliotecas:.....	13
5.2. Estrutura.....	13
5.3 VinhoNode* alocaVinho.....	13
5.4 VinhoNode* leVinho.....	14
5.5 int comparaVinho.....	14
5.6 void imprimeVinho.....	15
5.7 void freeVinho.....	15
6. Produto.....	16
6.1: Inclusão de bibliotecas:.....	16
6.2: Estrutura:.....	16
6.3 Função Produto* alocaProduto:.....	16
6.4: Função void printProduto:.....	17
6.5 Função int comparaProduto:.....	17
7. Lista.....	17
8. Main.....	27
9. Makefile.....	35
10. Casos Teste:.....	36

1. Introdução

O relatório detalha as principais funções do nosso programa e realiza uma série de teste para verificar o seu funcionamento.

2. Chocolate

Para a categoria chocolate, criamos uma biblioteca chamada chocolate.h e um arquivo chocolate.c.

2.1- Inclusão de bibliotecas no .h:

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

2.2. Estrutura

```
typedef struct Chocolate{
    char* nome;
    char* marca;
    char* tipo;
    char* porcentagem;
    char* origem;
    char* peso;
    char* ano_de_fabricacao;
    char* validade;
}chocoNode;
```

char* nome = armazena uma string para o nome do chocolate;
char* marca = armazena uma string para a marca do chocolate;
char* tipo = armazena uma string para o tipo do chocolate;
char* porcentagem = armazena uma string para a porcentagem de cacau do chocolate;
char* origem = armazena uma string para a origem do chocolate;
char* peso = armazena uma string para o peso do chocolate;
char* ano_de_fabricacao = armazena uma string para o ano de fabricação do chocolate;
char* validade = armazena uma string para a validade do chocolate;

2.3. Função chocoNode* alocaChoco

Função responsável por alocar uma estrutura que corresponde a um chocolate para ser inserido. A função recebe os parâmetros e realiza a alocação dinâmica, fazendo um calloc

com tamanho da estrutura e das variáveis do tipo char*, e atribui os dados fornecidos aos campos.

Parâmetros:

- char* nome = armazena uma string para o nome do chocolate;
- char* marca = armazena uma string para a marca do chocolate;
- char* tipo = armazena uma string para o tipo do chocolate;
- char* porcentagem = armazena uma string para a porcentagem de cacau do chocolate;
- char* origem = armazena uma string para a origem do chocolate;
- char* peso = armazena uma string para o peso do chocolate;
- char* ano_de_fabricacao = armazena uma string para o ano de fabricação do chocolate;
- char* validade = armazena uma string para a validade do chocolate;

Retorno: Retorna a estrutura chocoNode com os dados fornecidos.

2.4. chocoNode* leChoco();

```
chocoNode* leChoco(){
    char nome[MAX_TAM];
    char marca[MAX_TAM];
    char tipo[MAX_TAM];
    char porcentagem[MAX_TAM];
    char origem[MAX_TAM];
    char peso[MAX_TAM];
    char ano_de_fabricacao[MAX_TAM];
    char validade[MAX_TAM];

    getchar(); // Limpa qualquer '\n' no buffer antes da primeira entrada

    printf("NOME: ");
    scanf("%[^\n]%*c", nome);
    printf("\nMARCA: ");
    scanf("%[^\n]%*c", marca);
    printf("\nTIPO DE CHOCOLATE: ");
    scanf("%[^\n]%*c", tipo);

    if(strcmp(tipo, "amargo") == 0){
        printf("\nPORCENTAGEM DE CACAU: ");
        scanf("%[^\n]%*c", porcentagem);
    }
    else strcpy(porcentagem, "-");
    printf("\nNACIONALIDADE OU ORIGEM: ");
    scanf("%[^\n]%*c", origem);

    printf("\nPESO: ");
    scanf("%[^\n]%*c", peso);
    //getchar(); // Limpa o '\n' deixado pelo scanf de número

    printf("\nANO DE FABRICACAO: ");
    scanf("%[^\n]%*c", ano_de_fabricacao);
    //getchar(); // Limpa o '\n' deixado pelo scanf de número
```

Função responsável por ler os dados necessários da estrutura chocoNode. Realiza também a verificação caso o usuário tenha digitado o tipo de chocolate AMARGO, e se sim, exige a leitura da porcentagem de cacau.

Parâmetros: Não possui parâmetros;

Retorno: Retorna um chocolate após a leitura de todos os dados.

```
printf("\nVALIDADE: ");
scanf(" %[^\\n]*c", validade);

return alocaChoco(nome, marca, tipo, porcentagem, origem, peso, ano_de_fabricacao, validade);
}
```

2.5. int comparaChoco

```
int comparaChoco(void* dado, void* chave) {
    chocoNode* d = (chocoNode*)dado;
    char* c = (char*)chave;

    return strcmp(d->nome, c);
}
```

Função responsável por comparar uma chave e o nome do chocolate (chave de busca) contido na estrutura; Realiza o casting da estrutura e da chave;

Parâmetros:

- void* dado = estrutura do chocolate a ser comparada;
- void* chave = chave para ser comparada;

Retorno:

Retorna o valor da comparação pela função strcmp, que compara o nome do chocolate e a chave. Retorna 0 se igual, negativo se o nome é menor que a chave, e positivo se o nome é maior que a chave (em ordem alfabética).

2.6. void imprimeChoco

```
void imprimeChoco(void* dado){
    chocoNode* node = (chocoNode*)dado;
    printf("NOME: %s\\n", node->nome);
    printf("MARCA: %s\\n", node->marca);
    printf("TIPO DE CHOCOLATE: %s\\n", node->tipo);
    printf("PORCENTAGEM DE CACAU: %s%%\\n", node->porcentagem);
    printf("NACIONALIDADE OU ORIGEM: %s\\n", node->origem);
    printf("PESO: %sg\\n", node->peso);
    printf("ANO DE FABRICACAO: %s\\n", node->ano_de_fabricacao);
    printf("VALIDADE: %s\\n", node->validade);
}
```

Função responsável por imprimir todos os dados contidos na estrutura chocoNode.

Parâmetros: void* dado = estrutura do chocolate a ser imprimida;

Retorno: Função não possui retorno;

2.7. void freeChoco

```
void freeChoco(void *dado){
    chocoNode* node = (chocoNode*)dado;
    if(node!=NULL){
        free(node->nome);
        free(node->marca);
        free(node->tipo);
        free(node->porcentagem);
        free(node->peso);
        free(node->ano_de_fabricacao);
        free(node->origem);
        free(node->validade);
        free(node);
    }
}
```

Função responsável por dar free nas variáveis que foram alocadas dinamicamente na estrutura.

Parâmetros: void* dado = estrutura do chocolate a ser desalocada.

Retorno: Função não possui retorno;

3. Filme

Para a categoria filme, criamos uma biblioteca chamada filme.h e um arquivo filme.c.

