

PROJETO:

GIRLS STORE

DEVELOPED BY:
ANA CLARA, FERNANDA E WINNIE

Produto.c (Genérico)

Struct

```
1 typedef struct produto{
2     void *dado;
3     void(*print)(void* dado);
4     int(*compara)(void* dado, void* chave);
5     void(*desaloca)(void* dado);
6     struct produto *next;
7 }Produto;
8
9 Produto* inicializaProduto();
```

Alocação de Memória

```
1  Produto* inicializaProduto(){
2      Produto* ProdutoNode = (Produto*)calloc(1,sizeof(Produto));
3      if(ProdutoNode==NULL){
4          printf("ERRO_DE_ALOCACAO!");
5          exit(1);
6      }
7
8      return ProdutoNode;
9  }
10
11 Produto* alocaProduto(void* dado, void(*print)(void *dado), int(*compara)(
12     void* dado, void* chave), void(*desaloca)(void* dado)){
13     Produto* ProdutoNode = (Produto*)calloc(1,sizeof(Produto));
14     if(ProdutoNode==NULL){
15         printf("ERRO_DE_ALOCACAO!");
16         exit(1);
17     }
18     ProdutoNode->dado = dado;
19     ProdutoNode->compara = compara;
20     ProdutoNode->print = print;
21     ProdutoNode->desaloca = desaloca;
22
23     return ProdutoNode;
24 }
```

Produto.c (Genérico)

- Funções de Impressão
- Função de Comparação
- Função de Liberação de Memória

```
1 void printProduto(Produto *P){  
2     P->print(P->dado);  
3 }  
4  
5 int comparaProduto(Produto *P, void* chave){  
6     return P->compara(P->dado, chave);  
7 }  
8  
9 void freeProduto(Produto *P){  
10    if(P!=NULL){  
11        P->desaloca(P->dado);  
12        free(P);  
13    }  
14 }
```

Pilha



```
1 typedef struct pilha{  
2     Produto* P;  
3     struct pilha *next;  
4 }Pilha;
```

Lista.c

Inicializa Pilha



```
1 Pilha* inicializaNoPilha(Produto *Lista){  
2     Pilha* pilha = malloc(sizeof(Pilha));  
3     pilha->next=pilha;  
4     pilha->P = Lista;  
5     return pilha;  
6 }  
7  
8 Pilha* inicializaPilha(){  
9     Pilha* pilha = malloc(sizeof(Pilha));  
10    pilha->next= pilha;  
11    pilha->P = NULL;  
12    return pilha;  
13 }
```

Inserir na Pilha



```
1 Pilha* inserePilha(Produto* Lista, Pilha* pilha){  
2     Pilha* newpilha=inicializaNoPilha(Lista);  
3     if(pilha == NULL){  
4         pilha=newpilha;  
5         pilha->next=newpilha;  
6     }  
7     else{  
8         newpilha->next = pilha->next;  
9         pilha->next = newpilha;  
10        pilha=newpilha;  
11    }  
12  
13    return pilha;  
14 }
```

Remove Pilha

```
1 Pilha* removePilha(Pilha* pilha){  
2     Pilha* auxpilha = pilha->next;  
3  
4     if (pilha == NULL) return NULL;  
5     if(auxpilha==pilha){  
6         free(auxpilha);  
7         return NULL;  
8     }  
9     Pilha* predpilha = auxpilha;  
10    while(predpilha->next!=pilha) predpilha = predpilha  
11        ->next;  
12        predpilha->next=auxpilha;  
13        free(pilha);  
14    return predpilha;  
15 }
```

Pilha

- Função para Excluir
- Funções Redo e Undo

```
1 void limpaRedo(Pilha **pilhaaux, Pilha **redo){  
2     if((*redo)==NULL) return;  
3     if((*pilhaaux) == NULL){  
4         *pilhaaux = (*redo);  
5         *redo = NULL;  
6     }  
7     else{  
8         Pilha* Predo = (*redo)->next;  
9         Pilha* Paux = (*pilhaaux)->next;  
10        (*redo)->next = Paux;  
11        (*pilhaaux)->next = Predo;  
12        *redo=NULL;  
13    }  
14}  
15}  
16  
17 //AO FAZER UNDO, PEGAMOS O ULTIMO ELEMENTO DA PILHA DE UNDO E INSERIMOS EM REDO, LOGO APÓS APAGAMOS O TOPO DA PILHA DE UNDO;  
18 Produto* undo(Produto* List, Pilha** undo, Pilha** redo){  
19     if (*undo == NULL) return List;  
20     Produto* new = (*undo)->p;  
21     *redo = inserePilha(List, (*redo));  
22     *undo = removePilha(*undo);  
23     return new;  
24 }
```

