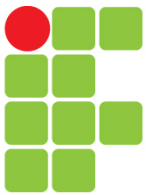
Modularização e dados abstratos

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

12 de março de 2019



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

Tópico

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

Objetivos

- Aprender a utilizar técnica de programação baseada na definição de *tipos de dados abstratos*
- Criar programas modularizados
- Utilizar o conceito de "dados abstratos e estruturados"

Tópico

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

Modularização

A Linguagem C permite que o programador utilize vários aquivos ou módulos através de diretivas de inclusão (**include**):

- inclusão de bibliotecas de sistema
- inclusão de bibliotecas personalizadas
- definição de tipos em arquivos distintos
- definição de cabeçalhos de funções

Modularização

Para programas pequenos, utilização de vários arquivos ou módulos pode não se justificar. Porém, é conveniente modularizar a implementação na medida que o software fica mais complexo ou quando utilizamos diversas estruturas personalizadas. O projeto tende a ficar:

- main.c: arquivo de nível hierárquico alto que contém a função main. Main é a função que é executada quando o Sistema Operacional executa a aplicação.
- modulo_x.h: definição de tipos e cabeçalhos de funções de **x** utilizados pela aplicação
- modulo_x.c: implementação das funções de **x**
- modulo_y.h: (...)

Tópico

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

Tipos abstratos de dados

Quando um módulo define um **novo** tipo de dado e seu conjunto de operações, este módulo representa um tipo abstrato de dado.

Exemplos:

- Ponto
- Matriz

Tópico

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

TaD Ponto

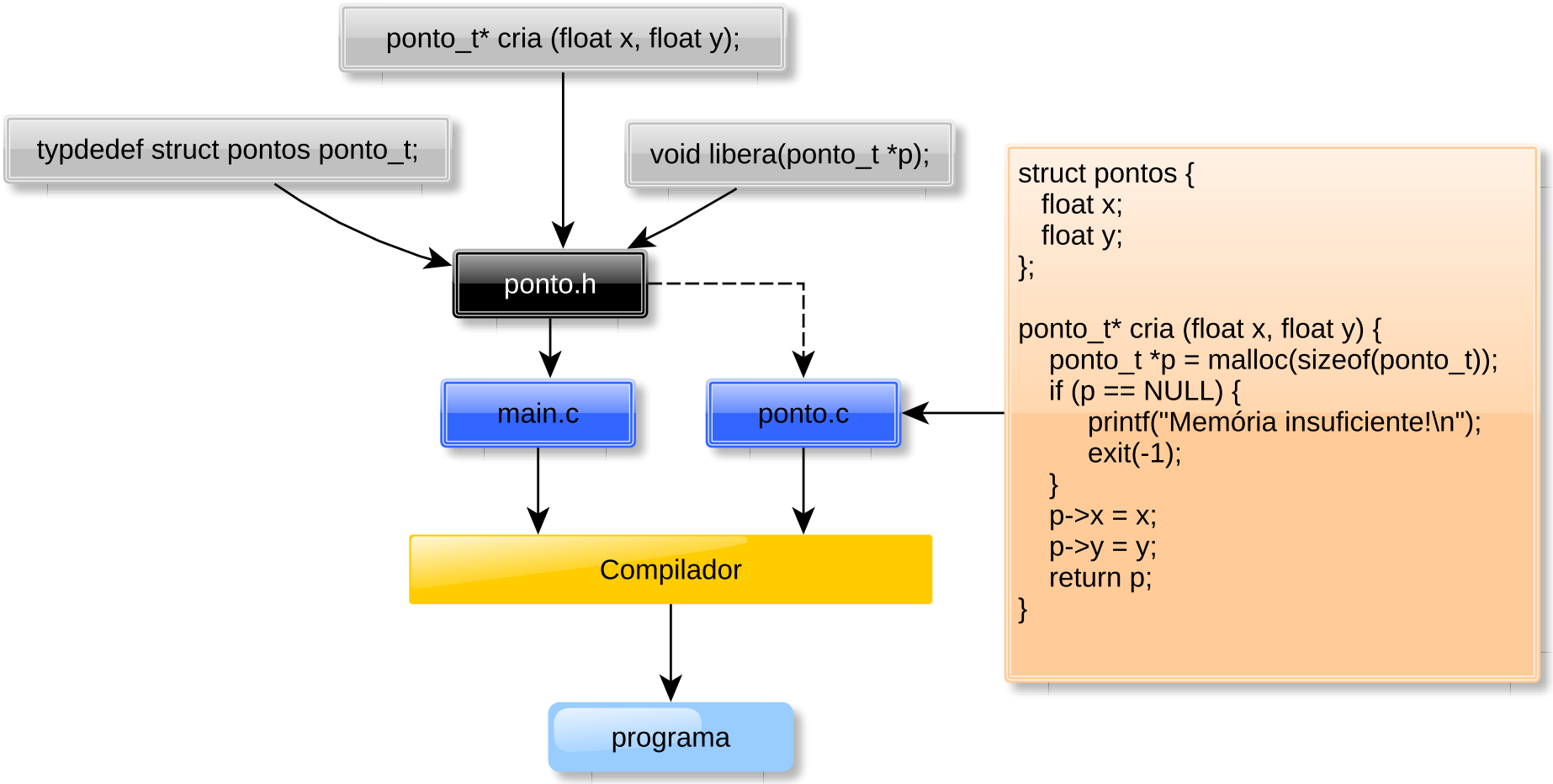
Objetivo: representação de um ponto no sistema R^2

