

# Estruturas de Dados

Prof. Rodrigo Martins

[rodrigo.martins@francomontoro.com.br](mailto:rodrigo.martins@francomontoro.com.br)

# Cronograma da Aula

- Vetores ou Arrays
- Matrizes ou Arrays Multidimensionais
- Funções
- Ponteiros
- Módulos
- Exemplos e Exercícios

# Vetores ou Arrays

- Um array é uma coleção de um ou mais objetos, do mesmo tipo, armazenados em endereços adjacentes de memória. Cada objeto é chamado de elemento do array.
- Da mesma forma que para variáveis simples, damos um nome ao array. O tamanho do array é o seu número de elementos. Cada elemento do array é numerado, usando um inteiro chamado de índice.
- Em C++ , a numeração começa com 0 e aumenta de um em um.
- Assim, o último índice é igual ao número de elementos do array menos um.

# Arrays – exemplo1-array.cpp

```
*exemplo6.cpp x
1  #include <iostream>
2
3  using namespace std;
4  #define ESTUDANTES 5
5
6  int main()
7  {
8      int indice;
9      float total, nota[ESTUDANTES];
10     indice = 0;
11
12     //preenche o vetor
13     while (indice < ESTUDANTES)
14     {
15         cout << "Entre a nota do estudante " << indice + 1 << ": ";
16         cin >> nota[indice];
17         indice = indice + 1;
18     }
19
20     cout << "-----" << endl;
21
22     total = 0;
23     int qtd = 1;
```

# Arrays – exemplo1-array.cpp

```
24 //imprime o vetor
25 for (int i = 0; i < ESTUDANTES; i++)
26 {
27     cout << "Nota " << qtd << ": " << nota[i] << endl;
28     total = total + nota[i];
29     qtd++;
30 }
31
32 cout << endl << "Media: " << total / ESTUDANTES << endl;
33 return 0;
34 }
35
```

# Arrays – exemplo2-array.cpp

```
*exemplo7.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  #define NOTAS 5
6
7  float mediaValoresVetor(int vet[], int tam)
8  {
9      float soma = 0;
10     for (int i = 0; i < NOTAS; i++)
11     {
12         soma += vet[i];
13     }
14     return soma / tam;
15 }
16
17 int main()
18 {
19     int vet[NOTAS], acima = 0;
20     float media;
21
22     for (int i = 0; i < NOTAS; i++)
23     {
24         cout << "Digite a nota " << i + 1 << ": " << endl;
25         cin >> vet[i];
26     }
27 }
```

# Arrays – exemplo2-array.cpp

```
28     media = mediaValoresVetor(vet, NOTAS);
29     cout << "Media: " << media << endl;
30
31     for (int i = 0; i < NOTAS; i++)
32     {
33         if (vet[i] > media)
34         {
35             acima++;
36         }
37     }
38     cout << "Valores acima da media: " << acima << endl;
39     cout << "Valores abaixo da media: " << NOTAS - acima;
40
41     return 0;
42 }
43
```