Pilha

- Função Redo

```
1 Produto* redo(Produto* List, Pilha** undo, Pilha** redo
){
2     if (*redo == NULL) return List;
3     Produto* new = (*redo)->P;
4     *undo = inserePilha(List, *undo);
5     *redo = removePilha(*redo);
6     return new;
7 }
```

Lista

- Função para Inserir

```
1 Produto* Insere(Produto *List, void *dados, void(*print)(void *dados), int(*compara)
2 )(void* dados, void* chave), void(*desaloca)(void* dados), Pilha** undo, Pilha **
3 redo, Pilha **auxpilha, void* chave){
4     Produto *node = alocaProduto(dados, print, compara, desaloca);
5     Produto* old = List;
6     Produto* new = NULL;
7     Produto *aux = List;
8     Produto *pred = NULL;
9     Produto *new_aux = NULL;
10
11
12    if(list == NULL){
13        new = node;
14        *undo = inserePilha(NULL, (*undo));
15        return new;
16    }
17    else{
18        *undo = inserePilha(old, (*undo));
19        while (aux) {
20            Produto *copia = alocaProduto(aux->dados, aux->print, aux->compara,
21 aux->desaloca);
```

Cont...

- Função para Inserir

```
1 if (new == NULL) {
2     new = copia;
3     new_aux = new;
4 } else {
5     new_aux->next = copia;
6     new_aux = new_aux->next;
7 }
8 if (aux->compara(aux->dado, chave) < 0) {
9     pred = copia;
10 } else {
11     break;
12 }
13 aux = aux->next;
14 }
15 if (pred == NULL) {
16
17     node->next = new;
18     new = node;
19 } else {
20
21     node->next = pred->next;
22     pred->next = node;
23 }
24
25 return new;
26
27 }
28 }
29 }
```

Lista

- Função de Exclusão

```
1 Produto* pop(Produto *list, void* chave, Pilha** undo, Pilha** redo, Pilha **  
auxpilha){  
2     Produto *auxnode = NULL;  
3     Produto *prednode = NULL;  
4     Produto* old = list;  
5     Produto* new = NULL;  
6     Produto* auxnew = NULL;  
7     auxnode = list;  
8  
9     Produto* node = NULL;  
10  
11    limpaRedo(auxpilha, redo);  
12  
13    *undo = inserePilha(old, (*undo));  
14    if (list == NULL){  
15        return NULL;  
16    }
```

Cont...

- Função de Exclusão

```
1 else{
2     while(auxnode!=NULL && (auxnode->compara(auxnode->dado, chave) != 0)){
3         prednode = auxnode;
4         if(prednode!=NULL){
5             node = alocaProduto(auxnode->dado, auxnode->print, auxnode->
6             compara, auxnode->desaloca);
7             if(new ==NULL){
8                 new = node;
9                 auxnew=new;
10            }
11            else{
12                while(auxnew->next!=NULL) auxnew=auxnew->next;
13                auxnew->next=node;
14            }
15            auxnode = auxnode->next;
16        }
17        if(prednode==NULL) new = auxnode->next;
18        else{
19            node->next=auxnode->next;
20        }
21    }
22    return new;
23 }
```

Lista

- Função de Busca



```
1 Produto *busca(Produto *list, void* chave){  
2     Produto *auxnode = list;  
3     while(auxnode!=NULL){  
4         if(auxnode->compara(auxnode->dado, chave)==0){  
5             return auxnode;  
6         }  
7         auxnode = auxnode->next;  
8     }  
9     return auxnode;  
10 }  
11 }
```