3.1. Inclusão de bibliotecas:

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

3.2. Estrutura:

```
typedef struct Filme{
    char* titulo;
    char* diretor;
    char* genero;
    char* distribuidor;
    char* PaisDeOrigem;
    char* duracao;
}FilmeNode;
```

char* titulo = armazena uma string para o titulo do Filme;

char* diretor = armazena uma string para o diretor do Filme;

char* genero = armazena uma string para o genero do Filme;

char* distribuidor = armazena uma string para o distribuidor do Filme

char* PaisDeOrigem = armazena uma string para o pais de origem do Filme;

char* duracao = armazena uma string para o pais de origem do Filme;

3.3 FilmeNode* alocaFilme

```
FilmeNode* alocaFilme(char *titulo, char* diretor, char* genero, char* distribuidor, char* PaisDeOrigem, char* duracao){
    FilmeNode* novo = (FilmeNode*)calloc(1,sizeof(FilmeNode));
    if(novo==NULL){
        printf("ERRO_DE_ALOCACAO\n");
        exit(1);
    }

    novo->titulo = (char*)malloc(sizeof(char) * (strlen(titulo) + 1));
    novo->diretor = (char*)malloc(sizeof(char) * (strlen(diretor) + 1));
    novo->genero = (char*)malloc(sizeof(char) * (strlen(genero) + 1));
    novo->distribuidor = (char*)malloc(sizeof(char) * (strlen(distribuidor) + 1));
    novo->PaisDeOrigem = (char*)malloc(sizeof(char) * (strlen(PaisDeOrigem) + 1));
    novo->duracao = (char*)malloc(sizeof(char) * (strlen(duracao) + 1));

    strcpy(novo->titulo, titulo);
    strcpy(novo->diretor, diretor);
    strcpy(novo->genero, genero);
    strcpy(novo->distribuidor, distribuidor);
    strcpy(novo->duracao, duracao);
    strcpy(novo->PaisDeOrigem, PaisDeOrigem);

    return novo;
}
```

Função responsável por alocar uma estrutura que corresponde a um filme para ser inserido. A função recebe os parâmetros e realiza a alocação dinâmica, fazendo um calloc com tamanho da estrutura e das variáveis do tipo char*, e atribui os dados fornecidos aos campos.

Parâmetros:

- char* titulo = armazena uma string para o titulo do Filme;
- char* diretor = armazena uma string para o diretor do Filme;
- char* genero = armazena uma string para o genero do Filme;
- char* distribuidor = armazena uma string para o distribuidor do Filme
- char* PaisDeOrigem = armazena uma string para o pais de origem do Filme;
- char* duracao = armazena uma string para o pais de origem do Filme;

Retorno: Retorna a estrutura FilmeNode com os dados fornecidos.

3.4 FilmeNode* leFilme

```
FilmeNode* leFilme(){
    char titulo[MAX_TAM];
    char diretor[MAX_TAM];
    char genero[MAX_TAM];
    char distribuidor[MAX_TAM];
    char duracao[MAX_TAM]; // Corrigi a acentuação para evitar problemas
    char PaisDeOrigem[MAX_TAM];

    getchar(); // Limpa qualquer '\n' residual antes da primeira entrada

    printf("TÍTULO: ");
    scanf(" %[^\\n]*c", titulo);

    printf("DIRETOR: ");
    scanf(" %[^\\n]*c", diretor);

    printf("GÊNERO: ");
    scanf(" %[^\\n]*c", genero);

    printf("NACIONALIDADE OU ORIGEM: ");
    scanf(" %[^\\n]*c", distribuidor);

    printf("DURAÇÃO: ");
    scanf(" %[^\\n]*c", duracao); // Removi o '&'

    printf("PAÍS DE ORIGEM: ");
    scanf(" %[^\\n]*c", PaisDeOrigem);

    return alocaFilme(titulo, diretor, genero, distribuidor, PaisDeOrigem, duracao);
}
```

Função responsável por ler os dados necessários da estrutura FilmeNode.

Parâmetros: Não possui parâmetros;

Retorno: Retorna um filme após a leitura de todos os dados;

3.5. int comparaFilme

```
int comparaFilme(void* dado, void* chave) {
    FilmeNode* d = (FilmeNode*)dado;
    char* c = (char*)chave;

    return strcmp(d->titulo, c);
}
```

Função responsável por comparar uma chave e o nome do Filme contido na estrutura; Realiza o casting da estrutura e da chave.

Parâmetros: void* dado = estrutura do chocolate a ser comparada;
void* chave = chave para ser comparada;

Retorno: Retorna o valor da comparação pela função strcmp, que compara o nome do chocolate e a chave. Retorna 0 se igual, negativo se o nome é menor que a chave, e positivo se o nome é maior que a chave (em ordem alfabética).

3.6. void imprimeFilme

```
void imprimeFilme(void* dado){
    FilmeNode* node = (FilmeNode*)dado;
    printf("TITULO: %s\n", node->titulo);
    printf("DIRETOR: %s\n", node->diretor);
    printf("GENERO: %s\n", node->genero);
    printf("DISTRIBUIDORA: %s\n", node->distribuidor);
    printf("DURACAO: %s\n", node->duracao);
    printf("PAIS DE ORIGEM: %s\n", node->PaisDeOrigem);
}
```

Função responsável por imprimir todos os dados contidos na estrutura FilmeNode;

Parâmetros: void* dado = estrutura do Filme a ser imprimida;

Retorno: Função não possui retorno.

3.7 void freeFilme

```
void freeFilme(void *dado){
    FilmeNode* node = (FilmeNode*)dado;
    if(node!=NULL){
        free(node->titulo);
        free(node->diretor);
        free(node->genero);
        free(node->distribuidor);
        free(node->duracao);
        free(node->PaisDeOrigem);
        free(node);
    }
}
```

Função responsável por dar free nas variáveis que foram alocadas dinamicamente na estrutura.

Parâmetros: void* dado = estrutura do Filme a ser imprimida;

Retorno: Função não possui retorno.

4 . Livro

4.1 Inclusão de bibliotecas:

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

4.2 Estrutura:

```
typedef struct Livro{
    char* titulo;
    char* autor;
    char* editora;
    char* DataDePublicacao;
    char* idioma;
    int NumDePag;
}LivroNode;
```

char* titulo = armazena uma string para o título do Livro;

char* autor = armazena uma string para o autor do Livro;

char* editora = armazena uma string para a editora do Livro;

char* DataDePublicacao = armazena uma string para a data de publicação do Livro;

char* idioma = armazena uma string para o idioma do Livro;

int NumDePag = inteiro referente ao número de páginas do Livro;

4.3 LivroNode* alocaLivro:

```
LivroNode* alocaLivro(char *titulo, char* autor, char* editora, char* DataDePublicacao, int NumDePag, char* idioma){
    LivroNode* novo = (LivroNode*)calloc(1,sizeof(LivroNode));
    if(novo==NULL){
        printf("ERRO_DE_ALOCACAO\n");
        exit(1);
    }

    novo->titulo = (char*)malloc(sizeof(char) * (strlen(titulo) + 1));
    novo->autor = (char*)malloc(sizeof(char) * (strlen(autor) + 1));
    novo->editora = (char*)malloc(sizeof(char) * (strlen(editora) + 1));
    novo->DataDePublicacao = (char*)malloc(sizeof(char) * (strlen(DataDePublicacao) + 1));
    novo->idioma = (char*)malloc(sizeof(char) * (strlen(idioma) + 1));

    novo->NumDePag = NumDePag;

    strcpy(novo->titulo, titulo);
    strcpy(novo->autor, autor);
    strcpy(novo->editora, editora);
    strcpy(novo->DataDePublicacao, DataDePublicacao);
    strcpy(novo->idioma, idioma);

    return novo;
}
```

Função que aloca a memória necessária para uma estrutura do tipo livro.

A função também aloca individualmente a memória necessária para cada string contida na estrutura.

Parâmetros:

- **char* titulo** = armazena uma string para o título do Livro;
- **char* autor** = armazena uma string para o autor do Livro;
- **char* editora** = armazena uma string para a editora do Livro;**char* DataDePublicacao** = armazena uma string para a data de publicação do Livro;
- **char* idioma** = armazena uma string para o idioma do Livro;**int NumDePag** = inteiro referente ao número de páginas do Livro;

Retorno: Retorna a estrutura livroNode com os dados fornecidos.

