Laboratório de Programação I

Assunto de Hoje:
TADs + Bibliotecas

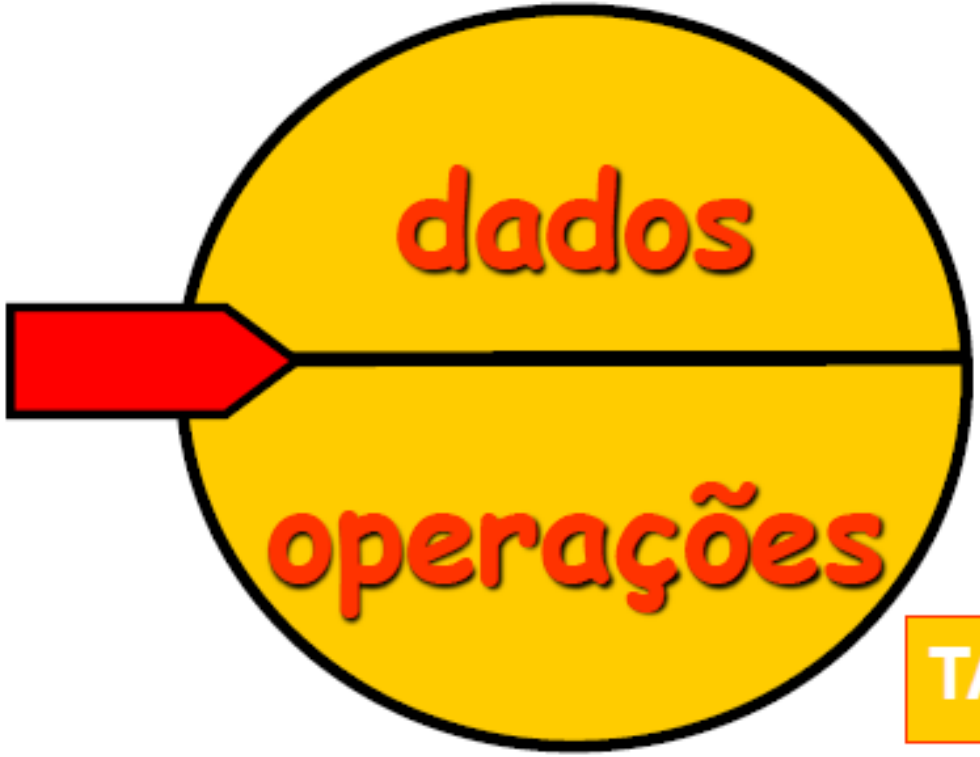
lucianobrum@unipampa.edu.br

Material de aula baseado nos slides do prof. Julio Saraçol

Abstração de Dados

- Abstraem a representação e as operações.
- Tipos de abstração de dados:
  - Tipos de dados primitivos;
  - Tipos de dados estruturados;
  - **Tipos abstratos de dados (TAD);**

Abstração de Dados



TAD

Tipos Abstratos de Dados

- Agrupa a estrutura de dados juntamente com as operações que podem ser feitas sobre esses dados.
- O TAD encapsula a estrutura de dados. Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados.
- Usuário do TAD x Programador do TAD:
- Usuário só “enxerga” a interface, não a implementação.

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

Tipos Abstratos de Dados

- Para utilizar um TAD é necessário conhecer a sua funcionalidade, mas não a sua implementação.

- Qualquer modificação nessa implementação fica restrita ao TAD:
  - Facilita a manutenção e reutilização de código.

Tipos Abstratos de Dados

- A representação ou a definição do tipo e as operações sobre variáveis desse tipo estão contidas numa única unidade sintática.