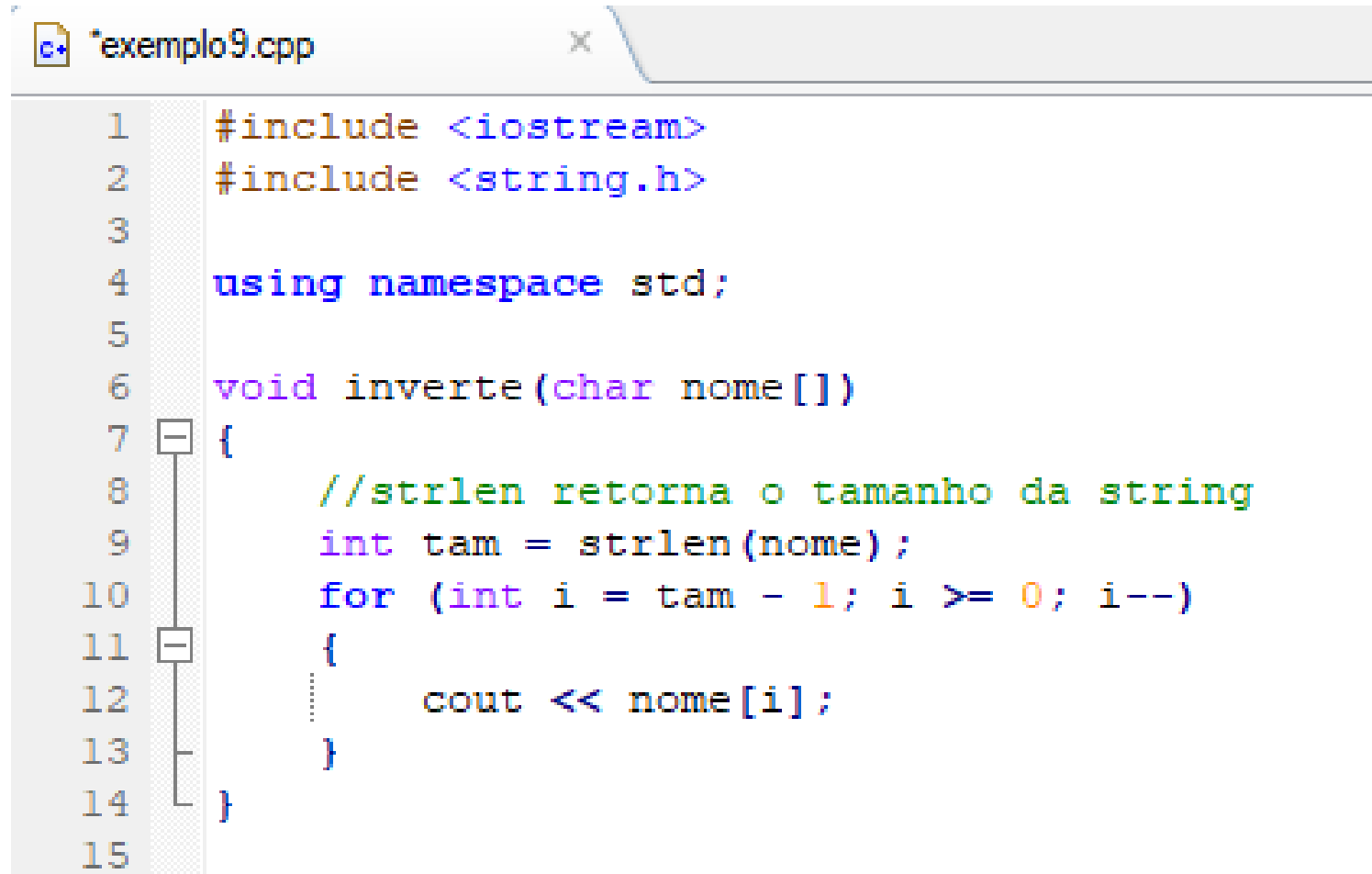
# Vetores de caracteres

## exemplo3-array.cpp

```
*exemplo8.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      //char nome[] = {'r', 'o', 'd', 'r', 'i', 'g', 'o', '\0'};
8      char nome[] = "rodrigo";
9      int i = 0;
10     /*
11     '\0' é um caracter null, com o valor numérico 0 é considerado false
12     Uma string é um array de caracteres, apesar de ser um array,
13     deve-se ficar atento para o fato de que as strings têm no elemento
14     seguinte a última letra da palavra/frase armazenada, um caractere '\0'.
15     */
16
17     //while (nome[i])
18     while (nome[i] != '\0')
19     {
20         cout << nome[i];
21         i++;
22     }
23     return 0;
24 }
```



# Arrays – exemplo4-array.cpp



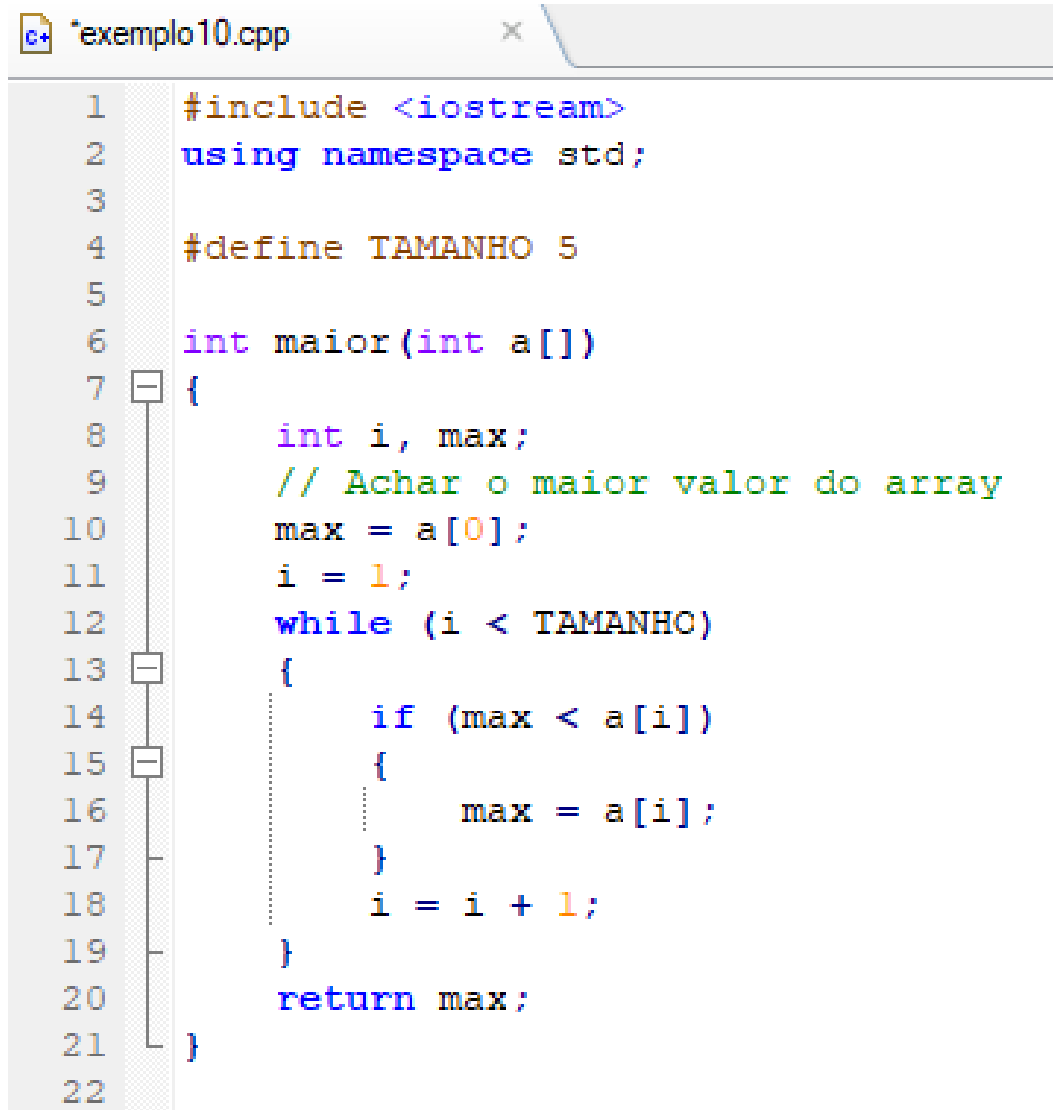
```
1  #include <iostream>
2  #include <string.h>
3
4  using namespace std;
5
6  void inverta(char nome[])
7  {
8      //strlen retorna o tamanho da string
9      int tam = strlen(nome);
10     for (int i = tam - 1; i >= 0; i--)
11     {
12         cout << nome[i];
13     }
14 }
15
```

The image shows a code editor window titled "exemplo9.cpp". The code is written in C++ and defines a function named "inverta" (note the typo in the original image) that takes a character array "nome" as input. The function uses "strlen" to determine the length of the string and then iterates from the end of the string to the beginning, printing each character. The code is formatted with line numbers on the left and uses color-coding for keywords (blue), comments (green), and identifiers (purple). There are also small square icons next to lines 7, 11, and 13, possibly indicating a debugger or a code folding tool.

