

Estruturas de Dados

Prof. Rodrigo Martins

rodrigo.martins@francomontoro.com.br

Cronograma da Aula

- Funções
- Módulos
- Escopo de Variável
- Vetores ou Arrays
- Matrizes ou Arrays Multidimensionais
- Exemplos e Exercícios

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;  
}
```

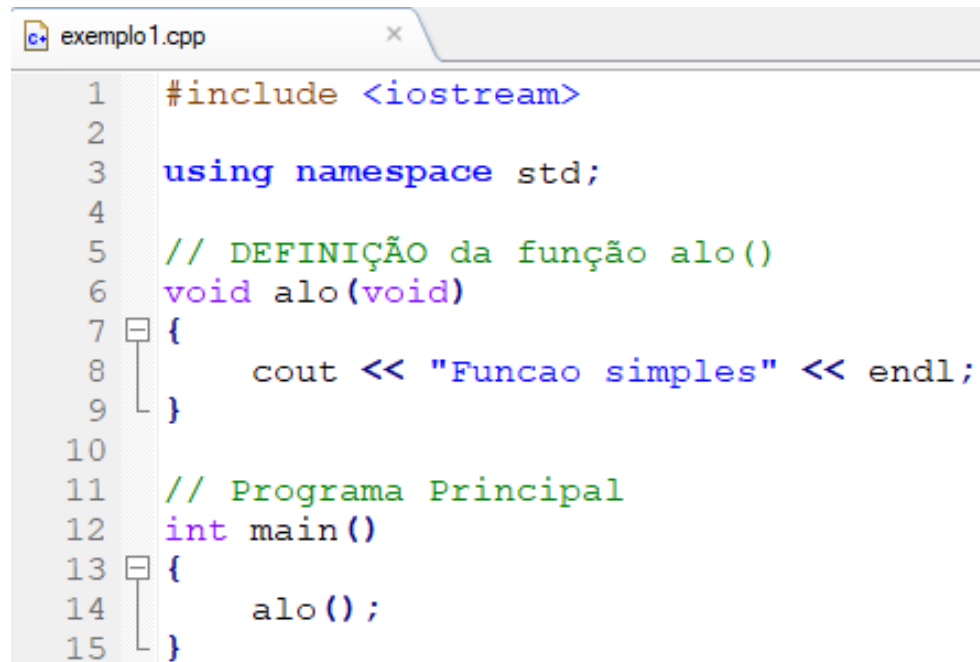
- Para chamar uma função em C++, basta escrever o nome da função seguido dos argumentos entre parênteses. Por exemplo:

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

Funções Simples – 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 que não retornam nada (void)

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 int, float e char.

Funções que retornam um valor

exemplo2.1.cpp

