# Introdução a Programação


## Tipos Abstratos de Dados – Implementando Pilha e Fila

# Abstração

**Abstração** é o processo ou resultado de generalização por **redução** do conteúdo da **informação** de um conceito ou fenômeno observável, normalmente para reter apenas a informação que é **relevante** para um propósito particular.

Fonte: Wikipedia

# Tópicos da Aula

- Hoje aprenderemos que existe uma forma de definir um tipo de dado pelas suas operações
  - Conceito de Tipo Abstrato de Dado
  - Dois Tipos Abstratos de Dados: Pilha e Fila
- Depois aprenderemos como implementar Pilhas e Filas
  - Implementação de Pilha usando Vetor
  - Implementação de Pilha usando Lista
  - Implementação de Fila usando Lista

# Tipo Abstrato de Dado

- Um **Tipo Abstrato de Dado** (TAD) é um conjunto organizado de informações e as operações que podem atuar sobre estas informações
  - Define um novo tipo de dado
- Um TAD é definido indiretamente pelas operações que podem atuar nele e os efeitos destas operações

# Exemplo de TAD: Pilha


- Operações sobre Pilha (Livros) : inserção no topo e remoção no topo
- Tipo é definido pelo comportamento
  - Conjunto de operações sobre o tipo

# Exemplo de TAD: Fila


- Operações sobre Fila (Pessoa) : inserção no fim e remoção no começo

# TAD x Estrutura de Dado

- O TAD é a descrição lógica de um dado e a estrutura de dado é a descrição real
- TAD (Tipo Abstrato de Dados) é a "figura lógica" dos dados e operações que manipulam os elementos componentes dos dados
- Estrutura de Dados é a implementação real dos dados e algoritmos que manipulam os elementos dos dados
- TAD $\rightarrow$ descrição das funcionalidades
- Estrutura de Dado $\rightarrow$ implementação

# Objetivos deTADs

- Abstrair (esconder) de quem usa um determinado tipo, a forma concreta com que ele foi implementado
  - O cliente utiliza o TAD de forma abstrata, apenas com base nas funcionalidades oferecidas pelo tipo (interface)
  - Desacoplar a implementação do uso, facilitando a manutenção e aumentando o potencial de reuso do tipo criado

# Implementando um TAD

- As operações definem a interface de um TAD
- Um código implementa corretamente um TAD desde que obedeça o que está sendo definido na interface do TAD

# Tipos Abstratos de Dados em C

- Em C:
  - Arquivos-fontes que agrupam funções afins são geralmente denominados de Módulos
  - Em programas modulares, cada módulo deve ser compilado separadamente e depois “linkados” para gerar executável
  - Quando módulo define um novo tipo de dado e o conjunto de operações para manipular dados deste tipo, dizemos que o módulo representa um Tipo Abstrato de Dado (TAD)

