ESTRUTURAS DE DADOS

Tipos Abstratos de Dados

Tipos Abstratos de Dados

- Tipo Abstrato de Dados (TAD) é uma maneira de encapsular (esconder) de quem usa um tipo a forma concreta como ele foi implementado
- TAD desacopla a implementação do uso, permitindo a reutilização de código
  - Modularização do código (arquivos)
  - Compilação por Módulo (em geral, arquivo .o ou .obj)
  - Ligador une os arquivos objetos (.o ou .obj) em um único executável
- Podemos desenvolver TAD para Ponto, Círculo,
  - Matriz, Carro, Pessoa etc.

Tipos Abstratos de Dados

Suponha que dois programas utilizem um mesmo tipo estruturado com funções semelhantes. Assim, podemos modularizar o desenvolvimento, utilizando a idéia de TAD


programa1.c


programa2.c


TAD.h

Define o nome do tipo e os protótipos das funções, ou seja, a assinatura do TAD.


TAD.c

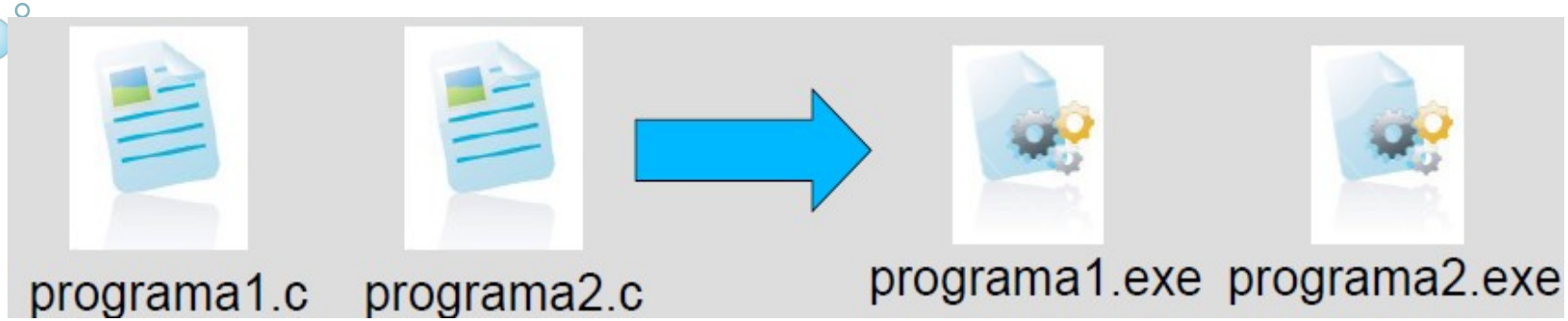
Define o tipo e implementa as funções do TAD. Deve importar o arquivo TAD.h

