

# Tipo Abstrato de Dados

Prof. Bruno Travençolo

# Tipo de Dado

---

## ▶ Tipo de Dado

- ▶ Conjunto de valores que uma variável, constante ou função podem assumir.

```
char c;
```

```
int a;
```

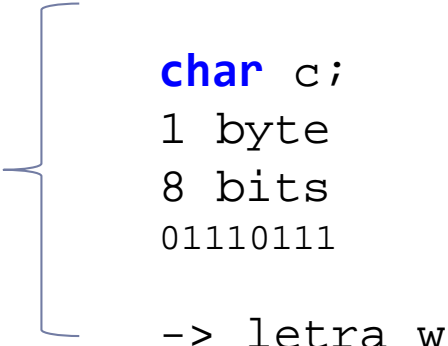
- ▶ Domínio: valores que o tipo de dados pode assumir
- ▶ Define as **operações** que podem ser realizadas com a variável
- ▶ Restrições (*constraints*) que se aplicam ao dado



# Tipo de Dado

- ▶ O tipo de dado permite a leitura de uma região de memória da forma correta (‘interpretar’ os bits)

Endereço	Blocos	Tamanho
11	01110111	(1 byte)
12	11010100	(1 byte)
13	00001000	(1 byte)
14	11010001	(1 byte)
15	00000000	(1 byte)
16	00000010	(1 byte)
17	10000010	(1 byte)
18	01000100	(1 byte)
....		



```
char c;  
1 byte  
8 bits  
01110111  
  
-> letra w
```

# Tipo de Dado

- ▶ O tipo de dado permite a leitura de uma região de memória da forma correta (‘interpretar’ os bits)

Endereço	Blocos	Tamanho
11	01110111	(1 byte)
12	11010100	(1 byte)
13	00001000	(1 byte)
14	11010001	(1 byte)
15	00000000	(1 byte)
16	00000010	(1 byte)
17	10000010	(1 byte)
18	01000100	(1 byte)
....		

**int** a;

4 bytes

32 bits

01110111110101000000100011010001

-> número 2.010.384.593

# Tipo de Dado

---

## ▶ Operações relacionadas aos tipos

### ▶ **int** a

- ▶ Soma
- ▶ Subtração
- ▶ Multiplicação
- ~~▶ Desreferenciar~~
- ▶ ...

### ▶ **int** \*p

- ▶ Soma/Subtração com inteiro (percorre memória)
- ▶ Desreferenciar (\*)
- ~~▶ Soma com outro ponteiro~~
- ▶ ...



# Estrutura de Dados

---

- ▶ Existe uma relação lógica entre os dados
  - ▶ O que é uma relação lógica?

- ▶ Criadas a partir de tipos básicos

- ▶ Vetores,
- ▶ Estruturas (*structs*)
- ▶ Union
- ▶ Enum

```
struct ponto {  
    int x;  
    int y;  
};  
// ponto não é um tipo básico
```

```
float inflation[12];  
// inflação para os 12 meses  
// do ano (array)
```

- ▶ Podem ser já pré-definidos da linguagem (ex: array) ou serem criados (*structs*)



# Estrutura de Dados

## ▶ Existe uma relação lógica

### ▶ Variáveis 'independentes' (sem relação lógica)

- ▶ Bloco
- ▶ TipoSangue
- ▶ LetraDigitada

### ▶ Variáveis relacionadas

- ▶ Sigla[0]
- ▶ Sigla[1]
- ▶ Sigla[2]
- ▶ Sigla[3]

### ▶ Sigla é um vetor, indicando que existe uma relação lógica entre os 4 endereços de memória

Endereço	Blocos	Variável	tipo
1			
2			
3	'H'	Bloco	char
4	'A'	TipoSangue	char
5	'j'	LetraDigitada	Char
6			
7	'U'	Sigla[0]	char[4]
8	'F'	Sigla[1]	
9	'U'	Sigla[2]	
10	'\0'	Sigla[3]	
11		a	int
12	19		
13			
14	....		

# Estrutura de Dados

## ► Existe uma relação lógica

### ► Struct

- Todas as informações são referentes a um aluno

```
struct aluno {  
    int num_aluno;  
    float nota1, nota2;  
    float media;  
};
```

```
struct aluno Maria  
Maria.num_aluno = 2323;  
Maria.nota1 = 10;  
Maria.nota2 = 8;
```

Endereço	Blocos	Variável	tipo
9			
10		Maira.num_aluno	int
11	2323		
12			
13			
14		Maria.nota1	float
15	10		
16			
17			
18		Maria.nota2	float
19	8		
20			
21			
22		Maria.media	float
23	1x		
24			
25			
26			
27			



# Estrutura de Dados

---

## ▶ Operações relacionadas às estruturas

### ▶ `int` `a[10]`

- ▶ Operações do `int`
- ▶ Indexação de elemento (e.g., `a[5]`)
- ▶ Desreferenciar (`*a`)
- ▶ ...

### ▶ `struct` `aluno`

- ▶ Operações de cada tipo da `struct`
- ▶ Atribuição
- ▶ ...