```
exemplo2.1.cpp x
1 // programa que verifica se 3 numeros podem ser os lados de um
2 // triangulo reto.
3
4 #include <iostream>
5 #include <locale.h>
6 using namespace std;
7
8 // funcao que calcula o quadrado de um numero
9
10 int quadrado(int n)
11 {
12     return n * n;
13 }
14
15 int main()
16 {
17     setlocale(LC_ALL, "Portuguese");
18     int s1, s2, s3;
19     cout << "Entre tres inteiros: ";
20     cin >> s1 >> s2 >> s3;
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 triangulo reto\n";
27     }
28     else
29     {
30         cout << " " << s1 << " " << s2 << " " << s3 << " nao podem formar um triangulo reto\n";
31     }
32 }
```

Funções que retornam um valor

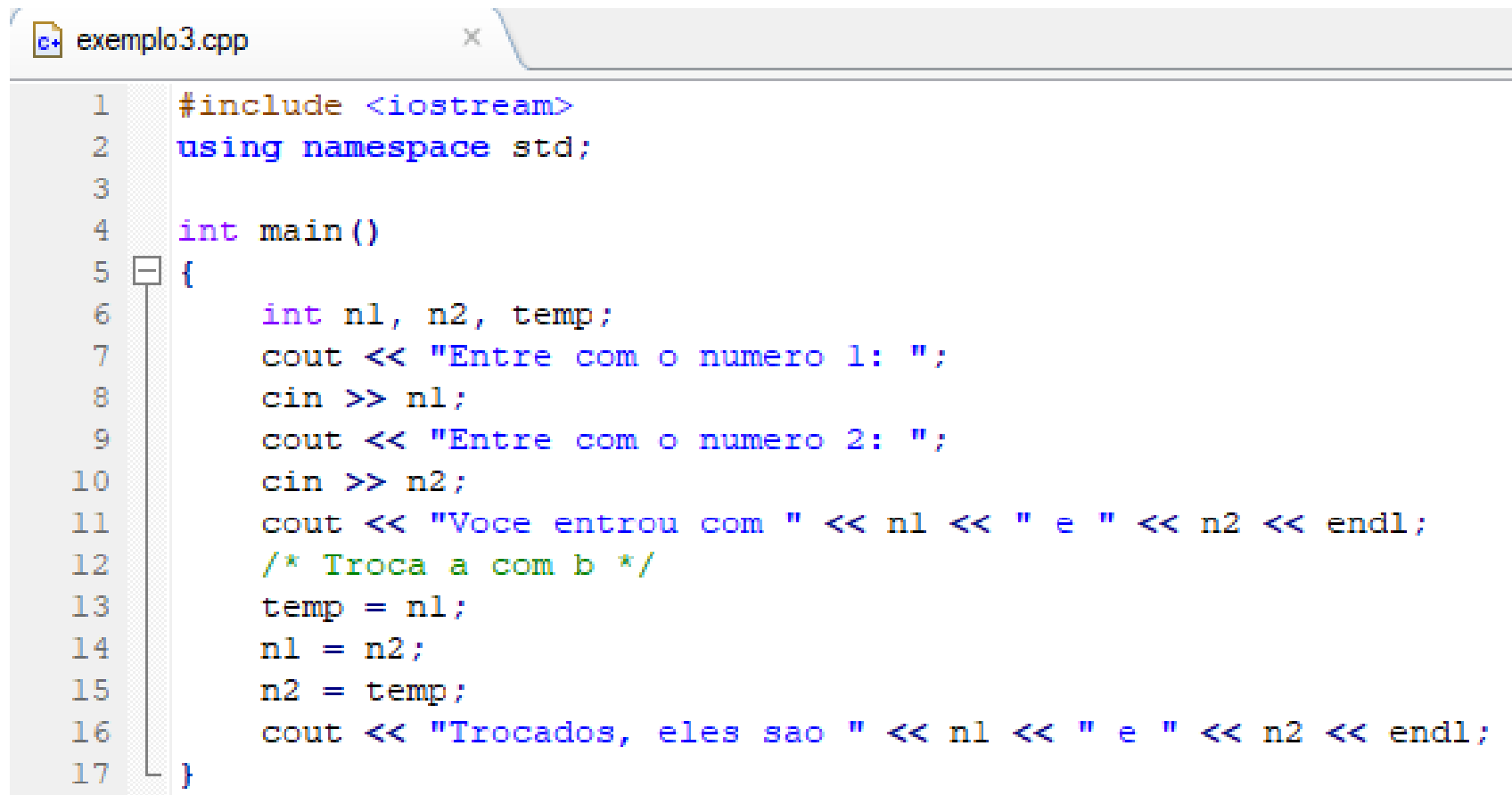
exemplo2.2.cpp