Lista

- Função de Impressão
- Função de Liberação de memória

```
1 void print_list(Produto *list){  
2     Produto *auxnode = list;  
3     if(list == NULL) {  
4         printf("LISTA VAZIA!\n");  
5         return;}  
6     printf("\n");  
7     while (auxnode){  
8         auxnode->print(auxnode->dado);  
9         printf("\n");  
10        auxnode = auxnode->next;  
11    }  
12 }  
13  
14 /*FUNÇÕES PARA A DESALOCAÇÃO DAS LISTAS E PILHAS */  
15 void freeLista(Produto* list){  
16     Produto *aux = list;  
17  
18     if(list==NULL) return;  
19  
20     while(list!=NULL){  
21         aux = list->next;  
22         freeProduto(list);  
23         list = aux;  
24     }  
25  
26 }
```

Pilha

- Função de Liberação de Memória

```
1 void freePilha(Pilha** pilha) {  
2     if (*pilha == NULL) return;  
3  
4     Pilha* atual = *pilha;  
5     Pilha* prox;  
6     Pilha* primeiro = *pilha;  
7  
8     do {  
9         prox = atual->next;  
10        freeLista(atual->P);  
11        atual->P = NULL;  
12        free(atual);  
13        atual = prox;  
14    } while (atual != primeiro);  
15  
16    *pilha=NULL;  
17 }
```

Chocolate (Departamento)



```
1  typedef struct Chocolate{
2      char* nome;
3      char* marca;
4      char* tipo;
5      char* porcentagem;
6      char* origem;
7      char* peso;
8      char* ano_de_fabricacao;
9      char* validade;
10 }chocoNode;
```

Chocolate (Departamento)

- Função de Alociação

```
1 #define MAX_TAM 100
2
3
4 chocoNode* alocaChoco(char *nome, char* marca, char* tipo, char* porcentagem,
5   char* origem, char* peso, char* ano_de_fabricacao, char* validade){
6   chocoNode* novo = (chocoNode*)calloc(1,sizeof(chocoNode));
7   if(novo==NULL){
8     printf("ERRO_DE_ALOCACAO\n");
9     exit(1);
10 }
11 novo->nome = (char*)malloc(sizeof(char) * (strlen(nome) + 1));
12 novo->marca = (char*)malloc(sizeof(char) * (strlen(marca) + 1));
13 novo->tipo = (char*)malloc(sizeof(char) * (strlen(tipo) + 1));
14 novo->origem = (char*)malloc(sizeof(char) * (strlen(origem) + 1));
15 novo->validade = (char*)malloc(sizeof(char) * (strlen(validade) + 1));
16
17 novo->porcentagem = (char*)malloc(sizeof(char) * (strlen(porcentagem) + 1));
18 novo->peso = (char*)malloc(sizeof(char) * (strlen(peso) + 1));
19 novo->ano_de_fabricacao = (char*)malloc(sizeof(char) * (strlen(
20   ano_de_fabricacao) + 1));
21
22 strcpy(novo->nome, nome);
23 strcpy(novo->marca, marca);
24 strcpy(novo->tipo, tipo);
25 strcpy(novo->origem, origem);
26 strcpy(novo->validade, validade);
27 strcpy(novo->porcentagem, porcentagem);
28 strcpy(novo->peso, peso);
29 strcpy(novo->ano_de_fabricacao, ano_de_fabricacao);
30
31 }
```

Chocolate (Departamento)

- Função de Leitura

```
1 chocoNode* leChoco(){
2     char nome[MAX_TAM];
3     char marca[MAX_TAM];
4     char tipo[MAX_TAM];
5     char porcentagem[MAX_TAM];
6     char origem[MAX_TAM];
7     char peso[MAX_TAM];
8     char ano_de_fabricacao[MAX_TAM];
9     char validade[MAX_TAM];
10
11    getchar(); // Limpa qualquer '\n' no buffer antes da primeira entrada
12
13    printf("NOME: ");
14    scanf(" %[^\n]*c", nome);
15    printf("\nMARCA: ");
16    scanf(" %[^\n]*c", marca);
17    printf("\nTIPO DE CHOCOLATE: ");
18    scanf(" %[^\n]*c", tipo);
19
20    if(strcmp(tipo, "amargo") == 0){
21        printf("\nPORCENTAGEM DE CACAU: ");
22        scanf(" %[^\n]*c", porcentagem);
23    }
24
25    else strcpy(porcentagem, "-");
26
27
28}
```

Chocolate (Departamento)