4.4 LivroNode* LeLivro:

```
LivroNode* leLivro(){
    char titulo[MAX_TAM];
    char autor[MAX_TAM];
    char editora[MAX_TAM];
    char dataDePublicacao[MAX_TAM]; // Evitei acentos para evitar erros
    int numDePag;
    char idioma[MAX_TAM];

    getchar(); // Remove qualquer '\n' residual

    printf("TÍTULO: ");
    scanf("%[^\n]*c", titulo);

    printf("AUTOR: ");
    scanf("%[^\n]*c", autor);

    printf("EDITORIA: ");
    scanf("%[^\n]*c", editora);

    printf("DATA DE PUBLICAÇÃO: ");
    scanf("%[^\n]*c", dataDePublicacao);

    printf("NÚMERO DE PÁGINAS: ");
    scanf("%d", &numDePag);
    getchar(); // Consome o '\n' deixado pelo scanf de inteiro

    printf("IDIOMA: ");
    scanf("%[^\n]*c", idioma);

    return alocalivro(titulo, autor, editora, dataDePublicacao, numDePag, idioma);
}
```

Função responsável por ler os dados necessários da estrutura LivroNode;

Parâmetros: Não possui parametros.

Retorno: Retorna um LivroNode após a leitura de todos os dados;

4.5 int comparaLivro:

```
int comparaLivro(void* dado, void* chave) {
    LivroNode* d = (LivroNode*)dado;
    char* c = (char*)chave;

    return strcmp(d->titulo, c);
}
```

Função responsável por comparar uma chave e o nome do Livro contido na estrutura; Realiza o casting da estrutura e da chave;

Parâmetros:

- void* dado = estrutura do Livro a ser comparada;
- void* chave = chave para ser comparada;

Retorno: Retorna o valor da comparação pela função strcmp, que compara o nome do chocolate e a chave. Retorna 0 se igual, um valor negativo se o nome é menor que a chave, e um valor positivo se o nome é maior que a chave (em ordem alfabética).

4.6 void imprimeLivro:

```
void imprimeLivro(void* dado){
    LivroNode* node = (LivroNode*)dado;
    printf("titulo: %s\n", node->titulo);
    printf("autor: %s\n", node->autor);
    printf("EDITORA: %s\n", node->editora);
    printf("DATA DE PUBLICACAO: %s\n", node->DataDePublicacao);
    printf("NUMERO DE PAGINAS: %d\n", node->NumDePag);
    printf("IDIOMA: %s\n", node->idioma);
}
```

Função responsável por imprimir todos os dados contidos na estrutura LivroNode;

Parâmetros: void* dado = estrutura do Livro a ser imprimida;

4.7 freeLivro:

```
void freeLivro(void *dado){
    LivroNode* node = (LivroNode*)dado;
    if(node!=NULL){
        free(node->titulo);
        free(node->autor);
        free(node->editora);
        free(node->DataDePublicacao);
        free(node->idioma);
        free(node);
    }
}
```

Função responsável por dar free nas variáveis que foram alocadas dinamicamente na estrutura.

Parâmetros: void* dado = estrutura do file a ser desalocada.

5. Vinho

5.1 Inclusão de bibliotecas:

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

5.2. Estrutura

```
typedef struct Vinho{
    char* nome;
    char* vinicola;
    char* tipo;
    char* pais;
    char* regioao;
    int ano_de_fabricacao;
    char* uva;
}VinhoNode;
```

char* nome = armazena uma string para o nome do Vinho;

char* vinicola = armazena uma string para a vinicola do Vinho;

char* tipo = armazena uma string para o tipo do Vinho;

char* pais = armazena uma string para o pais do Vinho;

char* regioao = armazena uma string a regioao do Vinho;

int ano_de_fabricacao = inteiro referente ao ano de fabricação do Vinho;

char* uva = armazena uma string para a uva do Vinho;

5.3 VinhoNode* alocaVinho

```
VinhoNode* alocaVinho(char *nome, char* vinicola, char* tipo, char* pais, char* regioao, int ano_de_fabricacao, char* uva){
    VinhoNode* novo = (VinhoNode*)calloc(1, sizeof(VinhoNode));
    if(novo==NULL){
        printf("ERRO_DE_ALOCACAO\n");
        exit(1);
    }

    novo->nome = (char*)malloc(sizeof(char) * (strlen(nome) + 1));
    novo->vinicola = (char*)malloc(sizeof(char) * (strlen(vinicola) + 1));
    novo->tipo = (char*)malloc(sizeof(char) * (strlen(tipo) + 1));
    novo->pais = (char*)malloc(sizeof(char) * (strlen(pais) + 1));
    novo->uva = (char*)malloc(sizeof(char) * (strlen(uva) + 1));
    novo->regiao = (char*)malloc(sizeof(char) * (strlen(regiao) + 1));

    novo->ano_de_fabricacao = ano_de_fabricacao;

    strcpy(novo->nome, nome);
    strcpy(novo->vinicola, vinicola);
    strcpy(novo->tipo, tipo);
    strcpy(novo->pais, pais);
    strcpy(novo->uva, uva);
    strcpy(novo->regiao, regioao);

    return novo;
}
```

Função responsável por alocar uma estrutura que corresponde a um vinho para ser inserido. A função recebe os parâmetros e realiza a alocação dinâmica, fazendo um calloc com tamanho da estrutura e das variáveis do tipo char*, e atribui os dados fornecidos aos campos.

Parâmetros: char* nome = armazena uma string para o nome do Vinho;
char* vinicola = armazena uma string para a vinicola do Vinho;
char* tipo = armazena uma string para o tipo do Vinho;
char* pais = armazena uma string para o pais do Vinho;
char* regioao = armazena uma string a regioao do Vinho;
int ano_de_fabricacao = inteiro referente ao ano de fabricação do Vinho;
char* uva = armazena uma string para a uva do Vinho;
Retorno: Retorna a estrutura VinhoNode com os dados fornecidos.

5.4 VinhoNode* leVinho

```
VinhoNode* leVinho(){
    char nome[MAX_TAM];
    char vinicola[MAX_TAM];
    char tipo[MAX_TAM];
    char pais[MAX_TAM];
    char regioao[MAX_TAM];
    int ano_de_fabricacao;
    char uva[MAX_TAM];

    getchar(); // Remove qualquer '\n' residual

    printf("NOME: ");
    scanf("%[^\n]%c", nome);

    printf("VINÍCOLA: ");
    scanf("%[^\n]%c", vinicola);

    printf("TIPO: ");
    scanf("%[^\n]%c", tipo);

    printf("PAÍS: ");
    scanf("%[^\n]%c", pais);

    printf("REGIÃO: ");
    scanf("%[^\n]%c", regioao);

    printf("ANO DE FABRICAÇÃO: ");
    scanf("%d", &ano_de_fabricacao);
    getchar(); // Consome o '\n' deixado pelo scanf de inteiro

    printf("UVA: ");
    scanf("%[^\n]%c", uva);

    return alocaVinho(nome, vinicola, tipo, pais, regioao, ano_de_fabricacao, uva);
}
```

Função responsável por ler os dados necessários da estrutura VinhoNode;

Parâmetros: Não possui parâmetros;

Retorno: Retorna um vinho após a leitura de todos os dados;

5.5 int comparaVinho

```
int comparaVinho(void* dado, void* chave) {
    VinhoNode* d = (VinhoNode*)dado;
    char* c = (char*)chave;

    return strcmp(d->nome, c);
}
```

Função responsável por comparar uma chave e o nome do Vinho contido na estrutura;

Realiza o casting da estrutura e da chave;

Parâmetros: void* dado = estrutura do chocolate a ser comparada;

void* chave = chave para ser comparada;

Retorno: Retorna o valor da comparação pela função strcmp, que compara o nome do chocolate e a chave. Retorna 0 se igual, negativo se o nome é menor que a chave, e positivo se o nome é maior que a chave (em ordem alfabética).