```
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 }
```

Mais sobre 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?

Quando return não é suficiente

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.

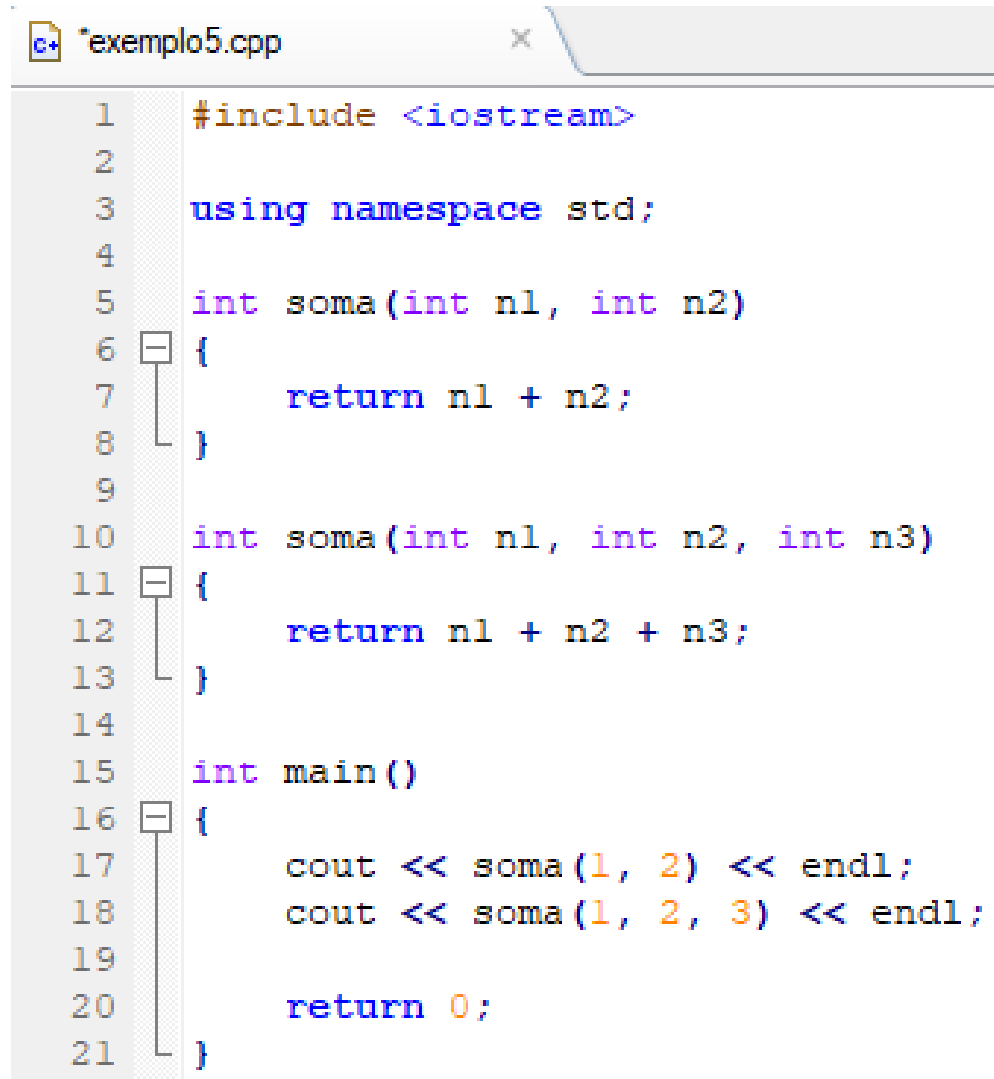
Outro de passagem por referência

exemplo4.cpp

```
*exemplo4.cpp
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 }
```

Sobrecarga de nomes de funções

exemplo5.cpp



