Tipos Abstratos de Dados

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

5 de setembro de 2017

Estrutura de Dados

Tipos de Dados, Tipos Abstratos de Dados e Estrutura de Dados

- Termos parecidos, mas com significados diferentes!
- **Tipos de Dados**
  - Em linguagens de programação o tipo de dado de uma variável, constante ou função define o conjunto de valores que a variável, constante ou função podem assumir.
    - Ex.: variáveis `int` podem assumir valores inteiros.
  - Programador pode definir novos tipos de dados em termos de outros já definidos.
    - Ex.: tipos estruturados, como vetores, pontos...

Tipos de Dados, Tipos Abstratos de Dados e Estrutura de Dados

- Termos parecidos, mas com significados diferentes!
- **Estrutura de Dados de Dados**
  - Um tipo estruturado é um exemplo de estrutura de dados.
    - Tipos estruturados são estruturas de dados já pré-definidas na linguagem de programação.
    - O programador pode definir outras estruturas de dados para armazenar as informações que seu programa precisa manipular.
    - Vetores, registros, listas encadeadas, pilhas, filas, árvores, grafos, são exemplos de estrutura de dados típicas utilizadas para armazenar informação em memória principal.

Perspectivas para Tipos de dados

- Tipos de dados podem ser vistos como métodos para interpretar o conteúdo da memória do computador.
- Mas, podemos interpretar o conceito de tipo de dados sob outra perspectiva.
  - não em termos do que um computador pode fazer (interpretar os bits...), mas em termos do que o usuário (programador) deseja fazer (ex.: multiplicar dois inteiros...)
  - O programador não se importa muito com a representação no hardware, mas sim com o conceito matemático de inteiro.
  - Um tipo inteiro "suporta" certas operações...

Tipo Abstrato de Dados

- Os tipos e estruturas de dados existem para serem usados pelo programa para acessar informações neles armazenadas, por meio de operações apropriadas.
- Do ponto de vista do programador, muitas vezes é conveniente pensar nas estruturas de dados em termos das operações que elas suportam, e não da maneira como elas são implementadas.
- Uma estrutura de dados definida dessa forma é chamada de um `Tipo Abstrato de Dados (=TAD)`.

Tipo Abstrato de Dados

- TAD, portanto, estabelece o conceito de tipo de dados divorciado da sua representação.
- Definido como um modelo matemático por meio de um par (v, o) em que:
  - v é um conjunto de valores.
  - o é um conjunto de operações sobre esses valores.
  - Ex.: tipo real.
    - $v = \mathbb{R}$
    - $o = \{+, -, *, /, =, <, >, <=, >=\}$.

Definição de TAD

- Requer que operações sejam definidas sobre os dados sem estarem atreladas a uma representação específica.
  - Ocultamento de informação (*information hiding*)
- Programador que usa um tipo de dado `real`, `integer`, `array` não precisa saber como tais valores são representados internamente.
  - mesmo princípio pode ser aplicado a listas, pilhas, filas...
  - se existe uma implementação disponível de uma lista, p. Ex.: um programador pode utilizá-la como se fosse uma "caixa preta", acessá-la por meio das operações que ela suporta.

Definição de TAD

- Para definir um TAD:
  - Programador descreve o TAD em dois módulos separados.
  - Um módulo contém a definição do TAD: representação da estrutura de dados e implementação de cada operação suportada.
  - Outro módulo contém a interface de acesso: apresenta as operações possíveis.
  - Outros programadores podem, por meio da interface de acesso, usar o TAD sem conhecer os detalhes representacionais e sem acessar o módulo de definição.

Definição de TAD

- Os módulos (ou *units*) são instalados em uma biblioteca e podem ser reutilizados por vários programas.
  - A execução do programa requer a linkedição dos módulos de definição (que podem ser mantidos já pré-compilados em uma biblioteca) junto com o programa.
  - Mas o programador não precisar olhar o código do módulo de definição para usar o TAD!
  - Basta conhecer a `interface` de acesso.

Implementação de um TAD

- Uma vez definido um TAD e especificadas as operações associadas, ele pode ser implementado em uma linguagem de programação.
- Uma estrutura de dados pode ser vista, então, como uma implementação de um TAD.
  - implementação do TAD implica na escolha de uma ED para representá-lo, a qual é acessada pelas operações que ele define.
- ED é construída a partir dos tipos básicos (`integer`, `real`, `char`) ou dos tipos estruturados (array, record) de uma linguagem de programação.

Características de um TAD

- Característica essencial de TAD é a separação entre a definição conceitual - par (v, o) e a implementação.
  - O programa só acessa o TAD por meio de suas operações.
  - "ocultamento de informação"
- Programador tem acesso a uma descrição dos valores e operações admitidos pelo TAD.
- Programador não tem acesso à implementação.
  - Idealmente, a implementação é "invisível" e inacessível.
  - Ex.: pode criar uma lista de clientes e aplicar operações sobre ela, mas não sabe como ela é representada internament.
- Quais as vantagens?

