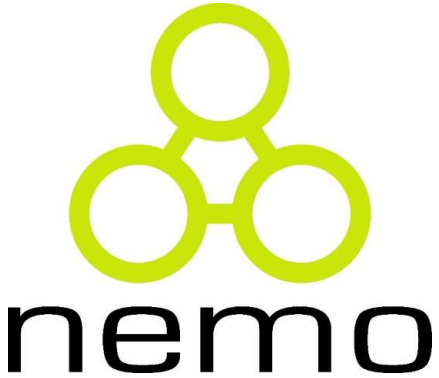


Tópicos Especiais em Programação I

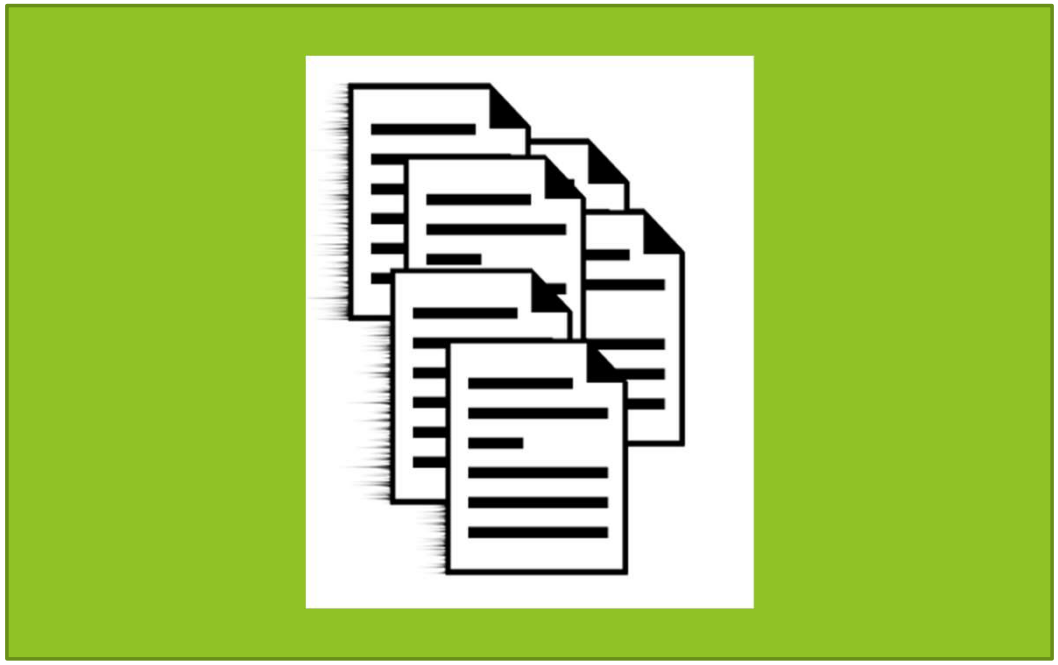
Jordana S. Salamon

jssalamon@inf.ufes.br

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO



Tipos Abstratos de Dados

nemo

Introdução

- As variáveis estudadas até o momento podem armazenar somente dados simples. Ex: int, float, char;
- Contudo, muitas vezes precisamos armazenar dados compostos, ou seja, várias variáveis que se referem ao mesmo conceito do mundo real.
- Exemplos:
  - Uma **pessoa** que possui nome, idade, peso e altura.
  - Um **aluno** que possui nome, número de matrícula e curso.
  - Um **livro** que possui título, gênero, autor(es), ISBN, etc.
  - Um **jogo** que possui nome, gênero, valor, empresa desenvolvedora.

Estruturas de Dados

- Se pensarmos no mundo real, não temos conjuntos de informações separadas, elas fazem parte de um todo.
- Desta forma seria melhor termos um “vetor de todos” ao invés de vetores separados de informações.
- Isso é possível utilizando as **estruturas de dados**.

