# Anotações* de Estrutura de Dados

## Tipos de dados

** As anotações apresentadas neste documento se tratam de um resumo do capítulo 2 do livro Estruturas de Dados, das autoras Nina Edelweiss e Renata Galante.*

---

A representação dos valores manipulados por uma aplicação pode ser feita por diferentes estruturas de dados. A maioria das linguagens de programação oferecem tipos de dados, como os inteiros, real, lógico, etc. Outras linguagens oferecem até mesmo tipos estruturados como os vetores, matrizes, registros, entre outros.

Apesar disso, os tipos de dados presentes nas linguagens de programação possuem limitações quando se trata de representar estruturas de dados mais complexas que envolvem diferentes tipos e associações entre eles. Com isso, surgiu a necessidade de criar estruturas de dados mais apropriadas e suas respectivas operações.

Basicamente, uma aplicação em Ciência da Computação é um programa de computador que manipula dados. A representação dos dados manipulados por uma aplicação pode ser feita por diferentes Estruturas de Dados. Essas estruturas precisam coincidir com o domínio do problema a ser resolvido. Para isso, as linguagens de programação devem possuir uma variedade de tipos e estruturas para que representam adequadamente os dados manipulados pelas aplicações.

### 1.0 Definição de tipo de dados

Um tipo de dado consiste da definição do domínio (conjunto de valores) que uma variável pode assumir ao longo da execução de um programa e do conjunto de operações que podem ser aplicadas sobre ele. Por exemplo, o tipo de dado inteiro pode assumir valores do tipo inteiro, como {..., -3, -2, -1, 0, 1, 2, 3...}. Sobre esses dados as possíveis operações são a soma, subtração, multiplicação, entre outras.

## 1.1 Tipos de dados básicos e estruturados

Os tipos de dados podem ser classificados como básicos e estruturados.

Os tipos de dados básicos (ou primitivos) não possuem estrutura sobre seus valores. Em outras palavras, não é possível decompor o tipo primitivo em partes menores, sendo assim indivisíveis. São exemplos de tipos de dados básicos: inteiro, real, lógico e caractere.

Os tipos de dados estruturados permitem agregar mais do que um valor em uma variável, existindo uma relação estrutural entre seus elementos. Em outras palavras, esses dados são compostos por diversos campos. Exemplos de tipos de dados estruturados são os arranjos (vetores ou matrizes), registros (que agregam componentes de tipos diferentes), sequências e conjuntos.

Existem ainda os tipos de dados definidos pelo usuário. Esses são tipos de dados estruturados, construídos hierarquicamente pelo usuário através de componentes, que podem ser de tipos diferentes, agrupados sob um único nome. Normalmente, os elementos desse tipo estruturado têm alguma relação semântica. Um exemplo de tipo de dado definido pelo usuário seria a criação do tipo de dado Aluno, onde Aluno é definido pelos seguintes campos (atributos): nome, matrícula, e-mail.

Dentro desse contexto surgem as estruturas de dados que especificam conceitualmente os dados. As estruturas de dados incluem operações para a manipulação dos seus dados. Esse nível conceitual de abstração, ou seja, essas estruturas, não é fornecido diretamente pelas linguagens de programação. As linguagens fornecem apenas os tipos de dados e os operadores que permitem a construção de uma estrutura de dados para o problema que está sendo definido. A forma mais próxima de implementação de uma estrutura de dados é por meio dos Tipos de Abstratos de Dados.

## 1.2 Tipos Abstratos de Dados

Os Tipos Abstratos de Dados (TADs) são estruturas de dados capazes de representar os tipos de dados que não foram previstos pelas linguagens de programação. Geralmente, os TADs são necessários para aplicações específicas. Essas estruturas são compostas por dois elementos principais que são os dados e as operações. Em outras palavras, um TAD corresponde a escolha da maneira de armazenar os dados e a definição de um conjunto adequado de operações para manipular esses dados.

> Formalmente, um TAD pode ser definido como um *par (v, o)*, onde *v* é o conjunto de valores e *o* representa o conjunto de operações aplicáveis sobre esses valores.

Um exemplo de TAD pode ser dado de acordo com o seguinte problema: *quantos dias existem entre 24 de abril e 2 de fevereiro?* É possível resolver esse problema implementando um TAD para Data. Sendo assim, o TAD pode ser dado pelo par (v, o), onde:

- v : é uma tripla formada por dia, mês e ano;
- o : são as operações aplicáveis sobre o tipo Data, como por exemplo, verificar se uma data é válida, calcular o dia da semana de uma determinada data, entre outras.