# Arrays – exemplo4-array.cpp

```
16  int main()
17  {
18      char nome[] = "rodrigo";
19      inverte(nome);
20      cout << endl;
21      //isalpha retorna true se caractere testado for alfabético
22      if (isalpha(nome[0]))
23      {
24          cout << "caractere alfabetico" << endl;
25      }
26      else
27      {
28          cout << "caractere numerico" << endl;
29      }
30
31      //isdigit retorna true se for um dígito
32      if (isalpha(nome[0]))
33      {
34          cout << "letra" << endl;
35      }
36      else
37      {
38          cout << "numero" << endl;
39      }
40
41      //isupper retorna true se o caractere for maiusculo
42      if (isupper(nome[0]))
43      {
44          cout << "maiusculo" << endl;
45      }
46      else
47      {
48          cout << "minusculo" << endl;
49      }
50      return 0;
51  }
```

# Arrays – exemplo5-array.cpp



```
*exemplo10.cpp x
1  #include <iostream>
2  using namespace std;
3
4  #define TAMANHO 5
5
6  int maior(int a[])
7  {
8      int i, max;
9      // Achar o maior valor do array
10     max = a[0];
11     i = 1;
12     while (i < TAMANHO)
13     {
14         if (max < a[i])
15         {
16             max = a[i];
17         }
18         i = i + 1;
19     }
20     return max;
21 }
22
```

# Arrays – exemplo5-array.cpp

```
23  int main()  
24  {  
25      int i, valor[TAMANHO];  
26      i = 0;  
27      while (i < TAMANHO)  
28      {  
29          cout << "Entre um inteiro: ";  
30          cin >> valor[i];  
31          i = i + 1;  
32      }  
33      cout << "O maior eh " << maior(valor) << endl;  
34  
35      return 0;  
36  }  
37
```

# Matrizes ou Arrays Multidimensionais

- Em C++, é possível também definir arrays com 2 ou mais dimensões.
- Eles são arrays de arrays.
- Um array de duas dimensões podem ser imaginado como uma matriz (ou uma tabela).

# Arrays Multidimensionais

## exemplo6-array.cpp

```
*exemplo11.cpp x
1  #include <iostream>
2
3
4  using namespace std;
5
6  #define LIN 2
7  #define COL 2
8
9
10 int main()
11 {
12     int matriz[LIN][COL], i, j;
13
14     //preenche a matriz
15     for (i = 0; i < 2; i++)
16     {
17         for (j = 0; j < 2; j++)
18         {
19             cout << "Digite um numero inteiro: ";
20             cin >> matriz[i][j];
21         }
22     }
23
24     cout << "===== " << endl;
```

# Arrays Multidimensionais

## exemplo6-array.cpp

```
25 //imprime a matriz na tela
26 for (i = 0; i < 2; i++)
27 {
28     for (j = 0; j < 2; j++)
29     {
30         cout << "O valor na posicao " << i << " " << j << " eh: "
31         << matriz[i][j] << endl;
32     }
33 }
34
35 return 0;
36 }
37
```

# Exercícios

1. Escreva um programa em C++ que permita a leitura dos nomes de 10 pessoas e armaze os nomes lidos em um vetor. Após isto, o algoritmo deve permitir a leitura de mais 1 nome qualquer de pessoa e depois escrever a mensagem ACHEI, se o nome estiver entre os 10 nomes lidos anteriormente (guardados no vetor), ou NÃO ACHEI caso contrário.
2. Escreva um programa em C++ que permita a leitura das notas de uma turma de 20 alunos. Calcular a média da turma e contar quantos alunos obtiveram nota acima desta média calculada. Escrever a média da turma e o resultado da contagem.
3. Ler um vetor A de 10 números. Após, ler mais um número e guardar em uma variável X. Armazenar em um vetor M o resultado de cada elemento de A multiplicado pelo valor X. Logo após, imprimir o vetor M.



# Exercícios

4. Faça um programa em C++ para ler 20 números e armazenar em um vetor. Após a leitura total dos 20 números, o algoritmo deve escrever esses 20 números lidos na ordem inversa.
5. Faça um programa em C++ para ler um valor N qualquer (que será o tamanho dos vetores). Após, ler dois vetores A e B (de tamanho N cada um) e depois armazenar em um terceiro vetor Soma a soma dos elementos do vetor A com os do vetor B (respeitando as mesmas posições) e escrever o vetor Soma.
6. Faça um programa em C++ para ler e armazenar em um vetor a temperatura média de todos os dias do ano. Calcular e escrever:
  - a) Menor temperatura do ano
  - b) Maior temperatura do ano
  - c) Temperatura média anual
  - d) O número de dias no ano em que a temperatura foi inferior a média anual

# Funções

- É importante lembrar que em C++, uma função é um bloco de código que é definido uma vez e pode ser chamado várias vezes a partir de diferentes partes do programa.
- As funções em C++ podem ter ou não argumentos, e podem ou não retornar valores.
- Para definir uma função em C++, você deve seguir o seguinte formato:

```
tipo_de_retorno nome_da_função (lista de parâmetros) {  
    // Corpo da função  
}
```

# Funções

- Onde
  - **tipo\_de\_retorno** é o tipo de dado que a função retorna (por exemplo, int, float, double, void, etc.).
  - **nome\_da\_função** é o nome que você dá para a função.
  - **lista\_de\_parâmetros** é a lista de argumentos que a função recebe. Cada argumento é composto por um tipo e um nome (por exemplo, int x, float y, double z, etc.).
  - **corpo\_da\_função** é o bloco de código que contém as instruções que serão executadas quando a função for chamada.