Sintaxe - Estruturas de Dados

```
struct <nome> {
    <declaração de variáveis>
    ...
    <declaração de variáveis>
};

<declaração de variáveis> =
    <tipo> <nome>, <nome>, ...;
```

```
struct paciente {
    char nome[20];
    int idade;
    float altura, peso;
};

struct ponto {
    int x,y;
};
```

Declarando e Manipulando uma Estrutura

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
};

int main(){
    struct paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
}
```

Typedef

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
};

typedef struct paciente Paciente;

int main(){
    Paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
}
```

Passando uma estrutura por parâmetro

```
void imprimePaciente(Paciente pac){
    printf("Dados de Pessoa\n");
    printf("Nome = %s\n", pac.nome);
    printf("Idade = %d\n", pac.idade);
    printf("Altura = %.2f\n", pac.altura);
    printf("Peso = %.2f\n", pac.peso);
}

int main(){
    Paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
    imprimePaciente(p);
}
```

Retornando uma estrutura

```
Paciente criaPaciente(){
    Paciente p;
    printf("Nome paciente: ");
    scanf("%s", p.nome);
    printf("Idade paciente: ");
    scanf("%d", &p.idade);
    printf("Altura paciente: ");
    scanf("%f", &p.altura);
    printf("Peso paciente: ");
    scanf("%f", &p.peso);
    return p;
}

int main(){
    Paciente p;
    p = criaPaciente();
}
```

Vetores de Estruturas

```
int main(){
    int i;
    Paciente vetor[10];
    for(i=0;i<10;i++){
        printf("Nome paciente: ");
        scanf("%s", vetor[i].nome);
        printf("Idade paciente: ");
        scanf("%d", &vetor[i].idade);
        printf("Altura paciente: ");
        scanf("%f", &vetor[i].altura);
        printf("Peso paciente: ");
        scanf("%f", &vetor[i].peso);
    }
    for(i=0;i<10;i++){
        imprimePaciente(vetor[i]);
    }
}
```

Vetores dentro das Estruturas

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
    char telefones[3][15];
};
typedef struct paciente Paciente;
int main(){
    int i;
    Paciente vetor[2];
    for(i=0;i<2;i++){
        printf("Telefone 1: ");
        scanf("%s", vetor[i].telefones[0]);
        printf("Telefone 2: ");
        scanf("%s", vetor[i].telefones[1]);
        printf("Telefone 3: ");
        scanf("%s", vetor[i].telefones[2]);
    }
}
```

Estruturas dentro das Estruturas

```c
struct endereco{
    char cidade[20];
    char bairro[20];
    int numero;
};

typedef struct endereco Endereco;

struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
    Endereco endereco;
};

typedef struct paciente Paciente;
```

Tipo União

- Localização de memória compartilhada por diferentes variáveis
  - Podem ser de tipos diferentes
  - São usadas para armazenar valores heterogêneos em um mesmo espaço de memória
  - Um único elemento da união pode estar armazenado!

Fonte: http://www.ic.uff.br/~aconci/PCV-modulo10-2015-2.pdf

Tipo União

- Exemplo de tipo união:

```
union exemplo {
    int i;
    char c;
}
```

- Para declarar a variável:
  - union exemplo v;
- Para acessar os elementos:
  - v.i = 10 ou v.c = ‘x’;

Fonte: http://www.ic.uff.br/~aconci/PCV-modulo10-2015-2.pdf

Tipo União

- A memória ocupada por uma união será grande o suficiente para caber o maior membro da união.
- Exemplo:

```c
#include <stdio.h>
#include <string.h>

union Data {
int i;
float f;
char str[20];
};

int main( ) {
union Data data;
data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");
printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);
return 0;
}
```

Qual a saída desse código?

Tipo União

- A memória ocupada por uma união será grande o suficiente para caber o maior membro da união.
- Exemplo:

```c
#include <stdio.h>
#include <string.h>

union Data {
int i;
float f;
char str[20];
};

int main( ) {
union Data data;
data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");
printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);
return 0;
}
```

valores de i e f foram corrompidos porque o valor final atribuído à variável ocupou a mesma posição de memória e por isso só o valor de str está sendo impressa corretamente!

nemo

Tipo Enumeração

- Uma enumeração em C é um tipo de dado que suporta apenas um conjunto finito de valores. A forma de uma enumeração é a seguinte:

```
enum diasemana {domingo, segunda, terca, quarta, quinta, sexta,sabado}; // definicao da enumeracao
enum mes {janeiro, fevereiro, marco, abril, maio, junho, julho, agosto, setembro, outubro, novembro,
    dezembro};
enum cor {branca, amarela, azul, verde, vermelha, preta};
```

