Introdução à Computação II – AULA 16

BCC Noturno - EMA896115B

Prof. Rafael Oliveira
olivrap@gmail.com

Universidade Estadual Paulista
“Júlio de Mesquita Filho”
UNESP

Rio Claro 2014 (Sem 2)

Estruturas de Dados
TAD Pilha – Implementação
Estática versus Dinâmica

Fontes Bibliográficas

- Livros:
  - Projeto de Algoritmos (Nivio Ziviani): Capítulo 3;
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): Capítulo 10;
  - Estruturas de Dados e seus Algoritmos (Szwarefiter, et. al): Capítulo 2;
  - Algorithms in C (Sedgewick): Capítulo 3;
- Slides baseados nas transparências disponíveis em: http://www.dcc.ufmg.br/algoritmos/transparencias.php

Pilhas

- É uma lista linear em que todas as inserções, retiradas e, geralmente, todos os acessos são feitos em apenas um extremo da lista.
- Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.
- O modelo intuitivo é o de um monte de pratos em uma prateleira, sendo conveniente retirar ou adicionar pratos individualmente na parte superior.

Propriedades e Aplicações das Pilhas

- Propriedade: o último item inserido é o primeiro item que pode ser retirado da lista. São chamadas listas **lifo** (“last-in, first-out”), ao contrário de uma *fila* (**fifo** – first-in, first out).
- Existe uma ordem linear para pilhas, do “mais recente para o menos recente”.
- É ideal para processamento de estruturas aninhadas de profundidade imprevisível.
- Uma pilha contém uma sequência de obrigações adiadas. A ordem de remoção garante que as estruturas mais recentes serão processadas antes das mais antigas.

Propriedades e Aplicações das Pilhas (2)

- Aplicações em estruturas aninhadas:
  - Quando é necessário caminhar em um conjunto de dados e guardar uma lista de coisas a fazer posteriormente.
  - O controle de sequências de chamadas de subprogramas.
  - A sintaxe de expressões aritméticas.
- As pilhas ocorrem em estruturas de natureza recursiva (como árvores). Elas são utilizadas para implementar a **recursividade**.

TAD Pilha

- Conjunto de operações:
  - FPVazia (Pilha). Faz a pilha ficar vazia.
  - Vazia (Pilha). Retorna *true* se a pilha estiver vazia; caso contrário, retorna *false*.
  - Empilha (x, Pilha). Insere o item x no topo da pilha. (operação *push*)
  - Desempilha (Pilha, x). Retorna o item x no topo da pilha, retirando-o da pilha. (operação *pop*)
  - Tamanho (Pilha). Esta função retorna o número de itens da pilha.