# Funções

- Por exemplo, uma função que recebe dois argumentos do tipo int e retorna a soma desses valores seria definida da seguinte forma:

```
int soma(int a, int b) {  
    return a + b;  
}
```

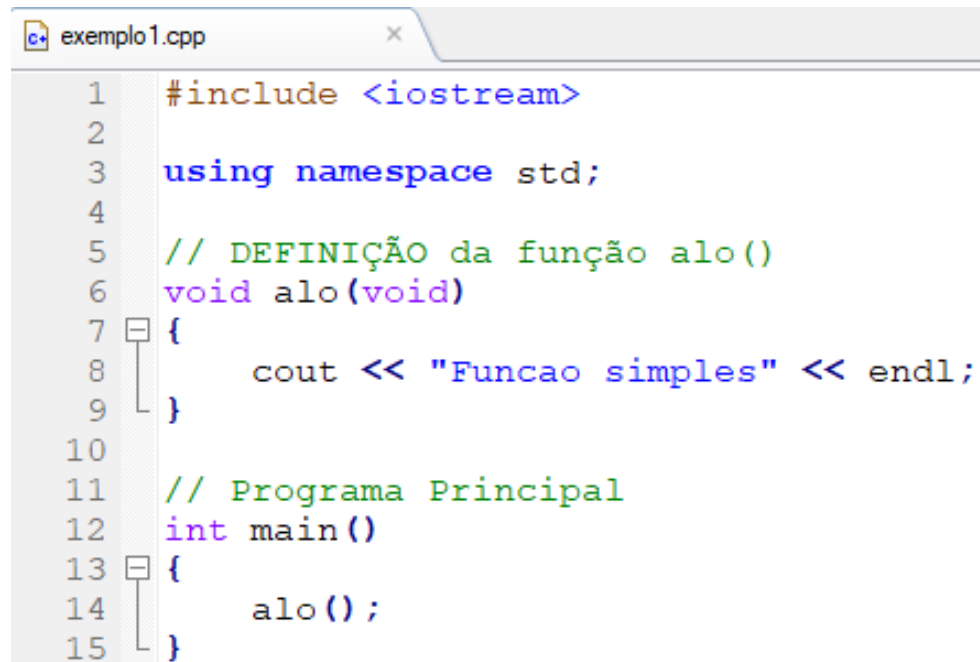
- Para chamar uma função em C++, basta escrever o nome da função seguido dos argumentos entre parênteses. Por exemplo:

```
int resultado = soma(2, 3);
```

# Funções – exemplo1.cpp

```
void nome-da-função (void)
{
    declarações e sentenças (corpo da função)
}
```

- O primeiro **void** significa que esta função não tem tipo de retorno (não retorna um valor), e o segundo significa que a função não tem argumentos (ela não precisa de nenhuma informação externa para ser executada).



```
exemplo1.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  // DEFINIÇÃO da função alo()
6  void alo(void)
7  {
8      cout << "Funcao simples" << endl;
9  }
10
11 // Programa Principal
12 int main()
13 {
14     alo();
15 }
```

# Funções - exemplo2.cpp

```
exemplo2.cpp x
1  #include <iostream>
2  #include <locale.h>
3
4  using namespace std;
5
6  //protótipo da função
7  bool par(int num);
8  void mensagem();
9
10 int main()
11 {
12     /*
13      comando de regionalização do C++ para que não somente acentue as palavras
14      corretamente, mas que mostre datas e horas em português.*/
15     setlocale(LC_ALL, "Portuguese");
16     int n = 0;
17
18     mensagem();
19
20     cout << "Digite um número: ";
21     cin >> n;
22
23     if (par(n))
24     {
25         cout << "O numero " << n << " eh par" << endl;
26     }
27     else
28     {
29         cout << "O numero " << n << " eh impar" << endl;
30     }
31     return 0;
32 }
```

# exemplo2.cpp

```
33 bool par(int num)
34 {
35     if (num % 2 == 0)
36     {
37         return true;
38     }
39     else
40     {
41         return false;
42     }
43 }
44
45 void mensagem()
46 {
47     cout << "Aula do Modulo 2" << endl;
48     cout << endl;
49 }
```

# Funções que retornam um valor

- Uma função pode retornar um valor para o programa que o chamou.
- Uma função que retorna um valor tem no cabeçalho o nome do tipo do resultado.
- O valor retornado pode ser de qualquer tipo, incluindo, bool, int, float e char.



# Funções - exemplo2.1.cpp

```
1 // programa que verifica se 3 numeros podem ser os lados de um
2 // triangulo reto.
3 #include <iostream>
4 #include <locale.h>
5 using namespace std;
6
7 // funcao que calcula o quadrado de um numero
8
9 int quadrado(int n)
10 {
11     return n * n;
12 }
13
```