5.6 void imprimeVinho

```
void imprimeVinho(void* dado){
    VinhoNode* node = (VinhoNode*)dado;
    printf("NOME: %s\n", node->nome);
    printf("VINICOLA: %s\n", node->vinicola);
    printf("TIPO: %s\n", node->tipo);
    printf("PAIS: %s\n", node->pais);
    printf("REGIAO: %s\n", node->regiao);
    printf("ANO DE FABRICACAO: %d\n", node->ano_de_fabricacao);
    printf("UVA: %s\n", node->uva);
}
```

Função responsável por dar free nas variáveis que foram alocadas dinamicamente na estrutura.

Parâmetros: void* dado = estrutura do Vinho a ser imprimida;

Retorno: Função não possui retorno.

5.7 void freeVinho

```
void freeVinho(void *dado){
    VinhoNode* node = (VinhoNode*)dado;
    if(node!=NULL){
        free(node->nome);
        free(node->vinicola);
        free(node->tipo);
        free(node->pais);
        free(node->regiao);
        free(node->uva);
        free(node);
    }
}
```

Função responsável por dar free nas variáveis que foram alocadas dinamicamente na estrutura.

Parâmetros: void* dado = estrutura do vinho a ser desalocada.

Retorno: Função não possui retorno.

6. Produto

6.1: Inclusão de bibliotecas:

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

6.2: Estrutura:

```
typedef struct produto{
    void *dado;
    void(*print)(void* dado);
    int(*compara)(void* dado, void* chave);
    void(*desaloca)(void* dado);
    struct produto *next;
}Produto;
```

void* dado: Ponteiro genérico que armazena uma estrutura com os dados específicos de cada tipo de produto (chocolate, livro, vinho ou filme);

void(*print)(void* dado): ponteiro para uma função de imprimir referente à estrutura que está sendo armazenada;

int(*compara)(void* dado, void* chave): ponteiro para uma função de comparar referente à uma estrutura que está sendo;

void(*desaloca)(void* dado): ponteiro para uma função de comparar referente à uma estrutura que está sendo armazenada;

struct produto *next: ponteiro para o próximo elemento da lista de produtos;

6.3 Função Produto* alocaProduto:

```
Produto* alocaProduto(void* dado, void(*print)(void *dado), int(*compara)(void* dado, void* chave), void(*desaloca)(void* dado)){
    Produto* ProdutoNode = (Produto*)calloc(1,sizeof(Produto));
    if(ProdutoNode==NULL){
        printf("ERRO_DE_ALOCACAO!");
        exit(1);
    }
    ProdutoNode->dado = dado;
    ProdutoNode->compara = compara;
    ProdutoNode->print = print;
    ProdutoNode->desaloca = desaloca;

    return ProdutoNode;
}
```

A função aloca o espaço necessário, checando se houve um erro de alocação, para uma estrutura do tipo produto e atribui os argumentos aos seus respectivos campos.

Parâmetros:

- void* dado = ponteiro para estrutura que está sendo armazenada(chocolate, vinho, livro ou filme);
- void(*print)(void* dado =: ponteiro para uma função de imprimir referente à estrutura que está sendo armazenada;
- int(*compara)(void* dado, void* chave) = ponteiro para uma função de comparar referente à uma estrutura que está sendo armazenada;
- struct produto *next = ponteiro para o próximo elemento da lista de produto;

Retorno:

Retorna um novo nó do tipo Produto.

6.4: Função void printProduto:

```
void printProduto(Produto *P){
    P->print(P->dado);
}
```

Função genérica responsável por imprimir os dados contidos no campo void* dado no nó da estrutura Produto utilizando a função de callback armazenada na estrutura.

Parâmetros:

Produto* P = nó do Produto.

6.5 Função int comparaProduto:

```
int comparaProduto(Produto *P, void* chave){
    return P->compara(P->dado, chave);
}
```

Função genérica responsável por comparar o campo dado com uma chave fornecida utilizando a função de callback contida na estrutura.

Parâmetros:

Produto* P = nó do Produto;

void* chave = chave para ser comparada;

Retorno:Retorna o resultado da comparação do dado com a chave (1 se for compatível e 0 caso contrário.

7. Lista

O código é dividido nessas funções:

- **Inicialização da pilha** (inicializaNoPilha, inicializaPilha)
- **Operações na pilha** (inserePilha, removePilha)
- **Undo & Redo** (limpaRedo, undo, redo)

- **Manipulação de listas** (Insere, pop, busca, print_list)
- **Liberação de memória** (freeLista, freePilha)

```
#include "lista.h"

/*INICIALIZAÇÕES*/
Pilha* inicializaNoPilha(Produto *lista){
    Pilha* pilha = malloc(sizeof(Pilha));
    pilha->next=pilha;
    pilha->P = lista;
    return pilha;
}

Pilha* inicializaPilha(){
    Pilha* pilha = malloc(sizeof(Pilha));
    pilha->next= pilha;
    pilha->P = NULL;
    return pilha;
}
```

A Pilha é implementada como uma lista circular encadeada. No InicializaNoPilha o ponteiro next apontando para si mesmo, criando uma estrutura circular.

Armazena um ponteiro para a lista de produtos no campo P.

Já a InicializaPilha é similar à anterior, mas inicializa uma pilha vazia (P = NULL).

```
//NA FUNÇÃO INSEREPILHA, ESTAMOS TRABALHANDO COM UMA LISTA CIRCULAR SIMPLES ENCADEADA, PARA AUXILIAR NA MANIPULAÇÃO
Pilha* inserePilha(Produto* lista, Pilha* pilha){
    Pilha* newpilha=inicializaNoPilha(lista);
    if(pilha == NULL){
        pilha=newpilha;
        pilha->next=newpilha;
    }
    else{
        newpilha->next = pilha->next;
        pilha->next = newpilha;
        pilha=newpilha;
    }

    return pilha;
}

Pilha* removePilha(Pilha* pilha){
    Pilha* auxpilha = pilha->next;

    if (pilha == NULL) return NULL;
    if(auxpilha==pilha){
        free(auxpilha);
        return NULL;
    }
    Pilha* predpilha = auxpilha;
    while(predpilha->next!=pilha) predpilha = predpilha->next;
    predpilha->next=auxpilha;
    free(pilha);

    return predpilha;
}
```

Já a função de `InsererPilha`, ela insere um novo elemento na pilha circular.

Caso a pilha estiver vazia, o novo nó é criado e aponta para si mesmo (`newpilha->next = newpilha`).

Entretanto se a pilha já contém elementos: o novo nó é inserido antes do primeiro elemento e o topo da pilha é atualizado para o novo nó.

A função `removePilha`, remove o nó do topo da pilha.

Caso a pilha está vazia, retorna `NULL`.

Se há apenas um nó, ele é removido e a pilha fica vazia.

Caso contrário:

- O nó anterior ao topo da pilha (`predpilha`) passa a ser o novo topo.
- O nó do topo antigo é liberado da memória.

```
/*OPERAÇÕES COM UNDO E REDO*/

void limpaRedo(Pilha **pilhaaux, Pilha **redo){
    if((*redo)==NULL) return;
    if((*pilhaaux) == NULL){
        *pilhaaux = (*redo);
        *redo = NULL;
    }
    else{
        Pilha* Predo = (*redo)->next;
        Pilha* Paux = (*pilhaaux)->next;
        (*redo)->next = Paux;
        (*pilhaaux)->next = Predo;
        *redo=NULL;
    }
}

//AO FAZER UNDO, PEGAMOS O ULTIMO ELEMENTO DA PILHA DE UNDO E INSERIMOS EM REDO, LOGO APÓS APAGAMOS O TOPO DA PILHA DE UNDO;
Produto* undo(Produto* list, Pilha** undo, Pilha** redo){
    if (*undo == NULL) return list;
    Produto* new = (*undo)->P;
    *redo = inserePilha(list, (*redo));
    *undo = removePilha(*undo);
    return new;
}

//AO FAZER REDO, É INSERIDO NA PILHA DE UNDO A ULTIMA VERSÃO DA LISTA, E É REMOVIDO DE REDO O SEU TOPO.
Produto* redo(Produto* list, Pilha** undo, Pilha** redo){
    if (*redo == NULL) return list;
    Produto* new = (*redo)->P;
    *undo = inserePilha(list, *undo);
    *redo = removePilha(*redo);
    return new;
}
```

Essas funções implementam as operações de Undo e Redo, permitindo desfazer (undo) e refazer (redo) modificações feitas em uma lista de produtos. Elas utilizam pilhas para armazenar os estados anteriores da lista, possibilitando que o usuário retorne a versões anteriores ou reexecute ações desfeitas.