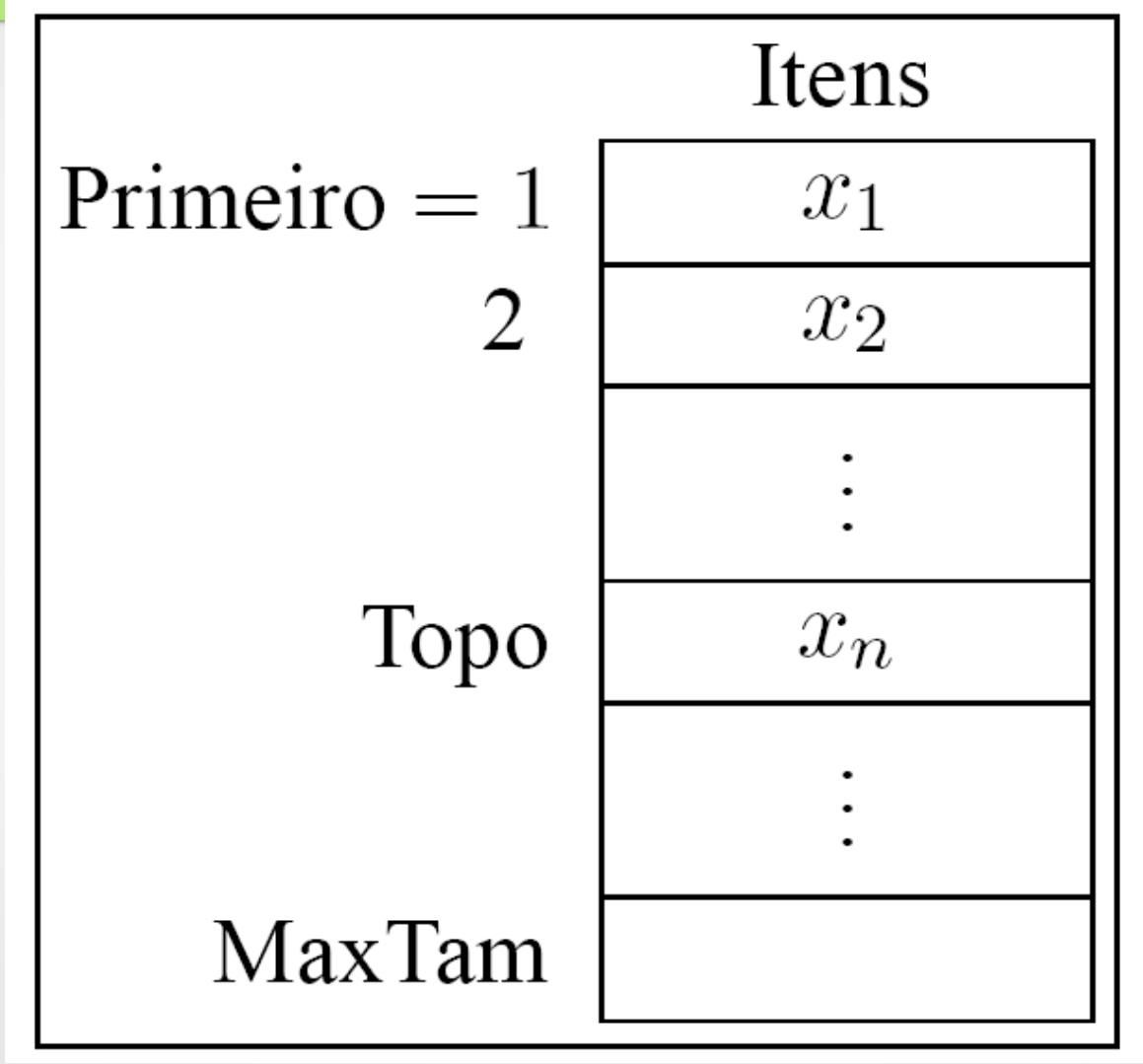
Implementação do TAD Pilha

- Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas.
- As duas representações mais utilizadas são as implementações por meio de *vetores* e de *estruturas encadeadas*.

Pilhas em Alocação Sequencial e Estática

- Os itens da pilha são armazenados em posições contíguas de memória (vetor de tamanho máximo).
- Como as inserções e as retiradas ocorrem no topo da pilha, um cursor chamado Topo é utilizado para controlar a posição do item no topo da pilha.

Pilhas em Alocação Sequencial e Estática (2)



Estrutura de Pilhas com Alocação Sequencial e Estática

- Os itens são armazenados em um **vetor** de tamanho suficiente para conter a pilha.
- O outro campo do mesmo registro contém o índice do item no topo da pilha.
- A constante MaxTam define o tamanho máximo permitido para a pilha.

Estrutura de Pilhas com Alocação Sequencial e Estática (2) --- pilha.h

```
#define MaxTam  1000
struct tipoitem {
  int valor;
  /* outros componentes */
};
typedef struct tipoitem TipoItem;

struct tipopilha {
  TipoItem* Item[MaxTam];
  int Topo;
};
typedef struct tipopilha TipoPilha;
```

Estrutura de Pilhas com Alocação Sequencial e Estática (2) --- pilha.h

```
TipoPilha* InicializaPilha();
void FPVazia(TipoPilha *Pilha);
int Vazia (TipoPilha* Pilha);
void Empilha (TipoItem* x, TipoPilha* Pilha);
void Desempilha (TipoPilha* Pilha, TipoItem*
  Item);
int Tamanho (TipoPilha* Pilha);
TipoItem* InicializaTipoItem (int n);
void Imprime (TipoPilha* pilha);
```

Implementação TAD Pilha com Vetores – pilha.c

```
#include <stdio.h>
#include <stdlib.h>
#include "pilha.h"
```