# Funções - exemplo2.1.cpp

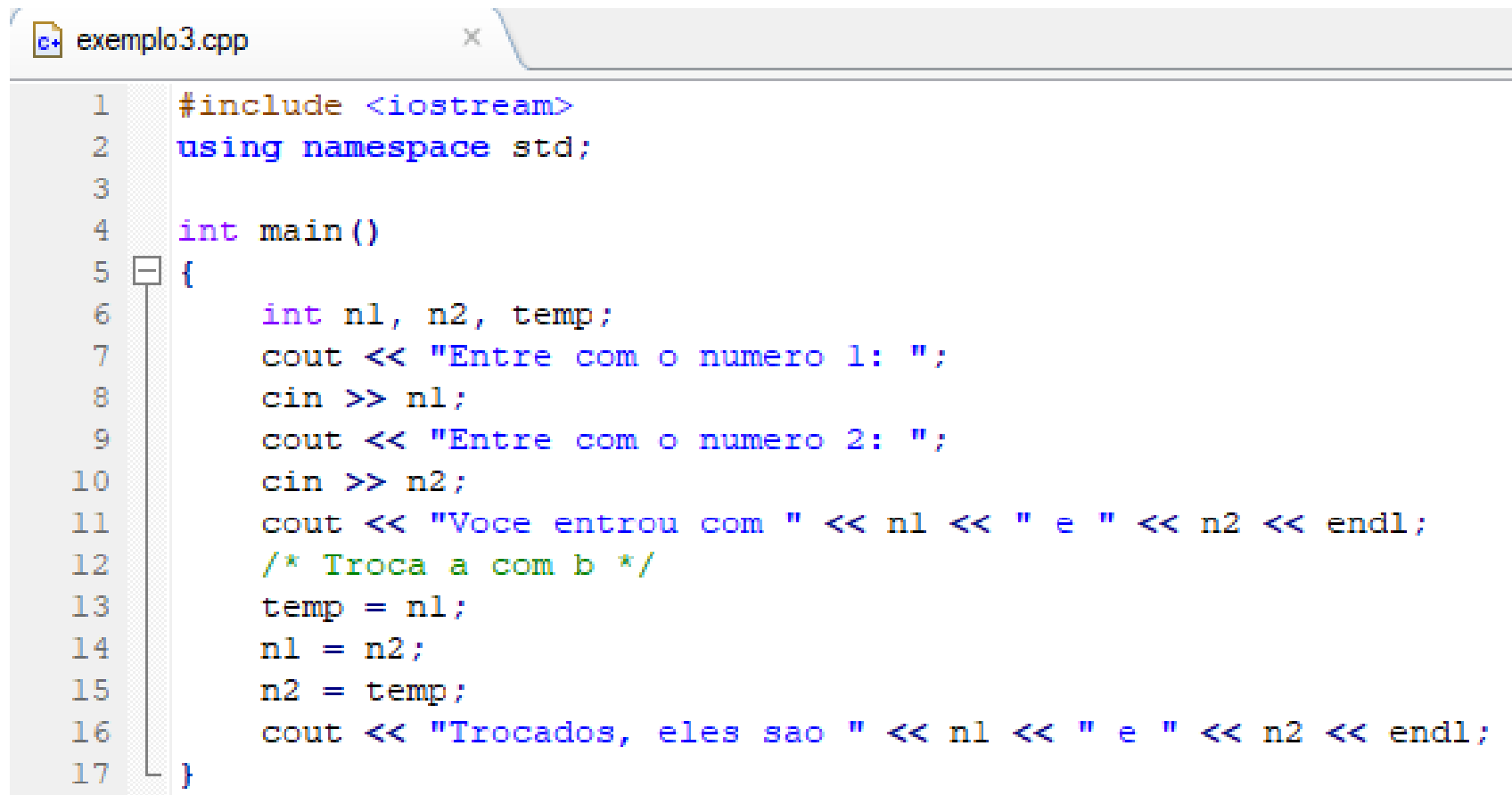
```
14  int main()
15  {
16      setlocale(LC_ALL, "Portuguese");
17      int s1, s2, s3;
18      cout << "Entre três inteiros: ";
19      cin >> s1 >> s2 >> s3;
20      // testar com os valores 3, 4 e 5
21      if ( s1 > 0 && s2 > 0 && s3 > 0 &&
22          (quadrado(s1) + quadrado(s2) == quadrado(s3)
23           || quadrado(s2) + quadrado(s3) == quadrado(s1)
24           || quadrado(s3) + quadrado(s1) == quadrado(s2)) )
25      {
26          cout << " " << s1 << " " << s2 << " " << s3 << " podem formar um triângulo reto\n";
27      }
28      else
29      {
30          cout << " " << s1 << " " << s2 << " " << s3 << " não podem formar um triângulo reto\n";
31      }
32  }
```

# Funções - exemplo2.2.cpp

```
exemplo2.2.cpp x
1  #include <iostream>
2  using namespace std;
3
4  int obtem_valor()
5  {
6      int valor;
7      cout << "Entre um valor: ";
8      cin >> valor;
9      return valor;
10 }
11
12 int main()
13 {
14     int a, b;
15     a = obtem_valor();
16     b = obtem_valor();
17     cout << "soma = " << a + b << endl;
18
19     return 0;
20 }
```

# Funções - exemplo3.cpp

Considere o programa abaixo que pede ao usuário dois inteiros, armazena-os em duas variáveis, troca seus valores, e os imprime.



```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n1, n2, temp;
7      cout << "Entre com o numero 1: ";
8      cin >> n1;
9      cout << "Entre com o numero 2: ";
10     cin >> n2;
11     cout << "Voce entrou com " << n1 << " e " << n2 << endl;
12     /* Troca a com b */
13     temp = n1;
14     n1 = n2;
15     n2 = temp;
16     cout << "Trocados, eles sao " << n1 << " e " << n2 << endl;
17 }
```

# exemplo3.1.cpp

É possível escrever uma **função** que executa esta operação de troca?

Faça você mesmo?

# exemplo3.1.cpp

- Como você já se viu nos exemplos anteriores, em C++ os argumentos são passados por valor.
- Uma vez que somente os valores das variáveis são passados, não é possível para a função **troca()** alterar os valores de a e b porque **troca()** não sabe onde está na memória estas variáveis armazenadas.
- Além disso, **troca()** não poderia ser escrito usando a sentença return porque podemos retornar **APENAS UM** valor (não dois) através da sentença return.

# Argumentos passados por referência

## exemplo3.2.cpp

- A solução para o problema acima é ao invés de passar os valores de n1 e n2, passar uma referência às variáveis n1 e n2.
- Desta forma, **troca()** saberia que endereço de memória escrever, portanto poderia alterar os valores de n1 e n2.

# Argumentos passados por referência

## exemplo3.2.cpp

```
*exemplo3.2.cpp
1  #include <iostream>
2  using namespace std;
3
4  void troca(int & px, int & py)
5  {
6      int temp;
7      temp = px;
8      px = py;
9      py = temp;
10 }
11
12
13 int main()
14 {
15     int n1, n2;
16     cout << "Entre com o numero 1: ";
17     cin >> n1;
18     cout << "Entre com o numero 2: ";
19     cin >> n2;
20     cout << "Voce entrou com " << n1 << " e " << n2 << endl;
21     // Troca a com b -- passa argumentos por referencia
22     troca(n1, n2);
23     cout << "Trocados, eles sao " << n1 << " e " << n2 << endl;
24 }
```



# Argumentos passados por referência

- Quando n1 e n2 são passados como argumentos para troca(), na verdade, somente seus valores são passados.
- A função não podia alterar os valores de n1 e n2 porque ela não conhece os endereços de n1 e n2.
- Mas se referências para n1 e n2 forem passados como argumentos ao invés de n1 e n2, a função troca() seria capaz de alterar seus valores; ela saberia então em que endereço de memória escrever.
- Na verdade, a função não sabe que os endereços de memória são associados com n1 e n2 , mas ela pode modificar o conteúdo destes endereços.
- Portanto, passando uma variável por referência (ao invés do valor da variável), habilitamos a função a alterar o conteúdo destas variáveis na função chamadora.

