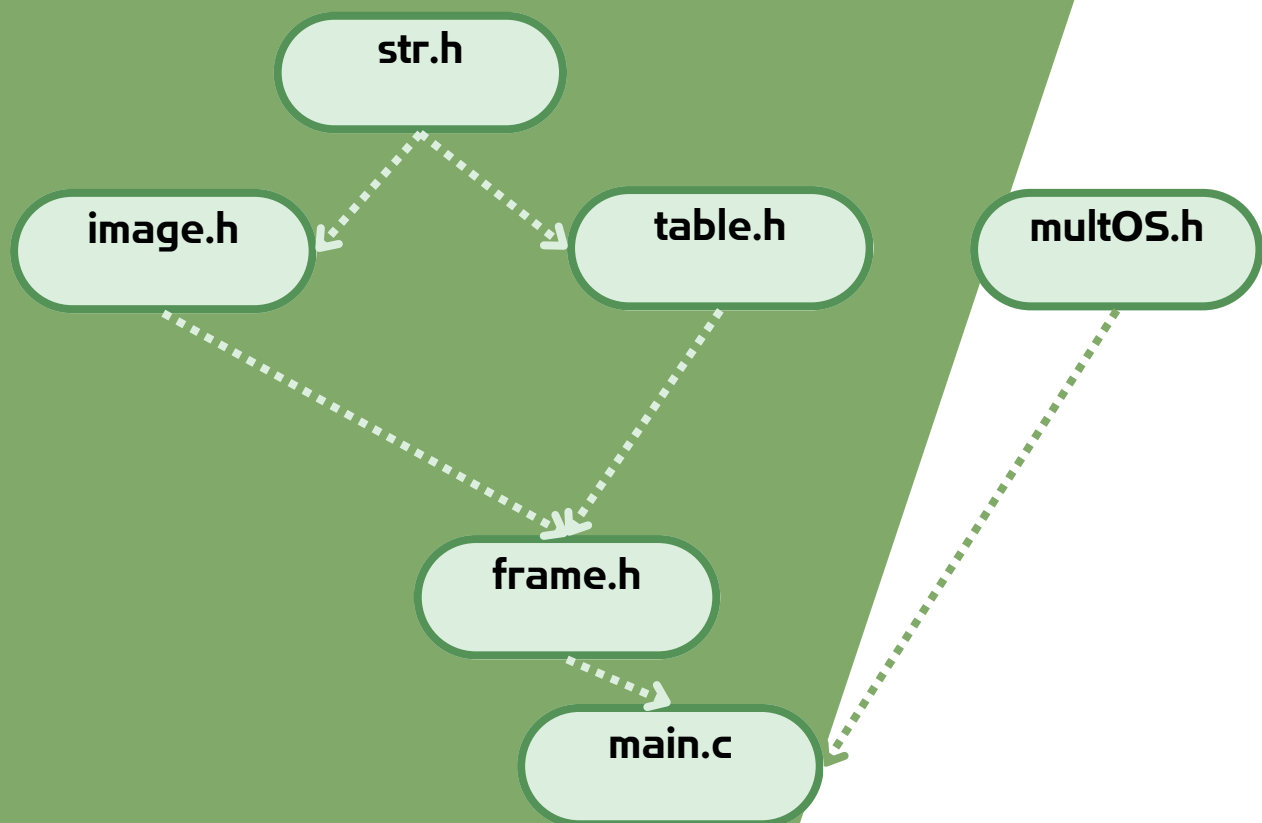


# BADT 1.0

## Definição

Essa biblioteca consiste na simplificação de desenvolvimento de aplicações em **C**. Contando principalmente com gerenciadores de janelas dinâmicas e manipulação -quase- direta de arquivos de forma não destrutiva.

## Visão Geral Dependências



# Visão Geral Resumo dos Códigos

## str.h

- Funções de manipulação de char \* criadas principalmente para solucionar tratativas recorrentes das estruturas.

## image.h

- A estrutura **Image** e seus métodos foram criados para gerenciar "imagens ASCII" de maneira eficiente.
- A imagem é armazenada em Image.pixels (char \*)

## table.h

- A estrutura **Table** e seus métodos foram criados para manipular arquivos. sua utilização depende unicamente de **str.h**.
- O arquivo todo fica armazenado em Table.registry (char \*).