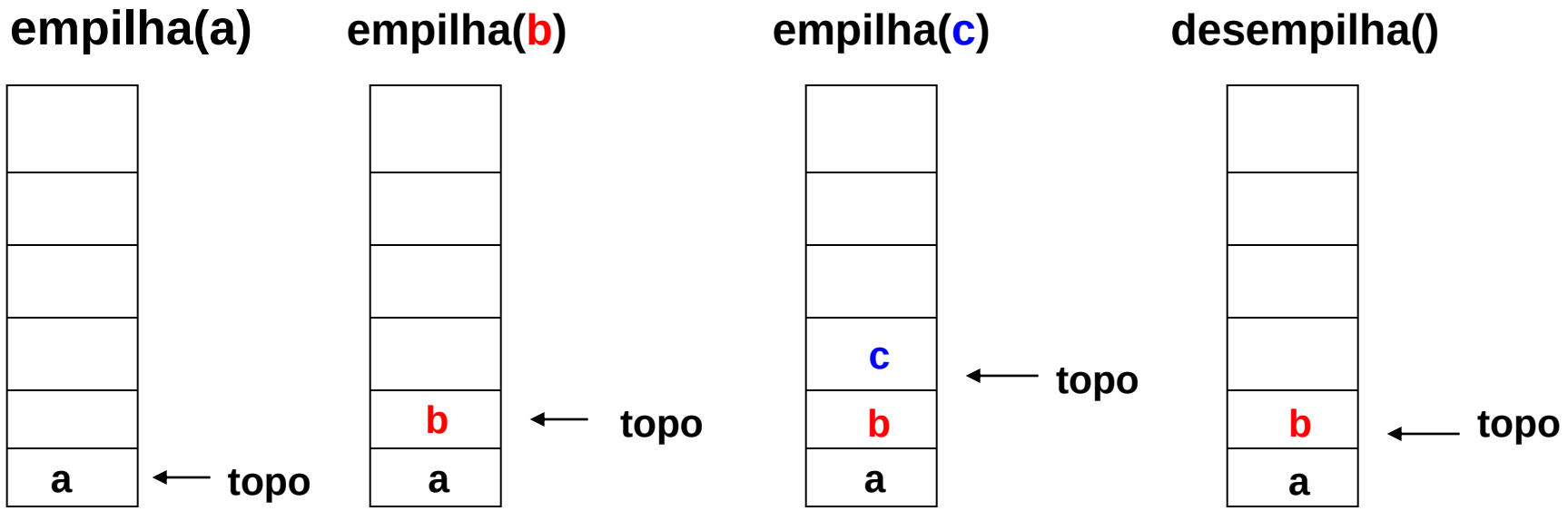
# Implementando TADs em C

- Geralmente a interface de um TAD é descrita em C nos arquivos .h
- O cliente só precisa dar um "include" no .h que contém a interface do TAD
  - Ao cliente também é dado o arquivo objeto (.o) com a implementação do TAD
    - Esconde (Abstrai) a implementação
- A interface de um TAD contém as assinaturas das funções que ele oferece e contém a definição do tipo de dado
  - Funciona como um "manual de uso" do tipo que está sendo definido

# Definição de Pilha

- Pilha é um tipo abstrato de dados onde:
  - Inserção e remoção de elementos no topo da pilha
  - O primeiro elemento que sai é o último que entrou (LIFO)
  - Operações básicas: push (empilhar) e pop (desempilhar)

# Funcionamento da pilha

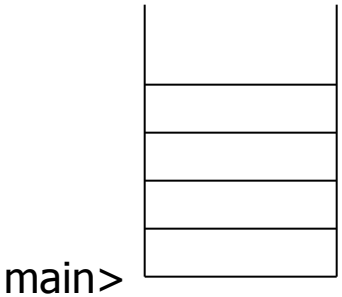


# Exemplo de Uso: Pilha de Execução de Funções

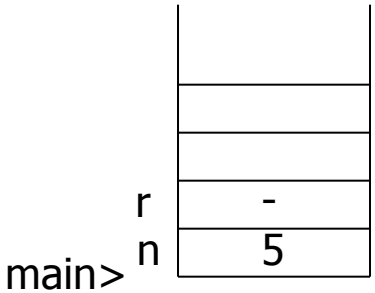
```
#include "stdio.h"
int  fat ( int  n ) ;
int main () {
  int  n = 5,  r ;
  r = fat(n) ;
  printf ( " Fatorial  = %d \n ",  r ) ;
  return 0 ;
}
int  fat  ( int  n ){
  int  f=1 ;
  while(n != 0) {
      f  *=  n ;
      n-- ;
  }
  return f ;
}
```

# Pilha de Execução

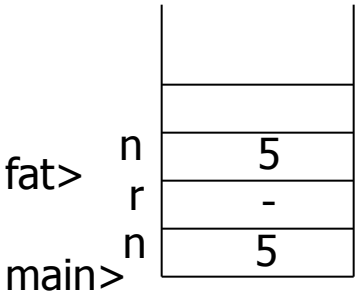
1 – Início do programa: pilha vazia




2 – Declaração de variáveis: n,r

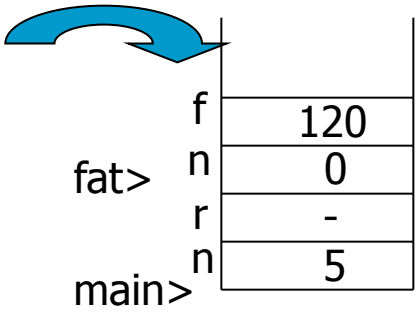



3 – Chamada da função: empilha variáveis

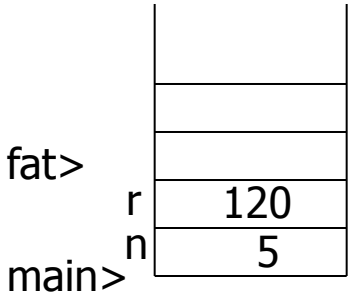



Acesso às variáveis que estão na função do topo

4 – Final do laço




5 – Retorno da função: desempilha



# Interface do tipo Pilha

- Criar pilha vazia;
- Inserir elemento no topo (push)
- Remover elemento do topo (pop)
- Verificar se a pilha está vazia
- Liberar a estrutura de pilha

