

Estruturas de Dados

Prof. Rodrigo Martins

rodrimartins2005@gmail.com



Cronograma da Aula

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

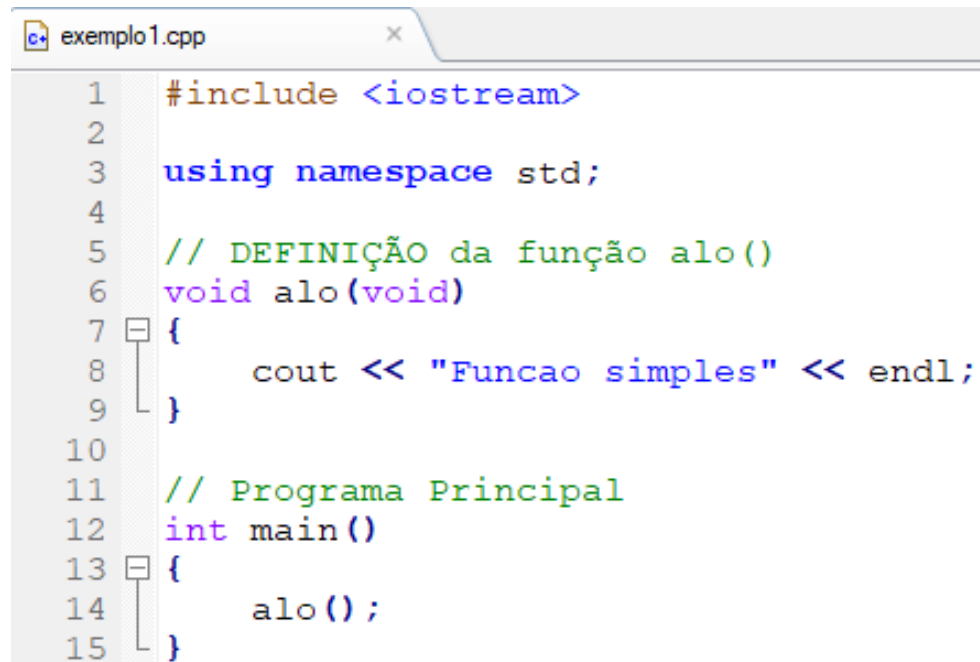
Funções

- Quando queremos resolver um problema, em geral tentamos dividi-lo em subproblemas mais simples e relativamente independentes, e resolvemos os problemas mais simples um a um.
- Uma função cria uma maneira conveniente de encapsular alguns detalhes de “processamento”, ou seja, como algum resultado é obtido. Quando esta “computação” é necessária, a função é chamada, ou invocada. Desta forma, quando uma função é chamada o usuário não precisa se preocupar como a computação é realizada.
- É importante saber o que a função faz (qual o resultado da execução de uma função) e também como se usa a função.
- Criando funções, um programa C++ pode ser estruturado em partes relativamente independentes que correspondem as subdivisões do problema.

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 }
```

Argumentos passados por valor

exemplo2.cpp

```
exemplo2.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  void cumprimenta(char inic1, char inic2); //protótipo da função
6  void cumprimenta2(string nome, string sobrenome);
7
8
9  int main()
10 {
11     char primeiro, segundo;
12     string nome, sobrenome;
13
14     cout << "Entre com duas iniciais (sem separacao): ";
15     cin >> primeiro >> segundo ;
16
17     cout << "Entre com nome e sobrenome: ";
18     cin >> nome >> sobrenome ;
19     cumprimenta(primeiro, segundo);
20     cumprimenta2(nome, sobrenome);
21 }
22
23 void cumprimenta2(string nome, string sobrenome)
24 {
25     cout << "Ola, " << nome << " " << sobrenome << "!" << endl;
26 }
27
28 void cumprimenta(char inic1, char inic2)
29 {
30     cout << "Ola, " << inic1 << inic2 << "!" << endl;
31 }
```

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 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
14 int main()
15 {
16     int s1, s2, s3;
17     cout << "Entre tres inteiros: ";
18     cin >> s1 >> s2 >> s3;
19     if ( s1 > 0 && s2 > 0 && s3 > 0 &&
20         (quadrado(s1) + quadrado(s2) == quadrado(s3)
21          || quadrado(s2) + quadrado(s3) == quadrado(s1)
22          || quadrado(s3) + quadrado(s1) == quadrado(s2)) )
23     {
24         cout << " " << s1 << " " << s2 << " " << s3 << " podem formar um triangulo reto\n";
25     }
26     else
27     {
28         cout << " " << s1 << " " << s2 << " " << s3 << " nao podem formar um triangulo reto\n";
29     }
30 }
```

