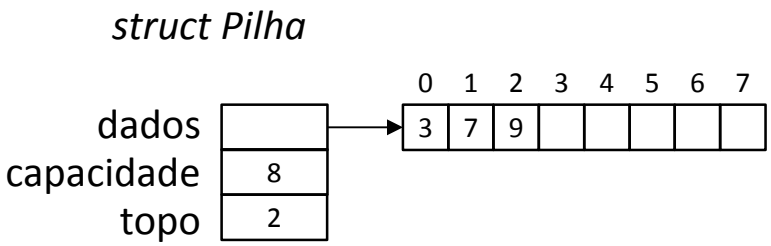
Implementação de Pilhas usando Vetor de Inteiros

A implementação mais simples de uma pilha consiste em uma *struct* que contém um ponteiro para o vetor de inteiros (alocado dinamicamente na inicialização), bem como um campo para guardar a capacidade do vetor, e o índice do valor no topo da pilha. A figura a seguir apresenta o modelo da pilha, instanciada com capacidade 8, e com os valores 3, 7 e 9 empilhados.



O código que implementa tal modelo é apresentado a seguir.

Pilha.h

```
#define ERRO_PILHA_CHEIA -1
#define ERRO_PILHA_VAZIA -2

typedef struct{
   int *dados;
   int capacidade, topo;
}Pilha;

void inicializa_pilha(Pilha *p, int c); // Aloca pilha com capacidade c.
int pilha_vazia(Pilha p); // Função booleana que verifica se pilha está vazia.
int pilha_cheia(Pilha p); // Função booleana que verifica se pilha está cheia.
int empilha(Pilha *p, int info); // Insere um valor no topo da pilha.
int desempilha(Pilha *p, int *info); // Remove o valor do topo da pilha.
int le_topo(PilhaGenerica p, int *info); // Somente leitura do valor no topo.
int mostra_pilha(Pilha p); // Mostra os dados da pilha na tela (console).
void desaloca_pilha(Pilha *p); // Libera a memória alocada na inicialização.
```

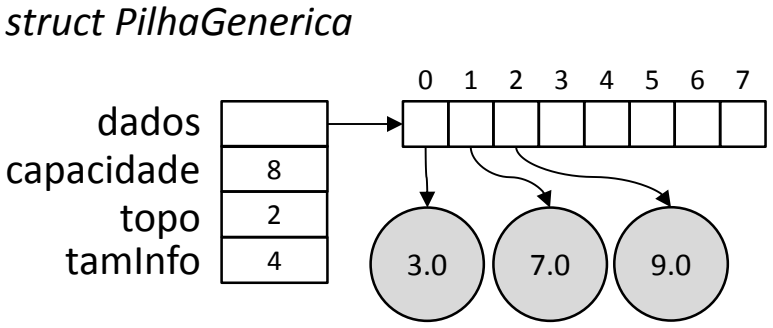
Pilha.c

```
#include "Pilha.h"
#include <stdio.h>
#include <stdlib.h>

void inicializa_pilha(Pilha *p, int c){
   p->dados = malloc( sizeof(void *) * c );
   p->capacidade = c;
   p->topo = -1;
}

int pilha_vazia(Pilha p) {
   return p.topo == -1;
}

int pilha_cheia(Pilha p) {
   return p.topo == p.capacidade - 1;
}

int empilha(Pilha *p, int info) {
   if(pilha_cheia(*p))
      return ERRO_PILHA_CHEIA;

   p->topo++;
   p->dados[p->topo] = info;
   return 1; // Sucesso
```

```
}

int desempilha(Pilha *p, int *info) {
   if(pilha_vazia(*p))
      return ERRO_PILHA_VAZIA;

   *info = p->dados[p->topo];
   p->topo--;
   return 1; // Sucesso

}

int le_topo(Pilha p, int *info) {
   if(pilha_vazia(p))
      return ERRO_PILHA_VAZIA;

   *info = p.dados[p.topo];
   return 1; // Sucesso
}

int mostra_pilha(Pilha p) {
   if(pilha_vazia(p))
      printf("Pilha vazia!\n");
   else{
      printf("Dados da Pilha:\n");
      int i;
      for(i=0; i<=p.topo; i++)
         printf("%d\n", p.dados[i]);
   }
}

void desaloca_pilha(Pilha *p) {
   free(p->dados);
}
```