## frame.h

- A estrutura **Frame** e seus métodos foram criados para desenvolver interfaces agradáveis para o usuário final de maneira intuitiva.
- Basicamente uma herança de **Image**, com mais funcionalidades.

## MultOS.h

- Possui funções de apoio que necessitam ser escritas de maneiras diferentes para SO's diferentes.

## INTRODUÇÃO

Para manipular arquivos é necessário importar **frame.h** e inicializar a tabela.

```
1 #include "frame.h"
2
3 void main(){
4
5     Table table;
6     tableSetup(&table, "dscif", "coluna 1,coluna 2,coluna 3,coluna 4,coluna 5");
7 }
```

Parâmetros:

- ponteiro de Table
- texto contendo o tipo de cada coluna
- nome das colunas separados por vírgula

```
8     printf("%s", table.registry);
luan@luan-Lenovo-IdeaPad-S145-15IWL:~/Documentos/GITHub_EDITAVEL/BADT$ ./a.out
| | | |
```

Ao “printar” o registro da tabela é possível notar que uma linha vazia já foi automaticamente criada.

## INSERINDO REGISTROS

Para inserir registros basta usar a coordenada da célula e um ponteiro do tipo adequado da coluna (definido na inicialização da tabela). **O tamanho máximo do tipo string é 70 caracteres!**

```
6
7     Table table;
8     tableSetup(&table, "dscif", "coluna 1,coluna 2,coluna 3,coluna 4,coluna 5");
9
10    tableInsert(&table, 0, 0, &numero);
11    tableInsert(&table, 1, 3, "texto");
12
13    printf("%s", table.registry);
14 }
15
luan@luan-Lenovo-IdeaPad-S145-15IWL:~/Documentos/GITHub_EDITAVEL/BADT$ ./a.out
1| | | |
| | | |
| | | |
| | | |
|texto| | |
```

\*note que novas linhas são criadas automaticamente quando necessário

## DELETANDO REGISTROS

Basta passar o ponteiro , coluna, linha.

```
13    tableDelete(&table, 1, 3);
luan@luan-Lenovo-IdeaPad-S145-15IWL:~/Documentos/GITHub_EDITAVEL/BADT$ ./a.out
1| | | |
| | | |
| | | |
| | | |
```

## ATUALIZANDO REGISTROS

Basta passar o ponteiro , coluna, linha e o novo valor, da mesma maneira que seria feito em tableInsert.

```
13 tableUpdate(&table,1,3,"novo texto");
luan@luan-Lenovo-IdeaPad-S145-15IWL:~/Documentos/GITHub_EDITAVEL/BADT$ ./a.out
1| | | |
| | | |
| | | |
| | | |
| | | |
|novo texto| | |
```

\*Não é necessário usar tableDelete() previamente, apenas atualizar diretamente já basta.

## ARQUIVOS...

A interação de tabelas com arquivos txt é feita por meio de duas funções:

- tableToTxt(Table \*table, char \*directory);
- txtToTable(Table \*table, char \*directory);

Na primeira execução é importante usar txtToTable() para salvar o arquivo formatado corretamente.

Nas próximas execuções segue a fórmula

- Setup
- txtToTable
- Modificações
- tableToTxt
- freeTable

```
1 #include "frame.h"
2
3 void main(){
4
5     int numero = 1;
6
7     Table table;
8     tableSetup(&table, "dscif","coluna 1,coluna 2,coluna 3,coluna 4,coluna 5");
9
10    //txtToTable(&table, "./txtTabela");
11
12    tableInsert(&table, 0, 0, &numero);
13    tableInsert(&table, 1, 3, "texto");
14
15    tableUpdate(&table,1,3,"novo texto");
16
17    printf("%s", table.registry);
18
19    tableToTxt(&table, "./txtTabela");
20
21    tableFree(&table);
22 }
23
```

\*Primeira execução, depois disso basta descomentar txtToTable.

