DOCUMENTAÇÃO DO SOFTWARE

Gerenciamento de locadora de filmes em C

IFMG – Campus Formiga 2018

Thiago Henrique Domingues Botelho

## Sumário

[Bibliotecas 3](#_Toc531344892)

[Estruturas 4](#_Toc531344893)

[Alocação de memória 4](#_Toc531344894)

[Administrativo 5](#_Toc531344895)

[Entrada de filmes (void *entradafilmes*( )) 5](#_Toc531344896)

[Locação de filmes (void *locacaofilmes*( )) 6](#_Toc531344897)

[Devolução de filmes (void *devolucaofilmes*( )) 6](#_Toc531344898)

[Verificar se o filme já pagou com as locações 6](#_Toc531344899)

[Gestão 7](#_Toc531344900)

[Edição de dados (void *edicaotipodado*\_campo( )) 7](#_Toc531344901)

[Exclusão de dados (void *excluirtipodado* ( )) 8](#_Toc531344902)

[Cadastros 8](#_Toc531344903)

[Finanças 9](#_Toc531344904)

[Contas a pagar (void *contaspagar\_filtro*( )) 9](#_Toc531344905)

[Contas a receber (void *receber\_contascl*( )) 9](#_Toc531344906)

[GUI (Graphic User Interface ) 9](#_Toc531344907)

[Menus 10](#_Toc531344908)

[Relatório (Feedback) 10](#_Toc531344909)

[Exportação 10](#_Toc531344910)

[Arquivo texto e binário 11](#_Toc531344911)

[Arquivo XML 11](#_Toc531344912)

[Importação de dados 12](#_Toc531344913)

[Arquivo texto 12](#_Toc531344914)

[Arquivo binário 13](#_Toc531344915)

[Verificação de dados 14](#_Toc531344916)

## Bibliotecas

Para melhor organização e manutenção do código fonte o projeto foi divido por bibliotecas. Foi criada um total de 12 bibliotecas diferentes onde foi fora agrupados funções cuja seu funcionamento e finalidade são semelhantes. Na tabela a seguir é possível visualizar todas as biblioteca e um breve resumo sobre ela. Explicações mais detalhadas serão apresentadas mais adiante no documento.

|  |  |
| --- | --- |
| NOME | CONTEÚDO |
| adminstracao.h | Funções da parte administrativa do estabelecimento |
| alocação.h | Funções de alocação de memoria |
| cadastros.h | Funções que recebem entradas de dados |
| fileBIN.h | Funções sobre a importação e exportação de dados em formato binário |
| fileTXT.h | Funções sobre a importação e exportação de dados em formato de arquivo de texto |
| fileXML.h | Funções sobre exportação de dados em formato XML |
| finanças.h | Funções que controlam a parte financeira do estabelecimento |
| gestão.h | Funções de edição e exclusão de dados |
| menus.h | Funções de menus e submenus |
| structs.h | Estruturas utilizadas no código |
| verificacaoDeDados.h | Funções que verificam se os dados estão corretas |
| GUI.h | Funções relacionadas a interface |

## Estruturas

Todas as estruturas utilizadas na construção do projeto estão localizadas na biblioteca ***struct.h***.. Estruturas contidas nesta biblioteca possui a seguinte declaração padrão:

1. **typedef** **struct** {
2. tipo campo1;
3. tipo campo2;
5. ...
7. } Strc\_TipoStruct;  P

## Alocação de memória

Todas as funções relacionadas a alocação de memória estão na biblioteca ***alocação.h***. Todos os tipos de dado são inicializados com um ponteiro do tipo NULL e um contador do tipo inteiro-estático iniciado em 0 e declarado como uma variável global. Este contador é utilizado para duas finalidades: controlar a quantidade de memoria solicitada pelos realocar e para controle de leitura em laços de repetição do tipo ***for***. Todas as funções de alocação possui o seguinte esqueleto.