Funções que retornam um valor

exemplo2.2.cpp

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


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.

```
exemplo3.cpp x
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a, b, temp;
7      cout << "Entre dois numeros: ";
8      cin >> a >> b;
9      cout << "Voce entrou com " << a << " e " << b << endl;
10     /* Troca a com b */
11     temp = a;
12     a = b;
13     b = temp;
14     cout << "Trocados, eles sao " << a << " e " << b << endl;
15 }
```

exemplo3.1.cpp

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

```
exemplo3.1.cpp x
1  #include <iostream>
2  using namespace std;
3
4  void troca(int x, int y)
5  {
6      int temp;
7      temp = x;
8      x = y;
9      y = temp;
10 }
11
12 int main()
13 {
14     int a, b;
15     cout << "Entre dois numeros: ";
16     cin >> a >> b;
17     cout << "Voce entrou com " << a << " e " << b << endl;
18
19     // Troca a com b
20     troca(a, b);
21     cout << "Trocados, eles sao " << a << " e " << b << endl;
22 }
```

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 a e b, passar uma referência às variáveis a e b. Desta forma, **troca()** saberia que endereço de memória escrever, portanto poderia alterar os valores de a e b.

Argumentos passados por referência

exemplo3.2.cpp

```
exemplo3.2.cpp x
1  #include <iostream>
2  using namespace std;
3
4  /* função troca(px, py)
5   * ação: troca os valores inteiros apontados por px e py
6   * entrada: apontadores px e py
7   * saída: valor de px e py trocados na origem da chamada da função
8   * suposições: px e py são apontadores válidos
9   * algoritmo: primeiro guarda o primeiro valor em um temporário e
10  * troca
11  */
12
13 void troca(int &px, int &py)
14 {
15     int temp;
16     temp = px;
17     px = py;
18     py = temp;
19 }
20
```

Argumentos passados por referência

exemplo3.2.cpp

```
21
22 int main()
23 {
24     int a, b;
25     cout << "Entre dois numeros: ";
26     cin >> a >> b;
27     cout << "Voce entrou com " << a << " e " << b << endl;
28     // Troca a com b -- passa argumentos por referencia
29     troca(a, b);
30     cout << "Trocados, eles sao " << a << " e " << b << endl;
31 }
```

Argumentos passados por referência

- Quando a e b são passados como argumentos para troca(), na verdade, somente seus valores são passados. A função não podia alterar os valores de a e b porque ela não conhece os endereços de a e b.
- Mas se referências para a e b forem passados como argumentos ao invés de a e b, 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 a e b, 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.

Funções que retornam um valor

exemplo4.cpp

```
*exemplo4.cpp
1  #include <iostream>
2  #include <locale.h>
3
4  using namespace std;
5
6  //protótipos
7  bool par(int num);
8  void mensagem();
9
10 int main(int argc, char *argv[])
11 {
12     setlocale(LC_ALL, "Portuguese");
13
14     int n=0;
15     mensagem();
16     cout << "Digite um numero: ";
17     cin >> n;
18     if(par(n))
19         cout << "O numero " << n << " eh par." << endl;
20     else
21         cout << "O numero " << n << " eh impar." << endl;
22     return 0;
23 }
```


Funções que retornam um valor

exemplo4.cpp

```
24
25 void mensagem()
26 {
27     cout << "Módulo 2 C++" << endl;
28 }
29
30 bool par(int num)
31 {
32     if(num % 2 == 0)
33         return true;
34     return false;
35 }
```

