

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`

9 de março de 2018



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

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

- 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”

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

A Linguagem C permite que o programador utilize vários arquivos 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

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`: (...)

- 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

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

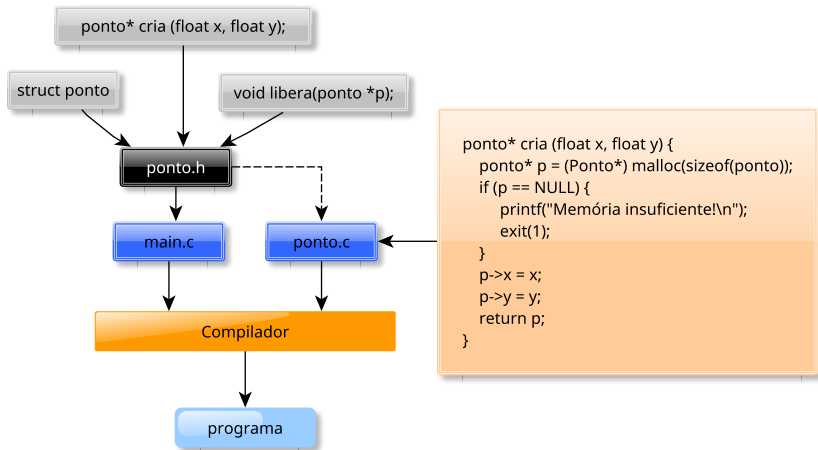
Objetivo: representação de um ponto no sistema R^2

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



TaD Ponto: ponto.h

```
#ifndef PONTO_H
#define PONTO_H

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

/* Funcoes exportadas */

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

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

(...)
```

TaD Ponto: ponto.h

```
(...)  
  
void acessa (ponto* p, float* x, float* y);  
  
/* Funcao atribui  
** Atribui novos valores as coordenadas de um ponto  
**/  
void atribui (ponto* p, float x, float y);  
  
/* Funcao distancia  
** Retorna a distancia entre dois pontos  
**/  
float distancia (ponto* p1, ponto* p2);  
  
#endif // PONTO_H
```

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

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

    return p;
}

(...)
```

```
void libera (ponto* p) {
    free(p);
}

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

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

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

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


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

- Considerando o conteúdo sobre estruturas de dados, alocação dinâmica e as implementações anteriores:
 - Obter winterGames.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 jogosInver {  
    unsigned char pos;  
  
    (...)  
};
```

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