- `limpaRedo(Pilha **pilhaaux, Pilha **redo)`
 - Essa função limpa a pilha de Redo sempre que uma nova operação é feita.
 - Se a pilha de Redo (redo) não estiver vazia, ela é transferida para pilhaaux.
 - Isso garante que, ao fazer uma nova modificação, não seja possível refazer (redo) ações anteriores.
- `undo(Produto* list, Pilha** undo, Pilha** redo)`
 - Desfaz a última operação feita na lista.
 - O estado atual da lista é movido para a pilha de Redo, permitindo que possa ser recuperado futuramente.
 - O topo da pilha de Undo (última versão salva) é restaurado como a lista principal.
 - Se não houver elementos na pilha de Undo, retorna a lista sem alterações.
- `redo(Produto* list, Pilha** undo, Pilha** redo)`
 - Refaz a última operação desfeita.
 - O estado salvo na pilha de Redo é restaurado e movido para a pilha de Undo.
 - Remove o topo da pilha de Redo, pois ele já foi reaplicado.

```

Produto* Insere(Produto *list, void *dado, void(*print)(void *dado), int(*compara)(void* dado, void* chave),void(*desaloca)(void* dado), Pilha** undo,
Pilha **redo, Pilha **auxpilha, void* chave){
    Produto *node = alocaProduto(dado, print, compara, desaloca);
    Produto* old = list;
    Produto* new = NULL;
    Produto *aux = list;
    Produto *pred = NULL;
    Produto *new_aux = NULL;

    limpaRedo(auxpilha, redo);

    if(list == NULL){
        new = node;
        *undo = inserePilha(NULL, (*undo));
        return new;
    }
    else{
        *undo = inserePilha(old, (*undo));
        while (aux) {
            Produto *copia = alocaProduto(aux->dado, aux->print, aux->compara, aux->desaloca);

            if (new == NULL) {
                new = copia;
                new_aux = new;
            } else {
                new_aux->next = copia;
                new_aux = new_aux->next;
            }
            if (aux->compara(aux->dado, chave) < 0) {
                pred = copia;
            } else {
                break;
            }
        }
    }
}

```

```

    }

    aux = aux->next;
}
if (pred == NULL) {

    node->next = new;
    new = node;
} else {

    node->next = pred->next;
    pred->next = node;
}

return new;
}
}

```

Primeiro, a função `Inserir` aloca um novo nó para o produto usando `alocaProduto`, que recebe os dados do novo produto e funções auxiliares (`print`, `compara`, `desaloca`).

Primeiro, se armazena o estado atual da lista em `undo`.

- Como será feita uma modificação, o estado anterior da lista é salvo na pilha de `undo`.
- Isso permite que a operação seja revertida posteriormente.

Limpa o `redo` caso uma nova inserção ocorra

- Antes de modificar a lista, a função `limpaRedo` é chamada. Isso impede que estados antigos sejam refeitos (evita inconsistências no histórico).

Se a lista está vazia (`list == NULL`)

- O novo nó se torna a cabeça da lista.
- O estado anterior (`NULL`, pois não havia lista) é salvo na pilha de `undo`.

Caso a lista já tenha elementos

- Percorre a lista encadeada para encontrar a posição correta do novo nó.
- Mantém uma cópia de cada nó antigo (cópia), garantindo que a nova lista seja uma nova versão sem modificar diretamente os nós anteriores.
- Se encontrar um nó maior que o novo produto, insere o novo nó antes dele.
- Se não encontrar, adiciona o nó no final.

Retorna a nova lista atualizada

- A função retorna a nova versão da lista, que inclui o produto recém-inserido.

```

Produto* pop(Produto *list, void* chave, Pilha** undo, Pilha** redo, Pilha **auxpilha){
    Produto *auxnode = NULL;
    Produto *prednode = NULL;
    Produto* old = list;
    Produto* new = NULL;
    Produto* auxnew = NULL;
    auxnode = list;

    Produto* node = NULL;

    limpaRedo(auxpilha, redo);

    *undo = inserePilha(old, (*undo));
    if (list == NULL){
        return NULL;
    }
    else{
        while(auxnode!=NULL && (auxnode->compara(auxnode->dado, chave) != 0)){
            prednode = auxnode;
            if(prednode!=NULL){
                node = alocaProduto(auxnode->dado, auxnode->print, auxnode->compara, auxnode->desaloca);
                if(new ==NULL){
                    new = node;
                    auxnew=new;
                }
                else{
                    while(auxnew->next!=NULL) auxnew=auxnew->next;
                    auxnew->next=node;
                }
            }
            auxnode = auxnode->next;
        }
        if(prednode==NULL) new = auxnode->next;
        else{
            node->next=auxnode->next;
        }
    }
    return new;
}

```

A função pop tem como objetivo remover um produto específico de uma lista encadeada, utilizando uma chave como identificador. Além disso, a função mantém um histórico das alterações feitas para permitir que a remoção possa ser desfeita (undo) futuramente. Para isso, a função utiliza pilhas auxiliares, que armazenam o estado anterior da lista antes da modificação.

Primeiramente, são declaradas várias variáveis para manipulação da lista. As variáveis auxnode e prednode são utilizadas para percorrer a lista e identificar o nó a ser removido. A variável old armazena o estado original da lista antes da remoção. As variáveis new e auxnew são utilizadas para construir a nova versão da lista após a remoção do elemento. Já a variável node será utilizada para criar cópias dos nós. Em seguida, auxnode é inicializado com a cabeça da lista, pois será utilizado para percorrê-la.

Antes de modificar a lista, a função chama limpaRedo(auxpilha, redo). Isso garante que, caso um novo produto seja removido, o histórico de Redo seja apagado, impedindo a repetição de ações desfeitas anteriormente. Em seguida, o estado atual da lista é salvo na

pilha undo, garantindo que a ação possa ser revertida futuramente. Caso a lista esteja vazia, a função retorna NULL, pois não há elementos a serem removidos.

A próxima etapa consiste em percorrer a lista para encontrar o elemento correspondente à chave fornecida. A função percorre a lista utilizando um loop que verifica se o nó atual contém a chave desejada, utilizando a função compara. Durante essa busca, prednode mantém uma referência ao nó anterior ao elemento que está sendo verificado. Isso é importante porque, ao remover um nó, o nó anterior precisa ajustar seu ponteiro para ignorar o nó que será excluído.

Se o nó a ser removido não for o primeiro da lista, a função cria uma nova versão da lista sem o elemento excluído. Para isso, cada nó encontrado na lista original é copiado para a nova lista, com exceção do nó que será removido. Se new for NULL, significa que este é o primeiro nó da nova lista. Caso contrário, a função percorre a nova lista e adiciona o novo nó ao final.

Após recriar a lista sem o elemento removido, a função ajusta os ponteiros para garantir que a estrutura continue válida. Se prednode for NULL, significa que o elemento removido era o primeiro da lista e, nesse caso, new aponta para o próximo nó da lista original. Caso contrário, o nó anterior ao removido é atualizado para apontar para o próximo nó, efetivamente eliminando o nó desejado da estrutura.

