

Tipos Abstratos de Dados

1. Tipos de Dados

Em uma linguagem de programação, os dados armazenados pelas constantes e variáveis são classificados de acordo com o tipo de informação que contém. Um tipo de dados caracteriza o conjunto de valores que pode ser assumido por uma variável ou gerado por uma função ou mesmo uma expressão.

Um tipo de dados **elementar** (também chamado de **simples** ou **primitivo**) é caracterizado por um conjunto (domínio) de valores, como por exemplo os valores definidos pelos tipos inteiro, real, caractere ou string.

Cada tipo de dados também possui um conjunto de operações que podem ser realizadas. Por exemplo, os tipos inteiro e real suportam as operações aritméticas (+, -, *, /, **, raiz quadrada, etc), enquanto o tipo string suporta as operações de concatenação e segmentação, etc.

2. Tipo Abstrato de Dados (TAD)

Um tipo abstrato de dados, (TAD) é uma especificação de um conjunto de dados (frequentemente heterogêneos) e operações que podem ser executadas sobre esses dados.

Por exemplo, podemos definir um tipo abstrato de dados estudante, para representar as informações necessárias para uma “entidade” estudante, contendo o nome, a idade e a matrícula de um estudante, e as operações de duplicar a informação, validar a matrícula, verificar a idade, etc.

Na prática, o TAD é implementado usando-se um tipo composto (struct/record – estrutura/registo) com os valores pertencentes ao TAD (nome, idade, matrícula). E por funções que operam essa estrutura.

Exemplo:

```
struct Estudante
{
    char nome[50];
    int idade;
    int matrícula;
}

int maiorDeIdade (Estudante e);
int matriculaValida (Estudante e);
```

No exemplo acima foi definida a estrutura Estudante, com os três campos mencionados, e duas funções com retorno booleano, maiorDeIdade, que retorna TRUE se a idade do estudante for maior ou igual a 18, e matriculaValida, que verifica se o campo matrícula do estudante é válida. Nesse exemplo somente os protótipos das funções estão apresentadas.

3. Lista

Um tipo abstrato de dados comum é a lista. Na prática, usamos a representação de lista no nosso dia a dia, para armazenar informações das mais variadas. Num modo bem simples, pode ser armazenada em uma folha de papel, como uma relação de itens. As operações usuais sobre uma lista em geral incluem:

- Criação da lista;
- Inclusão de um elemento na lista;
- Exclusão de um elemento da lista;
- Contar o número de elementos na lista;
- Exibir todos os elementos da lista;

Um mesmo tipo abstrato de dados pode ser implementado de mais de uma maneira diferente, e cada forma de implementação pode apresentar vantagens e desvantagens.

Por exemplo, a implementação de uma lista pode ser feita através de um vetor de strings, onde cada string contém um elemento da lista. Isso é útil, por exemplo, para armazenar uma lista de compras, onde cada item da lista é armazenado em um string do vetor. Essa forma é conhecida como lista com implementação estática.

Nesse caso específico, como todos os itens são representados somente com um string, não há necessidade de usar um tipo struct, e o tipo Lista pode ser definido como:

```
#define MaxItens 10
#define TamItem 20

typedef char tpLista[MaxItens][TamItem];
```

Nesse caso estamos definindo um tipo tpLista como um vetor de 10 strings, cada um deles com 20 caracteres.

E a alocação do espaço é definida quando da criação da variável Lista com base no tipo tpLista.

```
tpLista L1;
```

Assim, o espaço pode ser representado como na tabela abaixo, com 10 linhas de 20 colunas.

L1

0	c	a	r	n	e														
1	a	r	r	o	z														
2	a	l	f	a	c	e													
3																			
4																			
5																			
6																			
7																			
8																			
9																			

Na lista L1, do tipo tpLista, representada acima, foram inseridos os itens **carne**, **arroz** e **alface**.