```
*exemplo5.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int soma(int n1, int n2)
6  {
7      return n1 + n2;
8  }
9
10 int soma(int n1, int n2, int n3)
11 {
12     return n1 + n2 + n3;
13 }
14
15 int main()
16 {
17     cout << soma(1, 2) << endl;
18     cout << soma(1, 2, 3) << endl;
19
20     return 0;
21 }
```

Sobrecarga de nomes de funções

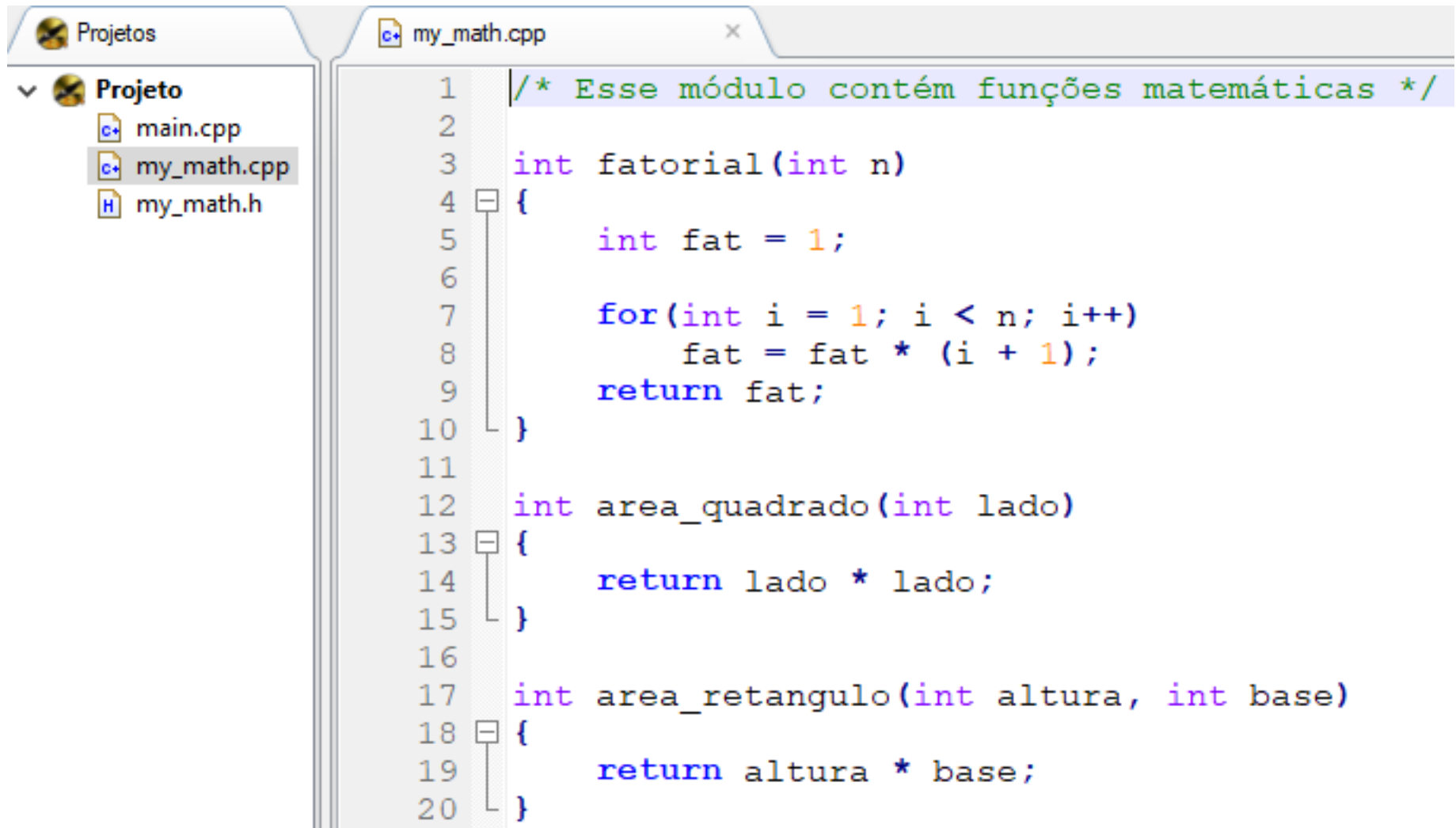
exemplo5.1.cpp

