Calculadora

Generated by Doxygen 1.8.16

Chapter 1

Main Page

1.1 Calculadora

Ainda em desenvolvimento.

1.1.1 O que ela faz?

Bem, é só uma calculadora.

1.1.2 A fazer

Ainda tem muita coisa pra fazer. Nem sei listar.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 t_list Struct Reference

Lista simplesmente lincada. Está longe de ser uma biblioteca adequada para essa estrutura de dados.

`#include <list.h>`

4.1.1 *

Public Types

- typedef t_list t_stack

  Definição de t_stack. Tem compatibilidade com t_list.

4.1.2 *

Public Member Functions

- t_node * create_node (void *data)

  Cria um nó dinamicamente. Normalmente, essa função não é chamada diretamente. É chamada por funções como insert_head() e push(). Pode ser considerada uma função privada.
- t_list * create_list (const char *data_type)

  Cria dinamicamente uma lista.
- void * get_head (t_list *list)

  Retorna o dado apontado pelo primeiro item da lista. Retorna `NULL` se a lista estiver vazia.
- void * get_tail (t_list *list)

  Retorna o dado apontado pelo último item da lista. Retorna `NULL` se a lista estiver vazia.
- int is_empty (t_list *list)

  Checa se a lista está vazia. É o caso de uma lista que acabou de ser criada.
- void print (t_list *list)

  Para fins de deparação, imprime a lista na tela. Só deve ser usada quando foi definida o tipo de dados da lista.
- void compare ()

  Não tá pronta. Não Preciso por agora.
- void insert_head (t_list *list, void *data)

Insere um item no começo da lista.

- void insert_tail (t_list *list, void *data)

  Insere um item no final da lista.

- void * remove_head (t_list *list)

  Remove o primeiro nó da lista e retorna seu dado e libera a memória apontada pelo nó.

- void delete_head (t_list *list)

  Remove o primeiro nó da lista e libera o espaço de memória apontado por `data` desse nó.

- void clear (t_list *list)

  *Limpa a lista cujos ponteiros para dado **apontam** para espaços alocados dinamicamente.*

- void soft_clear (t_list *list)

  *Limpa a lista cujos ponteiros para dado **não** apontam para espaços alocados dinamicamente.*

- void print_int_list (t_list *list)

  Para fins de depuração, é chamada por print() para listas com tipo de dados `int`. Pode ser considerada uma função privada.

- void print_double_list (t_list *list)

  Para fins de depuração, é chamada por print() para listas com tipo de dados `double`. Pode ser considerada uma função privada.

- void print_str_list (t_list *list)

  Para fins de depuração, é chamada por print() para listas com tipo de dados `char*` (ou "`string`"). Pode ser considerada uma função privada.

- void push (t_stack *stack, void *data)

  Adiciona um nó ao topo da pilha.

- void pop (t_stack *stack)

  Remove da pilha o nó no topo da pilha e libera o espaço de memória apontado pelo dado do nó.

- void * soft_pop (t_stack *stack)

  Remove o nó do topo da pilha.

- void * peek (t_stack *stack)

  Retorna o ponteiro para o dado do nó no topo da pilha.

4.1.3 *

Public Attributes

- t_node * head
- t_node * tail
- int length
- void(* **print**)(struct t_list *list)

4.1.4 Detailed Description

Lista simplesmente lincada. Está longe de ser uma biblioteca adequada para essa estrutura de dados.

Foi implementado apenas o suficiente para implementar a `calculadora em C`

Na documentação, as funções para lista estão na seção "Membros públicos". Apesar de C não ser uma linguagem que tem suporte para orienteção objeto, as funções foram indicadas com "`@memberof t_list`" para juntar a definição da estrutura de dados t_list e suas funções na mesma página, além de dar a ideia de uma classe e seus métodos.