1. **Strc\_tipoStruct**\* VariavelGlobal = NULL;
2. **int** **static** contTipo = 0;
3. **int** alocarTipo\_Dado(Strc\_tipoStruct\* ponteiro\_da\_struct) {
4. **if** (VariavelGlobal == NULL) {
5. Strc\_tipoStruct = malloc(**sizeof** (**Strc\_tipoStruct**));
6. } **else** {
7. ponteiro\_da\_struct=realloc(ponteiro\_da\_struct, (contTipo + 1) \* **sizeof** (**Strc\_tipoStruct**));
8. }
10. \*(VariavelGlobal+ contTipo) = \*ponteiro\_da\_struct;
11. Strc\_tipoStruct = contFilmes\_comprados;
13. **if** (VariavelGlobal == NULL) {
14. printf("Ocorreu um erro durante a alocação, \n");
15. exit(EXIT\_FAILURE);
16. }
18. contTipo++;
19. }

Além das funções de alocação de memoria, a biblioteca ***alocação.h*** possui também funções para manipulação desses dados. Os vetores alocados originalmente e seus respectivos contadores podem ser retornados para que sejam utilizados por outras funções. Essas duas funções possuem o seguinte esqueleto.

1. //Retornar vetor original
2. **Strc\_tipoStruct**return\_TipoDado() {
3. **return** VariavelGlobal;
4. }
6. //Retorno contador
7. **int** returnContTipo() {
8. **return** contTipo;
9. }

E caso esses vetores sejam alterados internamente em uma função e deseja que essas alterações reflitam nos vetores originais é possível realizar esta tarefa com as funções de alteração que possuem o seguinte esqueleto:

1. **void** alterarTipo(**Strc\_tipoStruct**\* ponteiro\_da\_struct) {
2. vetor\_original = ponteiro\_da\_struct;
3. }

## Administrativo

Todas as funções responsáveis pela parte administrativa do estabelecimento estão presentes na biblioteca ***adminstracao.h***. Sendo elas: entrada de novos filmes, locação e devolução de filmes, visualização do estoque do estabelecimento, geração de código únicos de notas fiscais e os filtros de verificação se determinado já foi pago com as locações. Segue a hierarquia de chamada das funções:



# Entrada de filmes (void *entradaFilmes*( ))

Esta função é responsável pela compra de novos títulos para o estabelecimento. Entradas de novos filmes são realizadas apenas por funcionários previamente já cadastrados no sistema. Além de pelo menos um funcionário já cadastrado é necessário também ter pelo menos um fornecedor e obviamente ter no mínimo um filme já cadastrado.

Os filmes comprados serão armazenados no campo **filmesComprados** da estrutura Locadora. Para economizar recursos é armazenados somente os códigos, sendo assim um vetor do tipo inteiro dinâmico. Na Após as entradas serem finalizadas, um menu com as opções de confirmação ou correção destes dados.. Após a confirmação, a nota fiscal da compra gerada, contendo os seguintes campos:

* **codForn:** Código do fornecedor;
* **código:** Código único da nota;
* **contItens:** Quantidade de itens comprados;
* **paga:** 
  + paga = 1, nota já foi paga;

paga = 0, nota em aberto;

* **itens:**
  + **codiFilme:** Armazena o código do filmes;
  + **quant:** Quantidade de exemplares;
  + **preco:** Preço de compra do filme;
  + **total:** Total da compra, que é calculado a partir dado como total = (preco de compra \* quantidade copias).
* **precoFrete:** Preço total do frete;
* **precoImposto:** Preço total dos impostos;
* **freteUnidade:**  Frete por mercadoia, é dado a partir de precoFrete / total de copias;
* **ImpostoUnidade:** Frete por mercadoia, é dado a partir de precoImposto / total de copias;
* **totaNF**: Valor total da venda;

O valor total da compra e a soma quantidade de exemplares pedidos serão armazenados nas variáveis ***somatorio*** e ***totalCopias*** respectivamente.