- Função de Leitura
- Função de Comparação

```
1 printf("\nNACIONALIDADE OU ORIGEM: ");
2     scanf(" %[^\n]*c", origem);
3
4 printf("\nPESO: ");
5     scanf(" %[^\n]*c", peso);
6 //getchar(); // Limpa o '\n' deixado pelo scanf de número
7
8 printf("\nANO DE FABRICACAO: ");
9     scanf(" %[^\n]*c", ano_de_fabricacao);
10 //getchar(); // Limpa o '\n' deixado pelo scanf de número
11
12 printf("\nVALIDADE: ");
13     scanf(" %[^\n]*c", validade);
14
15     return alocaChoco(nome, marca, tipo, porcentagem, origem, peso,
16     ano_de_fabricacao, validade);
17 }
18
19 int comparaChoco(void* dado, void* chave) {
20     chocoNode* d = (chocoNode*)dado;
21     char* c = (char*)chave;
22
23     return strcmp(d->nome, c);
```

Chocolate (Departamento)

- Função de Impressão

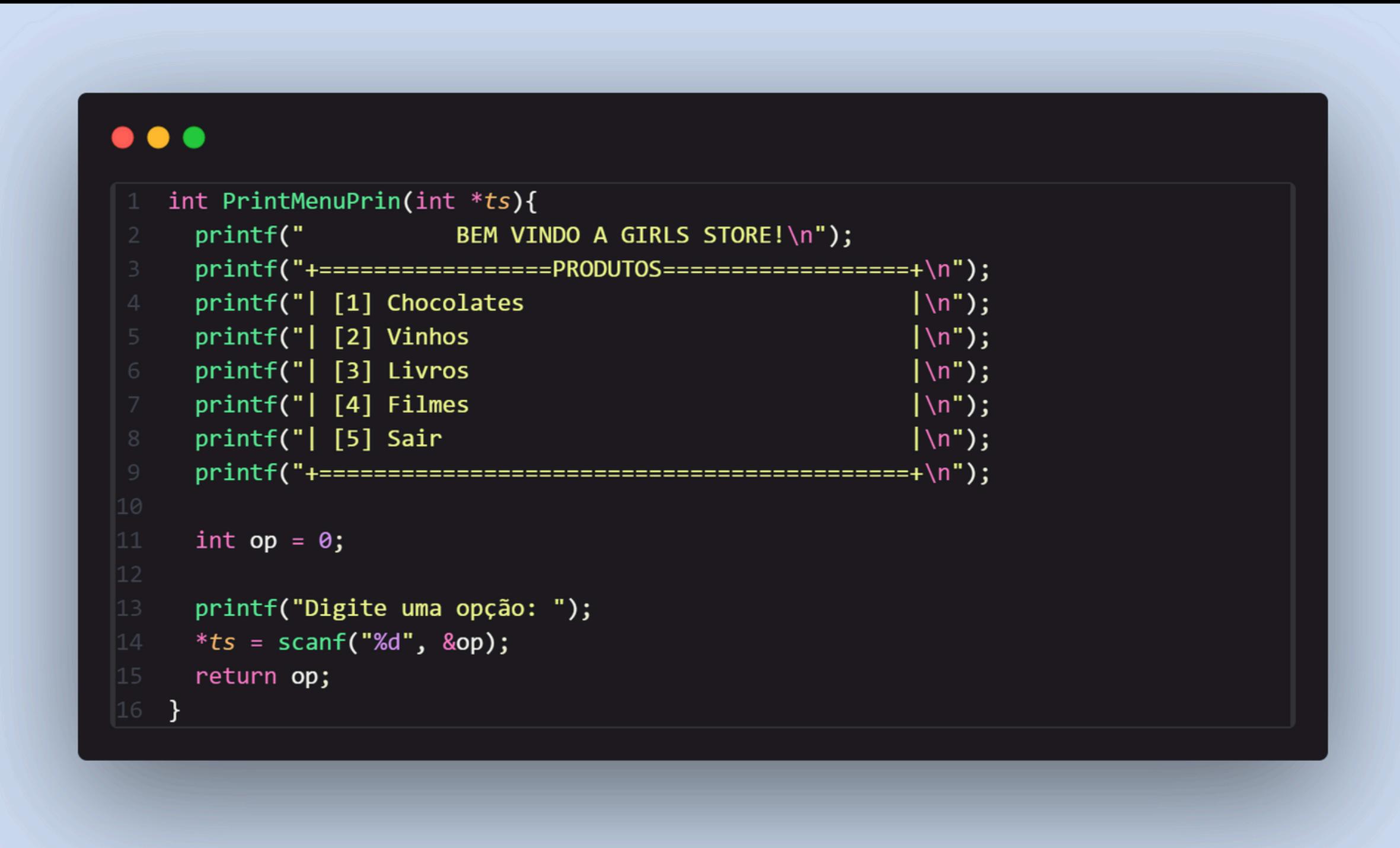
```
1 void imprimeChoco(void* dado){  
2     chocoNode* node = (chocoNode*)dado;  
3     printf("NOME: %s\n", node->nome);  
4     printf("MARCA: %s\n", node->marca);  
5     printf("TIPO DE CHOCOLATE: %s\n", node->tipo);  
6     printf("PORCENTAGEM DE CACAU: %s%%\n", node->porcentagem);  
7     printf("NACIONALIDADE OU ORIGEM: %s\n", node->origem);  
8     printf("PESO: %sg\n", node->peso);  
9     printf("ANO DE FABRICACAO: %s\n", node->ano_de_fabricacao);  
10    printf("VALIDADE: %s\n", node->validade);  
11}  
12}
```