Warning

Cabe ressaltar ao usuário que a lista implementada "não sabe" se o espaço de memória apontado `node->data` foi alocado dinamicamente ou não. Essa resposabilidade é deixada ao usuário (até porque não sei outra maneira de fazer isso). Esse aviso é ressaltado ao longo da documentação em algumas funções (remove_head(), delete_head(), soft_pop()). Resumidamente, essa biblioteca libera a memória apontada pelo nó, mas a memória apontada por `void* data` (em t_node) deve ser analisada em cada caso, dependendo da aplicação do usuário, que deve escolher qual função é mais apropriada.

Essa lista foi testada usando `valgrind` e a framework `Catch2`. Alguns trechos dos arquivos testes são usados como exemplo, por isso algumas macros do Catch2 estão presentes. Se você não tem familiriadade, cheque a parte `Testes e Seções`.

See also

Inspirada na lista implemetada em `link`.

4.1.5 Member Typedef Documentation

4.1.5.1 t_stack

```
typedef t_list t_stack
```

Definição de t_stack. Tem compatibilidade com t_list.

Honestamente, fiquei com preguiça de colocar esse tipo e suas funções em outro arquivo. São muito poucas. Talvez no futuro eu separe em um arquivo diferente.

4.1.6 Member Function Documentation

4.1.6.1 clear()

```
void clear (
            t_list * list )
```

Limpa a lista cujos ponteiros para dado **apontam** para espaços alocados dinamicamente.

Exemplo:

```
TEST_CASE("multiple insert_head", "[list]") {
    int* ptr_int = (int*) calloc(1, sizeof(int));
    t_list* list = create_list("int");
    int i = 0;
    while(i < 3) {
        *ptr_int = i;
        insert_head(list, ptr_int);
        ptr_int = (int*) calloc(1, sizeof(int));
        i++;
    }
    REQUIRE(list->length == 3);
    clear(list);
    free(list);
    free(ptr_int);
}
```

Parameters

list	Ponteiro para lista.

4.1.6.2 create_list()

```
t_list * create_list (
            const char * data_type )
```

Cria dinamicamente uma lista.

Além disso, associa as funções print() - e, futuramente, compare() - as suas funções com os seus devidos tipos. Por enquanto tem suporte para `int` e `char*`.

Note

Para novos tipos de dados, o usuário deve criar a sua própria versão de create_list() e definir, a seu gosto, como associar as funções print() e compare().

Warning

Quem chama essa função deve liberar o espaço depois de usá-lo.

Parameters

in	*data_type*	O tipo de dado.

Precondition

`"int"`, `"char*"` ou `"double"`

Returns

Ponteiro para a lista criada dinamicamente.

4.1.6.3 create_node()

```
t_node * create_node (
            void * data )
```

Cria um nó dinamicamente. Normalmente, essa função não é chamada diretamente. É chamada por funções como insert_head() e push(). Pode ser considerada uma função privada.

Note

Só é preciso se preocupar com vazamento memória se ela for chamada diretamente.

Parameters

data	Ponteiro genérico para um tipo de dado.

Returns

Ponteiro para o espaço alocado.

4.1.6.4 delete_head()

```
void delete_head (
            t_list * list )
```

Remove o primeiro nó da lista e libera o espaço de memória apontado por `data` desse nó.

Note

Deve ser usado *apenas* em listas cujos dados foram alocado dinamicamente.

Parameters

list	Ponteiro **não nulo** para lista.

4.1.6.5 get_head()

```
void * get_head (
            t_list * list )
```

Retorna o dado apontado pelo primeiro item da lista. Retorna `NULL` se a lista estiver vazia.

Parameters

list	Ponteiro **não nulo** para lista.

Returns

Ponteiro para dado. Retorna `NULL` se a lista estiver vazia.

4.1.6.6 get_tail()

```
void * get_tail (
            t_list * list )
```

Retorna o dado apontado pelo último item da lista. Retorna `NULL` se a lista estiver vazia.

Parameters

list	Ponteiro **não nulo** para lista.

Returns

Ponteiro para dado.

4.1.6.7 insert_head()

```
void insert_head (
            t_list * list,
            void * data )
```

Insere um item no começo da lista.