# Locação de filmes (void *locacaoFilmes*( ))

A locação de filmes é realizada apenas por funcionários previamente cadastrados no sistema. Após entrar com o código do funcionário, cliente e do filme, será solicitado a quantidade de exemplares no qual a função **verificarExemplares\_Filmes** (**int** id, **int** quant) que recebem como parâmetros o código do filme e a quantidade de filmes desejada. Caso a quantidade for suficiente será retornado 1, se não será retornado 0. Lembrando que o programa não continuará enquanto a quantidade de exemplares for valida.

As informações dos filmes selecionados serão armazenados em um vetor dinâmico do tipo **Strc\_MinimalFilmes** chamado de **FilmesLocados**para que possam ser lidas e trabalhadas posteriormente.. Assim que todas as entradas forem feitadas será mostarda a opção de confirmação ou correção destes dados.. A quantidade de exemplares será retirada do estoque da locadora.

Todas as funções relacionadas a pagamento estão presentes na função **PagamentoLocacao**da biblioteca **finanças*.*h.** Dentro desta função será determinada a forma de pagamento desejada: a vista ou a prazo. Caso a opção escolhida for a vista o valor total do aluguel será adicionado ao caixa. E se for a prazo, o cliente pode optar por dar ou não uma entrada inicial. Caso algum valor de entrada seja dado,, o mesmo ira direto para o caixa e o restante pode ser parcelado em até 3 vezes. Todas as contas pendentes estão disponíveis no meu contas a receber.

# Devolução de filmes (void *DevolucaoFilmes*( ))

A função de devolução de filmes é bastante simples, consiste em ler o código do cliente e verificar se ele possui filmes locados e se sim com, quantos dias de atraso. Em seguida todos os exemplares são devolvidos ao estoque.

# Verificar se o filme já pagou com as locações

Essas funções realizam o calculo de quantos o filme rendeu de lucro com as locações. O calculo é realizado da seguinte forma: **quantidade de vezes que o filme foi alugado \* preço do aluguel**. Caso o resultado seja maior que o custo da compra então é retornada a mensagem superávit, caso contrario a mensagem déficit é retornada.

Essa função pode ser filtrada pelas seguintes maneiras:

* Imprimir todos os filmes comprados;
* Imprimir somente os filmes que já foram pago;
* Imprimir somente os filmes em déficit;
* Pesquisa por determinado filme através do código;
* Pesquisa entre faixa de código (1 a 10 por exemplo).

## Gestão

Essa biblioteca possui apenas dois tipo de funções: as de exclusão e de edição de dados.

# Edição de dados (void *edicaoTipoDado*\_campo( ))

Os tipo dados disponíveis para edições são: clientes, filmes, categorias, funcionários e fornecedores. As edições podem ser realizadas por campos separadamente ou de todas de uma única vez. As funções de edição de dados possui o seguinte esqueleto declaração:

1. /\* @param: posição que deseja realizar a edição \*/
2. **void** edicaoTipo\_campo(**int** i);

Para acessar estas funções foi criado um menu para cada tipo de estrutura . O menu abaixo é da estrutura de categorias.

1. **int** edicaoCategoria() {
2. ...
3. ...
4. ...
5. ...
6. printf("\nO que deseja editar: \n"
7. "\t1. Nome \n"
8. "\t2. Descrição \n"
9. "\t3. Valor da multa \n"
10. "\t4. Tudo \n"
11. "\t5. Voltar ao menu anterior \n");
13. **switch** (selecao()) {
14. **case** 1:
15. edicaoCategoria\_nome(i);
16. **break**;
18. **case** 2:
19. edicaoCategoria\_descricao(i);
20. **break**;
22. **case** 3:
23. edicaoCategoria\_valor(i);
24. **break**;
26. **case** 4:
27. edicaoCategoria\_tudo(i);
28. **break**;
30. **case** 5:
31. menuGestao(i);
32. **break**;
34. **default**:
35. printf("Opção inválida. \n");
37. }
38. } **else** {
39. printf("Código inválido. \n");
40. }
41. }

