

Estruturas de Dados

Módulo 11 – Pilhas



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,
Introdução a Estruturas de Dados, Editora Campus
(2004)

Capítulo 11 – Pilhas

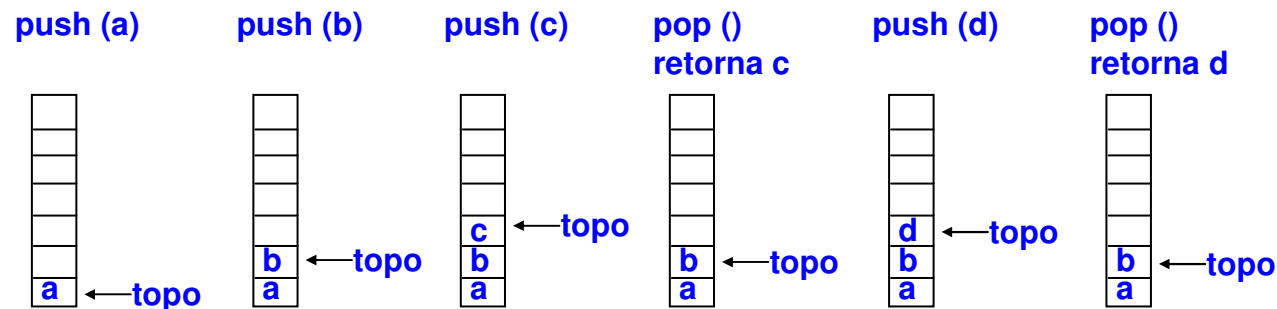
Tópicos

- Introdução
- Interface do tipo pilha
- Implementação de pilha com vetor
- Implementação de pilha com lista
- Exemplo de uso: calculadora pós-fixada

Introdução

- Pilha

- novo elemento é inserido no topo e acesso é apenas ao topo
 - o primeiro que sai é o último que entrou (LIFO – *last in, first out*)
- operações básicas:
 - empilhar (*push*) um novo elemento, inserindo-o no topo
 - desempilhar (*pop*) um elemento, removendo-o do topo



Interface do tipo pilha

- Implementações:
 - usando um vetor
 - usando uma lista encadeada
 - simplificação:
 - pilha armazena valores reais

Interface do tipo pilha

- Interface do tipo abstrato Pilha: *[pilha.h](#)*
 - função *[pilha_cria](#)*
 - aloca dinamicamente a estrutura da pilha
 - inicializa seus campos e retorna seu ponteiro
 - funções *[pilha_push](#)* e *[pilha_pop](#)*
 - inserem e retiram, respectivamente, um valor real na pilha
 - função *[pilha_vazia](#)*
 - informa se a pilha está ou não vazia
 - função *[pilha_libera](#)*
 - destrói a pilha, liberando toda a memória usada pela estrutura.

```
typedef struct pilha Pilha;
```

```
Pilha* pilha_cria (void);
```

```
void pilha_push (Pilha* p, float v);
```

```
float pilha_pop (Pilha* p);
```

```
int pilha_vazia (Pilha* p);
```

```
void pilha_libera (Pilha* p);
```

tipo Pilha:

- definido na interface
- depende da implementação do struct pilha

Implementação de pilha com vetor

- Implementação de pilha com vetor
 - vetor (vet) armazena os elementos da pilha
 - elementos inseridos ocupam as primeiras posições do vetor
 - elemento `vet[n-1]` representa o elemento do topo

```
#define N 50      /* número máximo de elementos */

struct pilha {
    int n;        /* vet[n]: primeira posição livre do vetor */
    float vet[N]; /* vet[n-1]: topo da pilha */
                /* vet[0] a vet[N-1]: posições ocupáveis */
};
```