Chocolate (Departamento)

- Função de Liberação de memória

```
1 void freeChoco(void *dados){  
2     chocoNode* node = (chocoNode*)dados;  
3     if(node!=NULL){  
4         free(node->nome);  
5         free(node->marca);  
6         free(node->tipo);  
7         free(node->porcentagem);  
8         free(node->peso);  
9         free(node->ano_de_fabricacao);  
10        free(node->origem);  
11        free(node->validade);  
12        free(node);  
13    }  
14 }
```

Main



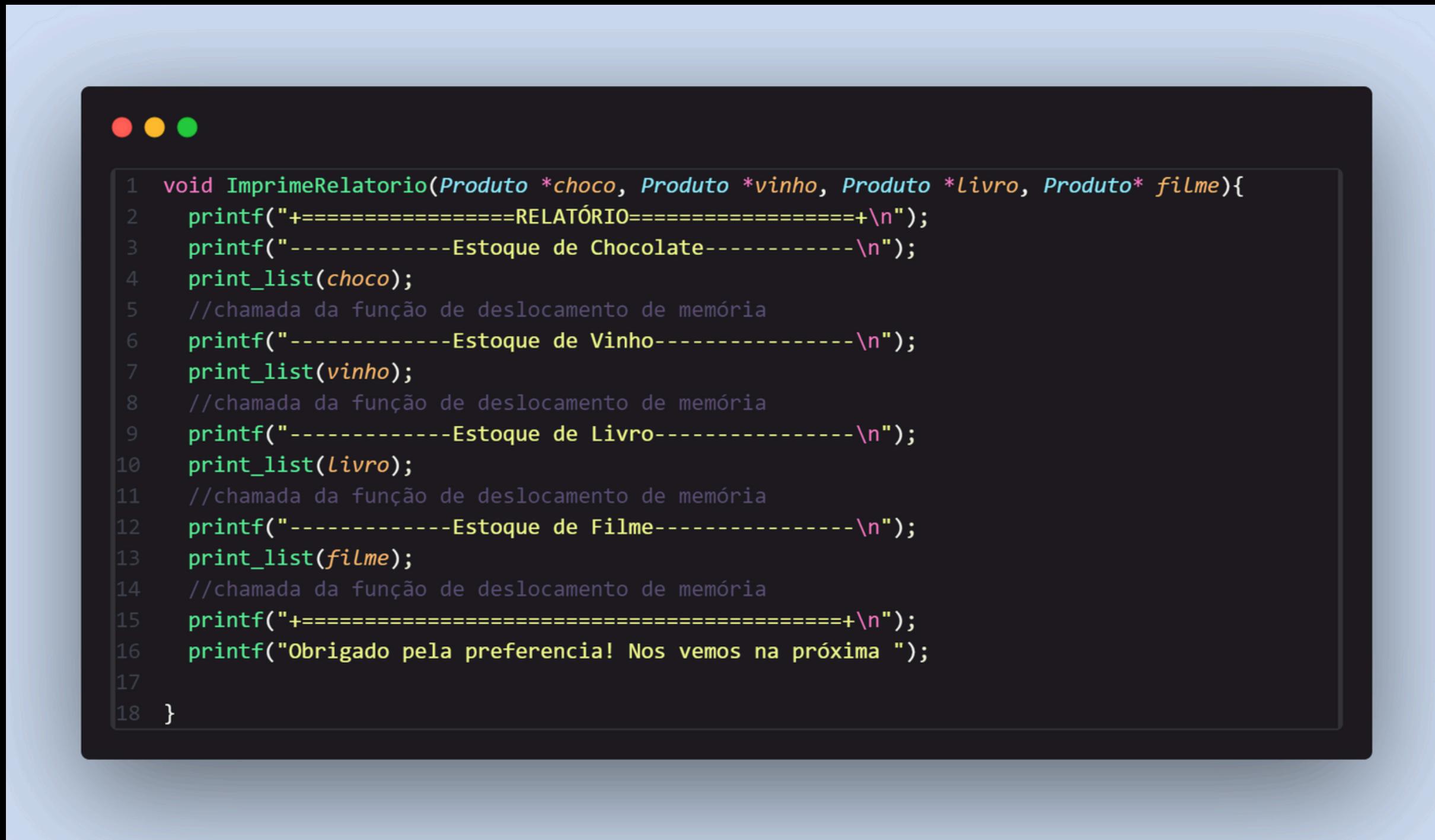
A screenshot of a terminal window on a Mac OS X desktop. The window has a dark theme with red, yellow, and green window control buttons at the top left. The terminal itself is black with white text. It displays a C program with line numbers from 1 to 16. The code defines a function `PrintMenuPrin` that prints a menu for a store. The menu includes options for chocolates, wines, books, movies, and exiting. The code uses `printf` for output and `scanf` for input.

```
1 int PrintMenuPrin(int *ts){
2     printf("          BEM VINDO A GIRLS STORE!\n");
3     printf("+=====PRODUTOS=====+\n");
4     printf("| [1] Chocolates      |\n");
5     printf("| [2] Vinhos          |\n");
6     printf("| [3] Livros          |\n");
7     printf("| [4] Filmes          |\n");
8     printf("| [5] Sair             |\n");
9     printf("+=====+\n");
10
11    int op = 0;
12
13    printf("Digite uma opção: ");
14    *ts = scanf("%d", &op);
15    return op;
16 }
```