**Nunca esqueça de usar tableFree() antes de encerrar o programa.** essa função é responsável por liberar as alocações dinâmicas necessárias para o funcionamento das tabelas.

## INTRODUÇÃO

Antes de usar de fato os **frames** é importante conhecer um pouco de seu funcionamento.

Todo **Frame** é composto de **Image** e possui o parâmetro **Frame.mode** que define sua função. Atualmente existem 3 modos de frames:

Parâmetros:

- menu - gerencia uma tela que permite múltipla escolha.
- table - dedica o Frame à visualização de uma tabela.
- input - permite a entrada de dados do usuário.

## MENU MODE

É o modo com o qual todos os frames são inicializados (isso será ajustado em versões futuras).

Então para construir um menu basta inicializar o frame normalmente e usar a função **frameInsertOption()** para adicionar opções à tela. também é possível usar a função **frameInsert()** e **frameInsertCenter()** para escrever livremente pelo Frame.

**\*imageShow()** exibe  
o frame.

50 corresponde a  
**largura.**

10 a **altura.**

O **último parâmetro**  
deve ser sempre 'n'.

```
1 #include "frame.h"
2
3 void main(){
4
5     Frame frame;
6     frameSetup(&frame,"menu", 50, 10,'n');
7
8     frameInsert(&frame, "escrevendo normal...", 1,2);
9     frameInsertCenter(&frame,"(centralizado)", 3);
10
11     imageShow(&frame.image);
12 }
```

```
=====menu=====
|
| escrevendo normal...
|                      (centralizado)
|
| >
|
|
|
|-----|
```

## EDITANDO OPÇÕES

As funções de edição de opções adicionam uma opção à próxima linha ou limpam todas as opções.

```
3 void main(){
4
5     Frame frame;
6     frameSetup(&frame,"menu", 50, 10,'n');
7
8     frameInsert(&frame, "escrevendo normal...", 1,2);
9     frameInsertCenter(&frame,"(centralizado)", 3);
10
11    frameInsertOption(&frame,"opcao 1");
12    frameCleanOption(&frame);
13    frameInsertOption(&frame,"ver tabela");
14    frameInsertOption(&frame,"ver input");
15    frameInsertOption(&frame,"Sair");
16
17    imageShow(&frame.image);
18 }
19
```

```
=====menu=====
|
| escrevendo normal...
|                (centralizado)
|
| >ver tabela
|   ver input
|   Sair
|
|
|-----|
```

```
15    frameInsertOption(&frame,"Sair");
16
17    int run = true;
18    char buffer = '\0';
19
20    while(run){
21        frame.inputBuffer[0] = buffer;
22        frameRefresh(&frame);
23
24        clear();
25        imageShow(&frame.image);
26
27        if(buffer != 10)
28            buffer = getch();
29    }
30
31    imageShow(&frame.image);
32 }
33
```

\*Note que **ainda** não é possível mover o cursor. Para isso é necessário criar o laço de repetição do programa.

### LAÇO DE REPETIÇÃO SIMPLES

As atualizações do programa (e seus frames) depende da entrada do usuário. essa entrada alimenta o buffer de input do Frame e permite que as devidas alterações sejam feitas para a próxima exibição.

A seguinte **organização** é recomendada para o laço:

- Alimentação do input do Frame.
- Atualização do Frame.
- Limpeza da tela.
- Exibir Frame atualizado.

Essa estrutura se aplica a **atualização de todos os modos** de frames. A comparação com 10 (ASCII para enter) impede que o programa entre em um "loop de confirmações".

