BCC202 - Estrutura de Dados I

Aula 03: Tipos Abstratos de Dados (TADs)

Reinaldo Fortes

Universidade Federal de Ouro Preto, UFOP
Departamento de Ciência da Computação, DECOM

Website: **www.decom.ufop.br/reifortes**
Email: **reifortes@iceb.ufop.br**

Material elaborado com base nos slides do Prof. Túlio Toffolo (curso de 2013/01).

2013/02

Conteúdo

Conteúdo

Algoritmos e Estruturas de Dados

- **Algoritmo:**
  - Sequência de ações executáveis para a solução de um determinado tipo de problema.
  - Exemplo: "Receita de Bolo"
  - Em geral, algoritmos trabalham sobre estruturas de dados.
- **Estruturas de Dados:**
  - Conjunto de dados que representa uma situação real.
  - Uma abstração da realidade.
  - Estruturas de dados e algoritmos estão intimamente ligados.

Representação dos Dados

- Dados podem ser representados (estruturados) de diferentes maneiras.
- Normalmente, a escolha da representação é determinada pelas operações que serão utilizadas sobre eles.
- Exemplo: números inteiros.
  - Representação por palitinhos: `II + III = IIIII`.
    - Boa para números pequenos (operações simples).
  - Representação decimal: `1278 + 321 = 1599`.
    - Boa para números maiores (operações complexas)

Programas

- Um programa é uma formulação concreta de um algoritmo abstrato, baseado em representações de dados específicas.
- Os programas são feitos em uma linguagem que pode ser entendida e seguida pelo computador.
  - **Linguagem de máquina**: pouco inteligível para o programador.
  - **Linguagem assembly**: mais compreensível, porém ainda pouco inteligível para o programador (mais próxima da máquina do que do programador).
  - **Linguagem de alto nível**: muito mais compreensível para o programador do que para a máquina.
    - Nesta disciplina a linguagem utilizada é **C**.

Linguagem C

- Criada no início da década de 70 para a programação do sistema operacional Unix.
- Uma das linguagens mais utilizadas no mundo, e serviu como base para outras como C++, C#, etc.
- Filosofia: **"O programador sabe o que está fazendo"**.
- Lembre-se disto:

Só se aprende a programar PROGRAMANDO!

Linguagem C - Exemplo

```c
#include <stdio.h>

#define MAX 10

int LeInteiro() {
  int num;
  printf("Digite numero: ");
  scanf("%d", &num);
  printf("\n");
  return num;
}
```

```c
void main() {}
  int v[10], cont, aux, i,
    soma;
  double media;
  cont = 0;
  aux = LeInteiro();
  while(cont < MAX) {
    v[cont] = aux;
    aux = LeInteiro();
    cont++;
  }
  soma = 0;
  for(i = 0; i < cont; i++)
    soma += v[i];
  media = soma/ (double) cont;
  printf("resultado: %lf\n",
    media);
}
```

Conteúdo

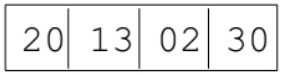
Conceito

- 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 *vs.* Programador do TAD.
  - Usuário só "enxerga" a interface, não a implementação.

Exemplo: Lista de números inteiros

- Insere um número no começo da lista.

Implementação por Vetor:




```
1 void Insere(int x, Lista L) {
2   for(i=0;...) {...}
3   L[0] = x;
4 }
```

Implementação por Lista Encadeada:


```
1 void Insere(int x, Lista L) {
2   p = CriaNovaCelula(x);
3   L.primeiro= p;
4   ...
5 }
```

Programa do usuário da TAD:

```
1 int main() {
2   Lista L;
3   int x;
4   x = 20;
5   FazListaVazia(L);
6   Insere(x,L);
7   ...
8 }
```

Motivação

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

Implementação

- 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).*

Estrutura (`struct`)

- Uma estrutura é uma coleção de uma ou mais variáveis colocadas juntas sob um único nome para manipulação conveniente.
- Por exemplo, para representar um aluno são necessárias as informações **nome**, **matrícula**, **conceito**.
- Ao invés de criar três variáveis, é possível criar uma única variável contendo três campos.
- Em C, usa-se a construção `struct` para representar esse tipo de dado.

Estrutura (struct)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct Aluno{
5    char nome[100];
6    int matricula;
7    char conceito;
8  };
9
10 void main() {
11   struct Aluno al, aux;
12
13   strcpy(al.nome, "Pedro");
14   al.matricula = 200712;
15   al.conceito = 'A';
16   aux = al;
17   printf("%s", aux.nome);
18 }
```

al:

Pedro	
200712	A

aux:

Pedro	
200712	A

Declaração de Tipos (typedef)

- Para simplificar, uma estrutura ou mesmo outros tipos de dados podem ser definidos como um novo tipo.
- Uso da construção **typedef**.

