Estrutura de Dados II

Prof. Sérgio Portari - 2016

Plano de Ensino

- **EMENTA**:

Filas, Árvores e suas generalizações: árvores binárias, árvores de busca, árvores balanceadas, árvores B e B+. Aplicações de árvores

- BIBLIOGRAFIA BÁSICA:

DROZDEK, A. Estrutura de Dados e Algoritmos em C++. São Paulo: Editora Pioneira Thomson Learning, 2002.

MORAES, C.R. Estruturas de Dados e Algoritmos – Uma abordagem didática. São Paulo: Editora Berkeley Brasil, 2001.

TERADA, R. Desenvolvimento de Algoritmo e Estruturas de Dados. São Paulo:

Makron Books, 1991.

WIRTH, N. Algoritmos e Estruturas de Dados. Rio de Janeiro: LTC Editora. 1999.

- BIBIOGRAFIA COMPLEMENTAR

SALVETTI, D. D. & BARBOSA L. M. Algoritmos. São Paulo: Makron Books, 1998

PEREIRA, S. do L. Estrutura de Dados Fundamentais. São Paulo: Editora Érica.1996

TENENBAUM, A. M. Data Structures Using Pascal. São Paulo Prentice-Hall Inc, 1986

Objetivo

- Possibilitar que o aluno compreenda, utilize e implemente estruturas de dados avançadas com acesso à memória em C, que possam ser utilizados nas diversas aplicações pertinentes.

Metodologia

- Aulas expositivas;
- Aulas Práticas em Laboratório;
- Atividades individuais e em grupo.

Avaliação

- Trabalhos individuais ou em grupos, intra ou extra sala, totalizando até 30% do total semestral de pontos;
- Aplicação de 2 avaliações parciais, totalizando 40% do total semestral de pontos;
- Avaliação Semestral, totalizando 30% do total semestral de pontos.

Informações, Avisos e Contatos

- E-mail: **portari.uemgituiutaba@gmail.com**
  - Utilizem este e-mail para evitar outros endereços com SPAM que não acesso mais;

- Página: **www.sergioportari.com.br**

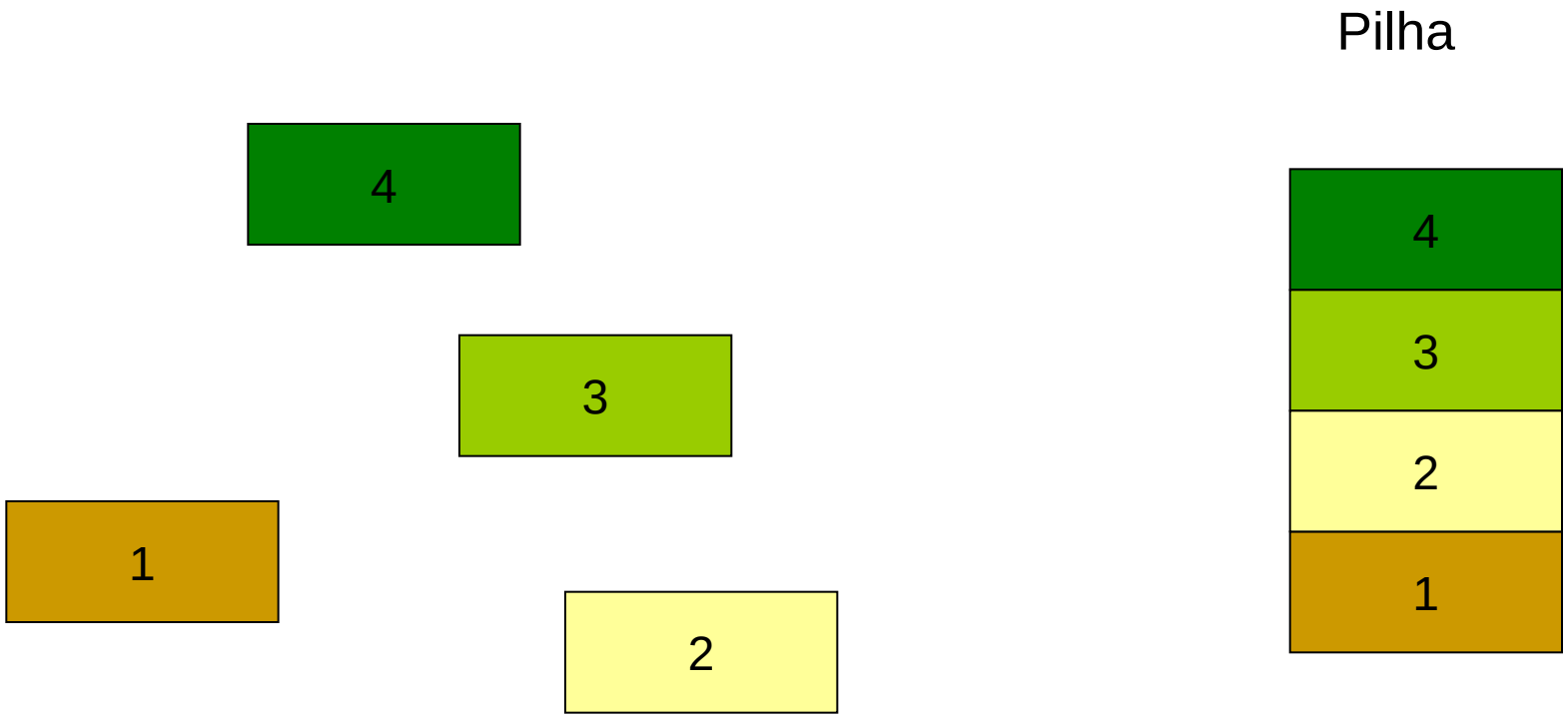
  - Na página você encontrará um link para informações da disciplina, datas importantes como entrega de trabalhos e provas, além de download dos materiais apresentados em sala.