Definido o TAD, então agora é necessário escolher uma estrutura de representação para suportar as operações definidas. Essa representação será uma coleção de campos primitivos ou uma estrutura complexa formada por vários campos primitivos. Considerando o exemplo citado acima, para o tipo de dado Data, pode-se definir a seguinte estrutura:

```
Data = registro
        dia:  inteiro
        mês: inteiro
        ano: inteiro
```

Além da estrutura descrita acima, a representação do TAD envolve a especificação de um conjunto de funções e procedimentos que podem ser executados sobre esse novo tipo de dado. Para o exemplo em questão, pode-se definir as seguintes operações aplicáveis sobre o TAD Data:

- **função InicializaData**

  **Entrada**: dia, mês, ano (inteiro)
  **Saída**: D (Data)
  *Essa função recebe três parâmetros inteiros, informando o dia, mês e ano, e retorna D, inicializado na data resultante.*

- **função AcrescentaDias**

  **Entrada:** D (Data), dias (inteiro)
  **Retorno:** (Data)
  *Soma um determinado número de dias a uma data recebida como parâmetro e retorna o resultado. Caso a operação não seja possível, retorna uma data cujo dia seja -1.*

- **função EscreveExtenso**

  **Entrada:** D (Data)
  **Retorno:** (lógico)
  *Recebe uma data e a escreve por extenso. Por exemplo, 01/01/2021 deve ser escrito como 1 de Janeiro de 2021. Retorna verdadeiro se a operação foi realizada com sucesso e, caso contrário, falso.*

Em linguagem de programação, a representação de um TAD é denominada de interface do TAD. O cliente do TAD (usuário) tem acesso somente à forma abstrata do tipo, com base simplesmente nas funcionalidades oferecidas pelo tipo. Assim, a forma como ele foi implementado torna-se um detalhe de implementação, que não deve interferir no uso do TAD em outros contextos. Portanto, independente da linguagem de programação que se use, a definição do TAD será sempre a mesma. É essencial separar o conceito da implementação.

O projeto de um TAD envolve a escolha de operações adequadas para uma estrutura de dados, definindo seu comportamento. Algumas dicas interessantes, são:

- definir um número pequeno de operações, com soluções simples, que combinadas possam realizar funções mais complexas;
- o conjunto de operações deve ser adequado o suficiente para realizar as computações necessárias às aplicações que utilizarão o TAD;
- cada operação deve ter um propósito definido, com um comportamento constante e coerente, sem muitos casos especiais e exceções.

## 2.0 Representação física dos dados

Por representação física de uma estrutura de dados considera-se somente a forma utilizada para implementar os relacionamentos entre os diferentes elementos da estrutura (nós ou nodos), e não a representação física das informações internas de cada nodo. Esse aspecto deve ser considerado na análise das estruturas de dados a serem estudadas, pois representam fisicamente os relacionamentos lógicos entre os dados.

Existem duas alternativas básicas de representação física de dados: estática (contiguidade física) e dinâmica (encadeamento).

## 2.1 Representação estática

Na representação física estática, os dados são armazenados em posições contíguas na memória. A ordem entre os elementos (nodos) da estrutura armazenada é definida implicitamente pela posição ocupada pelos nodos na memória. Sendo assim, cada posição contígua na memória armazena o conjunto de informações correspondente a um nodo. Um nodo pode ser simples ou complexo, ou seja, apresentando um campo ou vários campos.

As vantagens desse tipo de representação são:

- proteção de memória: a alocação é feita antes do início da execução do programa, garantindo a proteção da memória;

- **transferência de dados:** como todos os dados estão alocados em bloco, a transferência de dados entre memória principal e secundária fica facilitada;
- **estruturas simples:** é apropriada para armazenar estrutura simples, principalmente aquelas que usam ordem física em sua representação;
- **representação:** algumas estruturas de dados possuem uma representação lógica semelhante à contiguidade física, simplificando desta maneira a representação dos dados;
- **acesso:** qualquer elemento (nodo) pode ser diretamente acessado a qualquer momento, através de um índice associado à sua posição.

As desvantagens desse tipo de representação estática são:

- **compartilhamento de memória**: este tipo de alocação não permite o compartilhamento de memória;
- **previsão de espaço físico**: é necessário definir, antes da execução da aplicação, o número máximo de elementos a serem alocados. Isto pode constituir um grave problema quando não for possível estimar inicialmente o número de elementos que a estrutura vai apresentar;
- **estruturas complexas**: não é apropriado para estruturas complexas devido a natureza sequencial;
- **inserção e exclusão de componentes**: estas duas operações geralmente implicam no deslocamento de um número considerável de informações para conseguir preservar as relações lógicas representadas. Em outras palavras, existe a necessidade de reestruturação dos componentes da estrutura de dados durante sua manipulação.