- Em C

```
/* pilha.h */
typedef struct pilha Pilha;
Pilha* pilha_cria();
void pilha_push(Pilha* p,float v);
float pilha_pop(Pilha* p);
int pilha_vazia(Pilha* p);
void pilha_libera(Pilha* p);
```

# Implementação do tipo Pilha

- Existem várias implementações possíveis de uma pilha. Podem diferir da natureza dos elementos, maneira como são armazenados e operações disponíveis
- Iremos estudar 2 tipos de implementação:
  - Utilizando Vetor
  - Utilizando Lista Encadeada

# Implementando Pilha com Vetor

- Estrutura que representa pilha deve ser composta por um vetor para armazenar os elementos e o número de elementos armazenados
- Vantagem
  - Simplicidade
- Desvantagens
  - Deve-se saber de antemão o número máximo de elementos
  - Desperdício de espaço de memória

```
#define N 50
struct pilha {
   int n;
   float vet[N];
};
```

# Implementando Pilha com Vetor

- Função de Criação

Aloca estrutura dinamicamente

```
Pilha* pilha_cria ()
{
  Pilha* p = (Pilha*) malloc(sizeof(Pilha));
  p->n = 0;
  return p;
}
```

Inicializa com 0 elementos

# Implementando Pilha com Vetor

- Função de Inserção

```
void pilha_push (Pilha* p, float v) {
  if (p->n < N) {
      p->vet[p->n] = v;
      p->n++;
   } else {/* capacidade esgotada */
      printf("Capacidade da pilha estourou.\n");
   }

}
```

Verifica se tem espaço disponível

Insere na próxima posição livre e incrementa o número de elementos da pilha

# Implementando Pilha com Vetor

- Função de Remoção

```
float pilha_pop (Pilha* p)
{
  float v;
  if (pilha_vazia(p)) {
     printf("Pilha vazia.\n");
     exit(1);
  }
  v = p->vet[p->n-1];
  p->n--;
  return v;
}
```

Verifica se a pilha está vazia

Topo está na posição n - 1

Remoção se dá pelo decremento do número de elementos (próximo push sobrescreverá antigo topo)

# Implementando Pilha com Vetor

- Função que testa se pilha está vazia

```
int pilha_vazia (Pilha* p) {
  return (p->n==0);
}
```

- Função que libera memória alocada para a pilha

```
void pilha_libera (Pilha* p) {
  free(p);
}
```

# Implementando Pilha com Vetor

- Outras Funções Utilitárias

```
/* Função que informa o elemento do topo */
float pilha_topo (Pilha* p) {
  return (p->vet[p->n - 1]);
}
```

```
/* Função que imprime os elementos da pilha
void pilha_imprime (Pilha* p) {
  int i;
  for (i=p->n-1; i>=0; i--)
     printf("%f\n", p->vet[i]);
}
```

# Implementando Pilha com Lista

- Estrutura que representa pilha deve ser composta por um ponteiro para o primeiro nó da lista que armazena os elementos
- Vantagens
  - Otimização da memória
  - Tamanho irrestrito
- Desvantagem
  - Complexidade na manipulação de listas

```
typedef struct lista{
  float info;
  struct lista *prox;
} Lista;

struct pilha {
  Lista *topo;
};
```

# Implementando Pilha com Lista

- Função de Criação

Lista(topo) é inicializada com NULL

```
Pilha* pilha_cria () {
  Pilha* p = (Pilha*) malloc(sizeof(Pilha));
  p->topo = NULL;
  return p;
}
```

# Implementando Pilha com Lista

- Função de Inserção

```
void pilha_push (Pilha* p, float v) {
  Lista* novo = (Lista*) malloc(sizeof(Lista));
  novo->info = v;
  novo->prox = p->topo;
  p->topo = novo;
}
```

