Agenda

1. Introdução
2. Módulos e compilação em separado
3. Tipo abstrato de dados
    1. TAD Ponto
    2. TAD Círculo

Introdução

Introdução

- Ideia central
  - Encapsular de quem usa um determinado tipo a forma concreta com que ele foi implementado.

- Benefícios
  - Desacoplamento da implementação do uso;
  - Facilidade de manutenção;
  - Aumento do reuso.

Módulos e compilação em separado

Módulos e compilação em separado

- Módulo
  - Arquivo com funções que representam parte de uma implementação.
  - A implementação de um programa pode ser composta por um ou mais módulos.

Módulos e compilação em separado

- Programa composto por vários módulos:
  - Cada um compilado separadamente (*.o* ou *.obj*);
  - O ligador (*linker*) reúne todos os arquivos objeto em um único arquivo executável.

- Para programas de médio e grande porte, a sua divisão em vários módulos é fundamental.

Módulos e compilação em separado

- Exemplo do uso de módulos em C:
  - Arquivo *str.c* para manipulação de strings:
    - comprimento;
    - copia;
    - concatena.
  - Arquivo *prog1.c*

Módulos e compilação em separado

- Exemplo do uso de módulos em C (*str.c*):

```c
int comprimento (char* s) {
    int i;
    int n = 0;/* contador */
    for(i = 0; s[i] != '\0'; i++)
        n++;
    return n;
}

void copia (char* dest, char* orig) {
    int i;
    for (i = 0; orig[i] != '\0'; i++)
        dest[i] = orig[i];
    /* fecha a cadeia copiada */
    dest[i] = '\0';
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Módulos e compilação em separado

- Exemplo do uso de módulos em C (*str.c*):

```
void concatena (char* dest, char* orig) {
    int i = 0; /* índice usado na cadeia destino,
                  inicializado com zero */
    int j; /* índice usado na cadeia origem */
    /* acha o final da cadeia destino */
    while(dest[i] != '\0')
        i++;
    /* copia elementos da origem para o final do
    destino */
    for(j = 0; orig[j] != '\0'; j++) {
        dest[i] = orig[j];
        i++;
    }
    /* fecha cadeia destino */
    dest[i] = '\0';
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Módulos e compilação em separado

- Exemplo do uso de módulos em C (*prog1.c*):

```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 sequencia de caracteres: ");
    scanf(" %50[^\n]", str1);
    printf("Digite outra sequencia de caracteres: ");
    scanf(" %50[^\n]", str2);
    copia(str, str1);
    concatena(str, str2);
    printf("Comprimento da concatenacao: %d\n",
                                    comprimento(str));
    return 0;
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Módulos e compilação em separado

- Exemplo do uso de módulos em C:

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

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Módulos e compilação em separado

- Exemplo do uso de módulos em C:
  - Um programa pode utilizar as funções de *str.c* desde que conheça os seus protótipos;
  - Problema com a repetição de protótipos;
  - Todo módulo de funções C costuma ter associado um arquivo de protótipos que representa a sua interface (extensão *.h*).

Módulos e compilação em separado

- Exemplo do uso de módulos em C (*str.h*):

```
/* Funções oferecidas pelo módulo 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);
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Módulos e compilação em separado

- Exemplo do uso de módulos em C (*prog1.c*):

```c
#include <stdio.h>
#include "str.h"

int main (void) {
    char str[101], str1[51], str2[51];
    printf("Digite uma sequencia de caracteres: ");
    scanf(" %50[^\n]", str1);
    printf("Digite outra sequencia de caracteres: ");
    scanf(" %50[^\n]", str2);
    copia(str, str1);
    concatena(str, str2);
    printf("Comprimento da concatenacao: %d\n",
                                    comprimento(str));
    return 0;
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

Tipo abstrato de dados

- Tipo abstrato de dados (TAD):
  - Novo tipo de dados + conjunto de operações sobre dados desse tipo;
  - Se criarmos um tipo abstrato, podemos "esconder" a estratégia de implementação;
  - Benefícios:
    - Facilidade de manutenção;
    - Reutilização de código.

Tipo abstrato de dados

- Tipo abstrato de dados (TAD):
  - Uso de módulos e criação de TADs;
  - Interface de um TAD = nome do tipo + conjunto de funções exportadas;

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Operações:
    - cria;
    - libera;
    - acessa;
    - atribui;
    - distancia.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Interface do módulo (*ponto.h*):

```
/* TAD: Ponto (x,y) */

/* Tipo exportado */
typedef struct ponto Ponto;

/* Funções exportadas */
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);
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Note que a composição da estrutura Ponto (struct ponto) não é exportada pelo módulo;
  - O arquivo que usa o TAD deve, obrigatoriamente, incluir o arquivo responsável por definir sua interface.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:

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

int main(void) {
    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;
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Precisamos ligar o arquivo com a implementação do módulo para gerar um executável;
  - O arquivo de implementação do módulo (*ponto.c*) deve sempre incluir o arquivo de interface.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Implementação do módulo (*ponto.c*):

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

struct ponto {
    float x;
    float y;
};
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Implementação do módulo (*ponto.c*):

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

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Implementação do módulo (*ponto.c*):

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

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

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 1 - TAD Ponto:
  - Implementação do módulo (*ponto.c*):

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

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);
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 2 - TAD Círculo:
  - Operações:
    - cria;
    - libera;
    - area;
    - interior.

Tipo abstrato de dados

- Exemplo 2 - TAD Círculo:
  - Interface do TAD (*circulo.h*):

```
/* TAD: Circulo */

/* Dependência de módulos */
#include "ponto.h"

/* Tipo exportado */
typedef struct circulo Circulo;

/* Funções exportadas */
Circulo* circ_cria (float x, float y, float r);
void circ_libera (Circulo* c);
float circ_area (Circulo* c);
int circ_interior (Circulo* c, Ponto* p);
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 2 - TAD Círculo:
  - Implementação do TAD (*circulo.c*):

```
#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;
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 2 - TAD Círculo:
  - Implementação do TAD (*circulo.c*):

```
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);
}
```

Fonte: CELES; CERQUEIRA; RANGEL, 2004.

Tipo abstrato de dados

- Exemplo 2 - TAD Círculo:
  - Exercício: Implementar uma função *main* que utiliza as funcionalidades do TAD Círculo.

Referências

- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. **Introdução a Estruturas de Dados: com técnicas de programação em C.** Rio de Janeiro: Elsevier, 2004.
- OLIVEIRA, U. **Programando em C** – Volume I – Fundamentos. Ciência Moderna, 2008.