Dúvidas?

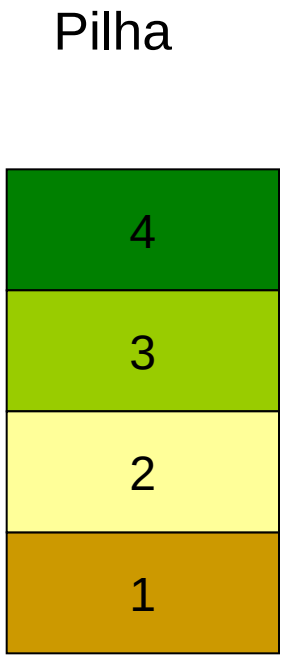
Revisão de Pilhas

Sérgio Carlos Portari Júnior

O que é uma pilha?



O que é uma pilha?



TAD Pilha

- Tipo Abstrato de dados com a seguinte característica:
  - O **último** elemento a ser inserido é o **primeiro** a ser retirado/ removido

    (LIFO – *Last in First Out*)

- Analogia: pilha de pratos, livros, etc.

- Usos: Chamada de subprogrmas, avalição de expressões aritméticas, etc.

TAD Pilha

- Há uma célula cabeça no topo para facilitar a implementação das operações empilha e desempilha quando a pilha estiver vazia.
- Para desempilhar o item $x_{(n)}$ basta desligar a célula cabeça da lista e a célula que contém $x_{(n-1)}$ passa a ser a célula cabeça.
- Para empilhar um novo item, basta fazer a operação contrária, criando uma nova célula cabeça e colocando o novo item na antiga.