# Exclusão de dados (void *excluirTipoDado* ( ))

Os tipos de dados de dados que podem ser excluídos são os mesmo que podem ser editados, sendo eles: clientes, filmes, categorias, funcionários e fornecedores. Como todas funções possuem funcionamento semelhante , explicando uma genérica automaticamente estará explicando as demais Para exemplificar o funcionamento das funções de exclusão segue um exemplo minimalista porem com logica de funcionamento semelhante. Seja o seguinte vetor:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| **0** | **1** | **2** | **3** |

* Suponha que excluímos a segunda posição ( **i** ). Ficando assim:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| **0** | **1** | **2** | **3** |

* Para não ficar um espaço vazio no meio do vetor temos que mover todos os demais uma posição para atrás.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 4 |  |
| **0** | **1** | **2** | **3** |

* Agora basta subtraímos uma unidade do tamanho original do vetor, tendo assim o seguinte vetor final:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 4 |
| **0** | **1** | **2** |

Escrevendo a operação acima pode ser escrita da seguinte maneira em código.

1. **do** {
2. Categoria[i] = Categoria[i + 1];
3. i++;
4. } **while** (i < contTipo);

## Cadastros

Todas as funções relacionadas com a entrada e dados estão na biblioteca cadastro.h e possuem o seguinte esqueleto. Ao final de cada novo cadastrado, os dados serão enviados para o vetor global presente em alocação.c no qual será utilizado em outras partes do código.

1. **void** cadastrarDado() {
2. **int** opcao;
4. Strc\_TipoDado Variavel;
5. ...
6. ...
7. ...
8. ...
10. /\*
11. \* Exportação de dados de acordo com a opção selecionada no inicio do
12. \* programa. Lembrando que:
13. \* returnModoArmazenamento == 1; Dados serao exportados em arquivo texto
14. \* returnModoArmazenamento == 2; Dados serao exportados em arquivo binario
15. \*/
16. **if** (returnModoArmazenamento() == 1) {
17. exportCliente\_txt(Cliente);
18. }
20. /\* Realiza a exportação para os vetores globai \*/
21. alocarDado(&Variave);
22. }

Além das funções de cadastros, essa biblioteca possui também funções para geração de códigos únicos para cada novo entrada. Esse códigos se iniciam em 0 e são incrementados em 1 cada vez que um dado do tipo é cadastrado. Essas funções utilizam variáveis estáticas do tipo inteiro para controle desses códigos e possui o seguinte esqueleto:

1. **int** gerarDado() {
2. **int** **static** codigo = 0;
4. codigo++;
5. **return** codigo;
6. }

## Finanças

Todas funções relacionadas com a parte financeira do estabelecimento como as de contas a pagar de fornecedores e contas a receber de clientes.

# Contas a pagar (void *contasPagar\_filtro*( ))

O pagamento de notas fiscais pode ser realizada de três maneiras: uma única nota de cada vez, pelo código do fornecedor ou todas de uma única vez. As parcelas são pagas uma de cada vez, e o valor do pagamento é descontado diretamente do caixa e é subtraído uma unidade do total de parcelas restantes contidas dentro da estruturas de notas fiscais.

# Contas a receber (void *receber\_ContasCL*( ))

Como as funções de conta as pagar o recebimento de contas é realizada em uma a uma. Para realizar o recebimento de conta basta entrar com o código do cliente que em seguida ira fazer a verificação de se este cliente possui alguma conta pendente e em caso de haver, o valor da parcela da parcela será adicionado ao caixa e será subtraído um do total de parcelas daquele cliente. E por fim será mostrada a quantidade de parcelas que ainda restam para pagar. E em caso de o cliente não possuir nenhum conta pendente será imprimido uma mensagem de erro.