Por fim, a função retorna a nova lista new, que contém todos os elementos da lista original, exceto aquele que foi removido. Dessa forma, a função pop não apenas remove um elemento da lista, mas também mantém um histórico de alterações, permitindo que a remoção seja revertida por meio da funcionalidade de undo.


```

Produto *busca(Produto *list, void* chave){
    Produto *auxnode = list;
    while(auxnode!=NULL){
        if(auxnode->compara(auxnode->dado, chave)==0){
            return auxnode;
        }
        auxnode = auxnode->next;
    }
    return auxnode;
}

void print_list(Produto *list){
    Produto *auxnode = list;
    if(list == NULL) {
        printf("LISTA VAZIA!\n");
        return;}
    printf("\n");
    while (auxnode){
        auxnode->print(auxnode->dado);
        printf("\n");
        auxnode = auxnode->next;
    }
}

```

O código também implementa duas outras funções fundamentais para a manipulação da estrutura: busca e print_list. A função busca percorre a lista encadeada em busca de um nó que contenha uma chave específica. Para isso, ela inicializa um ponteiro auxiliar auxnode apontando para a cabeça da lista e entra em um loop que percorre todos os nós. Em cada iteração, a função verifica se o dado armazenado no nó atual corresponde à chave fornecida, utilizando a função compara. Se encontrar um nó que contenha a chave, a função retorna esse nó. Caso contrário, continua avançando na lista até atingir o final. Se o elemento não for encontrado, a função retorna NULL, indicando que a chave não está presente na lista. Essa função é útil para localizar rapidamente um item dentro da estrutura e pode ser utilizada em conjunto com outras operações, como remoção ou atualização de dados.

Já a função print_list tem a finalidade de exibir os elementos da lista na tela. Inicialmente, ela verifica se a lista está vazia e, nesse caso, imprime a mensagem "LISTA VAZIA!" e retorna imediatamente. Caso contrário, a função percorre todos os nós da lista utilizando um ponteiro auxiliar auxnode, chamando a função print de cada elemento para exibir suas informações. Durante esse processo, cada dado é impresso seguido de uma quebra de linha, garantindo que a saída fique organizada e legível. Dessa forma, print_list permite visualizar os elementos armazenados na estrutura, facilitando o acompanhamento das operações realizadas sobre a lista.

```

/*FUNÇÕES PARA A DESALOCAÇÃO DAS LISTAS E PILHAS */
void freeLista(Produto* list){
    Produto *aux = list;

    if(list==NULL) return;

    while(list!=NULL){
        aux = list->next;
        freeProduto(list);
        list = aux;
    }
}

void freePilha(Pilha** pilha) {
    if (*pilha == NULL) return;

    Pilha* atual = *pilha;
    Pilha* prox;
    Pilha* primeiro = *pilha;

    do {
        prox = atual->next;
        freeLista(atual->P);
        atual->P = NULL;
        free(atual);
        atual = prox;
    } while (atual != primeiro);

    *pilha=NULL;
}

```

As funções freeLista e freePilha têm a finalidade de liberar a memória alocada dinamicamente para as listas encadeadas e pilhas circulares utilizadas na estrutura de dados. Essas funções garantem que não haja vazamento de memória ao encerrar o programa ou ao remover estruturas que não são mais necessárias.

A função freeLista é responsável por desalocar todos os nós de uma lista encadeada do tipo Produto. Primeiramente, um ponteiro auxiliar aux é inicializado com a cabeça da lista para facilitar a navegação. A função verifica se a lista está vazia, retornando imediatamente caso list == NULL. Em seguida, entra em um loop while que percorre cada nó da lista. Durante

esse processo, o ponteiro aux armazena a referência para o próximo nó antes de chamar freeProduto(list), que libera a memória associada ao produto armazenado no nó atual. Depois, list avança para aux, repetindo o processo até que todos os nós sejam removidos. Dessa forma, a função garante a liberação completa da memória associada à lista.

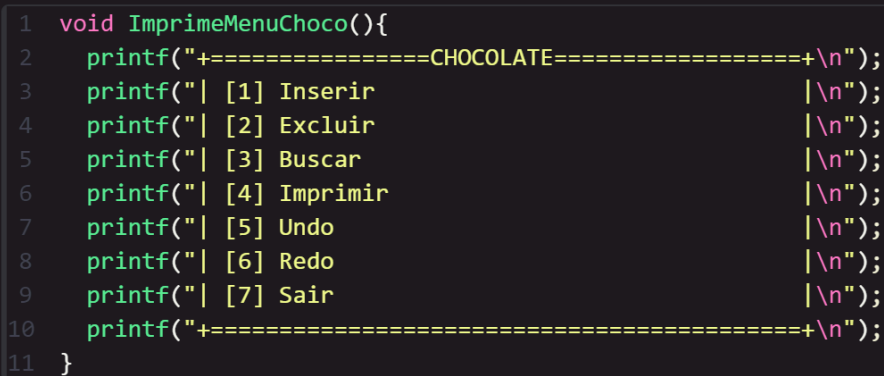
A função freePilha realiza a desalocação da estrutura de pilha circular utilizada no programa. Primeiramente, a função verifica se a pilha está vazia, retornando imediatamente se *pilha == NULL. Depois, os ponteiros auxiliares atual, prox e primeiro são inicializados, sendo primeiro utilizado para marcar o início da pilha e identificar o fim do loop. O código entra em um laço do-while, garantindo que pelo menos um nó seja processado. Em cada iteração, o ponteiro prox armazena a referência para o próximo nó da pilha. Em seguida, freeLista(atual->P) é chamado para desalocar a lista associada ao nó atual da pilha. Após isso, o ponteiro P do nó atual é definido como NULL e o próprio nó é liberado com free(atual). A pilha avança para o próximo nó e o processo se repete até que atual retorne ao nó inicial (primeiro). Por fim, o ponteiro da pilha é definido como NULL, garantindo que toda a memória associada à estrutura seja corretamente desalocada.

8. Main

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "chocolate.h"
5  #include "vinho.h"
6  #include "livro.h"
7  #include "filme.h"
8  #include "produto.h"
9  #include "lista.h"
10
11 int PrintMenuPrin(int *ts){
12     printf("          BEM VINDO A GIRLS STORE!\n");
13     printf("+=====PRODUTOS=====+\n");
14     printf("| [1] Chocolates                |\n");
15     printf("| [2] Vinhos                     |\n");
16     printf("| [3] Livros                     |\n");
17     printf("| [4] Filmes                     |\n");
18     printf("| [5] Sair                       |\n");
19     printf("+=====+\n");
20
21     int op = 0;
22
23     printf("Digite uma opção: ");
24     *ts = scanf("%d", &op);
25     return op;
26 }
```

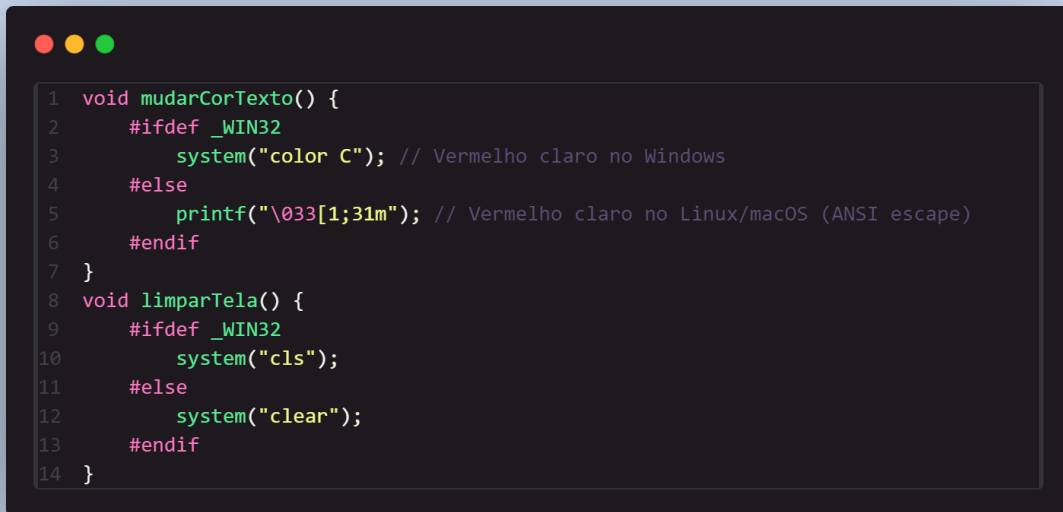
- Inclusão dos arquivos.h

- Função de impressão do menu principal. Recebe a opção digitada pelo usuário e retorna a variável;
- Recebe a lista que contém os produtos cadastrados de cada departamento e imprime na tela