Sobrecarga de nomes de funções

exemplo5.cpp

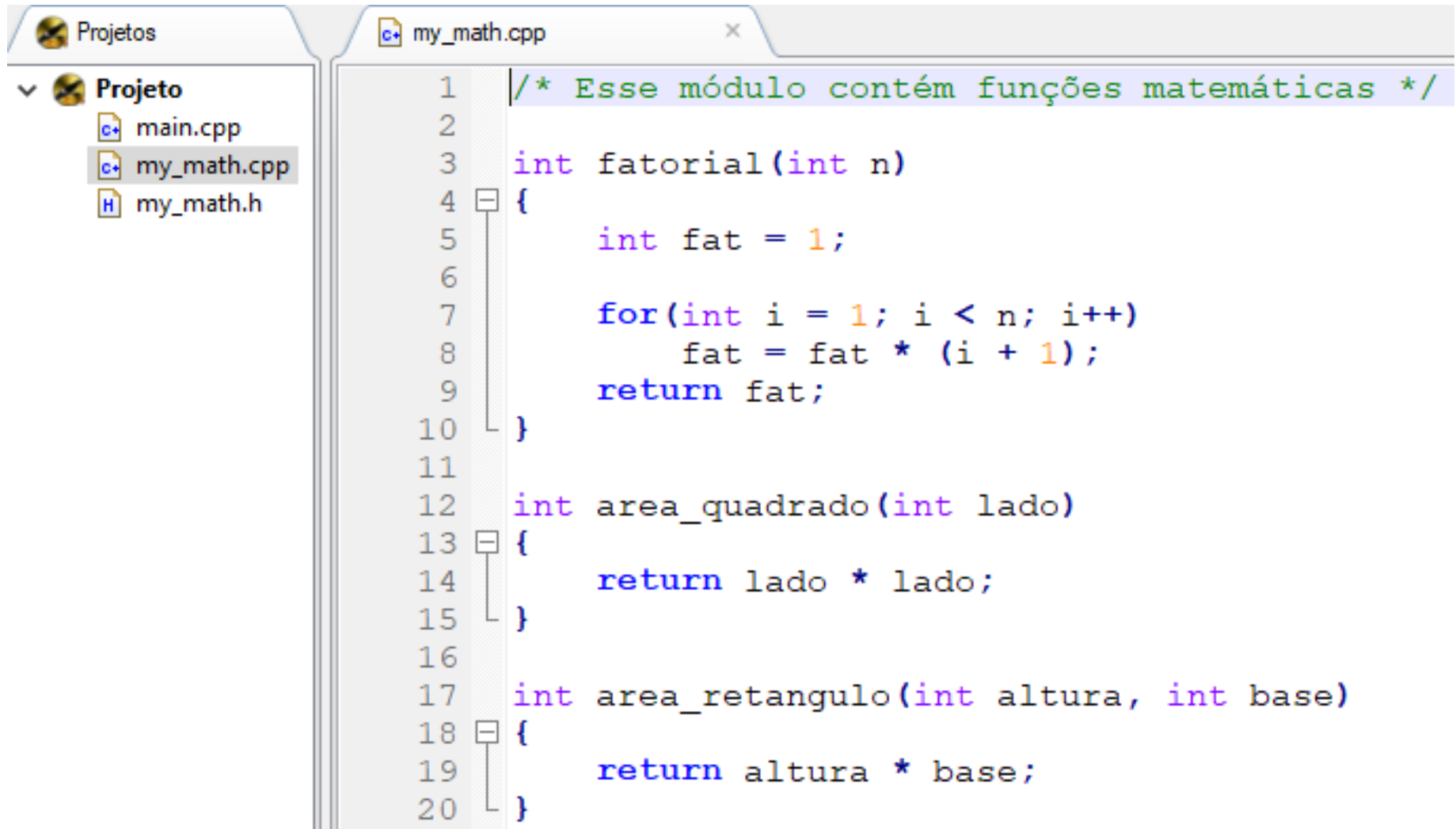
```
exemplo5.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  int soma(int n1, int n2)
6  {
7      return n1 + n2;
8  }
9
10 int soma(int n1, int n2, int n3)
11 {
12     return n1 + n2 + n3;
13 }
14
15 int main(int argc, char *argv[])
16 {
17     cout << soma(1, 2) << endl;
18     cout <<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

Sobrecarga de nomes de funções

exemplo5.1.cpp

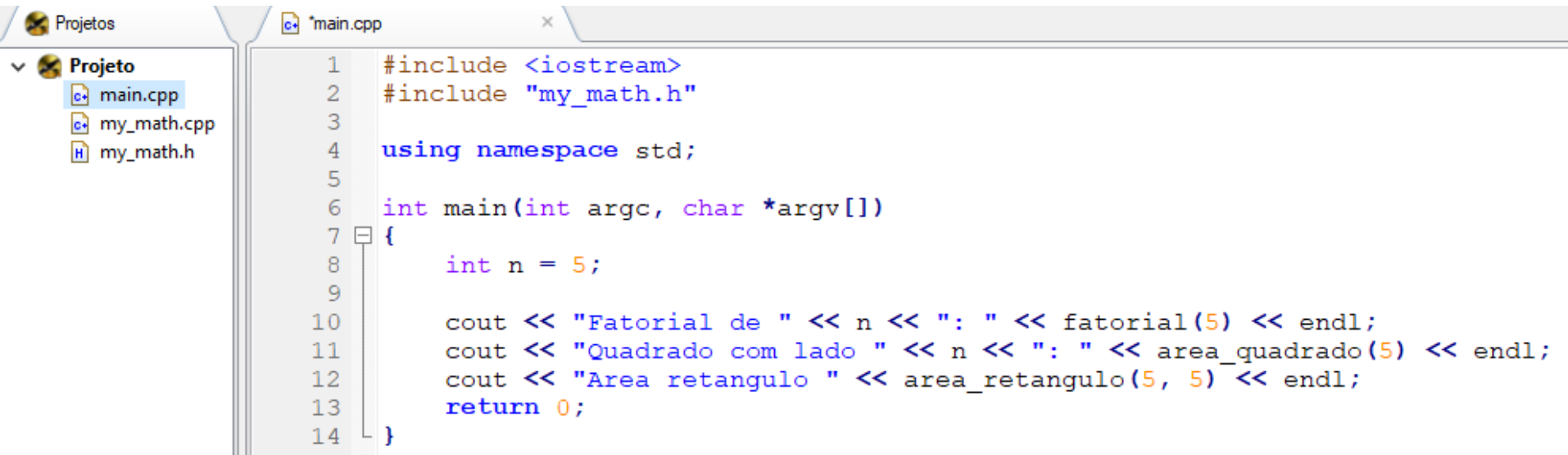
```
*exemplo5.1.cpp
1  #include <iostream>
2  #include <locale.h>
3
4  using namespace std;
5
6  void mensagem(int n){
7      cout << "numero: " << n << endl;
8  }
9
10 void mensagem(){
11     cout << "Exemplo de Sobrecarga de Função" << endl;
12 }
13
14 int main(int argc, char** argv)
15 {
16     setlocale(LC_ALL,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

