INF1007: Programação 2

9 – Tipos Abstratos de Dados

Tópicos

- Módulos e compilação em separado
- Tipo abstrato de dados
  - Exemplo 1: TAD Ponto
  - Exemplo 2: TAD Círculo
  - Exemplo 3: TAD Matriz

Módulos e Compilação em Separado

- Módulo
  - um arquivo com funções que representam apenas parte da implementação de um programa completo
- Arquivo objeto
  - resultado de compilar um módulo
  - geralmente com extensão *.o* ou *.obj*
- Ligador
  - junta todos os arquivos objeto em um único arquivo executável

Módulos e Compilação em Separado

- Exemplo:
  - *str.c:*
    - arquivo com a implementação das funções de manipulação de *strings:* “comprimento”, “copia” e “concatena”
    - usado para compor outros módulos que utilizem estas funções
      - módulos precisam conhecer os protótipos das funções em *str.c*

Módulos e Compilação em Separado

- Exemplo:
  - *prog1.c:* arquivo com o seguinte código

```
#include <stdio.h>
int comprimento (char* str);
void copia (char* dest, char* orig);
void concatena (char* dest, char* orig);
int main (void) {
  char str[101], str1[51], str2[51];
  printf("Digite uma seqüência de caracteres: ");
  scanf(" %50[^\n]", str1);
  printf("Digite outra seqüência de caracteres: ");
  ...
  return 0;
}
```

Módulos e Compilação em Separado

- Exemplo:
  - *prog1.exe:*
    - arquivo executável gerado em 2 passos:
      - compilando os arquivos *str.c* e *prog1.c* separadamente
      - ligando os arquivos resultantes em um único arquivo executável
    - seqüência de comandos para o compilador Gnu C (gcc):

```
> gcc -c str.c -o str.o
> gcc -c prog1.c -o prog1.o
> gcc -o prog1.exe str.o prog1.o
```

Módulos e Compilação em Separado

- Interface de um módulo de funções:
  - arquivo contendo apenas:
    - os protótipos das funções oferecidas pelo módulo
    - os tipos de dados exportados pelo módulo (typedef's, struct's, etc)
  - em geral possui:
    - nome: o mesmo do módulo ao qual está associado
    - extensão: *.h*

Módulos e Compilação em Separado

- Inclusão de arquivos de interface no código:

```
#include <arquivo.h>
```

  – protótipos das funções da biblioteca padrão de C

```
#include "arquivo.h"
```

  – protótipos de módulos do usuário

Módulos e Compilação em Separado

- Exemplo – arquivos *str.h* e *prog1.c*

```
/* Funções oferecidas pelo modulo str.c */
/* Função comprimento
** Retorna o número de caracteres da string passada como parâmetro
*/
int comprimento (char* str);
/* Função copia
** Copia os caracteres da string orig (origem) para dest (destino)
*/
void copia (char* dest, char* orig);
/* Função concatena
** Concatena a string orig (origem) na string dest (destino)
*/
void concatena (char* dest, char* orig);
```

Módulos e Compilação em Separado

```
#include <stdio.h>
#include "str.h"
int main (void) {
  char str[101], str1[51], str2[51];
  printf("Digite uma seqüência de caracteres: ");
  scanf(" %50[^\n]", str1);
  printf("Digite outra seqüência de caracteres: ");
  scanf(" %50[^\n]", str2);
  copia(str, str1);
  concatena(str, str2);
  printf("Comprimento da concatenação: %d\n",comprimento(str));
  return 0;
}
```

Tipo Abstrato de Dados

- Tipo Abstrato de Dados (TAD):
  - um TAD define:
    - um novo tipo de dado
    - o conjunto de operações para manipular dados desse tipo
  - um TAD facilita:
    - a manutenção e a reutilização de código
    - abstrato = “forma de implementação não precisa ser conhecida”
      - para utilizar um TAD é necessário conhecer a sua funcionalidade, mas não a sua implementação