```
1 void ImprimeMenuChoco(){
2     printf("+=====CHOCOLATE=====+\n");
3     printf("| [1] Inserir                               |\n");
4     printf("| [2] Excluir                                |\n");
5     printf("| [3] Buscar                                 |\n");
6     printf("| [4] Imprimir                              |\n");
7     printf("| [5] Undo                                 |\n");
8     printf("| [6] Redo                                 |\n");
9     printf("| [7] Sair                                 |\n");
10    printf("+=====+\n");
11 }
```

- Imprime o menu do departamento de chocolate.



```
1 void mudarCorTexto() {
2     #ifdef _WIN32
3         system("color C"); // Vermelho claro no Windows
4     #else
5         printf("\033[1;31m"); // Vermelho claro no Linux/macOS (ANSI escape)
6     #endif
7 }
8 void limparTela() {
9     #ifdef _WIN32
10        system("cls");
11    #else
12        system("clear");
13    #endif
14 }
```

A função `mudarCorTexto()` altera a cor do texto exibido no terminal para vermelho claro. Se o código estiver rodando no Windows (`_WIN32`), ele usa o comando `system("color C")`, que muda a cor do texto para vermelho claro no prompt de comando. Já em sistemas Linux/macOS, a função utiliza a sequência de escape ANSI `\033[1;31m` para obter o mesmo efeito.

A função `limparTela()` limpa o terminal. No Windows, ela executa `system("cls")`, enquanto no Linux/macOS, usa `system("clear")`, garantindo que a tela seja apagada independentemente do sistema operacional.

```
1  int main()
2  {
3
4
5      Produto* chocolate = NULL;
6      Pilha* chocoUndo = inicializaPilha();
7      Pilha* chocoRedo = inicializaPilha();
8      Pilha* auxpilhaC = inicializaPilha();
9
10     Produto* vinho = NULL;
11     Pilha* vinhoUndo = inicializaPilha();
12     Pilha* vinhoRedo = inicializaPilha();
13     Pilha* auxpilhaV = inicializaPilha();
14
15     Produto* livro = NULL;
16     Pilha* livroUndo = inicializaPilha();
17     Pilha* livroRedo = inicializaPilha();
18     Pilha* auxpilhaL = inicializaPilha();
19
20     Produto* filme = NULL;
21     Pilha* filmeUndo = inicializaPilha();
22     Pilha* filmeRedo = inicializaPilha();
23     Pilha* auxpilhaF = inicializaPilha();
```

- Implementação da função main: Declaração e inicialização das estruturas necessárias

```

1  Produto *b = NULL;
2  int op1 = 0;
3  int op2;
4  int ts = 0;
5  char chave[50];
6
7
8  do{
9
10     mudarCorTexto();
11     op1 = PrintMenuPrin(&ts);
12     if (ts != 1) { // Verifica se a entrada não é um número
13         limparTela();
14         printf("Entrada inválida! Digite um número válido.\n");
15         while (getchar() != '\n'); // Limpa o buffer de entrada
16         continue; // Volta para o início do loop sem executar o switch
17     }
18     limparTela();
19

```

- Função recursiva que imprime o menu principal e inicializa a variável op que guarda a opção escolhida pelo usuário, se opção for diferente de um número inteiro imprime uma mensagem de erro e volta para o início do laço. caso contrário entra em outro laço, que contém um switch case para cada departamento.

```

1  switch (op1) {
2  case 1:
3      do {
4          ImprimeMenuChoco();
5          printf("Digite uma opção: ");
6
7          if (scanf("%d", &op2) != 1) { // Verifica se a entrada não é um número
8              printf("Entrada inválida! Digite um número válido.\n");
9              while (getchar() != '\n'); // Limpa o buffer de entrada
10             continue; // Volta para o início do loop sem executar o switch
11         }
12
13         limparTela();

```

- Dentro do switch case do departamento de chocolate, solicita a opção desejada e caso a opção seja válida faz o tratamento de erro.
- No caso 1: Chama a função para inserir um elemento na lista.

```

1  case 2:
2      getchar();
3      printf("\nInsira uma chave para ser excluída: ");
4      scanf("%s", &chave);
5
6      b = busca(chocolate, chave);
7      if (b != NULL && b->compara(b->dado, chave) == 0) {
8          printf("Produto excluído com sucesso.\n");
9          chocolate = pop(chocolate, chave, &chocoUndo, &chocoRedo, &auxpilhaC);
10     } else {
11         printf("Erro ao excluir. Produto não encontrado ou lista vazia.\n");
12     }
13     break;
14
15 case 3:
16     getchar();
17     printf("\nInsira uma chave para ser buscada: ");
18     scanf("%s", &chave);
19
20     b = busca(chocolate, chave);
21     if (b != NULL) {
22         if (b->compara(b->dado, chave) == 0) printf("PRODUTO %s ENCONTRADO!\n", chave);
23         else printf("PRODUTO %s NAO ENCONTRADO.\n", chave);
24     }
25     else printf("PRODUTO %s NAO ENCONTRADO.\n", chave);
26     break;
27
28 case 4:

```


- No caso 2: chama função de exclusão de um produto na lista, caso exista imprime uma mensagem de operação realizada, caso contrário, exibe uma mensagem de erro
- No caso 3: chama a função de busca para buscar a chave recebida, caso exista imprime uma mensagem de chave existente, caso contrário, uma mensagem de erro.

```
1         default:
2             printf("Digite uma opção valida\n");
3             break;
4     }
5     } while (op2 != 7);
6     break;
```

- Caso o usuário digite um valor fora intervalo entre 1 e 7

```
1     case 5:
2         ImprimeRelatorio(chocolate, vinho, livro,
3 filme);
4         break;
5     default:
6         printf("Digite uma opção valida\n");
7         break;
8
9 }
```

- Caso o usuário no menu principal digite 5, imprime o relatório contendo os produtos cadastrados de todos os departamentos e encerra o programa;

```
1  /*DESALOCAÇÕES LISTAS E PILHAS*/
2
3      freeLista(chocolate);
4      freePilha(&chocoUndo);
5      freePilha(&chocoRedo);
6      freePilha(&auxpilhaC);
7      freeLista(vinho);
8      freePilha(&vinhoUndo);
9      freePilha(&vinhoRedo);
10     freePilha(&auxpilhaV);
11     freeLista(livro);
12     freePilha(&livroUndo);
13     freePilha(&livroRedo);
14     freePilha(&auxpilhaL);
15     freeLista(filme);
16     freePilha(&filmeUndo);
17     freePilha(&filmeRedo);
18     freePilha(&auxpilhaF);
19
20
21
22     return 0;
23 }
```

- Funções de liberação de memória de todas as estruturas alocadas ao longo do programa

9. Makefile

```
► make run
run: teste
    ./teste

► make teste
teste: chocolate.o vinho.o livro.o filme.o produto.o lista.o main.o
    @gcc -o teste chocolate.o vinho.o livro.o filme.o produto.o lista.o main.o -g

