Tipos Abstratos de Dados
TADs

Renata de Matos Galante

galante@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática

INF 01203 – Estruturas de Dados

UFRGS

Exemplo

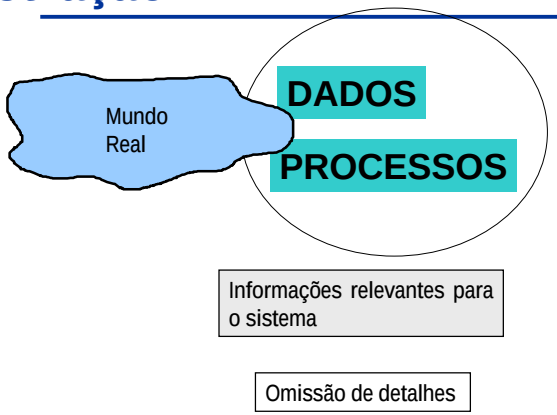
- Folha de freqüência

Disciplina: INF 01203 – Estruturas de Dados
Semestre: 2008-1 **Turma:** U
Professor: Renata de Matos Galante

matricula	nome	...	...	...
XXXX	Ana			
ZZZZ	Maria			
YYYY	Pedro			

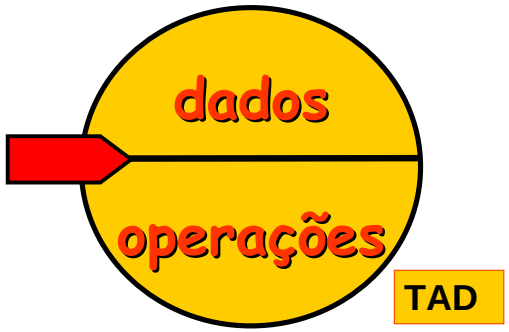
Programa: *manipula* dados a respeito dos alunos matriculados

Abstração



- **abstração de dados**
- **abstração de processos**

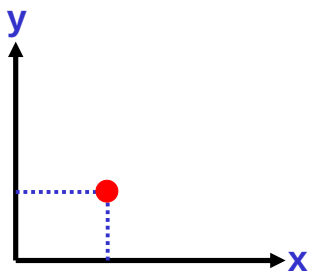
Abstração de Dados



Tipos Abstratos de Dados

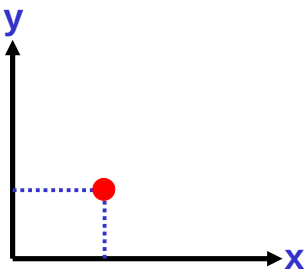
- Um TAD é uma forma de definir um novo tipo de dado juntamente com as operações que manipulam esse novo tipo de dado

Exemplo - Representar um Ponto



- **Modelo**
  Par ordenado (x,y)
- **Dados representando o modelo**
  - **Coordenada X**
  - **Coordenada Y**

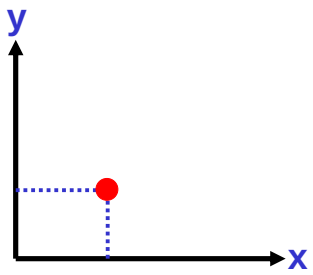
Exemplo - Representar um Ponto



Operações:

- **cria: operação que cria um ponto, alocando memória para as coordenadas x e y;**
- **libera: operação que libera a memória alocada por um ponto;**
- **acessa: operação que devolve as coordenadas de um ponto;**
- **atribui: operação que atribui novos valores às coordenadas de um ponto;**
- **distancia: operação que calcula a distância entre dois pontos.**

Exemplo - Representar um Ponto



Operações:

- **cria (x,y)**
- **libera (ponto P)**
- **acessa (ponto P)**
- **atribui (ponto P, x,y)**
- **distancia (ponto P1, ponto P2)**

Conceito em LP

- **Um tipo abstrato de dados (em LP) é um tipo de dado que satisfaz as condições:**
  - **A representação ou a definição do tipo e as operações sobre variáveis desse tipo estão contidas numa única unidade sintática**
    - **MÓDULO**
  - **A representação interna do tipo (a implementação) não é visível de outras unidades sintáticas, de modo que só as operações oferecidas na definição do tipo podem ser usadas com as variáveis desse tipo**