# typedef in C

---

- ▶ Em C existe uma declaração que serve para criar novos *nomes* para tipos de dados

```
// definindo o tipo 'Inteiro'
```

```
typedef int Inteiro
```

```
// utilizando o tipo definido para criar variáveis
```

```
Inteiro num, id;
```

```
Inteiro *vec;
```

```
// Declarando o tipo String, como sendo um 'char *'
```

```
typedef char *String;
```

```
// usando typedef junto com uma definição de estrutura
```

```
typedef struct estudante {
```

```
    Inteiro nmat;
```

```
    String nome;
```

```
} Estudante;
```

```
// neste exemplo struct estudante é chamada de Estudante
```

```
//criando uma declaração chamada PEstudante que é um ponteiro para estudante
```

```
typedef struct estudante *PEstudante
```

---



# typedef in C

---

- ▶ Por que usar typedef
  - ▶ Motivos estéticos
  - ▶ Melhor documentação
    - ▶ `PtrEstudante` é melhor documentado que `struct estudante *`
  - ▶ Parametrizar programas contra problemas de portabilidade
    - ▶ Por exemplo, quantos bits usar em um inteiro (`short`, `int`, `long`) pode variar dependendo da máquina.
    - ▶ `size_t` é um tipo da biblioteca padrão que é um exemplo deste caso. Dependendo do processador o compilador usará uma quantidade de bits diferentes para o `size_t`
      - `void *malloc(size_t n)`



# Tipo Abstrato de Dado (TAD)

---

- ▶ E com relação aos tipos de dados?
- ▶ A linguagem nos fornece alguns tipos “reais” “concretos”
  - ▶ `int`, `float`, `double`, `char`
- ▶ Nem sempre eles são suficientes para resolver os problemas
- ▶ As estruturas de dados também podem não ser suficientes, pois os números de operações são limitadas e relacionadas aos tipos básicos que foram essas estruturas.
  - ▶ `array`, `struct`
- ▶ Precisamos então “abstrair”, criar tipos mais elaborados



# Tipo Abstrato de Dado (TAD)

---

- ▶ *In English: abstract data type (ADT)*
- ▶ O que é abstrato?
  - ▶ Que existe no pensamento ou como uma ideia que não tem existência física ou concreta
  - ▶ que não é concreto; que resulta da abstração.
  - ▶ que possui alto grau de generalização.
- ▶ Exemplos?

(definições obtidas do google)



# Tipo Abstrato de Dado (TAD)

---

- ▶ O que é abstrato?
  - ▶ Algumas palavras relacionadas: “lógico”, “teórico”, “conceitual”
- ▶ Conjunto de dados estruturados e as operações permitidas sobre esses dados
- ▶ Implementação deve ficar oculta do usuário (o usuário neste caso refere-se ao programador que usará o TAD)
- ▶ “Caixa-preta” – usuário não tem acesso direto a informação armazenada no TAD.
- ▶ Implementação é desvinculada da utilização



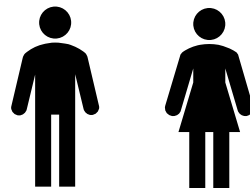
# Usuário x Idealizador/programador

---

- ▶ Normalmente quem desenvolve um TAD (seu idealizador e programador) o faz para uso em outros programas (próprios ou para terceiros – chamados de *usuários*)
- ▶ Para uso do TAD o usuário não necessita saber detalhes de sua implementação – basta ter acesso as operações que o TAD realiza

## Usuário

- Lê o manual para usar o TAD
- Inclui o TAD em seu software
- Não necessita do código fonte do TAD



## Idealizador/Programador

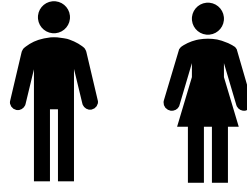
- Desenvolveu todo o Código
- Elaborou manuais de uso do TAD
- Realiza manutenção do TAD

# Modularização

---

## Usuário

- Lê o manual para usar o TAD
- Inclui o TAD em seu software
- Não necessita do código fonte do TAD



## Idealizador/Programador

- Desenvolveu todo o Código
- Elaborou manuais de uso do TAD
- Realiza manutenção do TAD

## ▶ Como usar um TAD sem seu código fonte?

### ▶ Modularizando

- ▶ É possível criar módulos que podem ser compilados separadamente do restante do programa
- ▶ Esses módulos possuem propósito bem definido
- ▶ O usuário pode até possuir o código fonte, mas não necessita usá-lo





# Modularização

---

## ▶ Sem modularizar

- ▶ Escrita do código fica difícil pois o arquivo fica muito grande
- ▶ Reutilização de código por copy+paste, dificultando a manutenção
- ▶ Qualquer modificação exige a recompilação de todo o código
- ▶ Fica no mesmo código entidades não relacionadas (ex. definição de uma Imagem com a definição de um Cadastro)

## ▶ Modularizando

- ▶ Divide-se o problema em problemas menores, facilitando o entendimento
- ▶ Encapsulamento, Segurança, Flexibilidade, Reutilização
- ▶ Apenas o que foi alterado precisa ser recompilado
- ▶ Facilita o trabalho em equipe
- ▶ Validação do módulo independente de outras partes do programa



# Exercício

---

- ▶ Crie um TAD para realizar operações com matrizes.
- ▶ Para fazer o TAD, precisamos definir quais requisitos ele atenderá
- ▶ Quais dados serão armazenados?
- ▶ Quais operações serão necessárias?
  - ▶ Ver no moodle “Basic Matrix Operations”