## GUI (Graphic User Interface )

Em português interface gráfica do usuário. Nesta biblioteca de mesmo nome (GUI.h) esta todas as funções relacionadas a interface; Em suma todos as saídas de dados estão nesta biblioteca. Sendo eles: locadora, categorias, clientes, funcionários, fornecedores, filmes, notas fiscais, filmes alugados, contas a receber e se o filme já se pagou com as locações.

## Menus

Esta biblioteca contém todos os menus e submenus utilizados para navegação pelas operações disponíveis do programa. Todos os menus possuem a mesma estrutura. Portanto não há necessidade explicar todos. As funções de menus possuem a seguinte estrutura:

1. **void** menu\_generico() {
2. printf("====== | MENU | ======\n"
3. "\t1. Opção 1 \n"
4. "\t2. Opçao 2 \n"
5. "\t3. Opção 3 \n
6. ...
7. ...
8. ...
9. "\t4. Voltar ao menu anterior \n");
10. **do** {
11. **switch** (selecao()) {
12. **case** 1:
13. funcao1();
14. **break**;
16. **case** 2:
17. funcao2();
18. **break**;
20. **case** 3:
21. funcao3();
23. **case** 4:
24. menuAnterior();
25. **break**;
27. **default**:
28. printf("Opção inválida. \n");
29. }
30. **break**;
31. } **while** (1);
32. }

## Relatório (Feedback)

Todas os dados de entrada do programa podem ser visualizados através das funções da biblioteca **filtrosRelatorios.h**. Por padrão todas as funções de filtros possuem a seguinte declaração **void filTipo\_FiltroImplementado.** Essa biblioteca contém todas as funções relacionada a filtros de busca por campos. Cada tipo de dado possui seus campos únicos. Como o funcionamento dessas funções são praticamente idênticos e simples, não será entrado em detalhes aqui na documentação para não se estender demais.

## Exportação

Para que os dados cadastrados possam continuar armazenados mesmo quando o encerramento do programa, foi incluída a opção de exportação. Os dados podem ser exportados em três formatos distintos, sendo: arquivo texto, arquivo binário e XML. Os arquivos exportados são salvos na pasta “arquivos/tipo” que pode ser acessada através da pasta home do projeto.

# Arquivo texto e binário

Arquivos no formato texto são salvos em “**arquivos/txt**”. A separação de dados neste arquivos possuem um padrão. Este padrão consiste em cada nova entrada se inicia com uma “**#**” e os campos são separados por virgulas e caso essa estrutura possuir um vetor como campo o mesmo é inicado com “**$”.** Isso pode ser observado mais facilmente abaixo:

|  |
| --- |
| **#, campo1,campo2,campo3, ..** |
| **$, campoV1,campoV2,campoV3 ...**  Vale ressaltar que dados em arquivo texto são armazenados ao final de cada entrada ao contrário de binário que é realizado tudo de uma única vez. Ambas as funções exportam os dados integralmente isto é, todos os dados do programa, não possuindo a opção de escolha de quais desejam exportar ou não.  **Esqueleto da função de exportação de dados em formato texto.**   1. **void** exportTipo\_txt(Strc\_TipoDado  var) { 2. **FILE** \* file = fopen("arquivos/nomeArquivo.txt", "a"); 4. //Verificação se o arquivo foi criado corretamente 5. **if** (file == NULL) { 6. printf("ERRO! \n"); 7. exit(EXIT\_FAILURE); 8. } 10. //Escrita no arquivo 11. fprintf(file, "#, %s,", var.campo1); 12. fprintf(file, "%d,", var.campo2); 13. fprintf(file, "%d,", var.campo3); 14. fprintf(file, "%d,", var.campo4); 15. }   **Esqueleto da função de exportação de dados em formato binário.**   1. **void** exportTipo\_bin() { 2. **FILE** \* file = fopen("arquivos/bin/nomeArquivo.bin", "wb"); 4. **if** (file == NULL) { 5. printf("ERRO! \n"); 6. exit(EXIT\_FAILURE); 7. } 8. Strc\_TipoDado\* var = return\_Vetor (); 9. **for** (**int** i = 0; i < returnCont\_tipoAlocado(); i++) { 10. fwrite(&va[i], **sizeof** (Strc\_TipoDado), 1, file); 11. } 13. fclose(file); 14. free(file); 15. } |