```
*exemplo5.1.cpp
1  #include <iostream>
2  #include <locale.h>
3
4  using namespace std;
5
6  void mensagem(int n)
7  {
8      cout << "numero: " << n << endl;
9  }
10
11 void mensagem()
12 {
13     cout << "Exemplo de Sobrecarga de Função" << endl;
14 }
15
16 int main()
17 {
18     setlocale(LC_ALL, "Portuguese");
19     mensagem();
20     mensagem(10);
21
22     return 0;
23 }
```

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

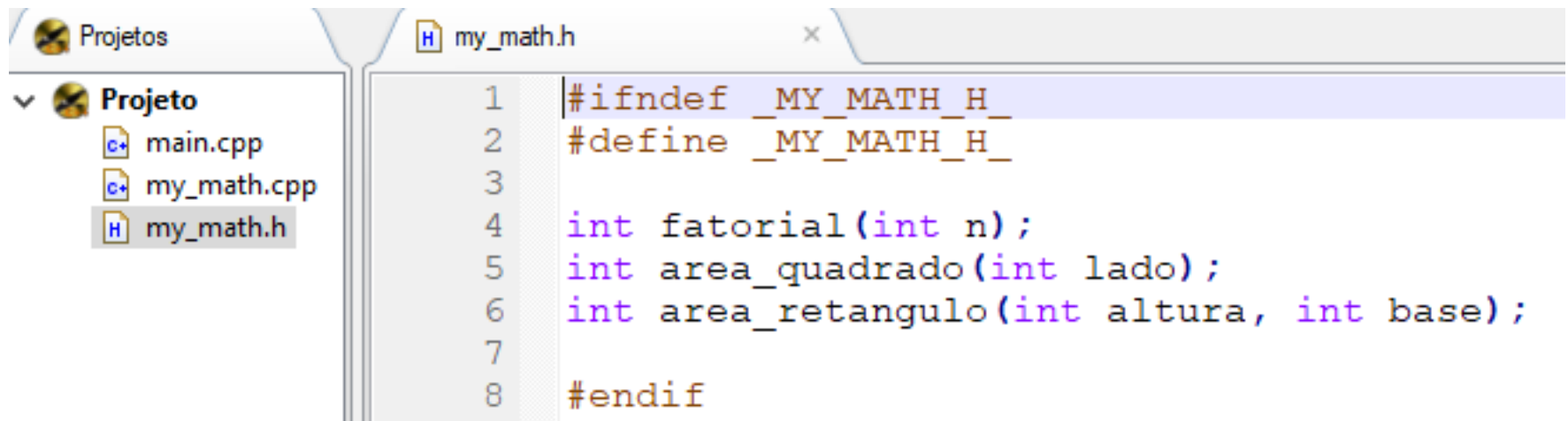


```
1  /* Esse módulo contém funções matemáticas */
2
3  int fatorial(int n)
4  {
5      int fat = 1;
6
7      for(int i = 1; i < n; i++)
8          fat = fat * (i + 1);
9      return fat;
10 }
11
12 int area_quadrado(int lado)
13 {
14     return lado * lado;
15 }
16
17 int area_retangulo(int altura, 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 editor shows the following C++ header file code:

```
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.

Escopo de variável

- Podemos dividir as variáveis quanto ao escopo em dois tipos:
 - variáveis locais e variáveis globais.
- **Variáveis locais**
 - São aquelas declaradas dentro do bloco de uma função.
 - Não podem ser usadas ou modificadas por outras funções.
 - Somente existem enquanto a função onde foi declarada estiver sendo executada.

Escopo de variável

- **Variáveis Globais**
 - São declaradas fora de todos os blocos de funções.
 - São acessíveis em qualquer parte do programa, ou seja, podem ser usadas e modificadas por todas as- outras funções.
 - Existem durante toda a execução do programa.

exemploEscopo.cpp

```
*exemploEscopo.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  // variável com escopo global
6  int num_global = 12;
7
8  void escopo()
9  {
10     //variável com escopo local
11     int num = 10;
12     static int num_static = 1;
13     cout << "variavel local: " << num << endl;
14     cout << "variavel global: " << num_global << endl;
15     num_static++;
16     cout << "variavel estatica: " << num_static << endl;
17 }
18
19 int main(int argc, char *argv[])
20 {
21     escopo();
22     escopo();
23     escopo();
24     return 0;
25 }
```

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.

Vetores ou Arrays – exemplo6.cpp

```
*exemplo6.cpp
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;
```

Vetores ou Arrays – exemplo6.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
```