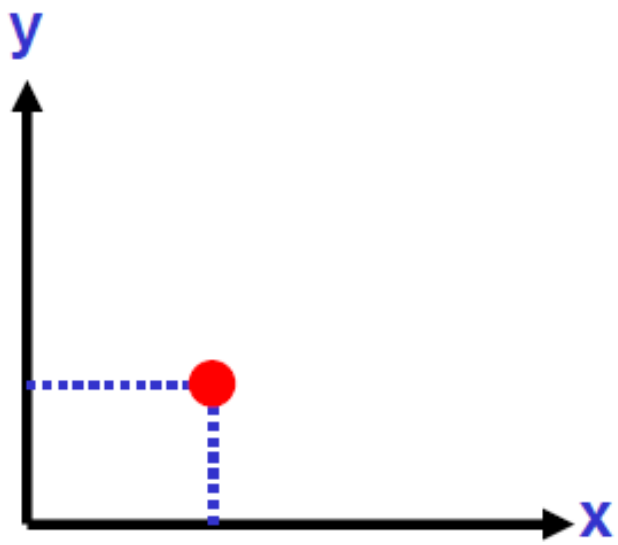
- A representação interna do tipo (a implementação) não é visível de outras unidades sintáticas, de modo que só as operações oferecidas na definição do tipo podem ser usadas com as variáveis desse tipo.

Tipos Abstratos de Dados

- Para o 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.
- REUTILIZAÇÃO!!

Tipos Abstratos de Dados

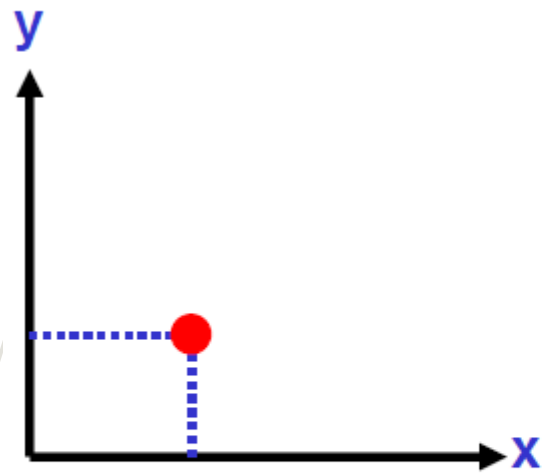


- **Modelo**

  Par ordenado (x,y)

- **Dados representando o modelo**
  - Coordenada X
  - Coordenada Y

Tipos Abstratos de Dados



Operações:

cria: operação que cria um ponto, alocando memória para as coordenadas x e y;

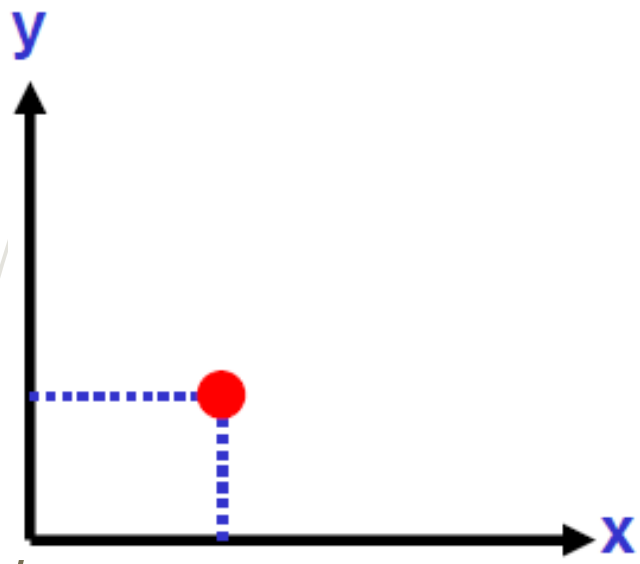
libera: operação que libera a memória alocada por um ponto;

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.

Tipos Abstratos de Dados



Operações:

cria (x,y)

libera (ponto P)

acessa (ponto P)

atribui (ponto P, x,y)

distancia (ponto P1, ponto P2)

Bibliotecas em C

- Uma boa técnica de programação é implementar os TADs em arquivos separados do programa principal.

- Para isso geralmente separa-se a declaração e a implementação do TAD em dois arquivos:
  - NomeDoTad.h: com a declaração.
  - NomeDoTad.c: com a implementação.

- O programa ou outros TADs que utilizam o seu TAD devem dar um #include no arquivo.h.

Bibliotecas em C

- A declaração/interface de um TAD define:
  - O nome do tipo.
  - Os nomes das funções exportadas.

- Os nomes das funções devem ser prefixada pelo nome do tipo, evitando conflitos quando tipos distintos são usados em conjunto.

- Exemplo:
  - pto_cria – função para criar um tipo Ponto.
  - circ_cria – função para criar um tipo Círculo.

