Linguagem C: Pilhas

Luis Martí
Instituto de Computação
Universidade Federal Fluminense
lmarti@ic.uff.br - http://lmarti.com

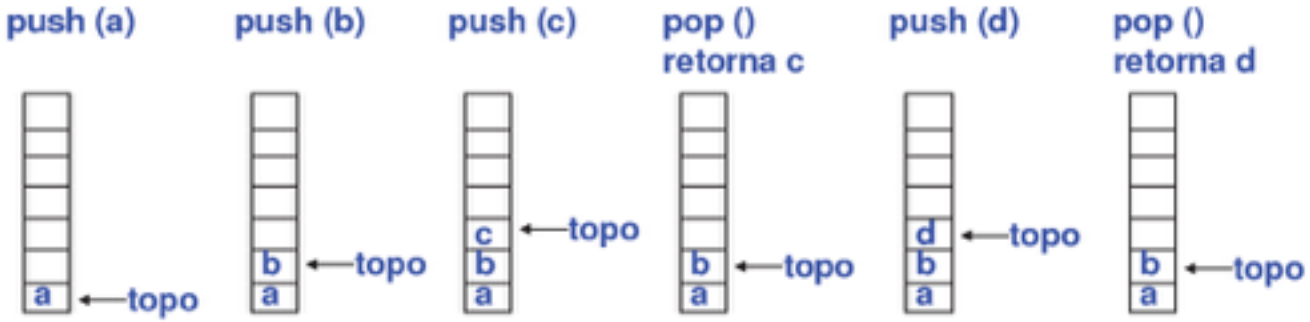
Tópicos Principais

- Conceito de Pilhas
- Interface tipo pilha
- Implementação com Lista Encadeada
- Exemplo de uso

Pilha

Características:

- 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

- 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);
```

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

- 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));
   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));
   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;
}
```

Exemplo de uso

- Verificação de expressões matemáticas
  - Considere 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;
- Expressões armazenadas como vetores do tipo char
  - v = {‘2’, ‘*’, ‘(‘, ‘3’, ‘+’, ‘4’, ‘+’, ‘{‘, ‘5’, ‘*’, ‘[‘, ‘2’, ‘+’, ‘3’, ‘}’, ‘]’, ‘)’};
- Protótipo da função:

int verifica(char *v, int n);

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. Outro caracteres (números ou operadores) são ignorados
    4. Ao final, a pilha deve estar vazia.

Exemplo de uso

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

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

Referências

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

- Capítulo 11 – Pilhas

Material adaptado por Luis Martí a partir dos slides de José Viterbo Filho que forem elaborados por Marco Antonio Casanova e Marcelo Gattas para o curso de Estrutura de Dados para Engenharia da PUC-Rio, com base no livro *Introdução a Estrutura de Dados*, de Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Editora Campus (2004).