Nome do Dado: *ponto*

Operações:

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

TaD Ponto


ponto_t* cria (float x, float y);
typdedef struct pontos ponto_t;
void libera(ponto_t *p);
ponto.h
main.c
ponto.c
Compilador
programa
struct pontos {
float x;
float y;
};
ponto_t* cria (float x, float y) {
ponto_t *p = malloc(sizeof(ponto_t));
if (p == NULL) {
printf("Memória insuficiente!\n");
exit(-1);
}
p->x = x;
p->y = y;
return p;
}

TaD Ponto: ponto.h

```c
#ifndef PONTO_H
#define PONTO_H

/* TAD: ponto (x,y) */
/* Tipo exportado */
typedef struct pontos ponto_t;

/* Funcoes exportadas */

/* Funcao cria
** Aloca e retorna um ponto com coordenadas (x,y)
*/
ponto_t* cria (float x, float y);

/* Funcao libera
** Libera a memoria de um ponto_t previamente criado.
*/
void libera (ponto_t *p);

(...)
```

TaD Ponto: ponto.h

```
(...)

void acessa (ponto_t *p, float* x, float* y);

/* Funcao atribui
** Atribui novos valores as coordenadas de um ponto
*/
void atribui (ponto_t  *p, float x, float y);

/* Funcao distancia
** Retorna a distancia entre dois pontos
*/
float distancia (ponto_t *p1, ponto_t *p2);

#endif // PONTO_H
```

TaD Ponto: ponto.c

```c
/* Includes */

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

struct pontos {
  float x;
  float y;
};

ponto_t* cria (float x, float y) {
  ponto_t *p = malloc(sizeof(ponto_t));
  if (p == NULL) {
    perror("cria ponto: ");
    exit(-1);
  }

  p->x = x;
  p->y = y;

  return p;
}

(...)
```

TaD Ponto: ponto.c

```c
void libera (ponto_t *p) {
  free(p);
}

void acessa (ponto_t  p, float* x, float* y) {
  *x = p->x;
  *y = p->y;
}

void atribui (ponto_t *p, float x, float y) {
  p->x = x;
  p->y = y;
}

float distancia (ponto_t *p1, ponto_t* p2) {
  float dx = p2->x - p1->x;
  float dy = p2->y - p1->y;

  return sqrt(dx*dx + dy*dy);
}
```

Tópico

1. Introdução
2. Modularização
3. Tipos abstratos de dados
4. TaD Ponto
5. Exercícios

Exercício

- Considerando o conteúdo sobre estruturas de dados, alocação dinâmica e as implementações anteriores:
  - Obter arquivo CSV do **Moodle**.
  - Criar a estrutura que comporte o dados deste arquivo (utilize typedef).
  - Criar uma função que leia o conteúdo deste arquivo e armazene em um vetor de estruturas alocado dinamicamente. Deve-se retornar um ponteiro do vetor de estruturas e também o número de dados lidos.

```
struct dados {
  (...)
};
```

  - Criar uma função que busque determinado nome e retorne as informações;
  - **Requisitos:**
    - As funções devem ser implementadas em um arquivo separado do **main.c**: xyz.c xyz.h
    - Utilize a macro que evita multi-inclusão de arquivos de cabeçalhos