UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

Programação

(CK0226 – 2022.2)

Curso: Ciência da Computação
Professor: Lincoln Souza Rocha
E-mail: lincoln@dc.ufc.br

Introdução à Linguagem C – Parte VI

Sumário

- Módulos e Compilação em Separado
- Tipo Abstrato de Dados (TAD)
- Exemplos de TAD

Módulos e Compilação em Separado

Em C, um **módulo** é um arquivo com funções que representam apenas parte da implementação de um programa completo.

O **arquivo objeto** é o resultado da compilação de um módulo. Geralmente possui a extensão **.o** ou **.obj**.

O programa **ligador** faz a junção de todos os arquivos objeto em um único arquivo binário, o executável.

Módulos e Compilação em Separado

Tomemos com exemplo o arquivo **str.c**

- Arquivo com a implementação de funções de manipulação de strings (ex., comprimento, copia e concatena) – um espécie de biblioteca
- Usado para compor outros módulos que utilização estas funções. Esses módulos precisam conhecer os protótipos das funções **str.c**

Módulos e Compilação em Separado

O arquivo **prog1.c** usa **str.c**

```
#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

O arquivo **prog1.exe** é o executável gerado em dois passos

1. Compilando os arquivos **str.c** e **prog1.c** separadamente
2. Ligando os arquivos resultantes em um único arquivo executável

A sequência 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

- 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** (de header)

Módulos e Compilação em Separado

Inclusão de arquivos de interface no código

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

- Para protótipos das funções da biblioteca padrão de C

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

- Para protótipos das funções dos módulos de usuário

Módulos e Compilação em Separado

O arquivo **str.h**

```
/* 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

O arquivo **str.h**

```
#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 (TAD)

Um TAD define um novo **tipo de dado** e o conjunto de operações para manipular dados desse tipo.

Um TAD facilita a manutenção e reutilização de código. A forma de implementação não precisa ser conhecida (abstrato). Ou seja, para utilizar um TAD é necessário conhecer a sua **funcionalidade**, mas não a sua **implementação**.

Interface de um TAD

A interface de um TAD (i.e., o arquivo **.h**) define o nome do tipo e o nome das funções exportadas.

OBS. os nomes das funções devem ser prefixada pelo nome do tipo, evitando conflitos quando tipos distintos são usados em conjunto. Exemplos:

- `pto_cria` – função para criar um `Ponto`
- `circ_cria` – função para criar um `Circulo`

Implementação de um TAD

O arquivo de implementação (**.c**) de um TAD deve incluir o arquivo de interface do TAD para: (i) permitir utilizar as **definições da interface**, que são necessárias na implementação; (ii) garantir que as **funções implementadas correspondem às funções da interface**, pois o compilador verifica se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos.

Implementação de um TAD

O arquivo de implementação (**.c**) de um TAD deve incluir as variáveis globais e funções auxiliares. Elas devem ser declaradas como estáticas, **visíveis apenas dentro do arquivo de implementação.**

Implementação de um TAD

O arquivo de implementação (**.c**) de um TAD deve incluir as variáveis globais e funções auxiliares. Elas devem ser declaradas como estáticas, **visíveis apenas dentro do arquivo de implementação.**

TAD Ponto

Representa 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

TAD Ponto (Interface)

A interface do TAD Ponto define o nome do tipo e o nome das funções exportadas. A composição da estrutura Ponto não faz parte da interface. Isto é, não é exportada pelo módulo, não faz parte da interface do módulo e não é visível para outros módulos.

Os módulos que utilizarem o TAD Ponto (i) não poderão acessar diretamente os campos da estrutura Ponto; e (ii) só terão acesso aos dados obtidos através das funções exportadas.

TAD Ponto (Interface)

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);
```

TAD Ponto (Implementação)

A implementação do TAD Ponto:

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

TAD Ponto (Interface)

ponto.c - arquivo com a implementação de 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) {
(...)
}
```

TAD Ponto (Interface)

Função para criar um ponto dinamicamente. Para isso ela (i) aloca a estrutura que representa o ponto; e (ii) 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;
}
```

TAD Ponto (Interface)

Função para liberar um ponto dinamicamente. Para isso ela deve apenas liberar a estrutura criada dinamicamente por meio da função pto_cria.

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

TAD Ponto (Interface)

Funções para recuperar e atribuir valores às coordenadas de um ponto. Permitem que um 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;
}
```

TAD Ponto (Interface)

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);
}
```

TAD Ponto (Cliente)

cliente.c - arquivo que usa o Ponto

```
#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;
}
```

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 Circulo (Interface)

circulo.h - arquivo com a interface de Circulo

```
/* 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_área(Circulo* c);

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

TAD Circulo (Implmentação)

circulo.c - arquivo com a implementação de 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;
   return c;
}

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

float circ_área(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 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

TAD Matriz (Interface)

matriz.h - arquivo com a interface da Matriz

```
/* 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);
```

TAD Matriz (Implementação)

matriz.c - arquivo com a implementação da Matriz

```
#include <stdlib.h>
#include "matriz.h"

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

   OU

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

(...)
```

UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

Programação

(CK0226 – 2022.2)

Curso: Ciência da Computação
Professor: Lincoln Souza Rocha
E-mail: lincoln@dc.ufc.br