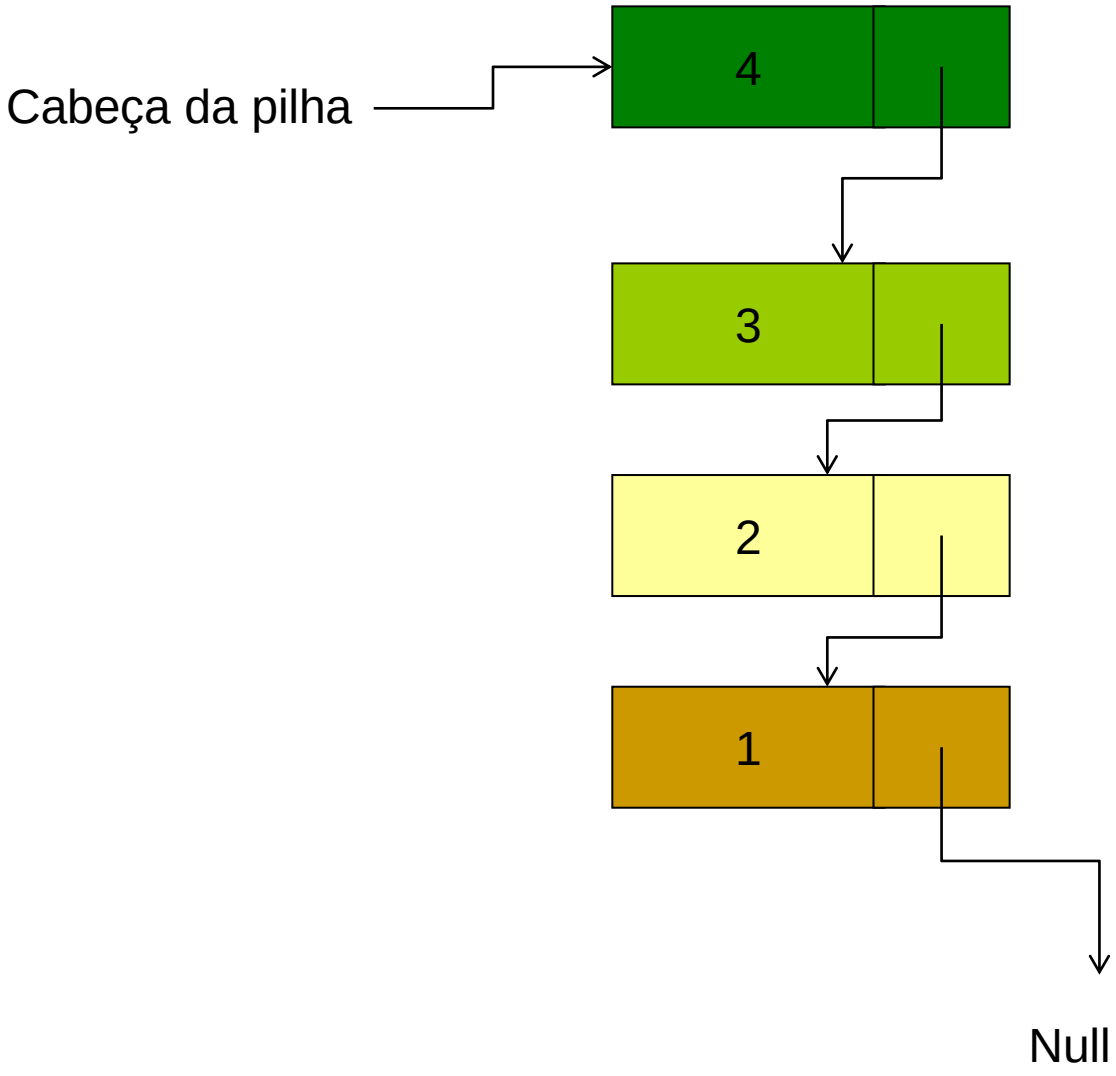
TAD Pilha

- Cada célula de uma pilha contém um item da pilha e um ponteiro para outra célula.

- O registro TipoPilha aPilha contém um ponteiro para o topo da pilha (célula cabeça)

- A última célula da pilha aponta para NULO

Pilha



Estrutura

- Para utilizar uma pilha, criamos uma estrutura de dados que guarda as informações e um ponteiro para o próximo elemento da pilha.
- Exemplo:

```
typedef struct pilha {
  int info;
  struct pilha *prox;
} Pilha;
```

Criando a pilha

- Para criar a pilha na memória, chamamos uma função que retorna um ponteiro para vazio, reservando um espaço na memória para o ponteiro que irá apontar a cabeça da pilha.

```
Pilha *criaPilha()
{
    return NULL;
}
```

Inserindo elementos

- Como dito, a inserção sempre será feita no início da pilha, então criamos uma função de inserção (PUSH) com a seguinte estrutura:

```
Pilha *Push (Pilha *p, int i)
{
   Pilha *novo = (Pilha*) malloc(sizeof(Pilha));
   novo->info = i;
   novo->prox = p;
return novo;
}
```

Testando se está vazia

- Em algumas situações, precisaremos saber se existe algum elemento na pilha, ou, se ela está vazia.

```
int PilhaVazia (Pilha *p)
{
   return (p == NULL);
}
//retorna 0 (falso) se existe elemento ou
//diferente de zero (verdadeiro) se vazia.
```

Removendo elementos

- De forma análoga, a remoção (Pop) também ocorre apenas na cabeça da pilha.

```
Pilha *Pop (Pilha *p)
{
  Pilha *aux=p;
  if (PilhaVazia(p))
    printf("\nPilha vazia");
  else
  {
     p=p->prox;
     free(aux);
     return p;
  }
}
```

Procura um elemento

- Para buscar um elemento específico na pilha, iremos percorre-la da cabeça para o fundo, procurando o elemento.

```
Pilha *Busca (Pilha *p, int v)
{ Pilha *aux;
for (aux=p; aux!=NULL; aux = aux->prox)
{
   if (aux->info == v)
      return aux; //retorna a célula se encontrou o elemento
}
return NULL; // não achou o elemento, retorna vazio
}
```

Liberando a pilha

- Para liberar a memória, percorremos célula a célula, da cabeça ao fundo, deligando as conexões e liberando a memória.

```
void liberaPilha (Pilha *p)
{
Pilha *l = p, *t;
while ( l != NULL)
   {
      t = l -> prox; /* guarda referência p/ próx. elemento */
      free(l); /* libera a memória apontada por p */
      l = t; /* faz p apontar para o próximo */
   }
}
```

Exercícios

- Organize as funções de pilha em C apresentadas no exemplo. Crie um programa que faça um menu para acionar as operações de uma pilha de acordo com a escolha do usuário
  - 1 – Criar pilha
  - 2 – Adicionar elemento
  - 3 – Buscar Elemento
  - 4 – Mostrar pilha
  - 5 – Liberar Pilha
  - 6 - Sair