# Outro de passagem por referência

## exemplo4.cpp

```
*exemplo4.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  void altera(int & n1, int & n2)
6  {
7      n1 = 100;
8      n2 = 200;
9  }
10
11
12 int main()
13 {
14     int n1 = 0, n2 = 0;
15
16     cout << "Digite um numero: " << endl;
17     cin >> n1;
18     cout << "Digite outro numero: " << endl;
19     cin >> n2;
20
21     cout << "Primeiro numero: " << n1 << endl;
22     cout << "Segundo numero: " << n2 << endl;
23
24     altera(n1, n2);
25     cout << "-----" << endl;
26     cout << "Primeiro numero alterado: " << n1 << endl;
27     cout << "Segundo numero alterado: " << n2 << endl;
28
29     return 0;
30
31 }
```

# Ponteiros

- A memória RAM de um computador é um conjunto de posições adjacentes.
- De uma maneira simplista podemos dizer que ela é um grande vetor e seu índice é formado pelos endereços individuais de memória.
- Os ponteiros em C++ são uma das características mais importantes da linguagem e permitem manipular diretamente a memória do computador.
- Com ponteiros, é possível criar estruturas de dados complexas e executar operações de baixo nível que não seriam possíveis com outros tipos de variáveis.

# Ponteiros

- Em C++, um ponteiro é uma variável que armazena o endereço de memória de outra variável.
- Um ponteiro é declarado usando o operador de asterisco \*, que é colocado antes do nome da variável. Por exemplo:

```
int* ptr;
```

- Nesse exemplo, ptr é um ponteiro para um valor inteiro.
- Ele não armazena o valor inteiro em si, mas sim o endereço de memória onde o valor está armazenado.

# Ponteiros

- Para obter o endereço de memória de uma variável, use o operador de endereço &, que é colocado antes do nome da variável. Por exemplo:

```
int x = 42;
```

```
int* ptr = &x;
```

- Nesse exemplo, &x retorna o endereço de memória da variável x e o valor desse endereço é armazenado no ponteiro ptr.

# Ponteiros

- Para acessar o valor armazenado em um endereço de memória apontado por um ponteiro, use o operador de referência \*, que é colocado antes do nome do ponteiro. Por exemplo:

```
int y = *ptr;
```

- Nesse exemplo, \*ptr retorna o valor armazenado no endereço de memória apontado pelo ponteiro ptr e esse valor é armazenado na variável y.

# Ponteiros

- Os ponteiros também podem ser utilizados para criar e manipular estruturas de dados complexas, como listas encadeadas, árvores binárias, grafos, entre outras.
- No entanto, é importante tomar cuidado ao manipular ponteiros, pois eles podem facilmente causar erros de segmentação (também conhecidos como "segfaults"), que ocorrem quando um programa tenta acessar uma área inválida da memória.

# Como uma variável é armazenada?

- Imagine que temos este comando em C++:

`char ch;`

- O compilador determina um endereço de memória para a variável.

Endereço	Variável	Conteúdo
1000		
1001		
1002	ch	
1003		
1004		
1005		



# Como uma variável é armazenada?

- Quando fazemos:

`ch = 'A';`

- estamos dizendo ao compilador para colocar o 'A' na posição de memória de ch.

Endereço	Variável	Conteúdo
1000		
1001		
1002	ch	A
1003		
1004		
1005		

# Como uma variável é armazenada?

- Imagine agora que nós criássemos um ponteiro ptr.
- Como qualquer outra variável ele ocuparia espaço.

Endereço	Variável	Conteúdo
1000	ptr	
1001		
1002	ch	A
1003		
1004		
1005		

# Como uma variável é armazenada?

- Se rodássemos o seguinte comando:

`ptr = &ch;`

- Temos o seguinte resultado e podemos dizer que ptr aponta para ch.

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	ch	A
1003		
1004		
1005		



# Declaração de ponteiros

- Devemos ter em mente que um ponteiro é uma variável como outra qualquer, e, por isso, deve ser declarado.

- Para declarar um ponteiro usamos:

`tipo *ponteiro;`

- Ex

`char a, b, *ptr , c, *x;`

`int v1, *p1;`

# Inicialização de ponteiros

- Todo ponteiro deve ser inicializado, como qualquer variável.
- Um ponteiro zerado nunca pode ser usado, mas podemos inicializa-lo.
- Existe uma constante em C++ que você pode utilizar para inicializar um ponteiro: NULL
- Ela nada mais é que um “apelido” para o número zero.

# Inicialização de ponteiros

- Vejamos o código:

```
int a=5, b=7;
```

```
int *ptr = NULL;
```

Endereço	Variável	Conteúdo
1000	ptr	NULL
1001		
1002	a	5
1003	b	7
1004		
1005		

# Inicialização de ponteiros

- Se agora usarmos:

`ptr = &a;`

- Agora temos:  $a \rightarrow 5$   $ptr \rightarrow 1002$   $*ptr \rightarrow 5$

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	a	5
1003	b	7
1004		
1005		



# Exemplo1-ponteiro.cpp

```
exemplo1.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv)
6  {
7      int var = 10; // declaração de variável padrão
8      int *pvar; // declaração de ponteiro
9      //sempre inicializar o ponteiro antes de utiliza-lo
10     pvar = &var; // &var --> leia como o endereço da variável var
11     *pvar = 20; // *pvar muda o valor de var, porque o ponteiro tem o seu endereço
12     cout << *pvar << endl; // *pvar mostra o conteúdo da variável apontada pelo ponteiro
13     cout << var << endl;
14     cout << &*pvar << endl;
15     return 0;
16 }
```

```
C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dado...
20
20
0x67fef8

O Processo retornou 0   tempo de execução : 0.077 s
Pressione uma tecla para continuar...
```



# Exemplo2-ponteiro.cpp

exemplo1.1.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv)
6  {
7      short usVar = 200;
8      long ulVar = 300;
9      int iVar = 400;
10     cout << "*** Valores e enderecos ***" << endl;
11     cout << "usVar: Valor = " << usVar << ", Endereco = " << &usVar << endl;
12     cout << "ulVar: Valor = " << ulVar << ", Endereco = " << &ulVar << endl;
13     cout << "iVar: Valor = " << iVar << ", Endereco = " << &iVar << endl;
14     return 0;
15 }
```

C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dados\exemplos\_Módulo\_3\

```
*** Valores e enderecos ***
usVar: Valor = 200, Endereco = 0x67fee4
ulVar: Valor = 300, Endereco = 0x67fee8
iVar: Valor = 400, Endereco = 0x67fee4
```

```
O Processo retornou 0   tempo de execução : 0.124 s
Pressione uma tecla para continuar...
```

# Exemplo3-ponteiro.cpp

