

INF1007 – Programação 2

9 – Pilhas



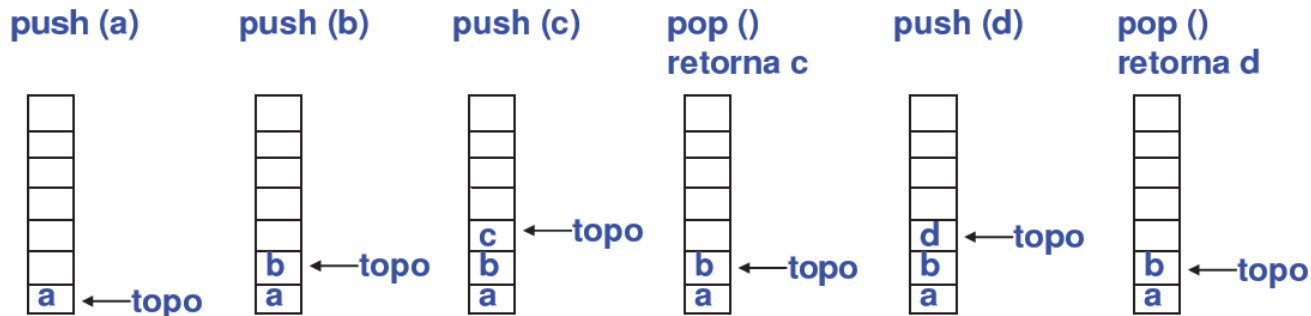
Tópicos Principais

- Introdução
- Interface do tipo pilha
- Exemplo de uso: verificação de expressões
- Implementação de pilha com lista encadeada
- Implementação de pilha com vetor

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.

Interface do tipo pilha

```
/* TAD: pilha de valores reais (float) */

typedef struct pilha Pilha;

/* Tipo Pilha, definido na interface, depende da
implementação do struct 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);
```

Exemplo de uso

- Verificação de expressões matemáticas
 - Considerando cadeias de caracteres com expressões matemáticas que podem conter termos entre parênteses, colchetes ou chaves, ou seja, entre os caracteres '(' e ')', ou '[' e ']', ou '{' e '}';
 - função que retorna 1, se os parênteses, colchetes e chaves de uma expressão aritmética *exp* são abertos e fechados corretamente, ou 0 caso contrário;
 - Para a expressão “2*{3+4*(2+5*[2+3])}” retornaria 1;
 - Para a expressão “2*(3+4+{5*[2+3]})” retornaria 0;
- Protótipo da função:

int verifica(char exp);*

Exemplo de uso

- Verificação de expressões matemáticas
 - A estratégia é percorrer a expressão da esquerda para a direita:
 1. Se encontra '(', '[' ou '{', empilha;
 2. Se encontra ')', ']' ou '}', desempilha e verifica o elemento no topo da pilha, que deve ser o caractere correspondente;
 3. Ao final, a pilha deve estar vazia.

Exemplo de uso

```
char fecho(char c) {
    if(c=='}') return '{';
    if(c=='[') return '[';
    if(c=='(') return '(';
}

int verifica(char* exp) {
    Pilha* p=pilha_cria();
    int i;
    for(i=0; exp[i]!='\0'; i++)
    {
        if(exp[i]=='{'||exp[i]=='['||exp[i]=='(')
            pilha_push(p,exp[i]);
        else if(exp[i]=='}'||exp[i]==']'||exp[i]==')')
        {
            if(pilha_vazia(p)) return 0;
            if(pilha_pop(p)!=fecho(exp[i])) return 0;
        }
    }
    if(!pilha_vazia(p)) return 0;
    pilha_libera(p);
    return 1;
}
```

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));
    if (p==NULL) exit (1);
    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 elemento {
    int info;
    struct elemento *prox
};
typedef struct elemento Elemento;

/* estrutura da pilha */
struct pilha {
    Elemento* 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)) ;
    if (p==NULL) exit(1) ;
    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)
{
    Elemento* n = (Elemento*) malloc(sizeof(Elemento));
    if (n==NULL) exit(1);
    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)
{
    Elemento* t;
    float v;
    if (pilha_vazia(p)) 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 todos os elementos da lista e depois libera a pilha

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

Implementação de pilha com lista

- função *pilha_vazia*
 - Retorna 1, se a pilha está vazia, ou 0, caso contrário

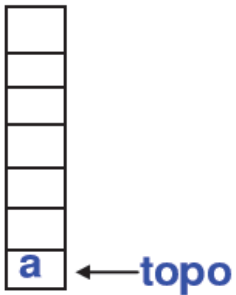
```
int pilha_vazia (Pilha* p)
{
    if(p->prim == NULL)
        return 1;
    return 0;
}
```

Resumo

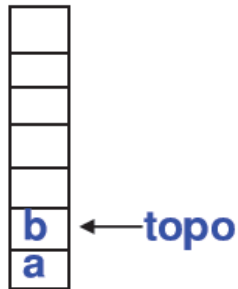
Pilha

- *push* - *insere* novo elemento no topo da pilha
- *pop* - *remove* o elemento do topo da pilha

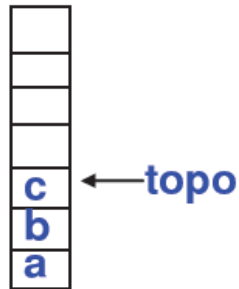
push (a)



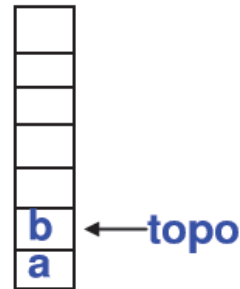
push (b)



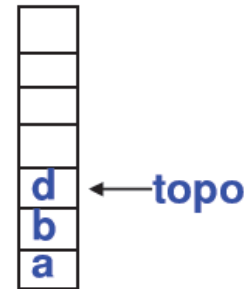
push (c)



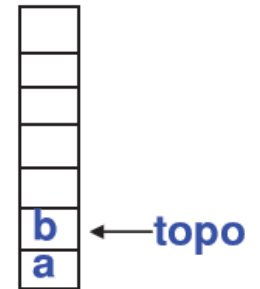
pop ()
retorna c



push (d)



pop ()
retorna d



Referências

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

Capítulo 11 – Pilhas