```
typedef enum diasemana diasemana;

diasemana Minha_Variavel;

Minha_Variavel = segunda;
```

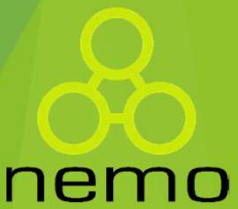
nemo

Tipo Enumeração

- As enumerações criadas podem ser utilizadas para criar variáveis que somente aceitem como valores os elementos que foram listados entre chaves.

- Internamente uma enumeração é representada por um número. Ou seja, no exemplo da primeira enumeração a palavra domingo vale a mesma coisa que 0 (valor padrão), segunda vale 1, terça 2 e assim por diante.

Fonte: https://repositorio.ufsc.br/bitstream/handle/123456789/125955/Tipos_Abstratos_de_dados.pdf?sequence=1&isAllowed=y



Estruturas de Dados x Tipos Abstratos de Dados

- Uma ED é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente, facilitando sua busca e modificação.

- Então, qual a diferença entre uma estrutura de dados e um tipo abstrato de dados?

Fonte: https://pt.wikipedia.org/wiki/Estrutura_de_dados

Tipos Abstratos de Dados

- Tipo abstrato de dado (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados.

- Além disso, é uma metodologia de programação que tem como proposta reduzir a informação necessária para a criação/programação de um algoritmo através da abstração das variáveis envolvidas em uma única entidade fechada, com operações próprias à sua natureza.

Fonte: https://pt.wikipedia.org/wiki/Tipo_abstrato_de_dado

Tipos Abstratos de Dados

- **Qualquer processamento a ser realizada sobre os dados encapsulados em um TAD só poderá ser executada por intermédio dos procedimentos definidos no modelo do TAD**, sendo esta restrição a característica operacional mais útil dessa estrutura.

- Um programa baseado em TAD deverá conter algoritmos e estruturas de dados que implementem, em termos da linguagem de programação adotada, os procedimentos e os modelos de dados dos TADs utilizados pelo programa.

Fonte: http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf

Tipos Abstratos de Dados

- Assim, a implementação de cada TAD pode ocupar porções bem definidas do programa: uma para a *definição das estruturas de dados* e outra para a *definição do conjunto de algoritmos*.

- **Nessas condições, quaisquer alterações realizadas na estrutura de um dado TAD não afetarão as partes do programa que utilizam esse TAD.**

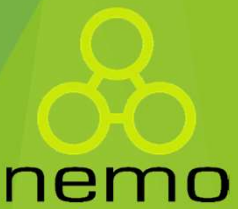
Fonte: http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf

Tipos Abstratos de Dados

- Para definir um TAD, o programador descreve o TAD em dois módulos separados:
- *Um módulo contém a definição do TAD*: representação da estrutura de dados e implementação de cada operação suportada
- *Outro módulo contém a interface de acesso*: apresenta as operações possíveis

- Outros programadores podem, por meio da interface de acesso, usar o TAD sem conhecer os detalhes representacionais e sem acessar o módulo de definição.

Fonte: http://wiki.icmc.usp.br/images/f/fd/AulaTAD.pdf



Tipos Abstratos de Dados

- Uma razão importante para programar em termos de TAD é o fato de que os elementos da estrutura do TAD são acessíveis somente através das operações definidas pelo TAD.

- Essa restrição conduz a uma forma eficiente de programação defensiva, protegendo os dados encapsulados no TAD contra manipulações inesperadas por parte de outros algoritmos.

Fonte: http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf

Tipos Abstratos de Dados

- Uma segunda razão, também importante, para programar em termos de TAD é o fato de que seu uso permite introduzir alterações nas estruturas definidas no nível de implementação visando, por exemplo, aumento de eficiência livre da preocupação de gerar erros no restante do programa.

- Tais erros não ocorrem porque a única conexão entre o TAD e o restante do programa é aquela constituída pela interface imutável dos algoritmos que implementam a estrutura ativa do TAD.

Fonte: http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf

Tipos Abstratos de Dados

- Por fim, pode-se dizer que um TAD bem construído pode tornar-se uma porção de código confiável e genérica, permitindo e aconselhando seu reuso em outros programas.

- Dessa forma, aumenta-se a produtividade na construção de programas e, sobretudo, garante-se a qualidade dos produtos gerados.

Fonte: http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf

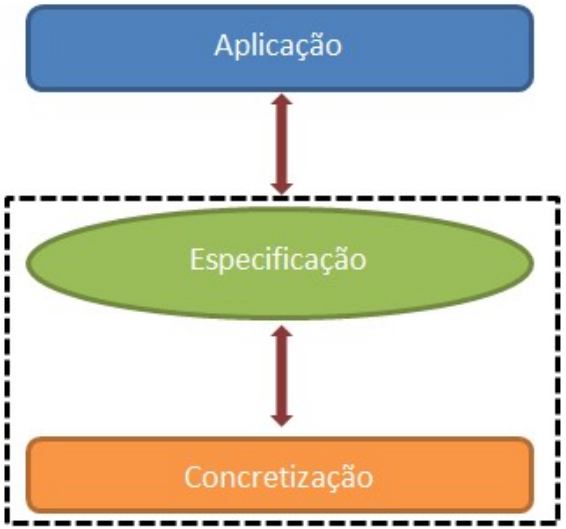
Tipos Abstratos de Dados

- Exemplo de TAD:

```
estrutura Estudante{
    Nome
    Idade
    Matricula
}

funcao Estudante_MaiorDeIdade(Estudante estudante) retorna booleano;
funcao Estudante_ValidaMatricula(Estudante estudante);
```

Tipos Abstratos de Dados

- Estrutura de um TAD:

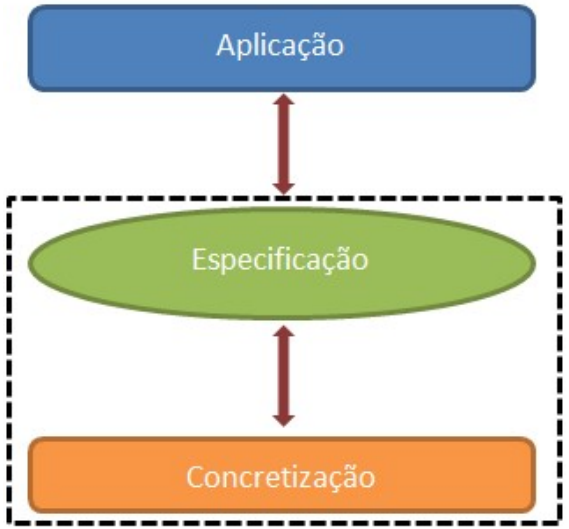


Tipos Abstratos de Dados

- Estrutura de um TAD:

- O TAD possui uma especificação, isto é, a definição do que o tipo abstrato representa, e uma implementação, isto é, a criação de um módulo que deve atender a especificação.

- Se esse módulo é encapsulado de forma que nenhum outro dependa de sua implementação, então esse módulo está de acordo com o princípio de *information hiding*.



Fonte: https://www.embarcados.com.br/ponteiro-em-c-tipo-de-dado-abstrato/

nemo

Módulos

- Um módulo na linguagem C pode ser criado utilizando arquivos de cabeçalho e de implementação:

- *Arquivo de cabeçalho*: Os arquivos '.h' são utilizados para especificar assinatura de funções, definições de constantes, tipos de dados criados pelo usuário etc. De modo geral, os arquivos de cabeçalho tem como função definir a interface de um módulo.

- *Arquivo de implementação*: Os arquivos '.c' implementam as funções definidas na interface.

Fonte: https://www.embarcados.com.br/ponteiro-em-c-tipo-de-dado-abstrato/

nemo

Módulos

- Assim, o arquivo de cabeçalho é utilizado para especificação do tipo de dado abstrato, isto é, das operações que podem ser realizadas e as estruturas de dados definidas.

- Já o arquivo de implementação concretiza as operações definidas na interface.

- Desse modo, uma aplicação pode fazer uso de um determinado módulo a partir da inclusão do seu arquivo de interface.

Fonte: https://www.embarcados.com.br/ponteiro-em-c-tipo-de-dado-abstrato/

Estruturas Opacas

- Após a apresentação dos conceitos e da visão geral sobre modularização, faremos o uso de estruturas opacas para criar a abstração de dados.

- Lembrando que o conceito envolvido na abstração é separar especificação de implementação, então *uma estrutura é opaca se apenas o seu nome é conhecido pela aplicação*.

- Isso é possível se declararmos uma estrutura em um arquivo de interface e sua descrição interna em um arquivo de implementação.

Fonte: https://www.embarcados.com.br/ponteiro-em-c-tipo-de-dado-abstrato/

nemo

Exemplo

- Vamos definir um TAD Ponto para representar um ponto no R^2.

- Neste exemplo, vamos considerar as seguintes operações:
- • cria: operação que cria um ponto com coordenadas x e y;
- • acessaX: operação que devolve a coordenada x de um ponto;
- • acessaY: operação que devolve a coordenada y de um ponto;
- • distancia: operação que calcula a distância entre dois pontos.

Fonte: http://www.ic.uff.br/~cbraga/ed/apostila/ed09-modulos.pdf

Exemplo

- Ponto.h