Implementação de pilha com vetor

- função pilha_cria
 - aloca dinamicamente um vetor
 - inicializa a pilha como sendo vazia, isto é, com o número de elementos igual a zero

```
Pilha* pilha_cria (void)
{
    Pilha* p = (Pilha*) malloc(sizeof(Pilha));
    p->n = 0; /* inicializa com zero elementos */
    return p;
}
```

tipo Pilha: definido na interface
struct pilha: determina a implementação

Implementação de pilha com vetor

- função pilha_push
 - insere um elemento na pilha
 - usa a próxima posição livre do vetor, se houver

```
void pilha_push (Pilha* p, float v)
{
    if (p->n == N) {                /* capacidade esgotada */
        printf("Capacidade da pilha estourou.\n");
        exit(1);                    /* aborta programa */
    }
    /* insere elemento na próxima posição livre */
    p->vet[p->n] = v;
    p->n++;                          /* equivalente a: p->n = p->n + 1 */
}
```

Implementação de pilha com vetor

- função pilha_pop
 - retira o elemento do topo da pilha, retornando o seu valor
 - verificar se a pilha está ou não vazia

```
float pilha_pop (Pilha* p)
{ float v;
  if (pilha_vazia(p)) { printf("Pilha vazia.\n");
                        exit(1); }                      /* aborta programa */
  /* retira elemento do topo */
  v = p->vet[p->n-1];
  p->n--;
  return v;
}
```

Implementação de pilha com lista

- Implementação de pilha com lista
 - elementos da pilha armazenados na lista
 - pilha representada por um ponteiro para o primeiro nó da lista

```
/* nó da lista para armazenar valores reais */  
struct lista {  
    float info;  
    struct lista* prox;  
};  
typedef struct lista Lista;  
  
/* estrutura da pilha */  
struct pilha {  
    Lista* prim;           /* aponta para o topo da pilha */  
};
```

Implementação de pilha com lista

- função pilha_cria
 - cria aloca a estrutura da pilha
 - inicializa a lista como sendo vazia

```
Pilha* pilha_cria (void)
{
    Pilha* p = (Pilha*) malloc(sizeof(Pilha));
    p->prim = NULL;
    return p;
}
```

Implementação de pilha com lista

- função pilha_push
 - insere novo elemento **n** no início da lista

```
void pilha_push (Pilha* p, float v)
{
    Lista* n = (Lista*) malloc(sizeof(Lista));
    n->info = v;
    n->prox = p->prim;
    p->prim = n;
}
```

Implementação de pilha com lista

- função pilha_pop
 - retira o elemento do início da lista

```
float pilha_pop (Pilha* p)
{
    Lista* t;
    float v;
    if (pilha_vazia(p)) { printf("Pilha vazia.\n");
                           exit(1); } /* aborta programa */

    t = p->prim;
    v = t->info;
    p->prim = t->prox;
    free(t);
    return v;
}
```

Implementação de pilha com lista

- função pilha_libera
 - libera a pilha depois de liberar todos os elementos da lista

```
void pilha_libera (Pilha* p)
{
    Lista* q = p->prim;
    while (q!=NULL) {
        Lista* t = q->prox;
        free(q);
        q = t;
    }
    free(p);
}
```


Exemplo de uso: calculadora pós-fixada

- Notação para expressões aritméticas
 - infix = operador entre os operandos $(1-2)*(4+5)$
 - pós-fixa = operador após operandos $1\ 2 -\ 4\ 5 + *$
 - pré-fixa = operador antes dos operandos $* -\ 1\ 2 +\ 4\ 5$
- Exemplo:
 - calculadora HP científica usa notação pós-fixa