# Armazenando a matriz como um vetor

---

## Matriz

i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1



# Armazenando a matriz como um vetor

---

Matriz

i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1



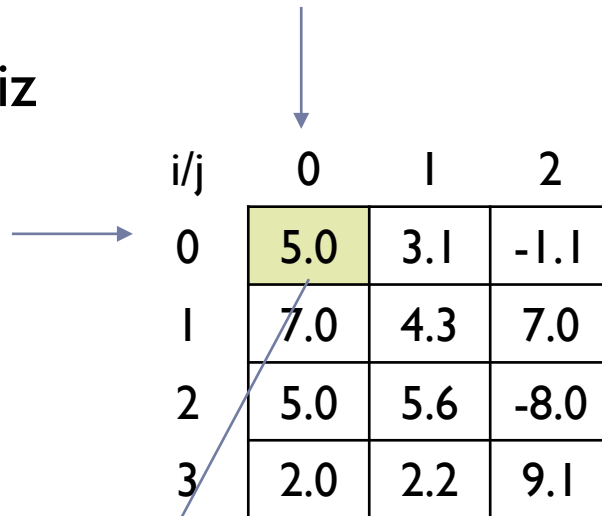
Vetor

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
-----	-----	-----	-----	-----	-----	-----	-----	------	-----	------	-----



# Armazenando a matriz como um vetor

Matriz



i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1

Dada uma posição (i,j), qual é a posição correspondente no vetor?

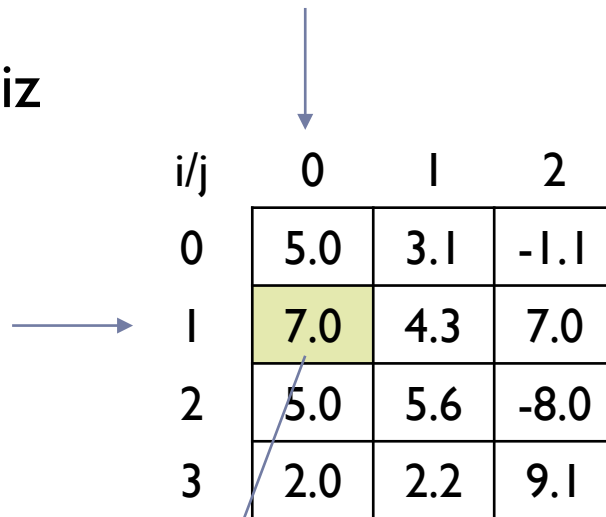
$(0,0) \rightarrow 0$

Vetor v

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	v[10]	v[11]

# Armazenando a matriz como um vetor

Matriz



i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1

Dada uma posição (i,j), qual é a posição correspondente no vetor?

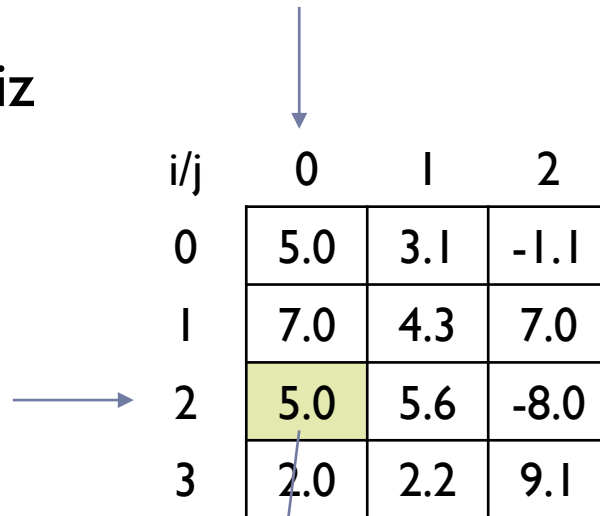
(1,0) → 1

Vetor v

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	v[10]	v[11]

# Armazenando a matriz como um vetor

Matriz



i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1

Dada uma posição (i,j), qual é a posição correspondente no vetor?

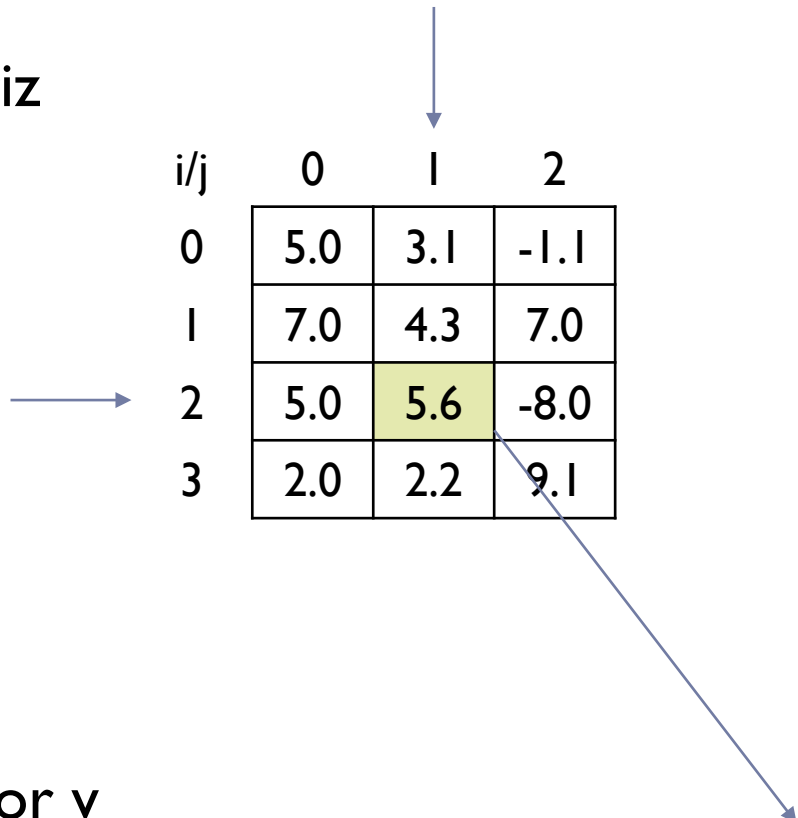
(2,0) → 2

Vetor v

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	v[10]	v[11]