Vantagens do uso de TADs

- **Reuso:** uma vez definido, implementado e restado, o TAD pode ser acessado por diferentes programas.
- **Manutenção:** mudanças na implementação do TAD não afetam o código fonte dos programas que o utilizam (decorrência do ocultamento de informação).
  - módulos do TAD são compilados separadamente.
  - uma alteração força somente a recompilação do arquivo envolvido e uma nova link-edição do programa que acessa o TAD.
  - O programa mesmo não precisa ser recompilado!
- **Correção:** TAD foi (ou deveria ter sido) testado e funciona (ou deveria) corretamente.

matrix.h

```
1  /*TAD: Matriz m por n*/
2
3  /*Tipo Exportado*/
4  typedef struct matriz Matriz;
5
6  /*Funções Exportadas*/
7  /* Função cria - Aloca e retorna matriz m por n */
8  Matriz* cria (int m, int n);
9  /* Função libera - Libera a memória de uma matriz */
10 void libera (Matriz* mat);
11 /* Função acessa - Retorna o valor do elemento [i][j]  */
12 float acessa (Matriz* mat, int i, int j);
13 /* Função atribui - Atribui valor ao elemento [i][j]   */
14 void atribui (Matriz* mat, int i, int j, float v);
15 /* Função linhas - Retorna o no. de linhas da matriz   */
16 int linhas (Matriz* mat);
17 /* Função colunas - Retorna o no. de colunas da matriz */
18 int colunas (Matriz* mat);
```

Biblioteca e definição da estrutura

```c
#include <stdlib.h> /*malloc, free, exit*/
#include <stdio.h> /*printf*/
#include "matriz.h"

struct matriz {
  int lin;
  int col;
  float* v;
};
```

Funções Libera e Cria

```
10 void libera (Matriz* mat){
11   free(mat->v);
12   free(mat);
13 }
14
15 Matriz* cria (int m, int n) {
16   Matriz* mat = (Matriz*) malloc(sizeof(Matriz));
17   if(mat == NULL) {
18     printf("Memória insuficiente!\n");
19     exit(1);
20   }
21   mat->lin = m;
22   mat->col = n;
23   mat->v = (float*) malloc(m*n*sizeof(float));
24   return mat;
25 }
```

Funções Acessa e Linhas

```
26 float acessa (Matriz* mat, int i, int j) {
27   int k; /* índice do elemento no vetor */
28   if(i<0 ||i>= mat->lin|| j<0 || j>= mat->col) {
29     printf("Acesso inválido!\n");
30     exit(1);
31   }
32   k = i*mat->col+ j; /*armazenamento por linha*/
33   return mat->v[k];
34 }
35
36 int linhas (Matriz* mat) {
37   return mat->lin;
38 }
```

Funções Atribui e Colunas

```
39 void atribui (Matriz* mat, int i, int j, float v) {
40   int k; /* índice do elemento no vetor */
41   if(i<0 || i>=mat->lin || j<0 || j>=mat->col) {
42     printf("Atribuição inválida!\n");
43     exit(1);
44   }
45
46   k = i*mat->col + j;
47   mat->v[k] = v;
48 }
49
50 int colunas (Matriz* mat) {
51   return mat->col;
```

Programa main que usa o TAD

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "matriz.h"
4  int main(int argc, char *argv[])
5  {
6    float a,b,c,d;
7    int i,j;
8    Matriz *M, *mat1, *mat2, *r;
9    // criação de uma matriz
10 //   M = cria(5,5);
11 //   // inserção de valores na matriz
12 //   atribui(M,1,2,40);
13 //   atribui(M,2,3,3);
14 //   atribui(M,3,0,15);
```

Programa main que usa o TAD

```
15 //  atribui(M,4,1,21);
16 //  // verificando se a inserção foi feita corretamente
17 //  a = acessa(M,1,2);
18 //  b = acessa(M,2,3);
19 //  c = acessa(M,3,0);
20 //  d = acessa(M,4,1);
21 //  printf ("M[1][2]: %4.2f \n", a);
22 //  printf ("M[2][3]: %4.2f \n", b);
23 //  printf ("M[3][0]: %4.2f \n", c);
24 //  printf ("M[4][1]: %4.2f \n", d);
25
26   mat1 = cria(2,2);
27   mat2 = cria(2,2);
28   for(i=0;i<2;i++){
29       for(j=0;j<2;j++){
30         atribui(mat1,i,j,i+j);
31       }
32   }
33   for(i=0;i<2;i++){
```

TAD Ponto

- Vamos considerar a criação de um tipo de dado para representar um ponto no $\mathbb{R}^2$.
- As operações do TAD são:
  - `cria`: operação que cria um ponto com coordenadores x e y;
  - `libera`: operação que libera a memória alocada por um ponto;
  - `acessa`: operação que atribui novos valores às coordenadas de um ponto;
  - `atribui`: operação que atribui novos valores às coordenadas de um ponto;
  - `distancia`: operação que calcula a distância entre dois pontos.

Tipos Abstratos de Dados

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

5 de setembro de 2017

Estrutura de Dados