Algoritmos e Estruturas de Dados: Pilha

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

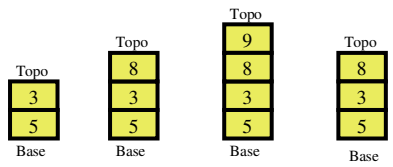
romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Maio/2011

Referências

- Mastering Algorithms with C
  Kyle Loudon
  O´Reilly, 1999
- Livros de algoritmos e estruturas de dados em geral

Pilha – Introdução

- Último a entrar é o primeiro a sair
- Operações básicas são chamadas de PUSH e POP
- Também é possível implementar uma operação PEEK
  - Olha o topo sem remove-lo



Pilha – Interface 1/6

- ***stack_init***
- void stack_init(Stack **stack*, void (**destroy*)(void **data*));
- **Retorno**
  - Nenhum.
- **Descrição**
  - Inicializa a pilha especificada por *stack*. Esta operação deve ser chamada para uma pilha antes que a pilha possa ser usada com qualquer outra operação. O argumento *destroy* fornece uma maneira de liberar dinamicamente dados alocados quando *stack_destroy* é chamada. Por exemplo, se a pilha contem dados dinamicamente alocados usando *malloc*, *destroy* deveria ser associado com *free* para liberar os dados quando a pilha é destruida. Para dados estruturados contendo diversos membros alocados dinamicamente, *destroy* deveria ser associado com uma função criada pelo usuário que chama *free* para cada membro alocado dinamicamente assim como para a estrutura ela própria. Para uma pilha contendo dados os quais não devem ser liberados, *destroy* deveria ser NULL.

Pilha – Interface 2/6

- ***stack_destroy***
- void stack_destroy(Stack **stack*);
- **Retorno**
  - Nenhum.
- **Descrição**
  - Destrói a pilha especificada por *stack*. Nenhuma outra operação é permitida após a chamada de *stack_destroy* a não ser que *stack_init* seja chamada novamente. A operação *stack_destroy* remove todos os elementos de uma pilha e chama a função passada como *destroy* para *stack_init* uma vez para cada elemento quando o mesmo é removido, desde que *destroy* não seja igual a NULL.

Pilha – Interface 3/6

- ***stack_ push***
- int stack_push(Stack **stack*, const void **data*);
- **Retorno**
  - 0 se a inserção na pilha teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Coloca um elemento no topo da pilha especificada por *stack*. O novo elemento contem um pointer para *data*, logo a memória referenciada por *data* deve permanecer válida pelo tempo que o elemento permanecer na pilha. É responsabilidade do chamador gerenciar a memória associada com *data.*

Pilha – Interface 4/6

- ***stack_ pop***
- int stack_pop(Stack **stack*, void ***data*);
- **Retorno**
  - 0 se a retirada do elemento teve sucesso, ou -1 caso contr ário.
- **Descrição**
  - Retira um elemento do topo da pilha especificada por *stack*. Após o retorno, *data* aponta para os dados armazenados no elemento que foi retirado. É responsabilidade do chamador gerenciar a memória associada com estes dados.

Pilha – Interface 5/6

- ***stack_ peek***
- void *stack_peek(const Stack **stack*);
- **Retorno**
  - Dados armazenados com o elemento no topo da pilha, ou NULL se a pilha estiver vazia.
- **Descrição**
  - Macro que obtem os dados armazenados no elemento no topo da pilha especificada por *stack*.

Pilha – Interface 6/6

- ***stack_size***
- int stack_size(const Stack **stack*);
- **Retorno**
  - Número de elementos na pilha.
- **Descrição**
  - Macro que obtem o número de elementos na pilha especificada por *stack*.

Pilha – Header 1/1

```
#ifndef STACK_H
#define STACK_H
#include <stdlib.h>

#include "list.h"

typedef List Stack;

#define     stack_init              list_init

#define     stack_destroy           list_destroy

int stack_push(Stack *stack, const void *data);

int stack_pop(Stack *stack, void **data);

#define stack_peek(stack)           ((stack)->head == NULL ? NULL : (stack)->head->data)

#define     stack_size              list_size

#endif
```


Pilha – Implementação 1/1

```
#include <stdlib.h>
#include "list.h"
#include "stack.h"

int stack_push(Stack *stack, const void *data) {
   return list_ins_next(stack, NULL, data);
}

int stack_pop(Stack *stack, void **data) {
   return list_rem_next(stack, NULL, data);
}
```


Pilha – Sumário

- Suporta operações PUSH, POP e PEEK
- Também conhecida como LIFO: Last-In First-Out
- Pode ser implementada como um tipo especial de lista
- Insere e retira sempre da frente