```
#ifndef PONTO_H_INCLUDED
#define PONTO_H_INCLUDED

/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;
/* Funções exportadas */
/* Função cria
** Aloca e retorna um ponto com coordenadas (x,y)
*/
Ponto cria (float x, float y);
/* Função acessaX
** Devolve o valor da coordenada x de um ponto
*/
float acessaX (Ponto p);
/* Função acessaY
** Devolve o valor da coordenada y de um ponto
*/
float acessaY (Ponto p);
/* Função distancia
** Retorna a distância entre dois pontos
*/
float distancia (Ponto p1, Ponto p2);

#endif // PONTO_H_INCLUDED
```

Fonte: http://www.ic.uff.br/~cbraga/ed/apostila/ed09-modulos.pdf

nemo

Exemplo

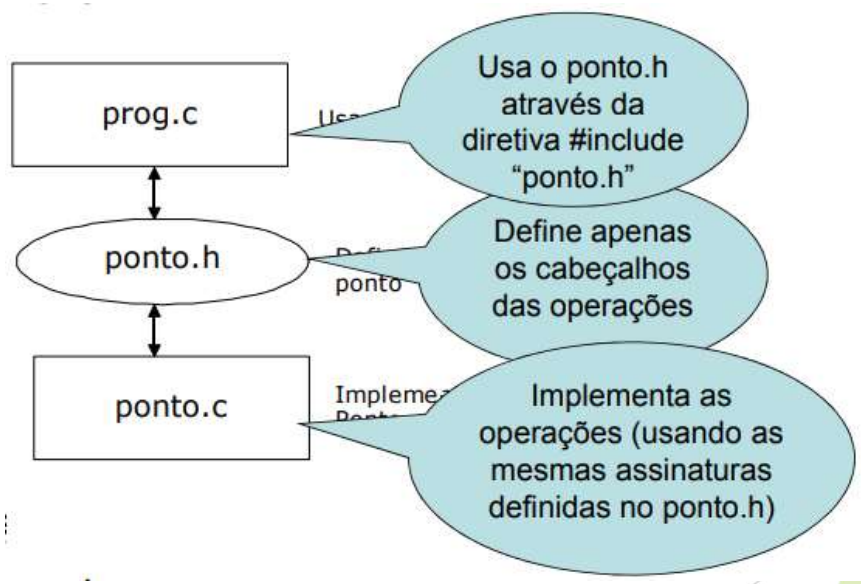
- Ponto.c

```c
#include <math.h> /* sqrt */
#include "ponto.h"

struct ponto {
float x;
float y;
};