# Armazenando a matriz como um vetor

Matriz



i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1

Dada uma posição (i,j), qual é a posição correspondente no vetor?

$(2,1) \rightarrow 6$

Vetor v

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	v[10]	v[11]



# Armazenando a matriz como um vetor

nrows (4)

i/j	0	1	2
0	5.0	3.1	-1.1
1	7.0	4.3	7.0
2	5.0	5.6	-8.0
3	2.0	2.2	9.1

$$pos = j * nrows + i$$

i	j	pos
0	0	$0*4 + 0 = 0$
1	0	$0*4 + 1 = 1$
2	0	$0*4 + 2 = 2$
..		
1	1	$1*4 + 1 = 5$
3	2	$2*4 + 3 = 11$

5.0	7.0	5.0	2.0	3.1	4.3	5.6	2.2	-1.1	7.0	-8.0	9.1
-----	-----	-----	-----	-----	-----	-----	-----	------	-----	------	-----

---

## ► TMat2D.c

```
#include "TMat2D.h"
struct TMat2D
{
    int nrows; // número de linhas
    int ncolums; // número de colunas
    double *data; // ponteiro para os dados da matriz
};
```

## ► TMat2D.h

```
typedef struct TMat2D TMat2D;
```

---



---

## ► TMat2D.h

```
typedef struct TMat2D TMat2D;
```

```
TMat2D *mat2D_create(int nrows, int ncolums);
```

```
int mat2D_free(TMat2D *mat);
```

```
int mat2D_set_value(TMat2D *mat, int i, int j, double val);
```

```
int mat2D_get_value(TMat2D *mat, int i, int j, double *val);
```



---

► main.c

➡ `#include "TMat2D.h"`

```
int main(){  
    TMat2D* mat;  
    mat = mat2d_create();
```

Incluir o .h do TAD para ter acesso aos tipos de dados e operações do TAD



---

► main.c

```
#include "TMat2D.h"
```

```
int main(){
```

```
    TMat2D* mat;  
    → mat = mat2d_create();
```

Cria uma ponteiro para o TAD. Note que ainda não houve alocação de memória para o TAD em si



---

► main.c

```
#include "TMat2D.h"
```

```
int main(){  
    TMat2D* mat;  
    mat = mat2d_create();
```



Cria a matriz, alocando a memória pré-definida e retorna o ponteiro (endereço) para a região alocada para o main.c



# Tipo Abstrato de Dado (TAD)

---

## ▶ Vantagens de uso

### ▶ Encapsulamento

- ▶ Usuário não precisa conhecer como é a implementação, somente como usar o TAD

### ▶ Segurança

- ▶ Usuário não tem acesso direto aos dados
- ▶ Dificulta a violação de memória

### ▶ Flexibilidade

- ▶ Pode-se modificar o código do TAD sem que seu cabeçalho (interface) seja alterado

### ▶ Reutilização

- ▶ O uso de módulos facilita o compartilhamento para diferentes programas e usuários



# Modularização em C

---

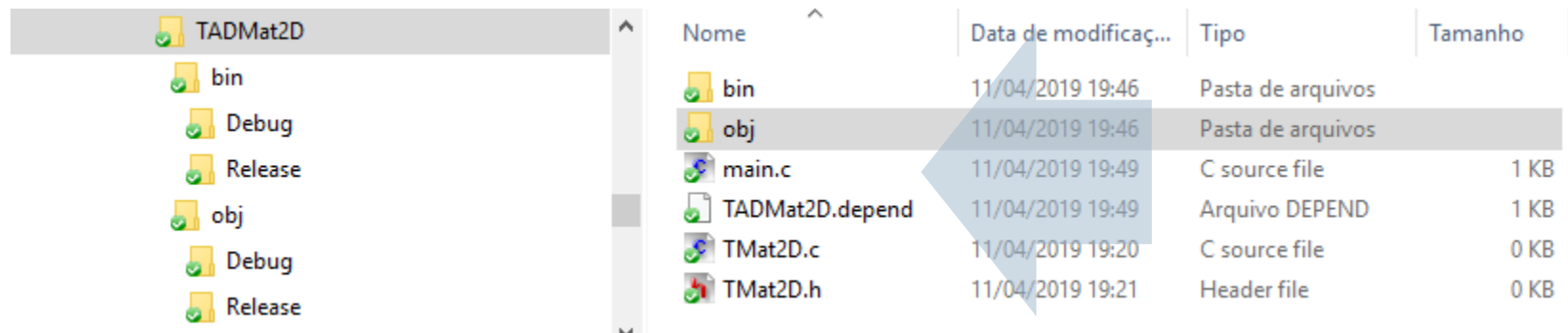
- ▶ Programas normalmente são divididos em vários arquivos
  - ▶ Arquivos fonte com extensão **.c** (módulos)
- ▶ Cada módulo deve ser compilado separadamente
  - ▶ Para isso usa-se um compilador
  - ▶ São gerados arquivos objeto não executáveis
  - ▶ Arquivos em linguagem de máquina com extensão **.o** ou **.obj**
- ▶ Arquivos objeto devem ser juntados em um **executável**
  - ▶ Para isso usa-se um ligador ou link-editor
  - ▶ Resultado: um único arquivo em linguagem de máquina
    - ▶ Usualmente com extensão **.exe** no Windows ou com propriedade **+x** no linux