Elemento é inserido no começo da lista - topo aponta para o começo da lista

# Implementando Pilha com Lista

- Função de Remoção

```
float pilha_pop (Pilha* p) {
  Lista* t;
  float v;
  if (pilha_vazia(p)) {
     printf("Pilha vazia.\n");
     exit(1);      /* aborta programa */
  }
  t = p->topo;
  v = t->info;
  p->topo = t->prox;
  free(t);
  return v;
}
```

Elemento é realmente removido, topo é atualizado

# Implementando Pilha com Lista

- Função que testa se pilha está vazia

```
int pilha_vazia (Pilha* p) {
  return (p->topo == NULL);
}
```

- Função que libera memória alocada para a pilha

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

Deve-se liberar todos os elementos da lista primeiro

# Implementando Pilha com Lista

- Outras Funções Utilitárias

```
/* Função que informa o elemento do topo */
float pilha_topo (Pilha* p) {
  return (p->topo->info);
}
```

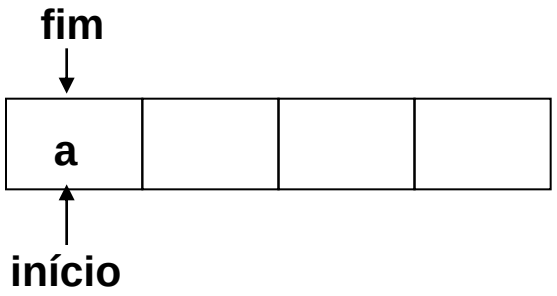
```
/* Função que imprime os elementos da pilha
void pilha_imprime (Pilha* p) {
  Lista* q;
  for (q=p->topo; q!= NULL;  q = q->prox)
     printf("%f\n", q->info);
}
```

# Definição de Fila

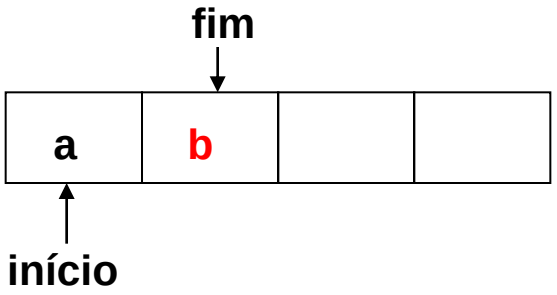
- Fila é um tipo abstrato de dados onde:
  - Inserção de elementos se dá no final e a remoção no início
  - O primeiro elemento que entra é o primeiro que sai (FIFO)
  - Ex de uso : fila de impressão

# Funcionamento da Fila

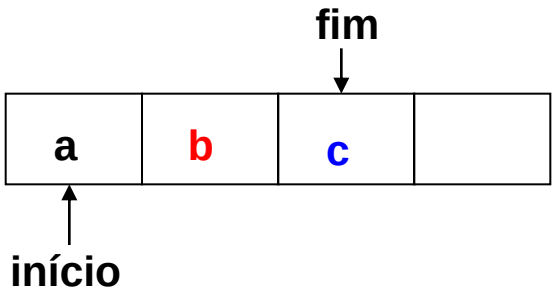
1- insere(a)



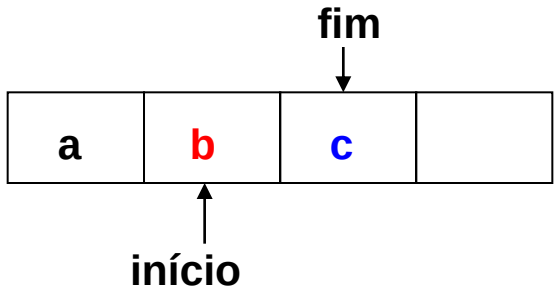

2- insere(b)




3- insere(c)




4- retira( )



# Interface do tipo Fila

- Criar uma fila vazia
- Inserir um elemento no fim
- Retirar o elemento do início
- Verificar se a fila está vazia
- Liberar a fila