Parameters

list	Ponteiro **não nulo** para lista.
data	Ponteiro **não nulo** para dado.

4.1.6.8 insert_tail()

```
void insert_tail (
            t_list * list,
            void * data )
```

Insere um item no final da lista.

Parameters

list	Ponteiro **não nulo** para lista.
data	Ponteiro **não nulo** para dado.

4.1.6.9 is_empty()

```
int is_empty (
            t_list * list )
```

Checa se a lista está vazia. É o caso de uma lista que acabou de ser criada.

Parameters

list	Ponteiro **não nulo** para lista.

Returns

1 se está vazio, 0 caso contrário.

4.1.6.10 peek()

```
void * peek (
            t_stack * stack )
```

Retorna o ponteiro para o dado do nó no topo da pilha.

Warning

Se o espaço apontado pelo ponteiro foi alocado dinamicamente, quem chamou essa função é reposável por liberá-lo.

Parameters

stack	Ponteiro para pilha.

Returns

Ponteiro para o dado do nó no topo da pilha.

4.1.6.11 pop()

```
void pop (
            t_stack * stack )
```

Remove da pilha o nó no topo da pilha e libera o espaço de memória apontado pelo dado do nó.

Note

Deve ser usado *apenas* em pilhas cujos dados foram alocados dinamicamente. Semelhante a remove_head().

Parameters

stack	Ponteiro para pilha.

4.1.6.12 print()

```
void print (
            t_list * list )
```

Para fins de deparação, imprime a lista na tela. Só deve ser usada quando foi definida o tipo de dados da lista.

Parameters

list	Ponteiro para lista.

4.1.6.13 print_double_list()

```
void print_double_list (
            t_list * list )
```

Para fins de depuração, é chamada por print() para listas com tipo de dados `double`. Pode ser considerada uma função privada.

Note

Não precisa ser chamada diretamente.

Parameters

list	Ponteiro para lista.

4.1.6.14 print_int_list()

```
void print_int_list (
            t_list * list )
```

Para fins de depuração, é chamada por print() para listas com tipo de dados `int`. Pode ser considerada uma função privada.

Note

Não precisa ser chamada diretamente.

Parameters

list	Ponteiro para lista.

4.1.6.15 print_str_list()

```
void print_str_list (
            t_list * list )
```

Para fins de depuração, é chamada por print() para listas com tipo de dados `char*` (ou "`string`"). Pode ser considerada uma função privada.

Note

Não precisa ser chamada diretamente.

Parameters

list	Ponteiro para lista.

4.1.6.16 push()

```
void push (
            t_stack * stack,
            void * data )
```

Adiciona um nó ao topo da pilha.

Parameters

stack	Ponteiro **não nulo** para pilha.
data	Ponteiro **não nulo** para o dado.

4.1.6.17 remove_head()

```
void * remove_head (
            t_list * list )
```

Remove o primeiro nó da lista e retorna seu dado e libera a memória apontada pelo nó.

Warning

Quem chama essa função dever ser o reponsável por liberar memória apontada por data.

Parameters

list	Ponteiro **não nulo** para lista.

Returns

O ponteiro para o primeiro dado da lista.

Em constraste, ver delete_head().

4.1.6.18 soft_clear()

```
void soft_clear (
            t_list * list )
```

Limpa a lista cujos ponteiros para dado **não** apontam para espaços alocados dinamicamente.

Nos exemplos abaixo não há chamada de funções como `malloc()` ou `calloc()` para alocar memória para os dados a serem armazenados na lista.

```
t_list* integers = create_list("int");
int a = 5;
insert_head(integers, &a);
soft_clear(integers);
insert_head(integers, &a);
soft_clear(integers);
free(integers);
t_list* strings = create_list("char*");
char str1[] = "lista";
char str2[] = "abacaxi";
char str3[] = "land rover";
insert_head(strings, str1);
insert_head(strings, str2);
insert_head(strings, str3);
char str[15];
strcpy(str, (char*) get_head(strings));
REQUIRE(strcmp(str, "land rover") == 0);
soft_clear(strings);
free(strings);
```

Em contraste, ver clear().