```
1  typedef struct {
2    char nome[100];
3    int matricula;
4    char conceito;
5  } TipoAluno;
6
7  typedef int[10] Vetor;
8
9  void main() {
10   TipoAluno al;
11   Vetor v;
12   ...
13 }
```

Práticas de programação de TAD

- 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 prática 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.

Práticas de programação de TAD

- Uma boa técnica 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 as declarações.
  - `NomeDoTAD.c`: com a implementação das declarações.
- Os programas, ou outros TADs, que utilizam o seu TAD devem dar um `#include` no arquivo `.h`.

Exemplo de TAD: Conta Bancária

- 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.
- Faça um pequeno programa para testar o seu TAD.

Declaração: contaBancaria.h

```
1  // definição do tipo
2  typedef struct{
3    int numero;
4    double saldo;
5  } ContaBancaria;
6
7  // cabeçalho das funções
8  void Inicializa(ContaBancaria*, int, double);
9  void Deposito(ContaBancaria*, double);
10 void Saque(ContaBancaria*, double);
11 void Imprime(ContaBancaria);
```

Implementação: contaBancaria.c

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

void Inicializa(ContaBancaria* pconta, int numero, double
    saldo) {
  pconta->numero = numero;
  pconta->saldo = saldo;
}

void Deposito(ContaBancaria* pconta, double valor) {
  pconta->saldo += valor;
}

void Saque(ContaBancaria* pconta, double valor) {
  pconta->saldo -= valor;
}

void Imprime(ContaBancaria conta) {
  printf("Numero: %d\n", conta.numero);
  printf("Saldo: %f\n", conta.saldo);
}
```

Utilização: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "contaBancaria.h"
4
5  int main(void) {
6    ContaBancaria conta1;
7    Inicializa(&conta1, 918556, 300.00);
8    printf("\nAntes da movimentacao:\n");
9    Imprime(conta1);
10   Deposito(&conta1, 50.00);
11   Saque(&conta1, 70.00);
12   printf("\nDepois da movimentacao:\n");
13   Imprime(conta1);
14
15   system("PAUSE");
16   return(0);
17 }
```

Como ficaria com alocação dinâmica?

Relembrando conceitos

- TAD 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 *vs.* Programador do TAD.
  - Usuário só "enxerga" a interface, não a implementação.
- Dessa forma, o usuário se abstrai da implementação específica.
- Qualquer modificação na implementação fica restrita ao TAD.
- A escolha de uma representação específica é fortemente influenciada pelas operações a serem executadas.

Conteúdo

Conclusão

- Nesta aula foram apresentados conceitos e exemplos de **Tipos Abstratos de Dados** (**TADs**).
- Do ponto de vista conceitual, maior atenção para o encapsulamento da estrutura de dados.
- Do ponto de vista de implementação, maior atenção para o uso das estruturas `struct` e `typedef`.
- *Próxima aula*: Análise de Algoritmos (Parte 1) – Noções de complexidade de algoritmos.
- Dúvidas?

Conteúdo

Exercício 01

- Implemente um TAD **Jogador de Futebol**.
  - Cada jogador possui os campos `nome`, `jogos`, `gols` e `assistências`.
  - Implemente as operações:
    - `Atribui`: atribui valores para os campos.
    - `Imprime`: imprime os dados/estatísticas do jogador.
    - `Soma`: soma estatísticas de dois jogadores.
    - `EhBom`: testa se o jogador é bom!!! (defina seu próprio critério de bom jogador)
- Crie o `main` para testar seu TAD.
- Utilize alocação dinâmica.

Exercício 02

- Incremente o exercício anterior, implementando uma TAD **Time de Futebol**.
  - Cada time possui os campos `nome`, `treinador`, `vitórias`, `empates`, `derrotas`, `jogadores` (um vetor de jogadores).
  - Implemente as operações:
    - `Atribui`: atribui valores para os campos. Um time precisa ter no mínimo cinco jogadores.
    - `Imprime`: imprime os dados/estatísticas do time.
    - `Pontuação`: retorna o número de pontos ganhos pelo time. Uma vitória corresponde a 3 pontos ganhos e um empate a 1 ponto ganho.
- Crie o `main` para testar seu TAD.
- Utilize alocação dinâmica.