TAD

- Um TAD consiste de um novo tipo de dado, juntamente com as operações que manipulam este tipo de dado
- O TAD é colocado em uma Unidade sintática separada
- Qualquer programa pode usar Unidade Sintática
- Se a implementação for modificada, os programas que utilizam a Unidade Sintática não serão alterados

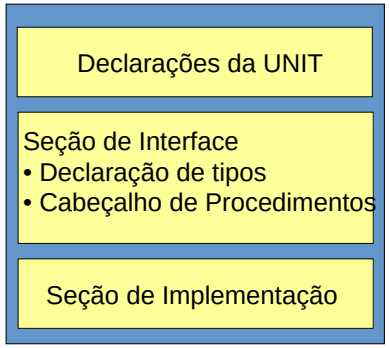
Para que servem?

- **Esquecer a forma de implementação (tipo concreto)**
- **Separar o código da aplicação do código da implementação**
- **Vantagens:**
  - **usar o mesmo tipo em diversas aplicações**
  - **pode-se alterar o tipo concreto usado sem alterar a aplicação**

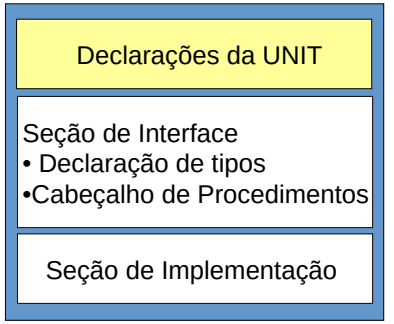
Implementação em Pascal
UNIT

UNIT para TAD

- Todas as partes da TAD podem ser colocadas na UNIT
  - Qualquer programador pode usar



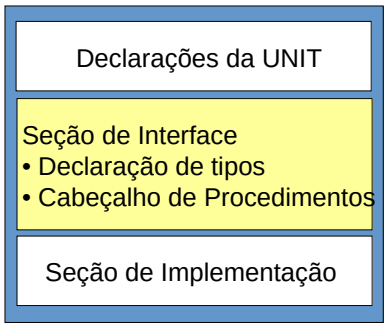
UNIT para TAD




```
unit <identificador>;  {arquivo deve ter mesmo nome.PAS}
```

UNIT para TAD

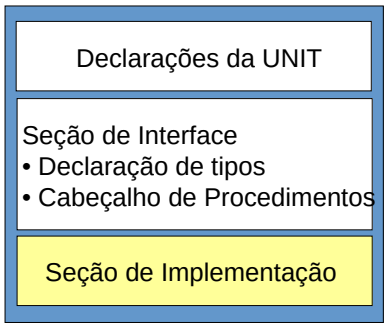
- Começa com a palavra **interface**
- Segue a **declaração** para o novo tipo
- Seguem os **cabeçalhos** para as funções e procedimentos




```
interface
  uses <lista de units>   {opcional}
  <declarações públicas>   {só cabeçalho}
```

UNIT para TAD

- Começa com a palavra **implementation** e termina com
- Seguem o cabeçalho+corpo das funções e procedimentos
- Termina com a palavra **end** e "." ponto



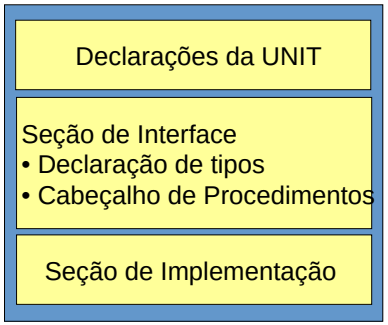

```
Implementation
Begin
  <inicializações>
End.
```

UNIT para TAD

```
unit <identificador>;

interface
   uses <lista de units>
  <declarações públicas>

implementation
Begin
   <inicializações>
End.
```



Exemplo 2: Funções Matemáticas

```
unit Matematica;

interface
    function SomaDois(x,y:integer): integer;
    function SubtraiDois(x,y:integer): integer;
implementation
     function SomaDois(x,y:integer): integer;
     begin
         SomaDois := x + y;
     end;
     function SubtraiDois(x,y:integer): integer;
     begin
         SubtraiDois := x - y;
     end;
end.
```

Exemplo 2: Funções Matemáticas

```
Program calculo;
uses Matematica;
var
   a, b, soma, sub: integer;

begin
    writeln('A'); readln(a);
    writeln('B'); readln(b);
    soma := SomaDois(a,b);
    writeln ('SomaDois: ', soma);
    readln;
    sub := SubtraiDois(a,b);
    writeln ('SubtraiDois:', sub);
    readln;
End.
```