Após a construção do laço:

```
=====menu=====
|
| escrevendo normal...
|                          (centralizado)
| >ver tabela
| ver input
| Sair
|
|=====
```

### Comandos:

W para cima  
S para baixo

## OBTENDO INFORMAÇÃO

A opção selecionada atualmente pode ser obtida em **Frame.optionSelect** para comparações necessárias. Caso queira apenas testar qual opção está selecionada use **frameShowOption()**.

```
=====menu=====
|
| escrevendo normal...
|                          Opcao selecionada: 0
| >ver tabela
| ver input
| Sair
|
|=====
```

**\*Isso  
sobrepõe  
informações  
na linha 3**

```
frameRefresh(&frame);
frameShowOption(&frame);
clear();
```

**Exemplo** do uso da opção selecionada após **pressionar enter**:

```
17 int run = true;
18 int state = 0;
19 char buffer = '\0';
20
21 while(run){
22     frame.inputBuffer[0] = buffer;
23     frameRefresh(&frame);
24
25     if(state == 1)
26         frameShowOption(&frame);
27
28     clear();
29     imageShow(&frame.image);
30
31     if(buffer != '\0'){
32         buffer = getch();
33     }
34     else{
35         if(frame.optionSelect == 1)
36             state = 1;
37
38         buffer = '\0';
39     }
40 }
41
42 imageShow(&frame.image);
43 }
44
```

\*Nesse exemplo, a opção atual só será exibida caso **state = 1** e, **state** só mudará para 1 **quando o usuário pressionar enter** na opção 1.

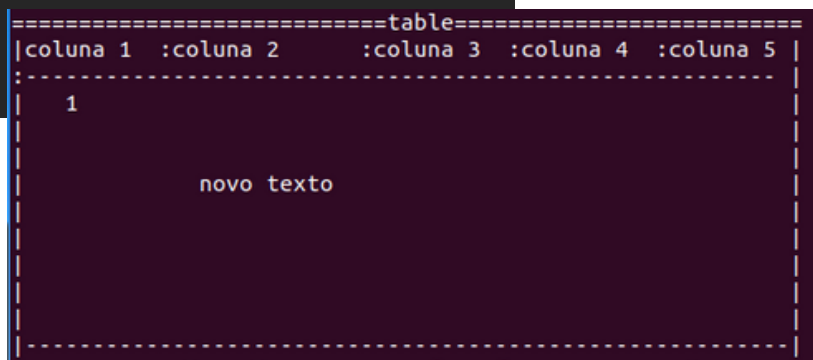
Esse gatilho pode ser usado para chamar outras funções até mesmo **exibir outro Frame** (como será demonstrado mais adiante).

## TABLE MODE

Esse modo é capaz de exibir qualquer tabela automaticamente. Inicializando o Frame e a Table como demonstrado antes é possível vincular a tabela ao Frame com **frameSetTable()**. Essa função deve ser seguida de **frameRefresh()**.

```
1 #include "frame.h"
2
3 void main(){
4
5     Frame frame;
6     frameSetup(&frame, "table", 10, 15, 'n');
7
8     Table table;
9     tableSetup(&table, "dscif", "coluna 1,coluna 2,coluna 3,coluna 4,coluna 5");
10    txtToTable(&table, "./txtTabela");
11
12    frameSetTable(&frame, &table);
13    frameRefresh(&frame);
14
15    clear();
16    imageShow(&frame.image);
17 }
18
```

**A altura mínima do Frame para comportar a tabela é 15!** caso inicialize menor isso será corrigido automaticamente



coluna 1	coluna 2	coluna 3	coluna 4	coluna 5
1	novo texto			

## EXIBINDO TABELAS MAIORES

A altura máxima de exibição é a altura do Frame, caso a tabela seja maior que isso, usando o laço de repetição simples a exibição da tabela é atualizada automaticamente.

```
13    frameRefresh(&frame);
14
15    int run = true;
16    char buffer = '\0';
17
18    while(run){
19        frame.inputBuffer[0] = buffer;
20        frameRefresh(&frame);
21
22        clear();
23        imageShow(&frame.image);
24
25        if(buffer != 10)
26            buffer = getch();
27        else
28            buffer = '\0';
29    }
30
31    imageShow(&frame.image);
32 }
```

Caso a **largura do Frame** seja **menor** que a largura necessária para exibir a tabela, ele ira se ajustar para exibir todas as colunas

A legenda dos comandos é gerada automaticamente!