```
exemplo2.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  //escopo global
6  int var = 0;
7
8  void passagemPorValor(int var){
9      var = 20;
10 }
11
12 int main(int argc, char** argv)
13 {
14     var = 10;
15     int *pvar;
16
17     pvar = &var;
18
19     passagemPorValor(var);
20
21     cout << var << endl;
22     return 0;
23 }
```

10

0 Processo retornou 0 tempo de execução : 0.061 s  
Pressione uma tecla para continuar...

# Exemplo4-ponteiro.cpp

```
exemplo3.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5
6  int var = 0;
7
8  void passagemPorReferencia(int* n){ // *n aponta para o endereço de pvar
9      *n = 20;
10 }
11
12 int main(int argc, char** argv)
13 {
14     var = 10;
15
16     int *pvar;
17     pvar = &var;
18
19     passagemPorReferencia(pvar);
20
21     cout << var << endl;
22     return 0;
23 }
```

20

O Processo retornou 0 tempo de execução : 0.085 s  
Pressione uma tecla para continuar...

# Exemplo5-ponteiro.cpp

```
*exemplo4.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  //exemplo de ponteiro de array
6
7  int main(int argc, char** argv)
8  {
9      int array[] = {1,2,3,4,5};
10     int* pArray = &array[0];
11
12     cout << *pArray << endl;
13     cout << endl;
14
15     for(int i=0;i<5;i++){
16         cout << *pArray << endl;
17         pArray++;
18     }
19
20     return 0;
21 }
```

```
C:\Users\roam\Google Drive\Unifesi\Estrutura de Da...
1
1
2
3
4
5
0 Processo retornou 0   tempo de execucao : 0.094 s
Pressione uma tecla para continuar...
```

# Exercícios

1. Indique verdadeiro ou falso
  - a) ( ) O operador & permite-nos obter o endereço de uma variável. Permite também obter o endereço de um ponteiro.
  - b) ( ) Se x é um inteiro e ptr um ponteiro para inteiros e ambos contêm no seu interior o número 100, então x+1 e ptr+1 apresentarão o número 101.
  - c) ( ) O operador \* nos permite obter o endereço de uma variável.
  - d) ( ) Os ponteiros são variáveis que apontam para endereços na memória.

# Exercícios

2. Qual o resultado?

```
cout << a << b << *ptr;
```

3. Se fizermos `ptr = &b`, qual o resultado?