Tipo Abstrato de Dados

- Interface de um TAD:
  - a interface de um TAD define:
    - o nome do tipo
    - os nomes das funções exportadas
      - os nomes das funções devem ser prefixada pelo nome do tipo, evitando conflitos quando tipos distintos são usados em conjunto
      - exemplo:

        pto_cria - função para criar um tipo Ponto

        circ_cria - função para criar um tipo Circulo

Tipo Abstrato de Dados

- Implementação de um TAD:
  - o arquivo de implementação de um TAD deve:
    - incluir o arquivo de interface do TAD:
      - permite utilizar as definições da interface, que são necessárias na implementação
      - garante que as funções implementadas correspondem às funções da interface
        - » compilador verifica se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos
    - incluir as variáveis globais e funções auxiliares:
      - devem ser declaradas como estáticas
      - visíveis apenas dentro do arquivo de implementação

Tipo Abstrato de Dados

- TAD Ponto
  - tipo de dado para representar um ponto no R^2 com as seguintes operações:

cria	cria um ponto com coordenadas x e y
libera	libera a memória alocada por um ponto
acessa	retorna as coordenadas de um ponto
atribui	atribui novos valores às coordenadas de um ponto
distancia	calcula a distância entre dois pontos

Tipo Abstrato de Dados

- Interface de Ponto
  - define o nome do tipo e os nomes das funções exportadas
  - a composição da estrutura Ponto não faz parte da interface:
    - não é exportada pelo módulo
    - não faz parte da interface do módulo
    - não é visível para outros módulos

  - os módulos que utilizarem o TAD Ponto:
    - não poderão acessar diretamente os campos da estrutura Ponto
    - só terão acesso aos dados obtidos através das funções exportadas

ponto.h - arquivo com a interface de *Ponto*

```
/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;

/* Funções exportadas */
/* Função cria - Aloca e retorna um ponto com coordenadas (x,y) */
Ponto* pto_cria (float x, float y);

/* Função libera - Libera a memória de um ponto previamente criado
*/
void pto_libera (Ponto* p);

/* Função acessa - Retorna os valores das coordenadas de um ponto
*/
void pto_acessa (Ponto* p, float* x, float* y);

/* Função atribui - Atribui novos valores às coordenadas de um
ponto */
void pto_atribui (Ponto* p, float x, float y);

/* Função distancia - Retorna a distância entre dois pontos */
float pto_distancia (Ponto* p1, Ponto* p2);
```

Tipo Abstrato de Dados

- Implementação de Ponto:
  - inclui o arquivo de interface de Ponto
  - define a composição da estrutura Ponto
  - inclui a implementação das funções externas

ponto.c - arquivo com o TAD Ponto

```
#include <stdlib.h>

#include "ponto.h"

struct ponto {
   float x;
   float y;
}

Ponto* pto_cria (float x, float y) ...

void pto_libera (Ponto* p) ...

void pto_acessa (Ponto* p, float* x, float* y) ...

void pto_atribui (Ponto* p, float x, float y) ...

float pto_distancia (Ponto* p1, Ponto* p2) ...
```

(Ver próximas transparências para implementação das funções externas)

- Função para criar um ponto dinamicamente:
  - aloca a estrutura que representa o ponto
  - inicializa os seus campos

```
Ponto* pto_cria (float x, float y)
{
   Ponto* p = (Ponto*) malloc(sizeof(Ponto));
   if (p == NULL) {
      printf("Memória insuficiente!\n");
      exit(1);
   }
   p->x = x;
   p->y = y;
   return p;
}
```

- Função para liberar um ponto:
  - deve apenas liberar a estrutura criada dinamicamente através da função *cria*

```
void pto_libera (Ponto* p)
{
   free(p);
}
```

- Funções para acessar e atribuir valores às coordenadas de um ponto:
  - permitem que uma função cliente acesse as coordenadas do ponto sem conhecer como os valores são armazenados

```
void pto_acessa (Ponto* p, float* x, float* y)
{
   *x = p->x;
   *y = p->y;
}

void pto_atribui (Ponto* p, float x, float y)
{
   p->x = x;
   p->y = y;
}
```