Implementação TAD Pilha com Vetores

```
TipoPilha* InicializaPilha(){
  TipoPilha* pilha
  =(TipoPilha*)malloc(sizeof(TipoPilha));
  return pilha;
}

void FPVazia(TipoPilha* Pilha){
  Pilha->Topo = 0;
}

int Vazia (TipoPilha* Pilha) {
  return (Pilha->Topo == 0);
}
```

Implementação TAD Pilha com Vetores – pilha.c

```
void Empilha (TipoItem* x, TipoPilha*
  Pilha) {
  if (Pilha->Topo == MaxTam)
     printf ("Erro: pilha esta cheia\n");
  else {
     Pilha->Item[Pilha->Topo] = x;
     Pilha->Topo++;
  }
}
```

Implementação TAD Pilha com Vetores (2) – pilha.c

```
void Desempilha (TipoPilha* Pilha,
  TipoItem* Item) {
  if (Vazia (Pilha))
     printf ("Erro: pilha esta vazia\n");
  else {
     Item = Pilha->Item[Pilha->Topo-1];
     Pilha->Topo--;
  }
}
int Tamanho (TipoPilha* Pilha) {
  return (Pilha->Topo);
}
```

Implementação TAD Pilha com Vetores (2) – pilha.c

```
TipoItem* InicializaTipoItem (int n)
{
   TipoItem* item = (TipoItem*)malloc(sizeof(TipoItem));
   item->valor = n;
   return item;
}

/*Imprime os itens da pilha */
void Imprime (TipoPilha* pilha)
{
   int Aux;
   printf ("Imprime Pilha Estatica de tamanho %d\n",
   Tamanho(pilha));

   for (Aux = 0; Aux < pilha->Topo; Aux++)
   {
       printf ("%d\n", pilha->Item[Aux]->valor);
   }
}
```

Implementação TAD Pilha com Vetores (2) – prog.c

```
#include <stdio.h>
#include <stdlib.h>
#include "pilha.h"

int main (int argc, char** argv)
{

   TipoPilha *pilha = InicializaPilha();
   FPVazia( pilha );

   TipoItem *top = NULL, *item = InicializaTipoItem(17);
   Imprime(pilha);
   Empilha (item, pilha);
   Imprime(pilha);
   Desempilha (pilha, top);
   Imprime(pilha);
   free(item);
   free(pilha);

   return EXIT_SUCCESS;
}
```

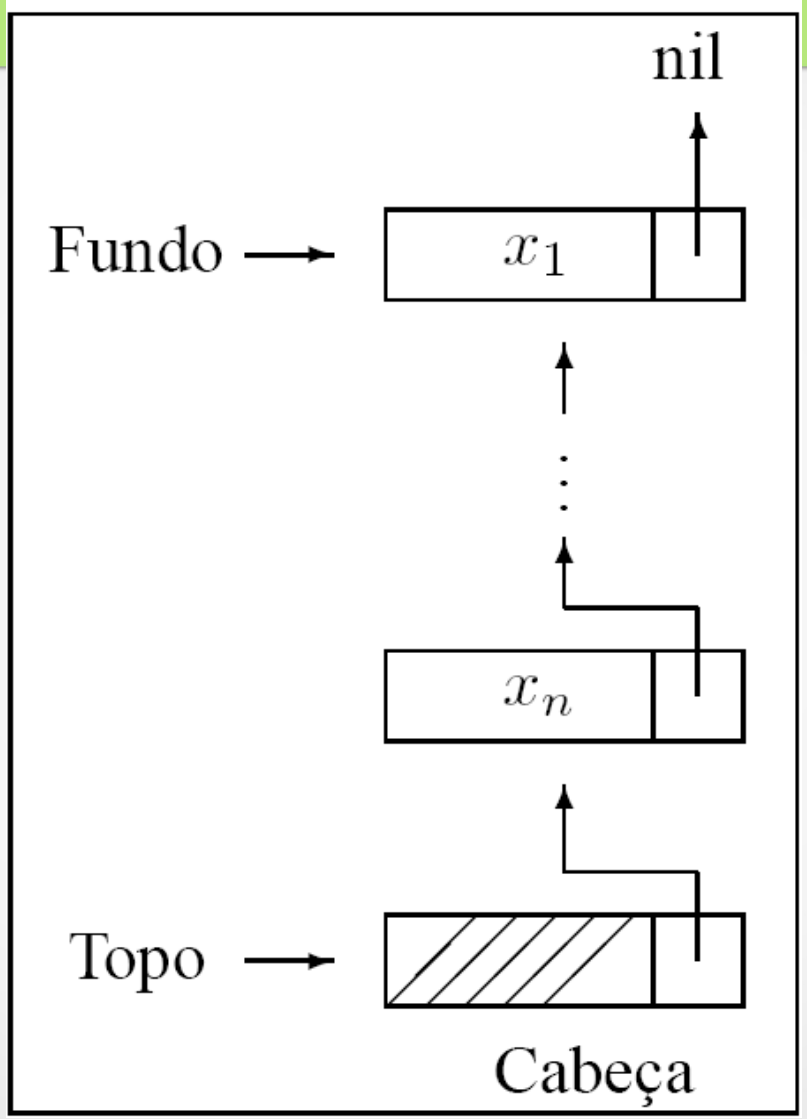
Pilhas com alocação não sequencial e dinâmica