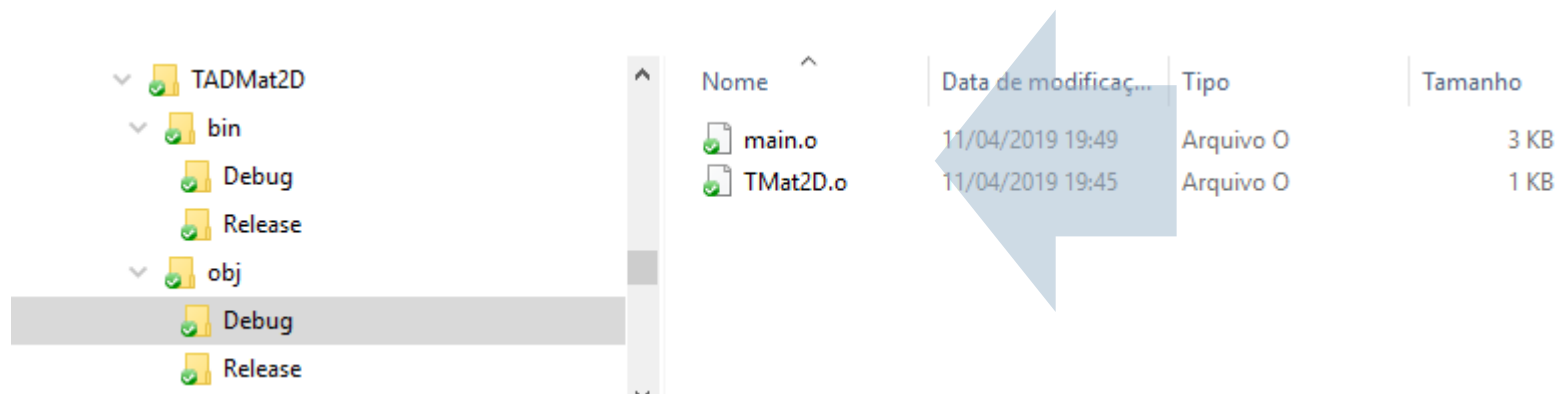
# Modularização em C

## ► Fonte e cabeçalhos (.c e .h)



Nome	Data de modificaç...	Tipo	Tamanho
bin	11/04/2019 19:46	Pasta de arquivos	
obj	11/04/2019 19:46	Pasta de arquivos	
main.c	11/04/2019 19:49	C source file	1 KB
TADMAT2D.depend	11/04/2019 19:49	Arquivo DEPEND	1 KB
TMat2D.c	11/04/2019 19:20	C source file	0 KB
TMat2D.h	11/04/2019 19:21	Header file	0 KB

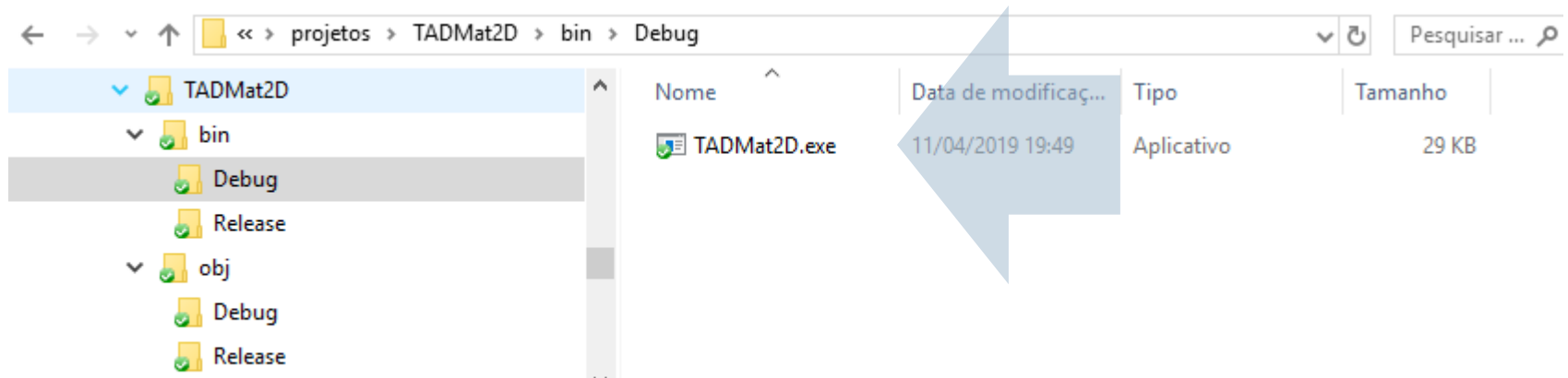
## ► Objetos (em Debug ou Release)