Vetores ou Arrays – exemplo7.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 }
```


Vetores ou Arrays – exemplo7.cpp

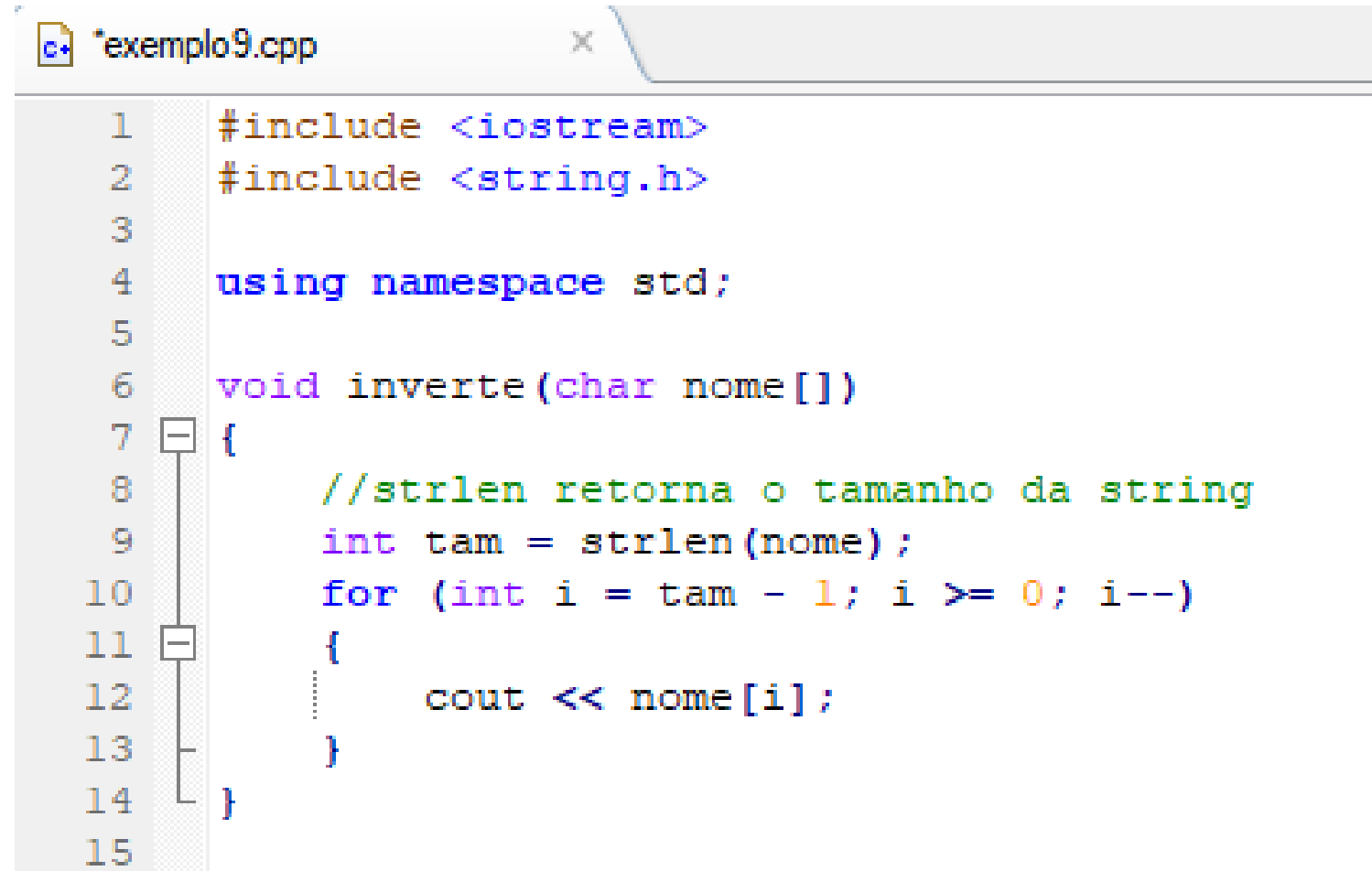
```
28     media = mediaValoresVetor(vet, NOTAS);
29     cout << "Media: " << media << endl;
30
31     for (int i = 0; i < NOTAS; i++)
32     {
33         if (vet[i] > media)
34         {
35             acima++;
36         }
37     }
38     cout << "Valores acima da media: " << acima << endl;
39    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