# Arquivo XML

Ao contrario de arquivos de texto e binário, a exportação em XML pode ser realiza por partes ou seja, o usuário pode escolher quais dados serão ou não exportados. Para ter acesso as opções de exportação em XML basta ir na opção 7 do menu principal, chamada de ”Exportação de dados no formato XML”. Após selecionar tal opção sera aberto um menu no qual o usuário seleciona quais tabelas deseja exportar. . Os arquivos XML gerados são salvos na em “**arquivos/xml”** e possuem a seguinte formatação.

1. <dados>
2. **<**tabela**tabela**=cliente**>**
3. **<registro>**
4. **<codigo>**1**</codigo>**
5. **<nome>**Fulano de Tal**</nome>**
6. **<endere**ço**>**Rua XYZ, 123, Bairro QWER, Formiga, MG**</endere**ço**>**
7. **<cpf>**111.111.111-11**</cpf>**
8. **<telefone>**3733213321**</telefone>**
9. **<email>**fulanodetal@fulanodetal.com.br**</email>**
10. **<sexo>**M**</sexo>**
11. **<est\_civil>**Casado**</est\_civil>**
12. **<data\_nascimento>**11/11/1988**</data\_nascimento>**
13. **</registro>**
14. **<registro>**
15. **<codigo>**2**</codigo>**
16. **<nome>**Ciclana Beltrano**</nome>**
17. **<endere**ço**>**Rua lkjdsfu, 1365423, Bairro qk2u3yi, Cidade, MG**</endere**ço**>**
18. **<cpf>**111.111.111-12**</cpf>**
19. **<telefone>**3733646821**</telefone>**
20. **<email>**ciclanafulano@ ciclanofulano.com.br**</email>**
21. **<sexo>**F**</sexo>**
22. **<est\_civil>**Casada**</est\_civil>**
23. **<data\_nascimento>**11/05/1958**</data\_nascimento>**
24. **</registro>**
25. **<registro>**
26. (...)
27. **</registro**
28. **</tabela>**

**IMPORTANTE:** Arquivos XML não podem ser importados pelo programa, nesse caso a exportação neste formato é utilizada somente para aplicações externas.

## Importação de dados

A opção de importação de um ou mais dados salvos em arquivo é dada durante a inicialização do software no qual a função **inicialização( )** é invocada. Esta função da ao usuário quatro opções: importar de arquivo, importar de arquivo binário, apagar todos os arquivos texto ou binários salvos anteriormente. Caso a opção de importação seja escolhida o usuário pode escolher quais dados deseja importar ou se deseja todos.

# Arquivo texto

O funcionamento das funções de importação de dados em arquivo texto se baseia em ler cada linha do arquivo e dividi-la em token’s menores que serão armazenados em cada campo. Como as funções de importação são maiores e mais complexas, a descrição de seu funcionamento foi realizada na forma de comentários dentro da própria função como pode ser visualizada no esqueleto abaixo:

1. **void** importDados\_txt() {
2. **FILE** \* file = fopen("arquivos/aqruivoGenerico.txt", "r");
4. /\* Verificação se foi possivel ler o arquivo \*/
5. **if** (file == NULL) {
6. printf("ERRO! \n");
7. inicializacao();
8. }

11. **char** line[256];  //Armazena a linha lida
12. **char**\* token;  //Armazena os token separados na linha
13. Strc\_tipoDado var;

