Estrutura de Dados I

Tipos abstratos de dados

Prof. MSc. Romuere Silva

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

- *operacoes.c:*
  - arquivo com a implementação das funções de manipulação de inteiros: “soma”, “subtração”, “multiplicação” e “divisão”;
  - usado para compor outros módulos que utilizem estas funções;
  - módulos precisam conhecer os protótipos das funções em *main.c*.

```
1   #include <stdio.h>
2
3   int soma(int a, int b){
4       return a+b;
5   }
6   int subtracao(int a, int b){
7       return a-b;
8   }
9   int multiplicacao(int a, int b){
10      return a*b;
11  }
12  int divisao(int a, int b){
13      return a/b;
14  }
15
16
```

Módulos e Compilação em Separado - Exemplo

- *main*.c: arquivo com o seguinte código

```
2   #include <stdlib.h>
3
4   int soma(int a, int b);
5   int subtracao(int c, int d);
6   int multiplicacao(int e, int f);
7   int divisao(int g, int h);
8
9   int main(void){
10
11      int so = soma(2,3);
12      int su = subtracao(2,3);
13      int mu = multiplicacao(2,3);
14      int di = divisao(2,3);
15
16      printf("A soma eh: %d\n", so);
17      printf("A multiplicacao eh: %d\n", mu);
18      printf("A subtracao eh: %d\n", su);
19      printf("A divisao eh: %d\n", di);
20      return 0;
21  }
22
```

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: igual ao 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: operacoes.h e main.c*

```
1
2   int soma(int a, int b);
3   int subtracao(int a, int b);
4   int multiplicacao(int a, int b);
5   int divisao(int a, int b);
6
```

```
#include <stdio.h>
#include <stdlib.h>
#include "operacoes.h"


int main(void){

    int so = soma(2,3);
    int su = subtracao(2,3);
    int mu = multiplicacao(2,3);
    int di = divisao(2,3);

    printf("A soma eh: %d\n", so);
    printf("A multiplicacao eh: %d\n", mu);
    printf("A subtracao eh: %d\n", su);
    printf("A divisao eh: %d\n", di);
    return 0;
}
```

Tipo Abstrato de Dados

- 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

- 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.
  - E deve também Incluir as variáveis globais e funções auxiliares.

Tipo Abstrato de Dados - Ponto

- Tipo de dado para representar um ponto 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 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.

Tipo Abstrato de Dados – Interface Ponto

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 do Ponto

- Inclui o arquivo de interface de Ponto;
- Define a composição da estrutura Ponto;
- Inclui a implementação das funções externas.

Tipo Abstrato de Dados – Implementação do Ponto

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) ...
```

Tipo Abstrato de Dados – Implementação do Ponto

```
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;
}
```

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

```
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;
}
```

Tipo Abstrato de Dados – Implementação do Ponto

```
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);
}
```

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

int main (void)
{
   float x, y;
   Point* p = pto_cria(2.0,1.0);
   Point* 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

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