Bibliotecas em C

- O arquivo de implementação de um TAD deve:
  - Incluir o arquivo de declaração/interface do TAD.
    - Permite utilizar as definições da interface, que são necessárias na implementação.
    - Garante que as funções implementadas correspondem às funções da interface.
      - Compilador verifica se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos.
  - Incluir as variáveis globais e auxiliares.

Bibliotecas em C

```
/* TAD: Ponto (x,y) */
struct ponto {
   float x;
   float 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);
/* Distância */
float pto_distancia(Ponto* p1, Ponto* p2);
```

ponto.h

Interface, chamada protótipo

Bibliotecas em C

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

Ponto* pto_cria (float x, float y)
{
   Ponto* p = (Ponto*) malloc(sizeof(Ponto));
   if (p == NULL)
      printf("Memória Insuficiente!\n");
 else{
      p->x = x;
      p->y = y;
      return p;
      }
}
```

ponto.C

Implementação da Interface

Bibliotecas em C

```
# include <stdio.h>
# include <stdlib.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("Distância entre ponto: %\n", d);
   pto_libera(q);
   pto_libera(p);
   system("pause");
   return 0;
}
```

main.c

Aplicação que usa a Interface

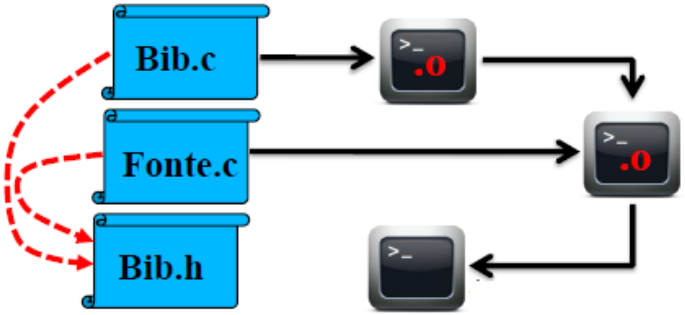
Bibliotecas em C

- Até o momento foram criados executáveis de forma simples (único arquivo fonte).
- Entretanto agora teremos 3 arquivos no mínimo (biblioteca.h, biblioteca.c, main.c).

• Compilação padrão:

gcc fonte.c –o exec

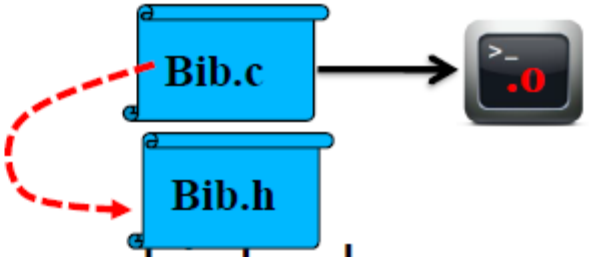
• Entretanto precisamos realizar a "*linkagem*" *do programa.*



Bibliotecas em C

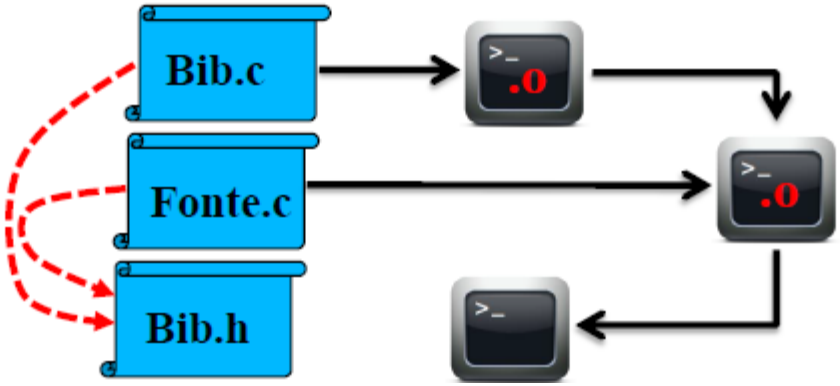
- Compilação da biblioteca

gcc bib.c –c



- Compilação do main + código objeto da bib.o

gcc fonte.c bib.o –o exec



Bibliotecas em C

- Exemplo com o arquivo de Pontos.

unipampa
Universidade Federal do Pampa

Engenharia de Computação

Dúvidas ?