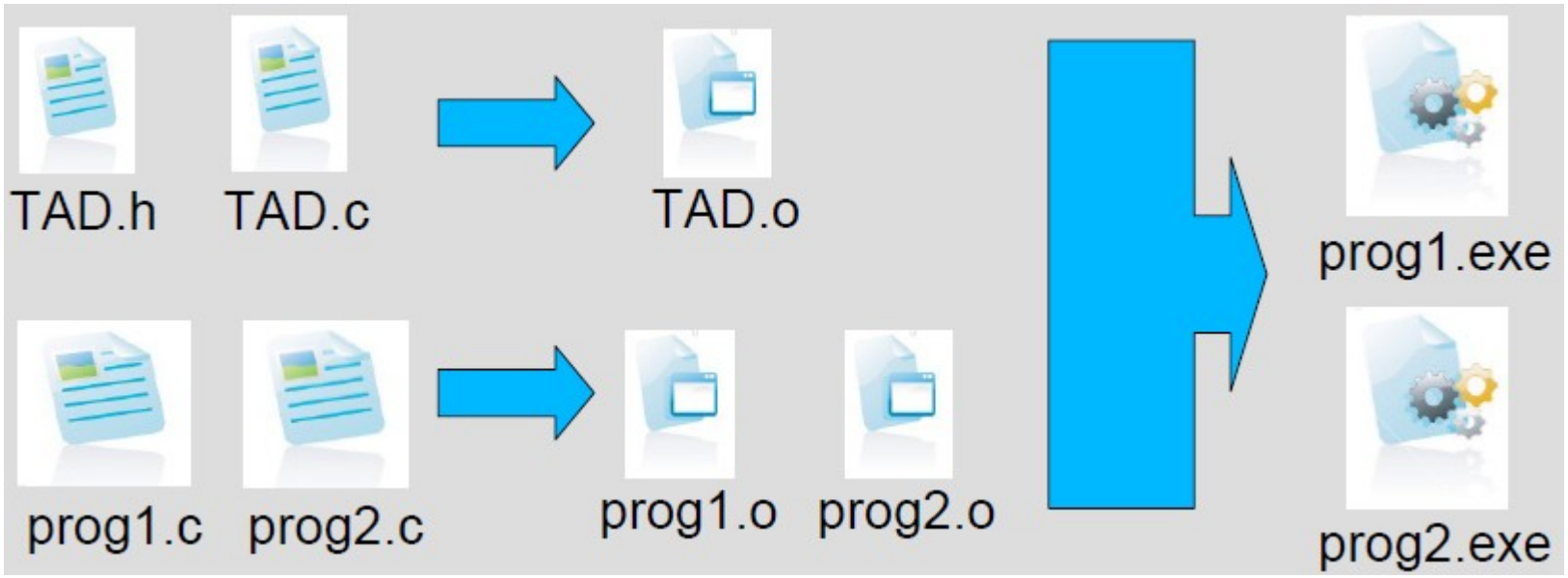
Os programas utilizam o mesmo TAD previamente implementado, importando o arquivo TAD.h


prog1.c


prog2.c

Compilação em Separado





Exemplos

- Programa que calcula a distância entre dois pontos com as seguintes funções
  - cria cria um ponto com coordenadas (x,y)
  - libera libera a memória alocada por um ponto
  - acessa retorna as coordenadas de um ponto
  - atribui altera os valores das coordenadas de um ponto
  - distancia calcula a distância entre dois pontos
- Programa que determina se um ponto está dentro de um círculo
  - As mesmas funções do programa anterior para o ponto e para o círculo
  - area calcula a área de um círculo
  - interior determina se um ponto está no interior de um círculo

Ex. Ponto – Versão sem TAD

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
typedef struct ponto {
     float x; float y;
} Ponto;

/*Protótipos das funções*/
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);
```

Ex. Ponto – Versão sem TAD

```
int main(void){
    Ponto *p1 = pto_cria(2.0,1.0);
    Ponto *p2 = pto_cria(3.4,2.1);
    float d = pto_distancia(p1,p2);
    printf("Distancia entre pontos: %f\n",d);
    pto_libera(p1); pto_libera(p2);
    return 0;
}
```

Ex. Ponto – Versão sem TAD

```
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;
}

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

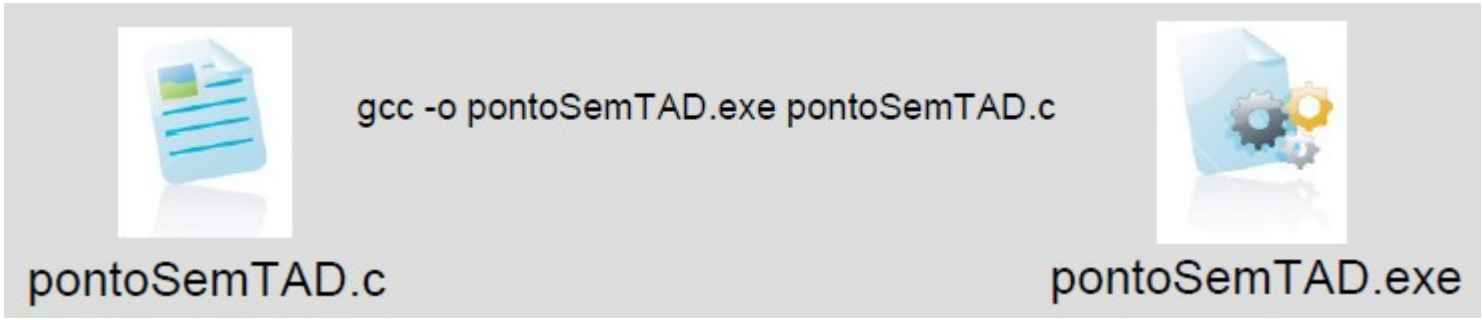
Ex. Ponto – Versão sem TAD

```
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;
}

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);
}
```

