



A Pilha

Aula 02

DPEE 1038 – Estrutura de Dados para Automação
Curso de Engenharia de Controle e Automação
Universidade Federal de Santa Maria

Prof. Rafael Concatto Beltrame
beltrame@mail.ufsm.br

Sumário

- Representação de pilhas em C
- Implementação das funções primitivas
 - empty
 - push
 - pop
 - stacktop → Não é primitiva



Representando pilhas em C

- Uma pilha é um **conjunto ordenado de itens**
 - Existem diferentes métodos de implementar uma pilha
- **Vetores**
 - O **nº de elementos** de um **vetor** é **fixado** em sua declaração
 - A **pilha** é, fundamentalmente, um **objeto dinâmico**, cujo tamanho muda conforme itens são empilhados/desempilhados
 - Um vetor pode ser declarado suficientemente grande para armazenar o tamanho máximo da pilha
 - Uma extremidade do vetor é o final fixo da pilha
 - O topo da pilha desloca-se constantemente
 - Logo, precisa-se de outro campo para rastrear a posição do topo da pilha

Representando pilhas em C

- Declaração básica de uma pilha: Estrutura
 - Estrutura contendo
 - Um vetor para armazenar os elementos da pilha
 - Um inteiro para indicar a posição atual do topo da pilha

```
#define STACKSIZE 100      // Dimensão da pilha

struct stack {
    int top;                // "Endereço" do topo
    int items[STACKSIZE];   // Vetor de dados
};

struct stack s;           // Declaração da pilha "s"
```

- Presumiu-se que todos os elementos de `s.items` são inteiros
- A pilha não conterá mais do que `STACKSIZE` itens

Representando pilhas em C

- Não há motivos para restringir uma pilha a conter somente inteiros

```
#define STACKSIZE 100      // Dimensão da pilha
#define INTGR     1      // Tipo "integer"
#define STRING    2      // Tipo "string"

struct stack_element {
    int etype;           // INTGR ou STRING
    union {
        int   ival;       // Possíveis tipos
        char *pnt_sval;
    } element;
};

struct stack {            // Definição da pilha
    int   top;
    struct stack_element items[STACKSIZE];
};
```

Representando pilhas em C

- Para imprimir o primeiro elemento da pilha, o script ficaria

```
struct stack s;                                // Declaração da pilha
struct stack_element s_aux; // Estrutura auxiliar

s_aux = s.items[s.top]; // Salva dados na estr. aux.

switch (s_aux.etype) {
    case INTGR : printf("%d \n", s_aux.ival); break;
    case STRING : printf("%s \n", s_aux.pnt_sval);
}
```

Representando pilhas em C

- A variável `top` precisa ser sempre declarada como `inteiro`
 - Representa a posição do elemento superior da pilha dentro de um vetor de itens
`s.top = 4` → Existem 5 elementos na pilha
`s.items[0], ..., s.items[4]`
- Sempre que um novo item for **inserido** na pilha
 - Incrementar `s.top`
- Sempre que um novo item for **retirado** da pilha
 - Decrementar `s.top`

Implementando a função EMPTY

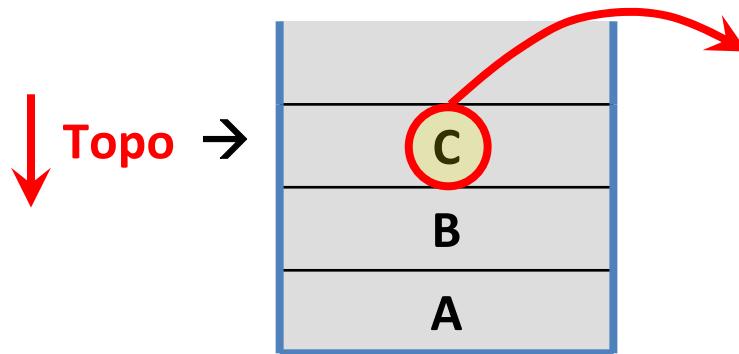
- A pilha vazia pode ser indicada por `s.top == -1`

```
if (s.top == -1)
    // A pilha está vazia
else
    // A pilha contém elementos
```

```
// Função empty()
empty(pnt_s)                      // Estrutura passada
    struct stack *pnt_s;             // como ponteiro
{
    if (pnt_s -> top == -1)
        return(TRUE);                // Pilha vazia
    else
        return(FALSE);               // Pilha com itens
}
```