## 2.2 Representação dinâmica

Neste tipo de representação os dados podem ser alocados à medida que se torna necessário, através da alocação dinâmica. Sempre que um espaço de memória para armazenar um dado seja necessário, este é solicitado pela aplicação, e alocado pelo

gerenciador de memória em algum espaço livre de memória, sendo seu endereço desenvolvido em uma variável especial. Várias linguagens de programação permitem esse tipo de alocação através de variáveis do tipo ponteiro.

Uma estrutura armazenada através de alocação dinâmica (encadeamento) apresenta seus elementos (nodos) alocados em posições aleatórias na memória, e não lado a lado. A disposição física dos componentes de uma estrutura independe de sua posição na estrutura lógica. Desse modo, para percorrer a estrutura é necessário que os nodos apresentem algum campo adicional, onde é colocado o endereço físico do próximo nodo.

As vantagens desse tipo de representação são:

- **compartilhamento de memória:** uma vez que os nodos de uma estrutura são indicados através de seus endereços, os mesmos nodos poderiam fazer parte de mais de uma estrutura;
- **maleabilidade:** a alocação e a liberação de memória feita de forma dinâmica favorece a maleabilidade dos programas;
- **facilidade para inserção e remoção de componente:** a inserção e a remoção de nodos é facilmente realizada, sendo somente necessário ajustar os campos de elo dos nodos envolvidos na operação, sem a necessidade de deslocamento de informações.

As desvantagens da representação dinâmica são:

- **transferência de dados:** é dificultada neste tipo de representação, uma vez que os dados estão espalhados na memória;
- **gerência de memória mais onerosa:** toda a manipulação da estrutura é feita através de alocação e/ou liberação de memória, o que deve ser realizado pelo gerenciador de memória;
- **procedimentos menos intuitivos:** a utilização de alocação dinâmica implica na construção de procedimentos que envolvam alocação e liberação de memória, e no encadeamento de nodos. Isso faz com que os procedimentos escritos para as operações sobre os dados sejam mais complexos;
- **acesso:** o processamento dos dados encadeados deve ser feito de forma serial, isto é, um nodo deve ser sempre acessado a partir de outro acessado anteriormente. Não é possível, como

nos casos dos vetores, acessar qualquer nodo a qualquer momento, a menos que todos os endereços sejam armazenados individualmente.

## Exercícios

1 - Considere uma aplicação para armazenar os seguintes dados de uma pessoa em uma agenda de endereços: nome, endereço e telefone. Especifique um TAD para armazenar os dados das pessoas e as operações necessárias para inserir, consultar e excluir os dados das pessoas.

2 - Considere uma aplicação para armazenar os seguintes dados de carros para uma garagem: placa, marca/modelo e cor. Especifique um TAD para armazenar os dados dos carros e as operações necessárias para inserir, consultar e excluir os dados dos carros.

3 - Considere uma empresa que precisa armazenar os seguintes dados de um cliente: nome completo, ano de nascimento, renda mensal do cliente. Especifique um TAD para armazenar os dados de um cliente e as operações necessárias para inserir, consultar e excluir os dados dos clientes. Especifique também operações para exibir o número de clientes com renda mensal acima da média, e exibir o número de clientes que nasceram entre 1980 e 2000.

4 - Considere um conjunto de informações relativas a alunos, constituído de nome, número de matrícula e data de nascimento. Especifique um TAD para armazenar os dados dos alunos e as operações necessárias para inserir, consultar e excluir esses dados. Implemente uma aplicação que utilize o tipo Aluno e ainda execute as seguintes operações:

- imprima os nomes e números de matrícula dos alunos que nasceram após uma determinada data (passada como parâmetro);

- imprima as informações relativas a um determinado aluno, cujo número de matrícula é passado como parâmetro.

5 - Os dados relativos aos clientes de uma empresa estão armazenados em um arquivo. Para cada cliente são registrados um código, o nome, o endereço, o telefone, a data em que fez sua primeira compra na empresa, a data da última compra e o valor da última compra. Especifique o TAD Cliente para armazenar os dados dos clientes e as operações necessárias para inserir, consultar e excluir esses dados. Implemente uma aplicação que utilize o tipo Cliente.