16. **while** (!feof(file)) {
18. /\* O primeiro caracter da string deve ser alterando manualmente para
19. \* evitar problemas durante a proxima leitura de dados \*/
20. line[0] = '\0';
22. /\* Realiza a leitura das linhas\*/
23. fgets(line, 256, file);
25. /\* Verifica se o primeiro caracter da linha é token '#'
26. \* Caso for s função strtok ira dividir linha em tokens separados por virgula \*/
27. **if** (line[0] == '#') {
28. token = strtok(line, ",");
30. /\* Laço sera encerrado quando todos os tokens forem lidos \*/
31. **while** (token != NULL) {
32. token = strtok(NULL, ",");
33. strcpy(var.campo1, token);
35. token = strtok(NULL, ",");
36. strcpy(var.campo2, token);
38. token = strtok(NULL, ",");
39. var.campo3 = atoi(token);
41. token = strtok(NULL, ",");
42. var.campo4 = atof(token);
44. /\* Apos ler e armazenar todos os dados em seus devidos campos a
45. estrutura será alocada no vetor global \*/
46. alocarDados(&var);
47. token = strtok(NULL, ",");
48. }
49. }
50. }
52. fclose(file);
53. free(file);
54. imprimeMsgm\_importSucesso();
55. }

**IMPORTANTE:** A leitura é realizada na mesma ordem que os dados foram exportados. Caso a ordem de leitura seja alterada irá causar problemas no armazenamento.

# Arquivo binário

A importação de arquivos é realizada através da função **fread( ).**

1. **void** importDados\_bin() {
2. Strc\_tipoDado  var;
3. **FILE**\* file = fopen("arquivos/bin/arquivo.bin", "rb");
4. **if** (file == NULL) {
5. printf("Erro ao ler o arquivo. \n");
6. exit(EXIT\_FAILURE);
7. }
9. /\* Laço para ler os arquivos armazenados.
10. \* laço só será encerrado quando o arquivo chegar ao fim
11. \*/
12. **do** {
13. fread(&var, **sizeof** (Strc\_tipoDado), 1, file);
14. alocarDado(&var);
15. } **while** (!feof(file));
17. fclose(file);
18. free(file);
19. }

## Verificação de dados

Essa biblioteca é possui diversas funções básicas de validação de entradas como, verificar se um número inteiro ou real é positivo ou negativo, verificar se o código de determinado tipo de dado existe ou não e outras funções com usos mais específicos que serão exemplificadas abaixo:

|  |  |  |
| --- | --- | --- |
| FUNÇÃO | PARAMETROS | USO |
| void modoArmazenamento( ) | // | Escolha do modo que o programa ira armazenar os dados, sendo 1 em formato texto e 2 em formato binário |
| int returnModoArmazenamento() | // | Retorna a opção de armazenamento escolhida |
| int verificarNumeroPositivoFLOAT(float x) | X: um número inteiro qualquer | Retorna 1 caso **x** for positivo;  Retorna 0 se for negativo |
| int verificarNumeroPositivoFLOAT(float x) | X: um número real qualquer | Retorna 1 caso **x** for positivo;  Retorna 0 se for negativo |
| int selecao( ) | // | Retorna a opção selecionada pelo usuário (utilizada bastante em menus) |
| parar\_ou\_ContinuarCadastro() | // | Utilizada nas opções de cadastros. Da opção do usuário continuar cadastrando ou parar e retornar ao menu principal |
| int chamarMenu\_Relatorio( ) | // | Continuar buscando novas informações nos relatórios ou retornar para o menu principal |
| int verificarCod\_TIPO(int cod) | Cod: código qualquer digitado pelo usuário | Caso o código existir ira ser retornado a posição em que ele se encontra, se não retornar -1 |
| int verificarExemplares\_Filmes(int id, int quant) | Id: código do filme solicitado;  quant: quantidade de exemplares; | Se a quantidade de exemplares solicitada estiver disponível será retornado 1, se não será retornado 0 |
| int verificarFilme\_Fornecedor(int posFor, int idFil, int cont) | posFor: Posição do fornecedor;  idFil: Código do filme;  cont: Total de fornecedores alocados | Verifica se um fornecedor possui determinado filme |
| void digiteAlguma\_teclaContinuar( ) | // | Irar limpar a tela caso alguma tecla for digitada; |