Parameters

list	Ponteiro para lista.

4.1.6.19 soft_pop()

```
void * soft_pop (
            t_stack * stack )
```

Remove o nó do topo da pilha.

Warning

Quem chama essa função dever ser o reponsável por liberar memória apontada por data.

Parameters

stack	Ponteiro para pilha.

Returns

Ponteiro para o dado do nó que estava no topo da pilha. Retorna `NULL` se a pilha está vazia.

4.1.7 Member Data Documentation

4.1.7.1 head

```
t_node* t_list::head
```

Ponteiro para o primeiro item.

4.1.7.2 length

```
int t_list::length
```

Comprimento da lista.

4.1.7.3 tail

```
t_node* t_list::tail
```

Ponteiro para o último item.

The documentation for this struct was generated from the following file:

- lib/list/list.h

4.2 t_node Struct Reference

Nó com ponteiro para dado genérico e com ponteiro para o próximo nó.

```
#include <list.h>
```

4.2.1 *

Public Attributes

- void * data
- struct t_node * next

4.2.2 Detailed Description

Nó com ponteiro para dado genérico e com ponteiro para o próximo nó.

4.2.3 Member Data Documentation

4.2.3.1 data

```
void* t_node::data
```

Ponteiro genérico para um tipo de dado.

4.2.3.2 next

```
struct t_node* t_node::next
```

Ponteiro para o próximo nó.

The documentation for this struct was generated from the following file:

- lib/list/list.h

Chapter 5

File Documentation

5.1 lib/list/list.h File Reference

Definição do tipo t_list e suas funções.

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

5.1.1 *

Classes

- struct t_node

  Nó com ponteiro para dado genérico e com ponteiro para o próximo nó.
- struct t_list

  Lista simplesmente lincada. Está longe de ser uma biblioteca adequada para essa estrutura de dados.

5.1.2 *

Typedefs

- typedef struct t_node t_node

  Nó com ponteiro para dado genérico e com ponteiro para o próximo nó.
- typedef struct t_list t_list

  Lista simplesmente lincada. Está longe de ser uma biblioteca adequada para essa estrutura de dados.

5.1.3 Detailed Description

Definição do tipo t_list e suas funções.

Author

Yudi Yamane de Azevedo

Date

18 de julho de 2019

É uma pequena lista de operações em listas simplemente lincadas. Inspirada em https://github.com/↩DevonCrawford/Video-Editing-Automation/blob/master/include/LinkedListAPI.h

5.1.4 Typedef Documentation

5.1.4.1 t_list

```
typedef struct t_list t_list
```

Lista simplesmente lincada. Está longe de ser uma biblioteca adequada para essa estrutura de dados.

Foi implementado apenas o suficiente para implementar a calculadora em C

Na documentação, as funções para lista estão na seção "Membros públicos". Apesar de C não ser uma linguagem que tem suporte para orienteção objeto, as funções foram indicadas com "@memberof t_list" para juntar a definição da estrutura de dados t_list e suas funções na mesma página, além de dar a ideia de uma classe e seus métodos.

Warning

Cabe ressaltar ao usuário que a lista implementada "não sabe" se o espaço de memória apontado node->data foi alocado dinamicamente ou não. Essa resposabilidade é deixada ao usuário (até porque não sei outra maneira de fazer isso). Esse aviso é ressaltado ao longo da documentação em algumas funções (remove_head(), delete_head(), soft_pop()). Resumidamente, essa biblioteca libera a memória apontada pelo nó, mas a memória apontada por void* data (em t_node) deve ser analisada em cada caso, dependendo da aplicação do usuário, que deve escolher qual função é mais apropriada.

Essa lista foi testada usando valgrind e a framework Catch2. Alguns trechos dos arquivos testes são usados como exemplo, por isso algumas macros do Catch2 estão presentes. Se você não tem familiriadade, cheque a parte Testes e Seções.

See also

Inspirada na lista implemetada em link.

Index