- 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 (e liberar) a célula cabeça da lista e a célula que contém x_n 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.

Estrutura da Pilhas usando ponteiros

- O campo Tamanho evita a contagem do número de itens na função Tamanho.
- Cada célula de uma pilha contém um item da pilha e um ponteiro para outra célula.
- O registro TipoPilha contém um ponteiro para o topo da pilha (célula cabeça) e um ponteiro para o fundo da pilha.

Estrutura da Pilhas usando ponteiros (2)



Estrutura da Pilhas usando ponteiros (3): pilha.h

```
struct tipoitem {
        int valor;
        /* outros componentes */
};
typedef struct tipoitem TipoItem;

struct celula_str {
        TipoItem Item;
        struct celula_str* Prox;
};
typedef struct celula_str Celula;

struct tipopilha {
        Celula *Fundo, *Topo;
        unsigned int Tamanho;
};
typedef struct tipopilha TipoPilha;

typedef int TipoChave;
```

Estrutura da Pilhas usando ponteiros (3): Pilha.h

```
TipoPilha* InicializaPilha();
void FPVazia(TipoPilha *Pilha);
int Vazia (TipoPilha* Pilha);
void Empilha (TipoItem* x, TipoPilha* Pilha);
void Desempilha (TipoPilha* Pilha, TipoItem* Item);
int Tamanho (TipoPilha* Pilha);
TipoItem* InicializaTipoItem (TipoChave n);
void Imprime (TipoPilha* pilha);
```

Implementação TAD Pilhas usando ponteiros (pilha.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "pilha.h"

TipoPilha* InicializaPilha(){
	TipoPilha* pilha = (TipoPilha*)malloc(sizeof(TipoPilha));
	return pilha;
}

void FPVazia (TipoPilha* Pilha)
{
	Pilha->Topo = (Celula*)malloc (sizeof(Celula));
	Pilha->Fundo = Pilha->Topo;
	Pilha->Topo->Prox=NULL;
	Pilha->Tamanho = 0;
}

int Vazia (TipoPilha* Pilha){
	return (Pilha->Topo == Pilha->Fundo);
}
```

Implementação TAD Pilhas usando ponteiros (pilha.c)

```
void Desempilha (TipoPilha *Pilha, TipoItem *Item){
        Celula* q;
        if (Vazia (Pilha))   {
                printf ("Erro: lista vazia \n");
                return;
        }
        q = Pilha->Topo;
        Pilha->Topo = q->Prox;
        *Item = q->Prox->Item;
        free (q);
        Pilha->Tamanho--;
}

int Tamanho(TipoPilha* Pilha){
        return (Pilha->Tamanho);
}
```

Implementação TAD Pilhas usando ponteiros (pilha.c)

```
void Imprime (TipoPilha* pilha){
	Celula* Aux;
	Aux = pilha->Topo->Prox;
	printf ("Imprime Pilha Encadeada de tamanho %d\n", Tamanho(pilha));
	while (Aux != NULL)
	{
		printf ("%d\n", Aux->Item.valor);
		Aux = Aux->Prox;
	}
}

TipoItem* InicializaTipoItem (int n)
{
	TipoItem* item = (TipoItem*)malloc(sizeof(TipoItem));
	item->valor = n;
	return item;
}
```

Implementação TAD Pilhas usando ponteiros (prog.c)

include <stdio.h> ... #include <stdlib.h> ... #include "pilha.h"