- Em C

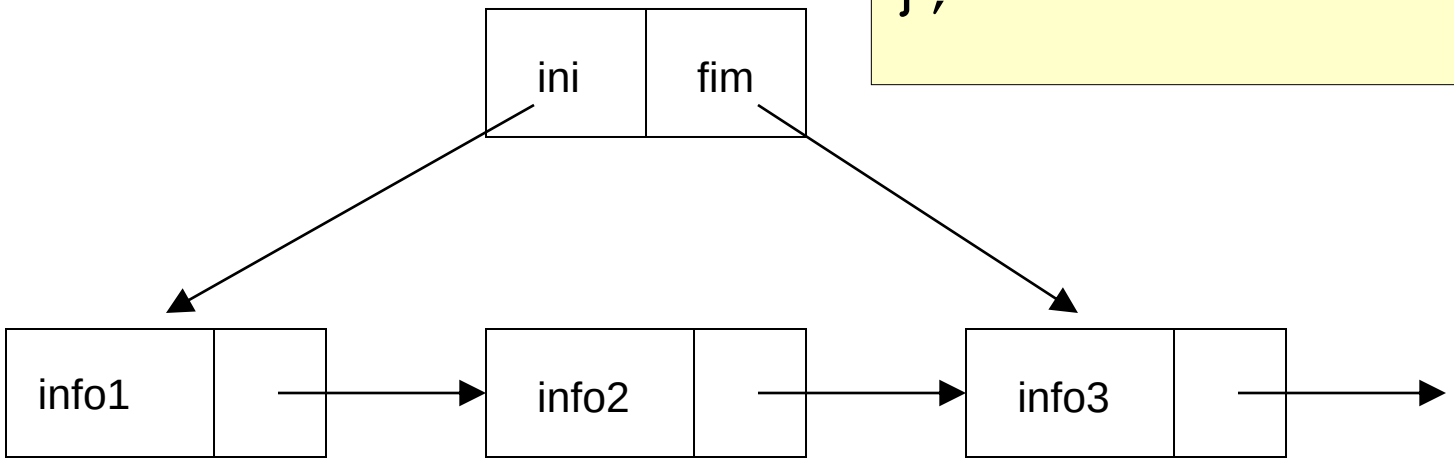
```
/* fila.h */
typedef struct fila Fila;
Fila* fila_cria();
void fila_insere(Fila* f,float v);
float fila_retira(Fila* f);
int fila_vazia(Fila* f);
void fila_libera(Fila* f);
```

# Implementando Fila com Lista

- Estrutura que representa fila deve ser composta por 2 ponteiros para a lista que armazena os elementos.
  - Um aponta para o primeiro elemento e o outro para o último elemento

```
typedef struct lista{
   float info;
   struct lista *prox;
} Lista;

struct fila {
   Lista *ini;
   Lista *fim
};
```



# Implementando Fila com Lista

- Função de Criação

```
Fila* fila_cria ( ) {
    Fila* f = (Fila*) malloc(sizeof(Fila));
    f->ini = f->fim = NULL;
    return f;
}
```

ini e fim são inicializados com NULL

# Implementando Fila com Lista

- Função de Inserção

```
void fila_insere (Fila* f, float v) {
   Lista* novo = (Lista*) malloc(sizeof(Lista));
   novo->info = v;
   novo->prox = NULL; /* novo nó passa a ser o
  último */
   if (!fila_vazia(f)) /* verifica se a fila não
  estava vazia */
        f->fim->prox = novo;
   else
      f->ini = novo;

   f->fim = novo;
}
```

Se a fila estava vazia novo elemento passa a ser o começo da fila

Novo elemento sempre é o último da fila

# Implementando Fila com Lista

- Função de Remoção

```
float fila_retira (Fila* f) {
   Lista* t;
   float v;
   if (fila_vazia(f)) {
       printf("Fila vazia.\n"); exit(1);
   }
   t = f->ini;
   v = t->info;
   f->ini = t->prox;
   if (f->ini == NULL)
       f->fim = NULL;
   free(t);
   return v;
}
```

Se a fila ficou vazia, deve-se atualizar ponteiro para o fim

# Implementando Fila com Lista

- Função que testa se a fila está vazia

```
int fila_vazia (Fila* f) {
   return (f->ini == NULL);
}
```

- Função que libera memória alocada para a fila

```
void fila_libera (Fila* f) {
   Lista* q = f->ini;
   while (q!=NULL) {
      Lista* t = q->prox;
      free(q);
      q = t;
   }
   free(f);
}
```

Deve-se liberar todos os elementos da lista primeiro

# Implementando Fila com Lista

- Outras Funções Utilitárias

```
/* Função que informa o primeiro elemento da fila
   */
float fila_primeiro (Fila* f) {
  return (f->ini->info);
}
```

```
/* Função que imprime os elementos da fila
void fila_imprime (Fila* f) {
   Lista* q;
   for (q=f->ini; q!=NULL; q =q->prox)
         printf("%f\n", q->info);
}
```