Ponto cria (float x, float y) {
Ponto p;
p.x = x;
p.y = y;
return p;
}

float acessaX (Ponto p) {
return p.x;
}

float acessaY (Ponto p) {
return p.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);
}
```

Fonte: http://www.ic.uff.br/~cbraga/ed/apostila/ed09-modulos.pdf

Encapsulamento

- Compilação
  - gcc –c ponto.c
  - gcc –c prog.c
- Linkagem
  - gcc –o prog ponto.o prog.o
- OU
  - gcc -o pontos ponto.c prog.c -lm



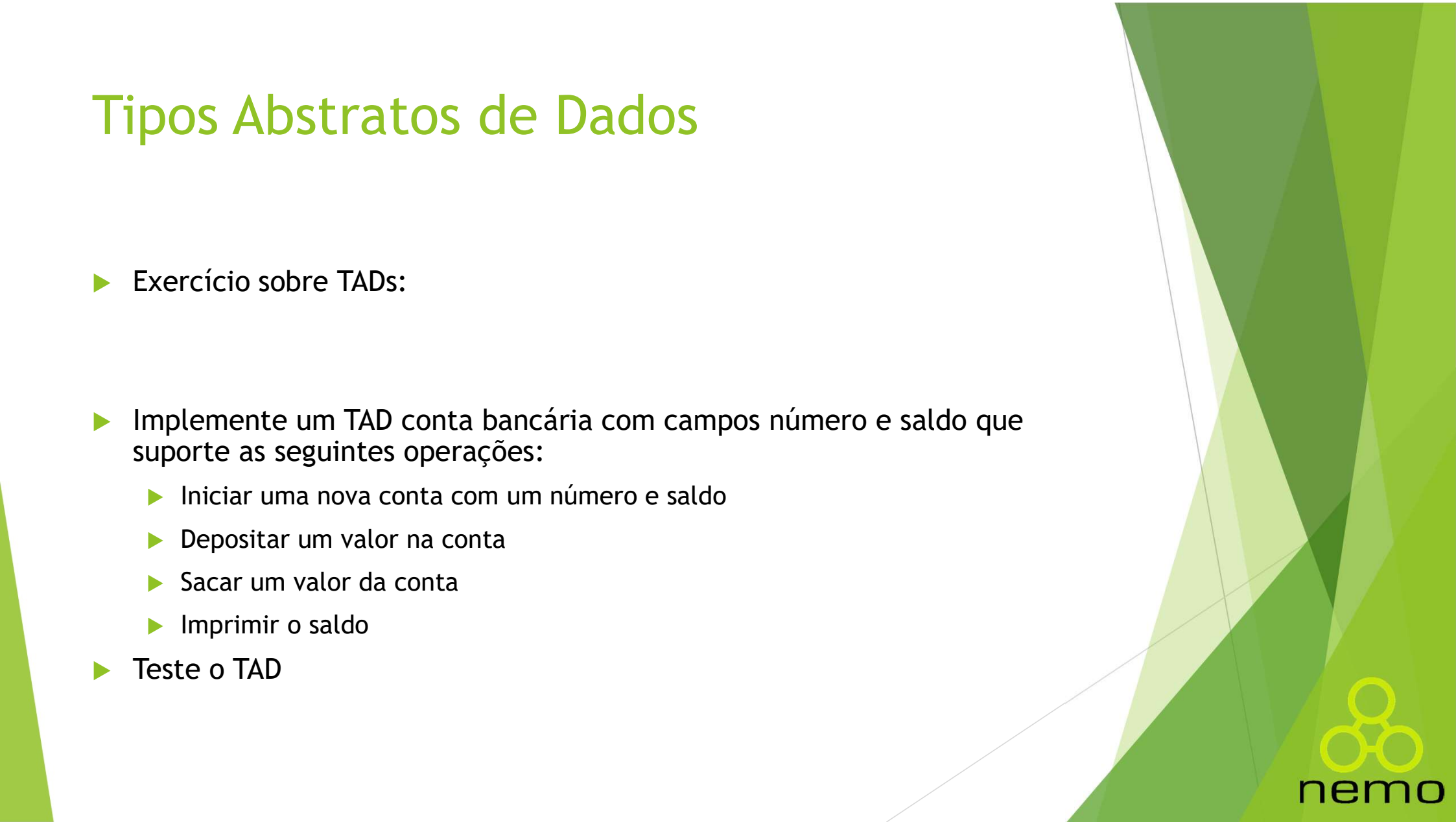
Fonte: http://www.inf.ufes.br/~pdcosta/ensino/2019-2-estruturas-de-dados/slides/Aula8%28TADs%29.pdf

nemo

Tipos Abstratos de Dados

- Exercício sobre TADs:

- Implemente um TAD conta bancária com campos número e saldo que suporte as seguintes operações:
  - Iniciar uma nova conta com um número e saldo
  - Depositar um valor na conta
  - Sacar um valor da conta
  - Imprimir o saldo
- Teste o TAD



Tipos Abstratos de Dados

- **ATENÇÃO**: Para fins de teste dos TADs, utilizaremos TADs sem estruturas opacas, ou seja, a definição da struct será inserida no arquivo de cabeçalho .h

- A partir da segunda parte da disciplina utilizaremos as estruturas de forma opaca!

That’s all Folks!