Exemplo de uso: calculadora pós-fixada

- Avaliação de expressões aritméticas pós-fixadas:
 - cada operando é empilhado numa pilha de valores
 - quando se encontra um operador
 - desempilha-se o número apropriado de operandos (dois para operadores binários e um para operadores unários)
 - realiza-se a operação devida
 - empilha-se o resultado
- Exemplo:
 - avaliação da expressão $1\ 2\ -\ 4\ 5\ +\ *$

empilhe os valores 1 e 2	1 2 - 4 5 + *	<div>2</div> <div>1</div>
quando aparece o operador “-”	1 2 - 4 5 + *	
desempilhe 1 e 2		
empilhe -1, o resultado da operação (1 - 2)		<div>-1</div>
empilhe os valores 4 e 5	1 2 - 4 5 + *	<div>5</div> <div>4</div> <div>-1</div>
quando aparece o operador “+”	1 2 - 4 5 + *	
desempilhe 4 e 5		<div>-1</div>
empilhe 9, o resultado da operação (4+5)		<div>9</div> <div>-1</div>
quando aparece o operador “*”	1 2 - 4 5 + *	
desempilhe -1 e 9		
empilhe -9, o resultado da operação (-1*9)		<div>-9</div>

Exemplo de uso: calculadora pós-fixada

- Interface da calculadora *calc.h*

```
typedef struct calc Calc;

/* funções exportadas */
Calc* calc_cria (char* f);
void calc_operando (Calc* c, float v);
void calc_operador (Calc* c, char op);
void calc_libera (Calc* c);

/* tipo representando a calculadora */
struct calc {
    char f[21];          /* formato para impressão (cadeia de caracteres) */
    Pilha* p;           /* pilha de operandos */
};
```

Exemplo de uso: calculadora pós-fixada

- função **cria**
 - recebe como entrada uma cadeia de caracteres com o formato que será utilizado pela calculadora para imprimir os valores
 - cria uma calculadora inicialmente sem operandos na pilha

```
Calc* calc_cria (char* formato)
{
    Calc* c = (Calc*) malloc(sizeof(Calc));
    strcpy(c->f,formato);      /* copia a cadeia de caracteres " formato "
                               para a cadeia de caracteres " c->f " */
    c->p = pilha_cria();        /* cria pilha vazia */
    return c;
}
```

Exemplo de uso: calculadora pós-fixada

- função **operando**
 - coloca no topo da pilha o valor passado como parâmetro

```
void calc_operando (Calc* c, float v)
{
    /* empilha operando */
    pilha_push(c->p,v);

    /* imprime topo da pilha */
    printf(c->f,v);
}
```

Exemplo de uso: calculadora pós-fixada

- função **operador**
 - retira dois valores do topo da pilha (operadores são binários)
 - efetua a operação correspondente
 - coloca o resultado no topo da pilha
 - operações válidas: '+', '-', '*' e '/'
 - se não existirem operandos na pilha, assume-se que são zero

```

void calc_operador (Calc* c, char op)
{
    float v1, v2, v;
    if (pilha_vazia(c->p))          /* desempilha operandos */
        v2 = 0.0;
    else
        v2 = pilha_pop(c->p);
    if (pilha_vazia(c->p))
        v1 = 0.0;
    else
        v1 = pilha_pop(c->p);
    switch (op) {                   /* faz operação */
        case '+':    v = v1+v2;    break;
        case '-':    v = v1-v2;    break;
        case '*':    v = v1*v2;    break;
        case '/':    v = v1/v2;    break;
    }
    pilha_push(c->p,v);             /* empilha resultado */
    printf(c->f,v);                 /* imprime topo da pilha */
}

```


Exemplo de uso: calculadora pós-fixada

```
/* Programa para ler expressão e chamar funções da calculadora */
```

```
#include <stdio.h>  
#include "calc.h"
```

```
int main (void)  
{  
    char c;  
    float v;  
    Calc* calc;
```

```
/* cria calculadora com formato de duas casas decimais */  
calc = calc_cria ("%f\n");
```

(ver livro)

Resumo

Pilha

<i>top</i>	retorna o topo da pilha
<i>push</i>	<i>insere</i> novo elemento no topo da pilha
<i>pop</i>	<i>remove</i> o elemento do topo da pilha