Nome	Data de modificaç...	Tipo	Tamanho
main.o	11/04/2019 19:49	Arquivo O	3 KB
TMat2D.o	11/04/2019 19:45	Arquivo O	1 KB

# Modularização em C

## ► Executável (em Debug ou Release)



## ► Executando

### ► Duplo clique/ chamada na linha de comando

```
C:\Users\Bruno>cd D:\projetos\TADMat2D\bin\Debug
```

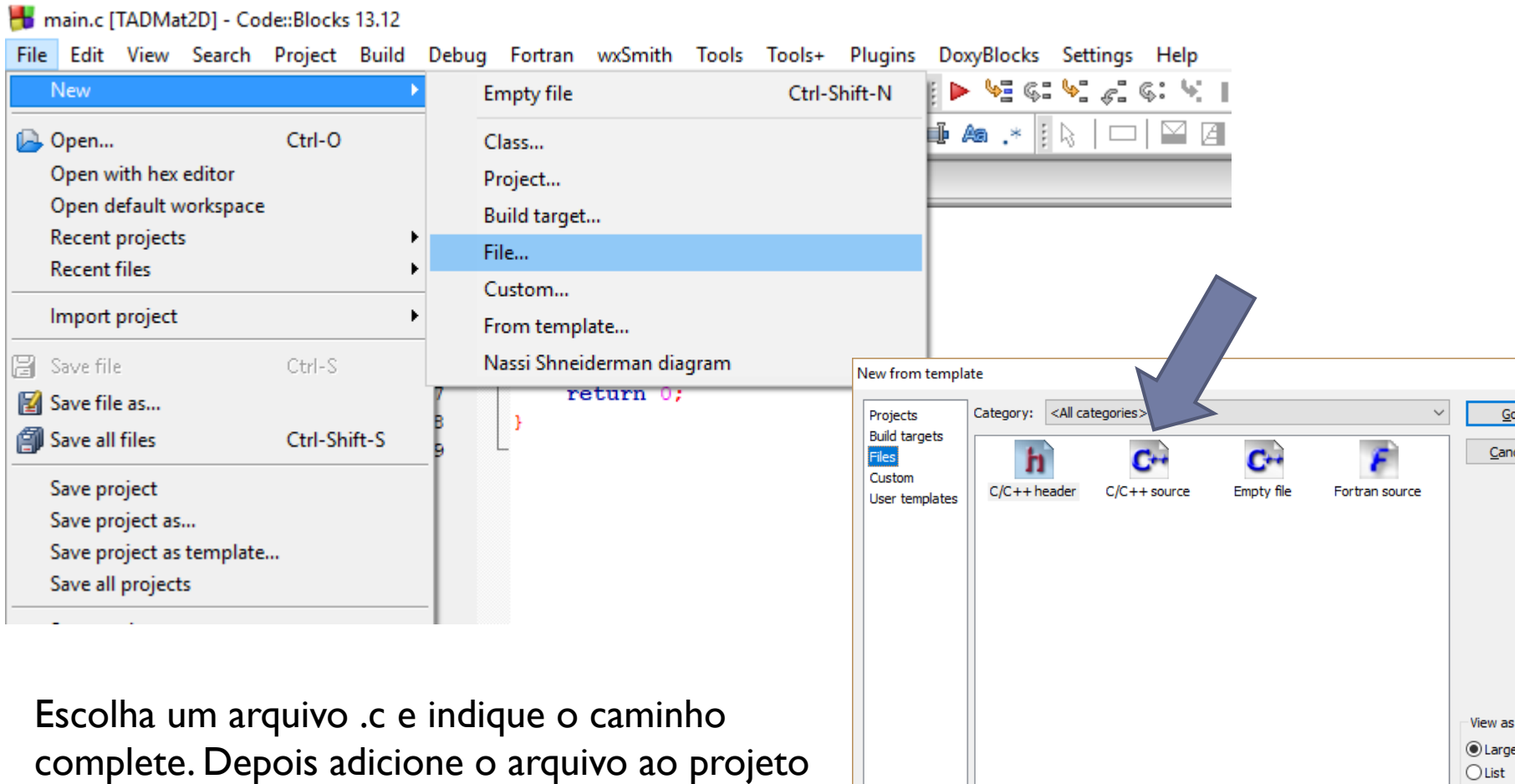
```
C:\Users\Bruno>d:
```

```
D:\projetos\TADMat2D\bin\Debug>.\TADMat2D.exe
```

```
Hello world!
```