Implementando a função POP

- `pop (&s)` → Retirar um item do topo (Desempilhar)



- Deve-se considerar a possibilidade de **underflow**
- Ações da função `pop ()`
 - 1) Se a pilha estiver **vazia**, imprimir uma **mensagem** de advertência e interromper a execução
 - 2) Remove o **primeiro** elemento da pilha
 - 3) Retorna esse elemento para o programa de chamada

Implementando a função POP

- **Implementação da função pop ()**

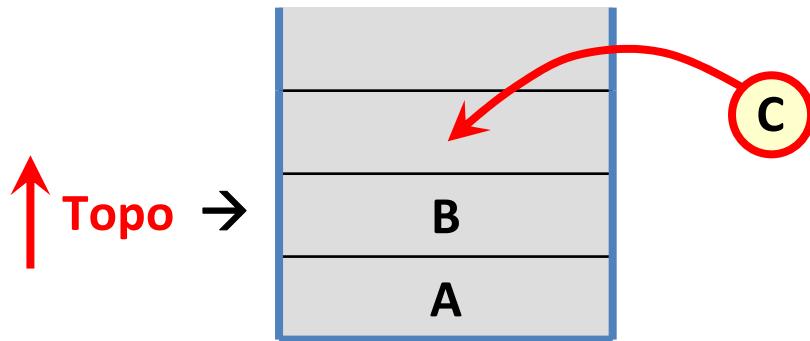
```
// Função POP
pop(pnt_s)                                // Estrutura passada
    struct stack *pnt_s;                    // como ponteiro
{
    // Testa underflow
    if (empty(pnt_s)) {
        printf("%s", "A pilha está vazia (underflow)");
        exit(1);
    }
    // Retorna item e decrementa top
    return(pnt_s -> items[pnt_s -> top--]);
}
```

Implementando a função POP

- **Interpretação do script**
 - Se a pilha não estiver vazia, o **primeiro valor** será retornado
 - Se **pnt_s->top** for igual a 87, existem apenas 88 itens na pilha
 - O valor de **pnt_s->items[87]** é retornado
 - O valor de **pnt_s->top** é decrementado para 86
 - Observar que **pnt_s->items[87]** mantém ainda seu antigo valor
 - O vetor **pnt_s->items** não é alterado com a chamada de **pop**
 - Porém a **pilha** é modificada, e tem agora **86** itens
 - **Lembre-se que vetor e pilha são objetos diferentes!**

Implementando a função PUSH

- `push (&s, x)` → Colocar um item do topo (**Empilhar**)
- Deve-se atentar para o tamanho do vetor
 - Evitar **overflow**



Implementando a função PUSH

- **Implementação da função push ()**

```
// Função PUSH
push(pnt_s, x)                      // Estrutura passada
    struct stack *pnt_s;              // como ponteiro
    int      x;
{
    if(pnt_s -> top == STACKSIZE-1) {
        printf("%s", "Estouro de pilha");
        exit(1);
    }
    else                            // Empilhamento de "x" em "s"
        // Incremento de "top"
        pnt_s -> items[++(pnt_s -> top)] = x;
    return;
}
```

Implementando a função STACKTOP

- Retorna o primeiro elemento da pilha sem removê-lo
- Não é uma operação primitiva
 - Decomposta em duas operações primitivas: `pop()` e `push()`

```
// Função STACKTOP
stacktop(pnt_s)                      // Estrutura passada
    struct stack *pnt_s;                // como ponteiro
{
    if (empty(pnt_s)) {               // Teste de underflow
        printf("%s", "Underflow");
        exit(1);
    }
    else                                // Como a pilha emprega um vetor,
                                            // não precisa desempilhar
        return(pnt_s -> items[pnt_s -> top]);
}
```

Resumo

- Uma pilha é um **conjunto ordenados de itens**
- **Representação em C**
 - Estrutura {
 vetor → para os dados (itens)
 inteiro → para o topo
- **Implementação das funções primitivas**
 - `empty (&s)` → Testa underflow
 - `push (&s, x)` → Empilha x e muda o topo (acima)
 - `pop (&s)` → Desempilha e muda o topo (abaixo)
 - `stacktop (&s)` → Lê o dado do topo, sem desempilhar