Considerações

- O uso e a implementação do tipo ponto estão acoplados
- Não permite a reutilização das funções em outros programas/módulos
- Para criar um TAD Ponto é necessário:
  - Criar um arquivo (ponto.h) com o nome do tipo Ponto e os protótipos das funções
  - Criar um arquivo (ponto.c) com a definição do tipo ponto e a implementação das funções. Deve importar o arquivo (ponto.h)

TAD Ponto


gcc -o pontoSemTAD.exe pontoSemTAD.c
pontoSemTAD.c
pontoSemTAD.exe


gcc -c ponto.c
gcc -o pontoComTAD.exe
ponto.o pontoComTAD.o
ponto.h
ponto.c
ponto.o
gcc -c pontoComTAD.c
pontoComTAD.exe
pontoComTAD.c
pontoComTAD.o

Ex. Ponto - Versão com TAD Ponto

- Arquivo ponto.h deve definir o nome do TAD e os protótipos das funções (documentados)

```
typedef struct ponto Ponto;

/*Função que cria um Ponto com as coordenadas (x,y)*/
Ponto* pto_cria(float x, float y);

/*Função que libera a memória de um ponto criado*/
void pto_libera(Ponto *p);

/*Função que acessa as coordenadas de um ponto*/
void pto_acessa(Ponto *p, float *x, float *y);
```

Ex. Ponto - Versão com TAD Ponto

```
/*Função que atribui novos valores às coordenadas
  (x,y)*/
   void pto_atribui(Ponto *p, float x, float y);

/*Função que calcula a distância entre dois pontos*/
   float pto_distancia(Ponto *p1, Ponto *p2);
```

Ex. Ponto - Versão com TAD Ponto

- Arquivo ponto.c deve definir o tipo Ponto e implementar as funções. Importar o arquivo ponto.h e as bibliotecas necessárias.

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

struct ponto {
   float x; float y;
};