Módulos



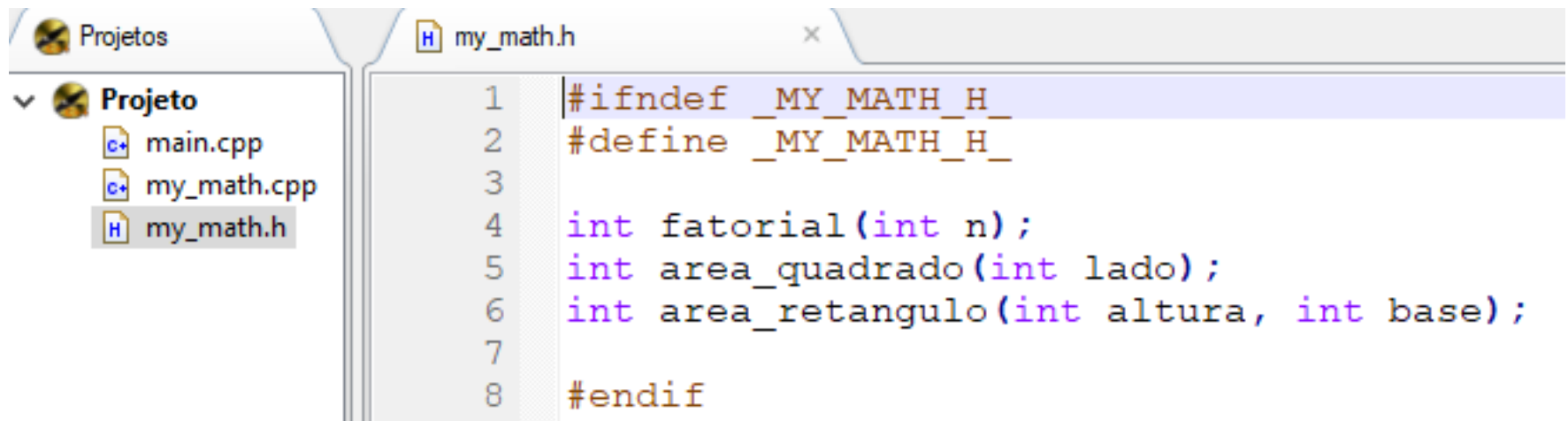
```
1  /* Esse módulo contém funções matemáticas */
2
3  int fatorial(int n)
4  {
5      int fat = 1;
6
7      for(int i = 1; i < n; i++)
8          fat = fat * (i + 1);
9      return fat;
10 }
11
12 int area_quadrado(int lado)
13 {
14     return lado * lado;
15 }
16
17 int area_retangulo(int altura,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

Módulos