```
int main (int argc, char** argv)
{     TipoPilha *pilha = InicializaPilha();
      FPVazia( pilha );

      TipoItem *item17 = InicializaTipoItem(17);
      TipoItem *item18 = InicializaTipoItem(18);
      Imprime(pilha);
      Empilha (item17, pilha);
      Imprime(pilha);
      Desempilha (pilha, item17);
      Imprime(pilha);
      Empilha (item17, pilha);
      Empilha (item18, pilha);
      Imprime(pilha);

      Desempilha (pilha, item17);
      Desempilha (pilha, item18);
      free(item17); free(item18);
      free(pilha->Topo); free(pilha);  return EXIT_SUCCESS; }
```

Vamos analisar outro exemplo!!!!

TAD Pilha com Alocação Dinâmica de Memória

Operações

- Não existe o teste de Pilha Cheia
  - Na Implementação do TAD Pilha com alocação Dinâmica não temos limite de espaço.
- Assim as operações do tad são as seguinte:
  - inicializaPilha
  - pilhaVazia
  - empilha
  - desempilha

Estrutura de dados:

```
typedef int Elemento; // tipo a ser
  armazenado
typedef struct nodo {
    Elemento item;
    struct nodo *prox;
}* Nodo;
typedef  struct {
                Nodo topo;
                 }Pilha;
```

Pré e pos condicoes

- Inicializa
  - Pre-condições: Não há
  - Pós-condições: topo aponta para NULL
- Empilha(elemento):
  - Pré-condições: não há
  - Pós-condições: topo aponta para novo nodo contendo o elemento. Prox de topo aponta para o topo anterior
- Desempilha
  - Précondições: pilhaVazia = FALSO
  - Pós-condições: remove o nodo do topo. Topo aponta para o próximo de topo.

Operações

```
void inicializaPilha(Pilha *);
int pilhaVazia(Pilha);
void empilha (Pilha *, Elemento);
int desempilha (Pilha * Elemento *);
Elemento mostraTopo(Pilha);
```

PilhaAD.h completo

```
/* * Interface do Tad PilhaAD* */
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define MSG_PILHAVAZIA "\nA Pilha está vazia\n"
#define MAX 10 // tamanho máximo da Pilha
#define TRUE 1
#define FALSE 0
typedef int Elemento; // tipo a ser armazenado
typedef struct nodo {
   Elemento item;
   struct nodo *prox;
}* Nodo;
typedef  struct {
              Nodo topo;
               }Pilha;
          int pilhaVazia(Pilha);
                  void inicializaPilha(Pilha *);
                  void empilha (Pilha *, Elemento);
                  int desempilha (Pilha * Elemento *);
                  Elemento mostraTopo(Pilha);
```

Implementação das Operações

```
/* tadPilha.c * Implementação das operações do Tad Pilha* */
#include "TadPilhaAD.h"
void inicializaPilha(Pilha *p) {
        p->topo=NULL;
}
int pilhaVazia(Pilha p){
     return (p.topo==NULL);
  }
void empilha (Pilha *p, Elemento ele){
    Nodo novoNodo;
    novoNodo = malloc(sizeof(struct nodo));
    novoNodo->item=ele;
    novoNodo->prox=p->topo;
    p->topo=novoNodo;
}
```

Implementação das Operações

```
int desempilha(Pilha *p,Elemento ele) {

    Nodo aux;
   if (pilhaVazia(*p)==FALSE) {
      *ele = p->topo->item;
      aux = p->topo;
      p->topo = p->topo->prox;
      free(aux);
  }
  else{
     fprintf(stderr, MSG_PILHAVAZIA);
    }
    return ele;
  }
```

Testando o tadPilha (principal.c)

```
#include"TadPilhaAD.h"
void testePilhaAD();
void testePilhaAD(){
  Pilha p;
  int i;
  Elemento e;
  inicializaPilha(&p);
  for(i=0;i<10;i++) {
     empilha(&p,i);
  }
  while(pilhaVazia(p)==FALSE){
     e = desempilha(&p);
     printf("%d\n",e);
  }
  fprintf(stderr,"Terminou ok.\n");

}
int main(){
  testePilhaAD();
return 0;
}
```

Exercício/Desafio

- Implementar uma solução para o problema das Torres de Hanoi utilizando Pilha