► make chocolate.o
chocolate.o: chocolate.c chocolate.h
    @gcc -c chocolate.c -Wall -Werror -Wextra -g

► make vinho.o
vinho.o: vinho.c vinho.h
    @gcc -c vinho.c -Wall -Werror -Wextra -g

► make livro.o
livro.o: livro.c livro.h
    @gcc -c livro.c -Wall -Werror -Wextra -g

► make filme.o
filme.o: filme.c filme.h
    @gcc -c filme.c -Wall -Werror -Wextra -g

► make produto.o
produto.o: produto.c produto.h
    @gcc -c produto.c -Wall -Werror -Wextra -g

► make lista.o
lista.o: lista.c lista.h
    @gcc -c lista.c -Wall -Werror -Wextra -g

► make main.o
main.o: main.c chocolate.h vinho.h livro.h filme.h produto.h lista.h
    @gcc -c main.c -Wall -Werror -Wextra -g

► make clean
clean:
    @rm -rf *.o teste
```

A primeira regra definida no Makefile é a regra run, que serve para executar o programa teste. Antes de rodá-lo, essa regra depende da regra teste, garantindo que o executável esteja atualizado antes de ser executado. A linha ./teste dentro dessa regra faz com que o programa compilado seja executado no terminal.

A regra teste é a principal do Makefile e tem a responsabilidade de gerar o executável final do projeto. Ela depende de diversos arquivos objeto (.o), como chocolate.o, vinho.o, livro.o, entre outros. Isso significa que, antes de criar o executável teste, todos esses arquivos precisam ser compilados. A linha gcc -o teste chocolate.o vinho.o livro.o filme.o produto.o lista.o main.o -g utiliza o compilador gcc para gerar o binário teste, unindo todos os arquivos objeto. A opção -g é utilizada para adicionar informações de depuração ao programa, facilitando a análise com ferramentas como o gdb.

Além da regra teste, o Makefile define regras individuais para compilar cada arquivo .c em um arquivo objeto .o. Por exemplo, a regra chocolate.o indica que esse arquivo depende de chocolate.c e chocolate.h. Se chocolate.c ou chocolate.h forem modificados, o Makefile recompilará chocolate.o. A linha gcc -c chocolate.c -Wall -Werror -Wextra -g é utilizada para compilar o código sem gerar o executável final. Os parâmetros -Wall, -Werror e -Wextra ativam avisos rigorosos, garantindo que o código esteja bem escrito e sem erros.

Esse mesmo padrão se repete para os demais arquivos do projeto, como vinho.o, livro.o, filme.o, produto.o e lista.o. Cada um deles possui sua regra específica, garantindo que apenas os arquivos modificados sejam recompilados, otimizando o tempo de compilação. Por fim, há a regra clean, que tem a função de limpar os arquivos gerados pela compilação. Essa regra contém o comando `rm -rf *.o` teste, que remove todos os arquivos objeto (.o) e o executável teste. Isso é útil para garantir que uma nova compilação seja feita do zero, sem resíduos de compilações anteriores.

10. Casos Teste:

- Teste 01:

Inicialmente, eu escolhi uma opção que não era possível.

```
./teste

      BEM VINDO A GIRLS STORE!
+=====PRODUTOS=====+
| [1] Chocolates          |
| [2] Vinhos              |
| [3] Livros              |
| [4] Filmes              |
| [5] Sair                |
+=====+
Digite uma opção: 10
```

Apareceu, para eu digitar uma opção válida:

```
Digite uma opção valida
```

Depois, eu selecionei a opção: Chocolate.

```
      BEM VINDO A GIRLS STORE!
+=====PRODUTOS=====+
| [1] Chocolates          |
| [2] Vinhos              |
| [3] Livros              |
| [4] Filmes              |
| [5] Sair                |
+=====+
Digite uma opção: 1
```

E fui em imprimir.

LISTA VAZIA!

Como esperado, deu lista vazia.

- Teste 02:

Continuando, eu fui em inserir e inserir consecutivamente dois produtos e imprimir eles.

```
NOME: Choco1
MARCA: ChocoBay
TIPO DE CHOCOLATE: Doce
PORCENTAGEM DE CACAU: -%
NACIONALIDADE OU ORIGEM: Brasileiro
PESO: 12g
ANO DE FABRICACAO: 12/2025
VALIDADE: 12/2026
```

```
NOME: Choco2
MARCA: ChocoBay
TIPO DE CHOCOLATE: Amargo
PORCENTAGEM DE CACAU: -%
NACIONALIDADE OU ORIGEM: Argetino
PESO: 12g
ANO DE FABRICACAO: 12/2025
VALIDADE: 12/2026
```

Por fim, eu fui eu seleccionei a opção undo e como esperado só apareceu o somente o Choco1.

```
NOME: Choco1
MARCA: ChocoBay
TIPO DE CHOCOLATE: Doce
PORCENTAGEM DE CACAU: -%
NACIONALIDADE OU ORIGEM: Brasileiro
PESO: 12g
ANO DE FABRICACAO: 12/2025
VALIDADE: 12/2026
```

Fui em undo, novamente e apareceu, Lista Vazia.

LISTA VAZIA!

Por fim, sai do programada e apareceu esse relatório.

```
+=====RELATÓRIO=====+
-----Estoque de Chocolate-----
LISTA VAZIA!
-----Estoque de Vinho-----
LISTA VAZIA!
-----Estoque de Livro-----
LISTA VAZIA!
-----Estoque de Filme-----
LISTA VAZIA!
+=====+
```

- Teste 03:

Inicialmente, eu a opção de filmes, depois escolhi a opção Inserir e digitei os valores e inseriu com sucesso.

```
Chave inserida com sucesso!
```

Depois, eu imprimir o que tinha digitado e procurei ele.

```
TITULO: Harry Potter
DIRETOR: Ana
GENERO: Ficcao
DISTRIBUIDORA: Inglesa
DURACAO: 120
PAIS DE ORIGEM: Inglesa
```

```
Insira uma chave para ser buscada: Harry Potter
PRODUTO Harry Potter ENCONTRADO!
```

Por fim, fiz Undo e deu certo:

```
LISTA VAZIA!
```

Depois, fiz Redo e voltou novamente.

```
TITULO: Harry Potter
DIRETOR: Ana
GENERO: Ficcao
DISTRIBUIDORA: Inglesa
DURACAO: 120
PAIS DE ORIGEM: Inglesa
```

Por fim, sai do programa.

E saiu imprimindo o relatório.

```

+=====RELATÓRIO=====+
-----Estoque de Chocolate-----
LISTA VAZIA!
-----Estoque de Vinho-----
LISTA VAZIA!
-----Estoque de Livro-----
LISTA VAZIA!
-----Estoque de Filme-----

TITULO: Harry Potter
DIRETOR: Ana
GENERO: Ficcao
DISTRIBUIDORA: Inglesa
DURACAO: 120
PAIS DE ORIGEM: Inglesa

+=====+
Obrigado pela preferencia! Nos vemos na próxima

```

- Teste 04:

Inicialmente, eu fui em Vinhos e adicionei dois vinhos, depois fui em Filmes e adicionei 1 também, depois fiz undo uma vez em vinho e sai do programa.

```

NOME: VinhoBranco
VINICOLA: Vini12
TIPO: Amargo
PAIS: Argentina
REGIAO: Norte
ANO DE FABRICACAO: 2025
UVA: aa

-----Estoque de Livro-----
LISTA VAZIA!
-----Estoque de Filme-----

TITULO: Harry Potter e o Calice de Fogo
DIRETOR: ana
GENERO: ficcao
DISTRIBUIDORA: inglesa
DURACAO: 120
PAIS DE ORIGEM: inglaterra

+=====+

```

Como podemos ver, o programa funcionou corretamente no relatório, só imprimiu 1 vinho e 1 filme.