Main

```
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 }
```

Main



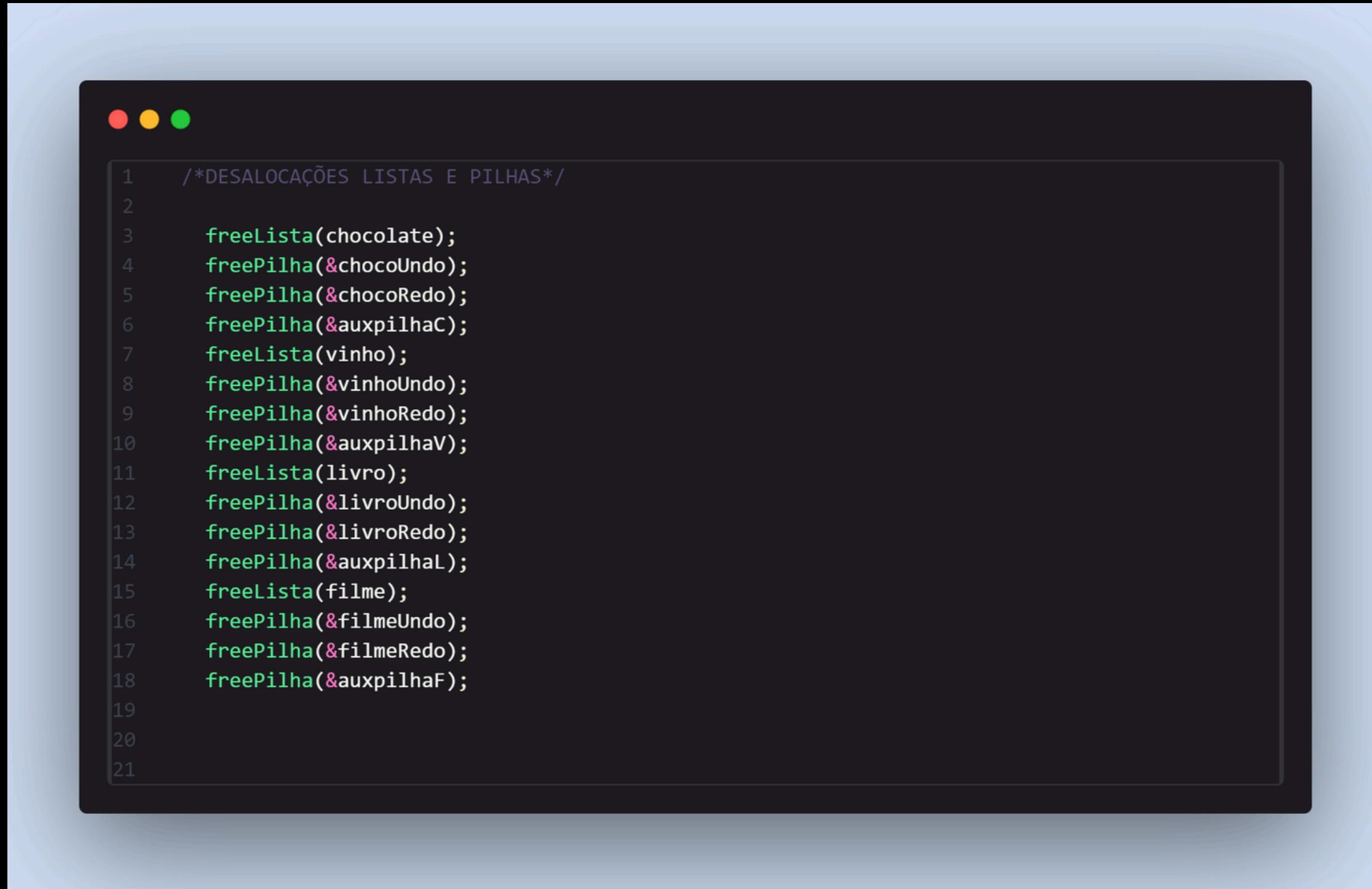
The image shows a terminal window with a dark background and light-colored text. At the top left are three small colored circles: red, yellow, and green. The terminal displays the following C code:

```
1 void ImprimeRelatorio(Produto *choco, Produto *vinho, Produto *Livro, Produto* filme){
2     printf("=====RELATÓRIO=====+\n");
3     printf("-----Estoque de Chocolate-----\n");
4     print_list(choco);
5     //chamada da função de deslocamento de memória
6     printf("-----Estoque de Vinho-----\n");
7     print_list(vinho);
8     //chamada da função de deslocamento de memória
9     printf("-----Estoque de Livro-----\n");
10    print_list(Livro);
11    //chamada da função de deslocamento de memória
12    printf("-----Estoque de Filme-----\n");
13    print_list(filme);
14    //chamada da função de deslocamento de memória
15    printf("=====+\n");
16    printf("Obrigado pela preferencia! Nos vemos na próxima ");
17
18 }
```

Main

```
1 do{
2
3     mudarCorTexto();
4     op1 = PrintMenuPrin(&ts);
5     if (ts != 1) { // Verifica se a entrada não é um número
6         limparTela();
7         printf("Entrada inválida! Digite um número válido.\n");
8         while (getchar() != '\n'); // Limpa o buffer de entrada
9         continue; // Volta para o início do loop sem executar o switch
10    }
11    limparTela();
12
13    switch (op1) {
14        case 1:
15            do {
16                ImprimeMenuChoco();
17                printf("Digite uma opção: ");
18
19                if (scanf("%d", &op2) != 1) { // Verifica se a entrada não é um número
20                    printf("Entrada inválida! Digite um número válido.\n");
21                    while (getchar() != '\n'); // Limpa o buffer de entrada
22                    continue; // Volta para o início do loop sem executar o switch
23                }
24            }
```

Main



The image shows a screenshot of a macOS terminal window. The window has a dark theme with red, yellow, and green title bar buttons. The code displayed is:

```
1  /*DESLOCAMENTOS 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
```

OBIGADA