# Criando um TAD no codeblocks

## ► Após criar o projeto com o main.c faça



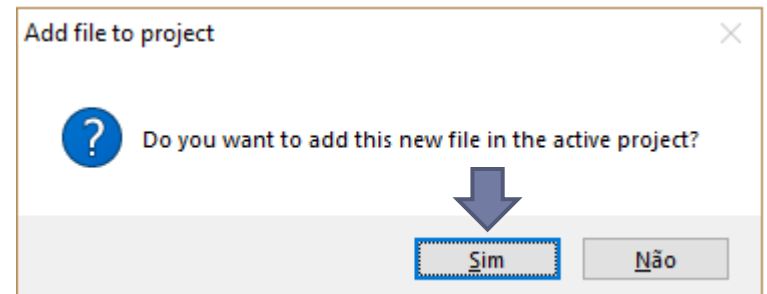
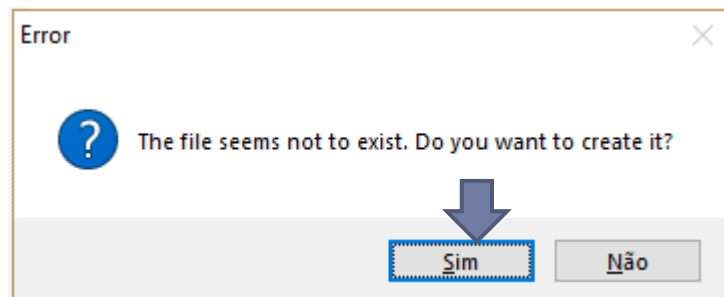
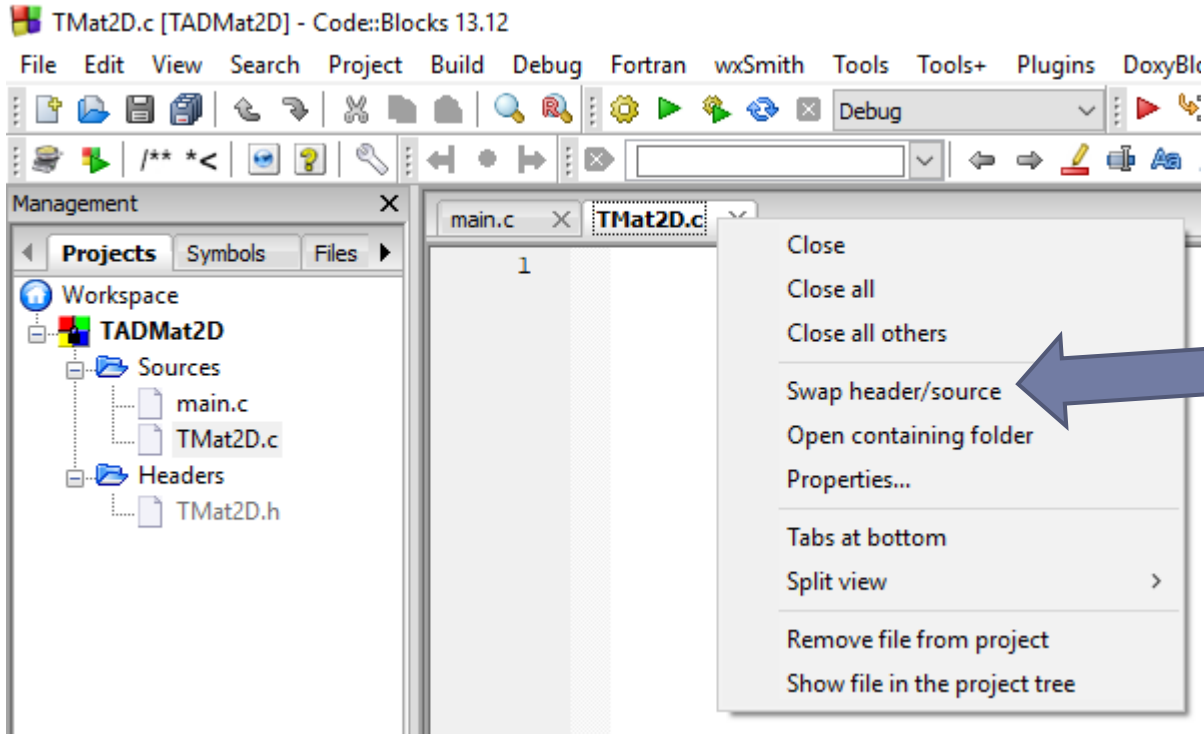
The screenshot shows the Code::Blocks 13.12 IDE. The 'File' menu is open, and the 'New' option is selected, leading to a submenu where 'File...' is highlighted. A blue arrow points from the 'File...' option in the submenu to the 'New from template' dialog box. In this dialog, the 'Category' is set to '<All categories>' and the 'C/C++ source' template is selected. The background shows a C program with a `return 0;` statement.

Escolha um arquivo .c e indique o caminho completo. Depois adicione o arquivo ao projeto