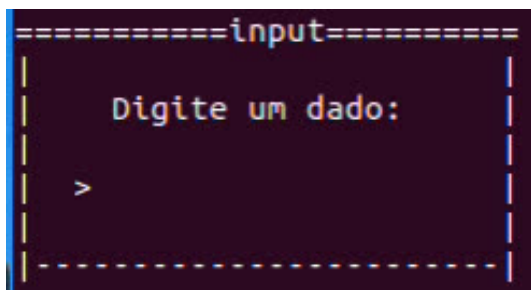
=====table=====				
coluna 1	:coluna 2	:coluna 3	:coluna 4	:coluna 5
00	NOME NOMEADOR UM NOME	a	10	15.20
01	NOME NOMEADOR dois	b	8	10.10
02	NOME NOMEADOR tres	b	8	10.10
03	NOME NOMEADOR UM	a	10	1587.45
04	NOME NOMEADOR dois	b	8	17.57
05	NOME NOMEADOR tres	b	8	1789.87
06	NOME NOMEADOR UM nome gran	a	1092721050	1587.45
07	NOME NOMEADOR dois	b	8	17.57
08	NOME NOMEADOR tres	b	8	1789.87
09	NOME NOMEADOR UM	a	10	1587.45
10	NOME NOMEADOR dois	b	8	17.57
11	NOME NOMEADOR tres	b	8	1789.87
S-Proximo				

## INPUT MODE

Esse modo permite a entrada de dados pelo usuário e armazena em **Frame.inputBuffer**, na prática, é apenas necessário atribuir o endereço do buffer à um ponteiro char.

**frameSetInput()** define o texto do cabeçalho e o tamanho máximo do input (15 caracteres nesse caso).

Todo o resto é gerido automaticamente



O tamanho máximo impede que qualquer dígito a mais seja efetuado

```

1 #include "frame.h"
2
3 void main(){
4
5     Frame frame;
6     frameSetup(&frame, "input", 5, 5, '\n');
7
8     frameSetInput(&frame, "Digite um dado:", 15);
9
10    int run = true;
11    char buffer = '\0';
12
13    while(run){
14        frame.inputBuffer[0] = buffer;
15        frameRefresh(&frame);
16
17        clear();
18        imageShow(&frame.image);
19
20        if(buffer != 10)
21            buffer = getch();
22        else
23            run = 0;
24    }
25
26    imageShow(&frame.image);
27 }
28

```

## EXTRAINDO INPUT

Para obter o valor digitado é necessário criar um ponteiro char e usar **frameGetInput()** para apontar para o buffer corretamente, a partir disso é possível fazer o uso do ponteiro normalmente, converter para outro tipo, inserir em uma tabela, etc.

```
8   frameSetInput(&frame, "Digite um dado:", 15);
9
10  char *out;
11  out = frameGetInput(&frame);
12
13  int run = true;
14  char buffer = '\\0';
15
16  while(run){
17      if(buffer != '\\n')
18          frame.inputBuffer[0] = buffer;
19
20      frameRefresh(&frame);
21
22      clear();
23      imageShow(&frame.image);
24
25      if(buffer != 10){
26          buffer = getch();
27      }
28      else{
29          run = 0;
30          buffer = '\\0';
31      }
32  }
33
34  imageShow(&frame.image);
35
36  printf("out: %s", out);
37 }
```

### Algumas alterações para melhor funcionamento:

Criado char \*out para apontar para inputBuffer.

filtro de entrada do buffer != '\\n'.

“reset” do buffer após enter.

print do ponteiro ao final para validar funcionamento.

## Considerações Finais

As funções foram tão **encapsuladas** quanto possível, e a prioridade escolhida entre memória X desempenho foi **desempenho**. Em quase todas as ocasiões que um valor é usado mais de uma vez e esse valor necessita de qualquer tipo de cálculo, o mesmo é atribuído à uma variável para que o custo de processar seu resultado ocorra apenas uma vez por função.

### Nunca esqueça de liberar todas as alocações dinâmicas!

```
clear();

tableToTxt(&table, "./teste");

frameFree(&frame[0]);
frameFree(&frame[1]);
tableFree(&table);
frameFree(&frame[2]);

printf("=====\n");
printf("|                                     |\n");
printf("| Programa encerrado corretamente! |\n");
printf("|                                     |\n");
printf("=====\n");
```

Exemplo de encerramento após o laço do programa.