Vetores de caracteres

exemplo8.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 }
```

Vetores ou Arrays – exemplo9.cpp

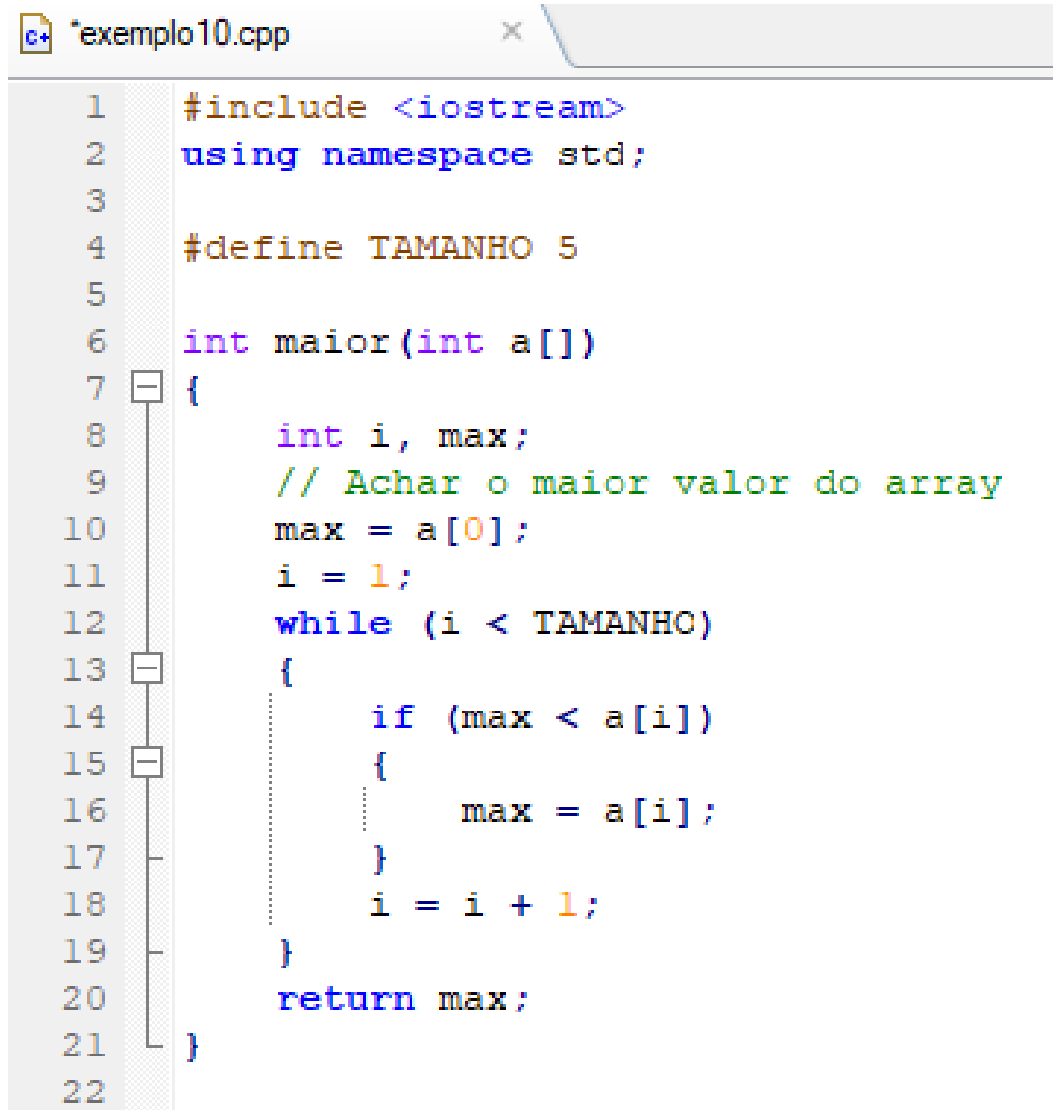


```
1  #include <iostream>
2  #include <string.h>
3
4  using namespace std;
5
6  void inverte(char nome[])
7  {
8      //strlen retorna o tamanho da string
9      int tam = strlen(nome);
10     for (int i = tam - 1; i >= 0; i--)
11     {
12         cout << nome[i];
13     }
14 }
15
```

Vetores ou Arrays – exemplo9.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  }
```

Vetores ou Arrays – exemplo10.cpp



```
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
```

Vetores ou Arrays – exemplo10.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).

Matrizes ou Arrays Multidimensionais

exemplo11.cpp

```
*exemplo11.cpp
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;
```


Matrizes ou Arrays Multidimensionais

exemplo11.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

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