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;
}
```

Ex. Ponto - Versão com TAD Ponto

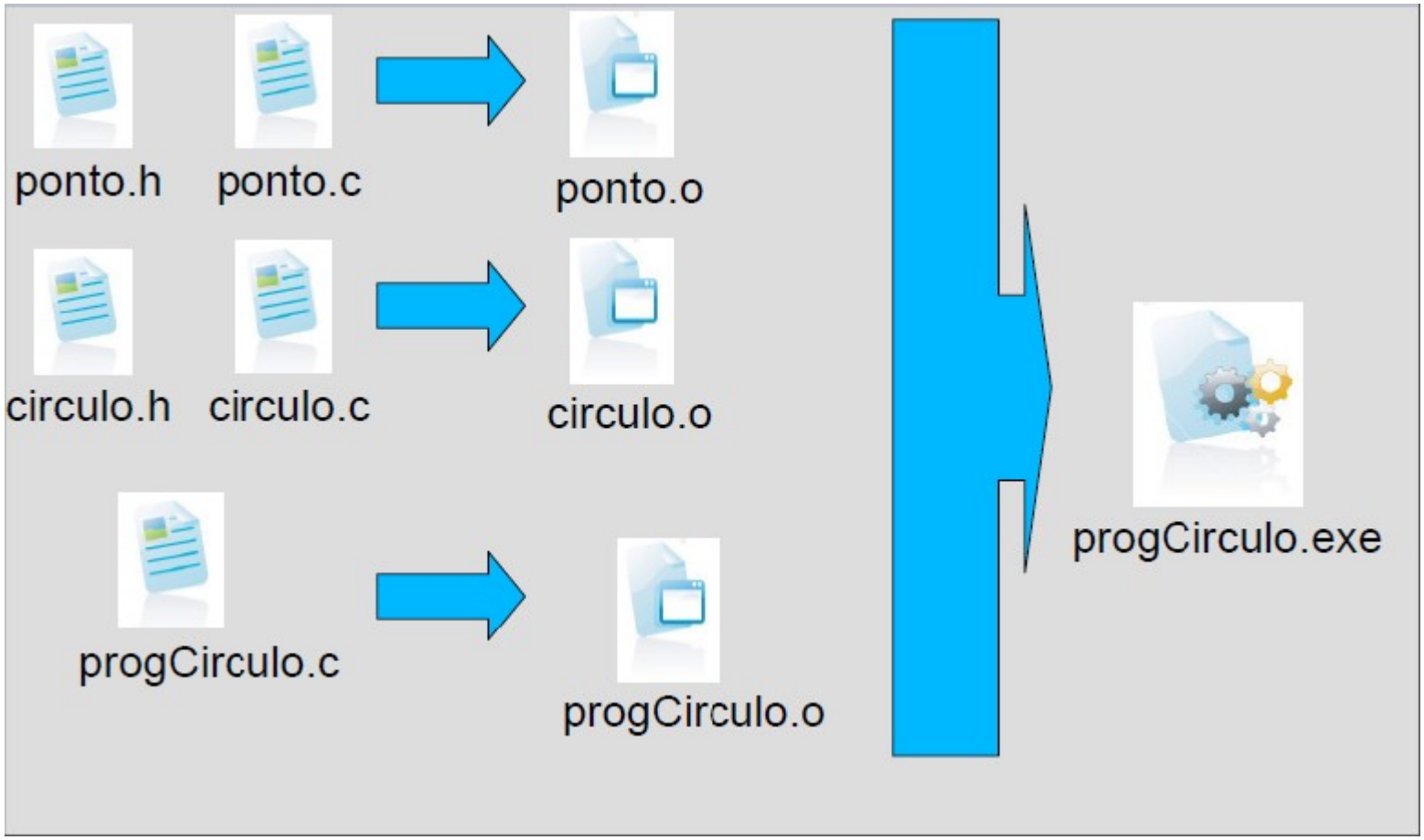
```
void pto_libera(Ponto *p){
  free(p);
}

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;
}

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);
}
```

Ex. Ponto - Versão com TAD Ponto

- Arquivo pontoComTAD.c utiliza o TAD ponto previamente desenvolvido

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

int main(void){
  Ponto *p1 = pto_cria(2.0,1.0);
  Ponto *p2 = pto_cria(3.4,2.1);
  float d = pto_distancia(p1,p2);
  printf("Distancia entre pontos: %f\n",d);
  pto_libera(p1); pto_libera(p2);
  return 0;
}
```

TAD Ponto e Círculo



Versão TAD Ponto e Círculo

- Arquivo circulo.h deve definir o nome do TAD círculo e os protótipos das funções documentados

```
#include "ponto.h"

typedef struct circulo Circulo;

/*Função que cria e retorna um círculo com centro (x,y) e raio r */
Circulo* circ_cria (float x, float y, float r);

/* Função que libera a memória de um círculo previamente criado */
void circ_libera (Circulo* c);

/* Função que calcula o valor da área do círculo */
float circ_area (Circulo* c);

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

Versão TAD Ponto e Círculo

- Arquivo circulo.c deve definir o TAD Círculo e implementar as funções. Importar o arquivo circulo.h

```
#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_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); }
```

Versão TAD Ponto e Círculo

- Arquivo progCirculo.c deve usar o TAD Círculo, importando o arquivo circulo.h

```
#include<stdio.h>
#include "circulo.h"
int main(void){
  Ponto *p = pto_cria(2.0,1.0);
  Circulo *c = circ_cria(3.4,2.1,2.0);
  float a = circ_area(c);
  printf("Area do circulo: %.2f\n",a);
  if(circ_interior(c,p))
     printf("Ponto dentro do Circulo\n");
  else
     printf("Ponto fora do Circulo\n");
  pto_libera(p); circ_libera(c);
  return 0;
}
```

- Slides baseados no livro **Introdução a Estruturas de Dados**, Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Editora Campus, 2004.