- Função para calcular a distância entre dois pontos

```
float pto_distancia (Ponto* p1, Ponto* p2)
{
   float dx = p2->x - p1->x;
   float dy = p2->y - p1->y;
   return sqrt(dx*dx + dy*dy);
}
```

- Exemplo de arquivo que usa o TAD Ponto

```
#include <stdio.h>
#include "ponto.h"

int main (void)
{
   float x, y;
   Ponto* p = pto_cria(2.0,1.0);
   Ponto* q = pto_cria(3.4,2.1);
   float d = pto_distancia(p,q);
   printf("Distancia entre pontos: %f\n",d);
   pto_libera(q);
   pto_libera(p);
   return 0;
}
```

Tipo Abstrato de Dados

- **TAD Circulo**
  - tipo de dado para representar um ponto círculo com as seguintes operações:

cria	cria um círculo com centro (x,y)e raio r
libera	libera a memória alocada por um círculo
area	calcula a área do círculo
interior	verifica se um dado ponto está dentro do círculo

```
/* TAD: Círculo */
/* Dependência de módulos */
#include "ponto.h"

/* Tipo exportado */
typedef struct circulo Circulo;

/* Funções exportadas */
/* Função cria - Aloca e retorna um círculo com centro (x,y) e raio r */
Circulo* circ_cria (float x, float y, float r);

/* Função libera - Libera a memória de um círculo previamente criado */
void circ_libera (Circulo* c);

/* Função area - Retorna o valor da área do círculo */
float circ_area (Circulo* c);

/* Função interior - Verifica se um dado ponto p está dentro do círculo */
int circ_interior (Circulo* c, Ponto* p);
```

circulo.h - arquivo com a interface do TAD

interface *ponto.h* incluída na interface pois a operação *interior* faz uso do tipo Ponto

circulo.c - arquivo com o TAD Circulo

```
#include <stdlib.h>
#include "circulo.h"
#define PI 3.14159
struct circulo { Ponto* p; float r; };

Circulo* circ_cria (float x, float y, float r)
{  Circulo* c = (Circulo*)malloc(sizeof(Circulo));
   c->p = pto_cria(x,y);
   c->r = r;
}

void circ_libera (Circulo* c)
{  pto_libera(c->p); free(c); }

float circ_area (Circulo* c)
{  return PI*c->r*c->r; }

int circ_interior (Circulo* c, Ponto* p)
{  float d = pto_distancia(c->p,p); return (d<c->r); }
```

TAD Círculo

usa

TAD Ponto

Tipo Abstrato de Dados

- **TAD Matriz**
  - tipo de dado para representar uma matriz com as seguintes operações:

cria	cria uma matriz de dimensão m por n
libera	libera a memória alocada para a matriz
acessa	acessa o elemento da linha i e da coluna j da matriz
atribui	atribui o elemento da linha i e da coluna j da matriz
linhas	retorna o número de linhas da matriz
colunas	retorna o número de colunas da matriz

  - ilustra diferentes maneiras de implementar um mesmo TAD

```
/* TAD: matriz m por n */

typedef struct matriz Matriz;

Matriz* mat_cria (int m, int n);
void mat_libera (Matriz* mat);
float mat_acessa (Matriz* mat, int i, int j);
void mat_atribui (Matriz* mat, int i, int j, float v);
int mat_linhas (Matriz* mat);
int mat_colunas (Matriz* mat);
```

matriz.h - arquivo com a interface do TAD

interface do módulo independe da estratégia de implementação

- a estrutura representando uma matriz
  na implementação como vetores simples

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

- a estrutura representando uma matriz
  na implementação como vetores de ponteiros

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

Resumo

Módulo	arquivo com funções que representam apenas parte da implementação de um programa completo
Arquivo objeto	resultado de compilar um módulo geralmente com extensão .o ou .obj
Interface de módulo	arquivo contendo apenas os protótipos das funções oferecidas pelo módulo, e os tipos de dados exportados pelo módulo
TAD	define um novo tipo de dado e o conjunto de operações para manipular dados do tipo
Interface de TAD	define o nome do tipo e das funções exportadas

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

Capítulo 9 – Tipos abstratos de dados