Implementação de Pilhas Genéricas

Com base na implementação anterior, é possível construir um modelo de pilha que permite armazenar qualquer tipo de dados. Neste caso, o ponteiro de dados apontará para um vetor de ponteiros *void* (void *). A justificativa é que um ponteiro *void* consegue armazenar o endereço que qualquer tipo de dado. Além disso, é necessário um campo extra para armazenar o tamanho do dado a ser armazenado na pilha. A figura a seguir apresenta o modelo da pilha, instanciada com capacidade 8, com o tamanho da informação 4 (para dados do tipo *float*), e com os valores 3.0, 7.0 e 9.0 empilhados.



O código que implementa tal modelo é apresentado a seguir. Os trechos destacados em vermelho representam alguma modificação ou inclusão de novo código, para refletir o novo modelo.

A função *mostra_pilha()* recebe agora um novo parâmetro, que é função para imprimir uma informação. Como a estrutura é genérica, este aspecto deve ser resolvido por uma função em nível de aplicação. Esse é um exemplo de *polimorfismo*, pois função *mostra_pilha()* tem um comportamento que pode variar dependendo da aplicação.

PilhaGenerica.h

```
#define ERRO_PILHA_CHEIA -1
#define ERRO_PILHA_VAZIA -2

typedef struct{
   void **dados;
   int capacidade, topo;
   int tamInfo;
}PilhaGenerica;

void inicializa_pilha(PilhaGenerica *p, int c, int t);
int pilha_vazia(PilhaGenerica p);
int pilha_cheia(PilhaGenerica p);
int empilha(PilhaGenerica *p, void *info);
int desempilha(PilhaGenerica *p, void *info);
int le_topo(PilhaGenerica p, void *info);
int mostra_pilha(PilhaGenerica p, void (*mostra)(void *) );
void desaloca_pilha(PilhaGenerica *p);
```

PilhaGenerica.c

```
#include "PilhaGenerica.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // necessário para a função memcpy()

void inicializa_pilha(PilhaGenerica *p, int c, int t){
   p->dados = malloc( sizeof(void *) * c );
   p->capacidade = c;
   p->topo = -1;
   p->tamInfo = t;
}

int pilha_vazia(PilhaGenerica p) {
   return p.topo == -1;
}

int pilha_cheia(PilhaGenerica p) {
   return p.topo == p.capacidade - 1;
}

int empilha(PilhaGenerica *p, void *info) {
   if(pilha_cheia(*p))
      return ERRO_PILHA_CHEIA;

   p->topo++;
   p->dados[p->topo] = malloc(p->tamInfo); // Aloca memória para o dado.
   memcpy(p->dados[p->topo], info, p->tamInfo); // Copia o dado.
   return 1; // Sucesso
}

int desempilha(PilhaGenerica *p, void *info) {
   if(pilha_vazia(*p))
      return ERRO_PILHA_VAZIA;

   memcpy(info, p->dados[p->topo], p->tamInfo); // Copia o dado.
   free(p->dados[p->topo]); // Libera a memória.
   p->topo--;
   return 1; // Sucesso
```

```
}

int le_topo(PilhaGenerica p, void *info) {
   if(pilha_vazia(p))
      return ERRO_PILHA_VAZIA;

   memcpy(info, p.dados[p.topo], p.tamInfo); // Copia o dado.
   return 1; // Sucesso
}

void mostra_pilha(PilhaGenerica p, void (*mostra)(void *) ) {
   if(pilha_vazia(p))
      printf("Pilha vazia!\n");
   else{
      printf("Dados da Pilha:\n");
      int i;
      for(i=0; i<=p.topo; i++)
         mostra(p.dados[i]); // Código polimórfico.
   }
}

void desaloca_pilha(PilhaGenerica *p) {
   int i;
   for(i=0; i<=p->topo; i++)
      free(p->dados[i]); // Libera a memória dos dados em si.
   free(p->dados); // Libera a memória do vetor de ponteiros.
}
```