Estrutura de Dados 1

Prof. Igor Calebe Zadi
igor.zadi@ifsp.edu.br

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

Tipos de Dados Abstratos (TAD)

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

Abstração

- Uma abstração é uma representação que inclui apenas os atributos mais significativos
- Reduz complexidade
- Permite foco no essencial
- Exemplo: `sort(vetor, tam)` – não importa o algoritmo, apenas o efeito

Tipos de Abstração

1. Processo: subprogramas (funções)
2. Dados: **tipos + operações**, **ocultando** a estrutura interna

- “A abstração de dados segue a de processo, pois depende de operações abstratas”

Definição de TAD

- Um TAD é um tipo de dado com representação escondida e interface clara
  - Características:
    - Representação escondida
    - Interface com operações bem definidas
    - Usado por meio de funções
    - **Instâncias são objetos**

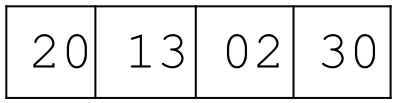
De outra maneira...

- Agrupa a estrutura de dados juntamente com as operações que podem ser feitas sobre esses dados
- O TAD **encapsula** a estrutura de dados.
  - Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados
- **Usuário do TAD x Programador do TAD**
  - Usuário só “enxerga” a interface, não a implementação

Exemplo

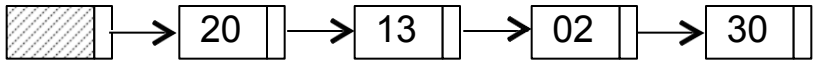
- Insere número no começo da lista

Implementação por Vetores:




```
        (i=0;...) {...}
    L[0]i=0;x.;..) {...}
    L[0] = x;
}
```

Implementação por Listas Encadeada




```
void Insere(int x, Lista L) { p
     = CriaNovaCelula(x);
     L.primeiro = p;
     ...
}
```

Programa usuário do TAD:

```
main() {
 Lista L;
 int x;
 x = 20;
 FazListaVazia(L);
 Insere(x,L);
 ...
}
```

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

O que se pode concluir

- Dessa forma, o usuário pode abstrair da implementação específica.
- Qualquer modificação nessa implementação fica restrita ao TAD.
- A escolha de uma representação específica é fortemente influenciada pelas operações a serem executadas.

Vantagens

- Ocultação de informação
- Confiabilidade
- Modularização
- Reusabilidade
- Segurança
- Mudanças internas não afetam o "cliente"

Implementação de TADs

- Em linguagens orientadas a objeto (C++, Java) a implementação é feita através de classes.
- Em linguagens estruturadas (C, pascal) a implementação é feita pela definição de **tipos** juntamente com a implementação de **funções**.
  - Conceitos de C (typedef e structs)

- Conceitos de orientação a objetos (classes, etc) serão vistos em outras disciplinas (POO).

Tipos em C (typedef)

```
typedef  struct   TipoAluno{
    char nome[50];
    int matricula;
    char conceito;
}  TipoAluno;
```

```
typedef int[10] Vetor;
```

```
int   main()      {
  TipoAluno   al;
  Vetor  v;
  ...
```

TADs em C

- Para implementar um Tipo Abstrato de Dados em C, usa-se a definição de tipos juntamente com a implementação de funções que agem sobre aquele tipo.
- Como boa regra de programação, evita-se acessar o dado diretamente, **fazendo o acesso somente através das funções**.
  - Mas, diferentemente de C++ e Java, não há uma forma de proibir o acesso.

TADs em C

- Uma boa prática de programação é implementar os TADs em arquivos separados do programa principal
- Para isso geralmente separa-se a declaração e a implementação do TAD em dois arquivos:
  - NomeDoTAD.h : com a declaração
  - NomeDoTAD.c : com a implementação
- O programa principal ou outros TADs que utilizam o TAD (criado) devem dar um **`#include`** no arquivo .h

Exemplo em C

- Implemente um TAD ContaBancaria, com os campos **número** e **saldo** onde os clientes podem fazer as seguintes operações:
  - Iniciar uma conta com um número e saldo inicial
  - Depositar um valor
  - Sacar um valor
  - Imprimir o saldo

conta.h

```
// definição do tipo
typedef struct ContaBancaria{
    int numero;
    double saldo;
} ContaBancaria;

// cabeçalho das funções
void Inicializa (ContaBancaria*, int, double);
void Deposito (ContaBancaria*, double);
void Saque (ContaBancaria*, double);
void Imprime (ContaBancaria);
```

conta.c

```c
#include <stdio.h>
#include "conta.h"

void Inicializa(ContaBancaria * ptr_conta, int numero, double saldo) {
     ptr_conta->numero = numero;
     ptr_conta->saldo = saldo;
}
void Deposito (ContaBancaria * ptr_conta, double valor) {
     ptr_conta->saldo += valor;
}
void Saque (ContaBancaria* ptr_conta, double valor) {
     ptr_conta->saldo -= valor;
}
void Imprime (ContaBancaria conta) {
     printf("Numero: %d\n", conta.numero);
     printf("Saldo: %f\n", conta.saldo);
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "conta.h"

int main (void) {
   ContaBancaria conta1;
   Inicializa(&conta1, 1234, 300);
   printf("\nAntes da movimentacao:\n ");
   Imprime(conta1);
   Deposito(&conta1, 50);
   Saque(&conta1, 70);
   printf("\nDepois da movimentacao:\n ");
   Imprime (conta1);

   return(0);
}
```

Exemplo em C

- Observações importantes:
  - Para executar compile os arquivos antes (gcc)
  - Os arquivos conta.c e main.c - **devem ser executados juntos**
  - Os arquivos devem ficar dentro de um "projeto" ou seja - no mesmo diretório

Encapsulamento

- “A interface é clara, a implementação é escondida”
  - Somente operações permitidas são acessíveis
  - Reduz o risco de erros
  - Permite alterações internas sem impacto externo

- Em C pode-se implementar (dependerá da codificação)
- Em outras linguagens Java, C++, etc existem mecanismos próprios para essa finalidade

Exercício

- Com base nos conceitos de Tipos Abstratos de Dados e encapsulamento, projete a interface de um TAD chamado Conjunto, que represente um conjunto de números inteiros sem repetição.
  Crie o arquivo conjunto.h com:

- A definição do tipo Conjunto (via typedef struct Conjunto Conjunto;)
- As funções públicas para criar, adicionar, remover e verificar elementos no conjunto
- Uma função para liberar memória alocada.
- Importante: não implemente as funções agora — concentre-se apenas na definição da interface (o que o TAD oferece, não como funciona).

Exercício

```
// conjunto.h
typedef struct Conjunto Conjunto;

Conjunto* conjunto_criar();
void conjunto_adiciona(Conjunto* c, int elemento);
void conjunto_remove(Conjunto* c, int elemento);
int conjunto_pertence(Conjunto* c, int elemento);
void conjunto_destruir(Conjunto* c);
```

Exemplo de pilha

- Arquivos:
  - pilha.h
  - pilha.c
  - main.c

Observações sobre o material eletrônico

- O material ficará disponível na pasta compartilhada que é acessada sob convite
- O material foi elaborado a partir de diversas fontes (livros, internet, colegas, alunos etc.)
- Alguns trechos podem ter sido inteiramente transcritos a partir dessas fontes
- Outros trechos são de autoria própria
- Esta observação deve estar presente em qualquer utilização do material fora do ambiente de aulas do IFSP - Catanduva

INSTITUTO FEDERAL
São Paulo
Campus Catanduva