```
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 = 5;
9
10    cout << "Fatorial de " << n << ": " << fatorial(5) << endl;
11    cout << "Quadrado com lado " << n << ": " << area_quadrado(5) << endl;
12    cout << "Area retangulo " << area_retangulo(5, 5) << endl;
13    return 0;
14 }
```

Módulos



```
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
```

Escopo de Variável

```
*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(int argc, char** argv)
7  {
8      int indice;
9      float total, nota[ESTUDANTES];
10     indice = 0;
11     //preenche o vetor
12     while (indice < ESTUDANTES)
13     {
14         cout << "Entre a nota do estudante " << indice + 1 << ": ";
15         cin >> nota[indice];
16         indice = indice + 1;
17     }
18
19     cout << "Notas: ";
20     total = 0;
21     indice = 0;
22
23     //imprime o vetor
24     while (indice < ESTUDANTES)
25     {
26         cout << nota[indice] << " ";
27         total = total + nota[indice];
28         indice = indice + 1;
29     }
30     cout << endl << "Media: " << total / ESTUDANTES << endl;
31     return 0;
32 }
```

Vetores ou Arrays – exemplo7.cpp

```
*exemplo7.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char *argv[])
6  {
7      char vetor[5] = {'1', '2', '3', '4', '5'};
8
9      cout << sizeof(vetor) << endl;
10     return 0;
11 }
```

Vetores de caracteres

exemplo8.cpp

exemplo8.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv)
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     //while (nome[i])
17     while (nome[i] != '\0')
18     {
19         cout << nome[i];
20         i++;
21     }
22     return 0;
23 }
```

Vetores ou Arrays – exemplo9.cpp

```
*exemplo9.cpp x
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
```

Vetores ou Arrays – exemplo9.cpp

```
16 int main(int argc, char** argv)
17 {
18     char nome[] = "rodrigo";
19     inverte(nome);
20     cout << endl;
21
22     //isalpha retorna true se caractere testado for alfabético
23     if (isalpha(nome[0]))
24     {
25         cout << "caractere alfabetico" << endl;
26     }
27     else
28     {
29         cout << "caractere numerico" << endl;
30     }
31
32     //isdigit retorna true se for um dígito
33     if (isdigit(nome[0]))
34     {
35         cout << "letra" << endl;
36     }
37     else
38     {
39         cout << "numero" << endl;
40     }
```

Vetores ou Arrays – exemplo9.cpp

```
41
42 //isupper retorna true se o caractere for maiusculo
43 if (isupper(nome[0]))
44 {
45     cout << "maiusculo" << endl;
46 }
47 else
48 {
49     cout << "minuscule" << endl;
50 }
51 return 0;
52 }
53
```

Vetores ou Arrays – exemplo10.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 }
```

Vetores ou Arrays – exemplo10.cpp

```
22
23  /* Programa principal */
24  int main()
25  {
26      int i, valor[TAMANHO];
27      i = 0;
28      while (i < TAMANHO)
29      {
30          cout << "Entre um inteiro: ";
31          cin >> valor[i];
32          i = i + 1;
33      }
34      cout << "O maior eh " << maior(valor) << endl;
35  }
```


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  #include <iomanip>
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         for (j=0;j<2;j++){
17             cout << "Digite um numero inteiro: ";
18             cin >> matriz[i][j];
19         }
20     }
21
22     //imprime a matriz na tela
23     for (i=0;i<2;i++){
24         for (j=0;j<2;j++){
25             cout << "O valor na posicao " << i << " " << j << " eh: "
26                 << matriz[i][j] << endl;
27         }
28     }
29
30     return 0;
31 }
```

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
- <http://www.cplusplus.com/reference/>

Obrigado

Rodrigo