```
cout << a << b << *ptr;
```

4. Se agora tivermos `*ptr = 20`, qual o resultado?

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	a	5
1003	b	7
1004		

# Exercícios

5. Qual caractere que se coloca na declaração de uma variável para indicar que ela é um ponteiro? Onde se coloca este caractere?
6. O que contém uma variável do tipo ponteiro?
7. Faça um programa em C++ que crie um vetor de 10 inteiros, coloque peça valores ao usuário e depois imprima todos os seus conteúdos na ordem normal e depois inversa. A impressão dos conteúdos deverá ser feita usando ponteiro.

# Exercícios

8. Escreva um programa em C++ que crie um array de 5 inteiros, preencha-o com valores fornecidos pelo usuário e, em seguida, calcule a média dos valores usando ponteiros.

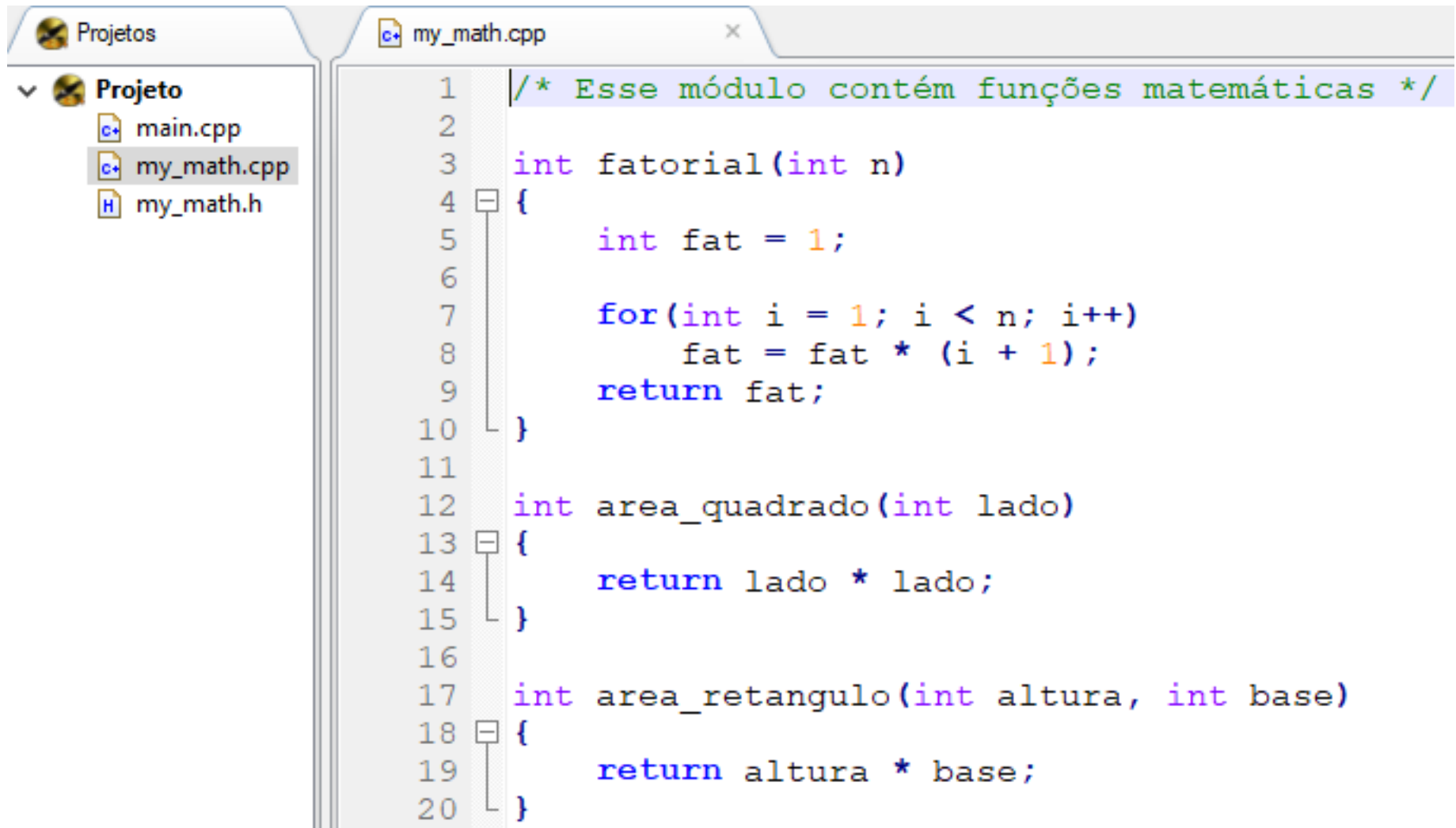
Dica: para calcular a média, some todos os valores do array e divida o resultado pelo número de elementos no array.



# Módulos

- Os módulos são uma funcionalidade importante em C++ desde a versão 20 da linguagem, que permitem uma nova forma de organizar e compartilhar código em projetos grandes.
- Antes dos módulos, a organização do código em arquivos de cabeçalho e arquivos de implementação podia ser um pouco confusa, com problemas de conflitos de definições e dependências circulares.
- Com os módulos em C++, é possível agrupar as definições e implementações de um conjunto de funcionalidades em um único módulo, que pode ser importado em outros módulos que dependem dessas funcionalidades.
- Dessa forma, é possível evitar as duplicações de código e as dependências circulares que dificultam a manutenção e evolução de projetos grandes.

# Módulos

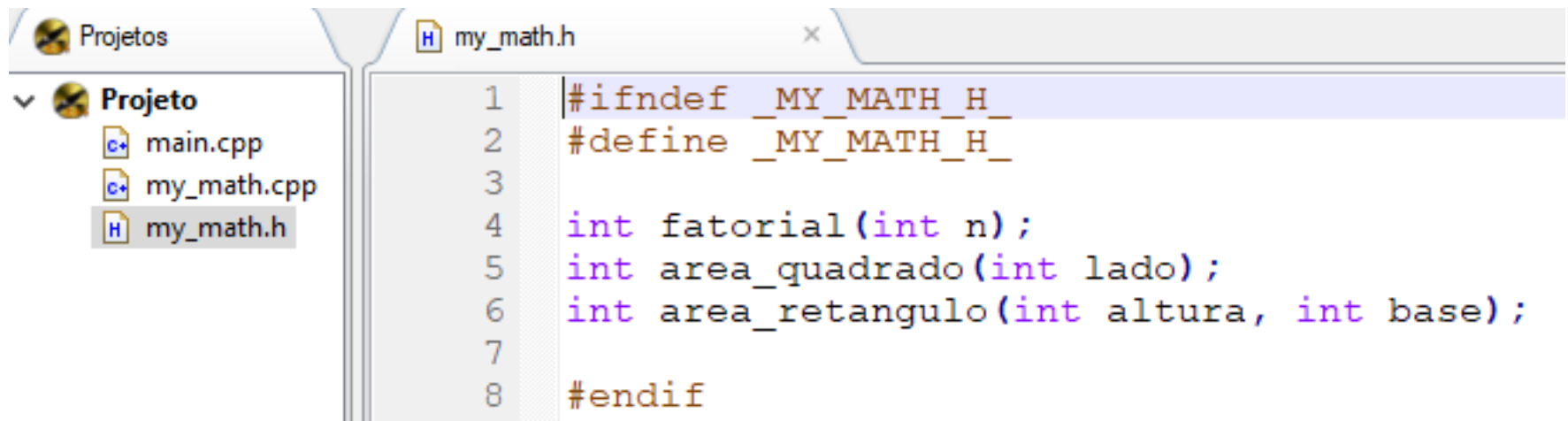


```
1  /* Esse módulo contém funções matemáticas */
2
3  int fatorial(int n)
4  {
5      int fat = 1;
6
7      for(int i = 1; i < n; i++)
8          fat = fat * (i + 1);
9      return fat;
10 }
11
12 int area_quadrado(int lado)
13 {
14     return lado * lado;
15 }
16
17 int area_retangulo(int altura, int base)
18 {
19     return altura * base;
20 }
```

# Módulos

```
main.cpp x
1  #include <iostream>
2  #include "my_math.h"
3
4  using namespace std;
5
6  int main(int argc, char *argv[])
7  {
8      int n = 0;
9
10     cout << "Digite o numero para calcular: ";
11     cin >> n;
12
13     cout << "Fatorial de " << n << ": " << fatorial(n) << endl;
14     cout << "Quadrado com lado " << n << ": " << area_quadrado(n) << endl;
15     cout << "Area retangulo " << area_retangulo(n, n) << endl;
16     return 0;
17 }
18
```

# Módulos



The image shows a code editor interface. On the left, a sidebar displays a project named 'Projeto' with three files: 'main.cpp', 'my\_math.cpp', and 'my\_math.h'. The 'my\_math.h' file is selected and open in the main editor window. The code in 'my\_math.h' is as follows:

```
1 #ifndef _MY_MATH_H_
2 #define _MY_MATH_H_
3
4 int fatorial(int n);
5 int area_quadrado(int lado);
6 int area_retangulo(int altura, int base);
7
8 #endif
```

# Módulos

- O arquivo .h em C++ é um arquivo de cabeçalho que contém definições e declarações de funções, classes, variáveis e outros elementos que serão utilizados em outras partes do programa.
- Os arquivos de cabeçalho são incluídos em outros arquivos de código-fonte (geralmente com extensão .cpp) utilizando a diretiva de pré-processador `#include`, que informa ao compilador que as definições e declarações contidas no arquivo de cabeçalho devem ser incluídas no arquivo de código-fonte durante a compilação.

# Referência desta aula

- Notas de Aula do Prof. Prof. Armando Luiz N. Delgado baseado em revisão sobre material de Prof.a Carmem Hara e Prof. Wagner Zola.
- VELOSO, P. et Alli. Estruturas de Dados. Ed. Campus, 1986.
- <http://www.cplusplus.com/reference/>

Obrigado

Rodrigo