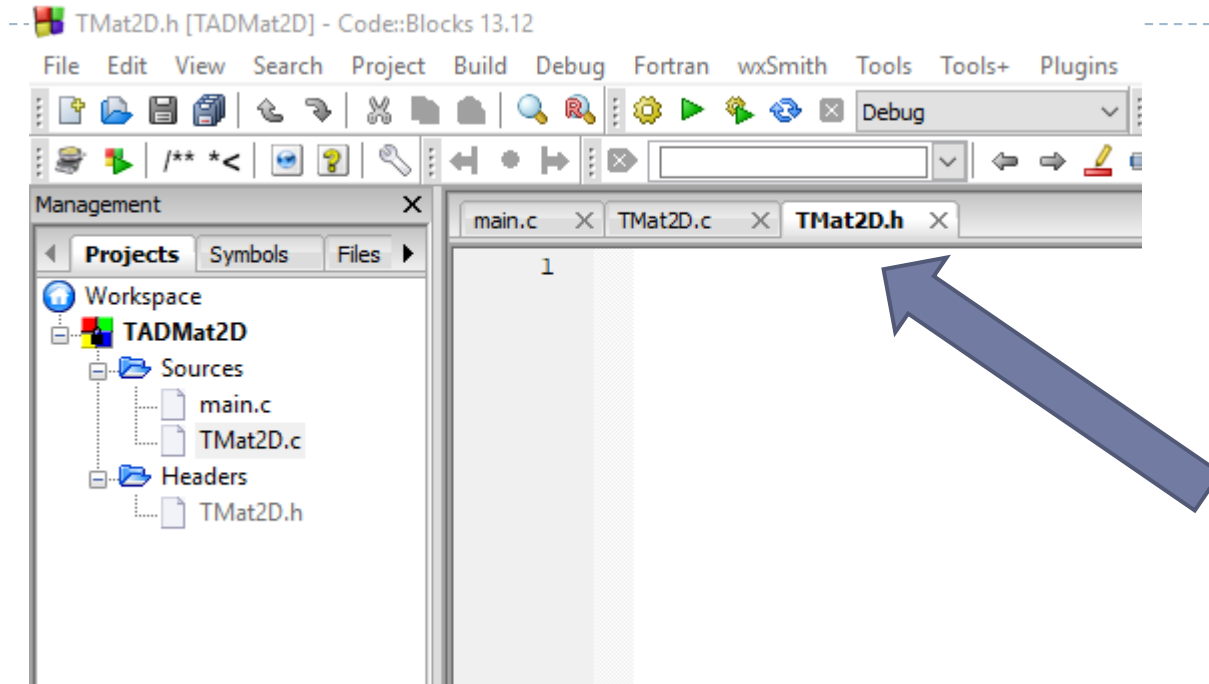
TIP: Try right-clicking an item

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

# Criando um TAD no codeblocks



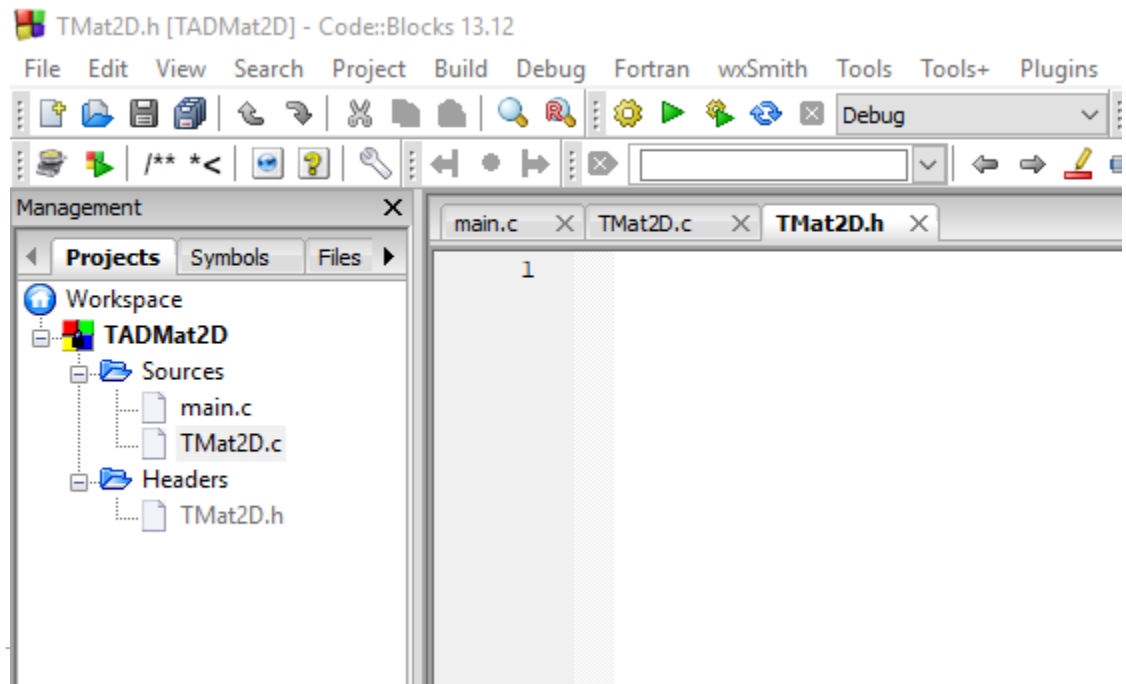
# Criando um TAD no codeblocks



# Arquivos e TAD

---

- ▶ 3 arquivos
  - ▶ main.c – programa principal
  - ▶ TMat2D.c codificação do TAD
  - ▶ TMat2D.h header (cabeçalho) do TAD



# Outras IDEs (ou direto no gcc)

---

- ▶ Crie os arquivos necessários em seu editor preferido
  - ▶ main.c; TMat2D.c ; TMat2D.h
  - ▶ Compile os arquivos .c
  - ▶ Ligue os arquivos gerados (arquivos objetos .o) e gere o arquivo executável (.exe no Windows; Linux sem necessidade de extensão)
  - ▶ Pesquisar: construir um Makefile

```
$>gcc -c main.c -o main.o  
$>gcc -c TMat2D.c -o TMat2D.o  
$>gcc -o TADMAT2D.exe main.o TMat2D.o
```



# Compilando vários arquivos

---

- ▶ Compilando um só arquivo, sem gerar executável (main)

```
$>gcc -c main.c -o main.o
```

- ▶ Compilando um só arquivo, sem gerar executável (TAD)

```
$>gcc -c TMat2D.c -o TMat2D.o
```

- ▶ Juntando (linking) todos os arquivos e gerando o executável

```
$>gcc -o TADMat2D.exe main.o TMat2D.o
```