Questão

- Se um benfeitor fornecer a você uma UNIT ***matemática***, mas você pode ler apenas a interface da UNIT. Você não pode ler as seções de implementação.
  - **Você pode escrever um programa que utiliza esta UNIT de tipo de dado *matemática*?**

Questão

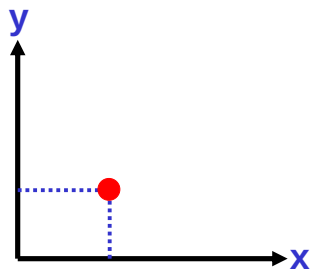
- Se um benfeitor fornecer a você uma UNIT ***matemática***, mas você pode ler apenas a interface da UNIT. Você não pode ler as seções de implementação.
  - **Você pode escrever um programa que utiliza esta UNIT de tipo de dado *matemática*?**
  - **SIM! O nome do tipo de dados é suficiente para declarar variáveis do tipo *matemática*. Você também conhece os cabeçalhos e especificações de cada operação.**

Outros Tipos de Soma e Subtração

- Exemplo soma/subtração: inteiros
  - Poderia ser: soma/subtração de reais, de string, de caracter, etc...
- Suponha que você deseja usar uma destas implementações. O que você terá que modificar na implementação atual?
- TAD Genéricas!

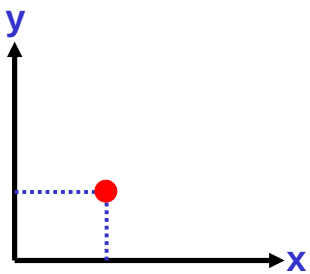
Implementação em C

Exemplo - Representar um Ponto



- **Modelo**
  Par ordenado (x,y)
- **Dados representando o modelo**
  - **Coordenada X**
  - **Coordenada Y**

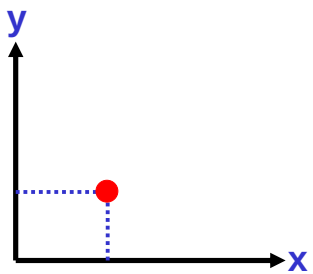
Exemplo - Representar um Ponto



Operações:

- **cria: operação que cria um ponto, alocando memória para as coordenadas x e y;**
- **libera: operação que libera a memória alocada por um ponto;**
- **acessa: operação que devolve as coordenadas de um ponto;**
- **atribui: operação que atribui novos valores às coordenadas de um ponto;**
- **distancia: operação que calcula a distância entre dois pontos.**

Exemplo - Representar um Ponto



Operações:

- **cria (x,y)**
- **libera (ponto P)**
- **acessa (ponto P)**
- **atribui (ponto P, x,y)**
- **distancia (ponto P1, ponto P2)**

TAD em C

- Interface (PROTÓTIPO)
  - # include "arquivo.h" -> *TAD usuários*
  - # include <arquivo.h> -> *TAD linguagem C*
- Programa.c
  - Código fonte

Exemplo - Implementação C

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

/* Funções Exportadas */

/* Cria coordenada */
Ponto* pto_cria (float x, float y);

/* Libera ponto */
void pto_libera(Ponto* p);

/* Acessa ponto */
void pto_acessa(Ponto* p, float* x, float* y);

/* Atribui */
void pto_atribui(Ponto* p, float* x, float* y);

/* Distrância */
void pto_distribui(Ponto* p1, Ponto* p2);
```

ponto.h

Interface, chamada protótipo

Exemplo - Implementação C

```
# include <studio.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(“Memória Insuficiente!”\n);
      exit(1);
}
p-> = x;
P->y = y;
return p;
}
```

Ponto.C

Implementação da Interface

Exemplo - Implementação C

```
# include <studio.h>
# include “ponto.h”

int main (void)
{
   Point* p = pto_cria(2.0, 1.0);
   Point* q = pto_cria(3.4, 2.1);
   float d = pto_distancia(p,q);
   printf(“Distância entre ponto: %\n”, d);
   pto_libera(q);
   pto_libera(p);
   return 0;
}
```

Aplicacao.C

Aplicação que usa a **Interface**