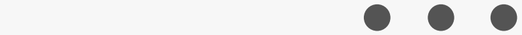
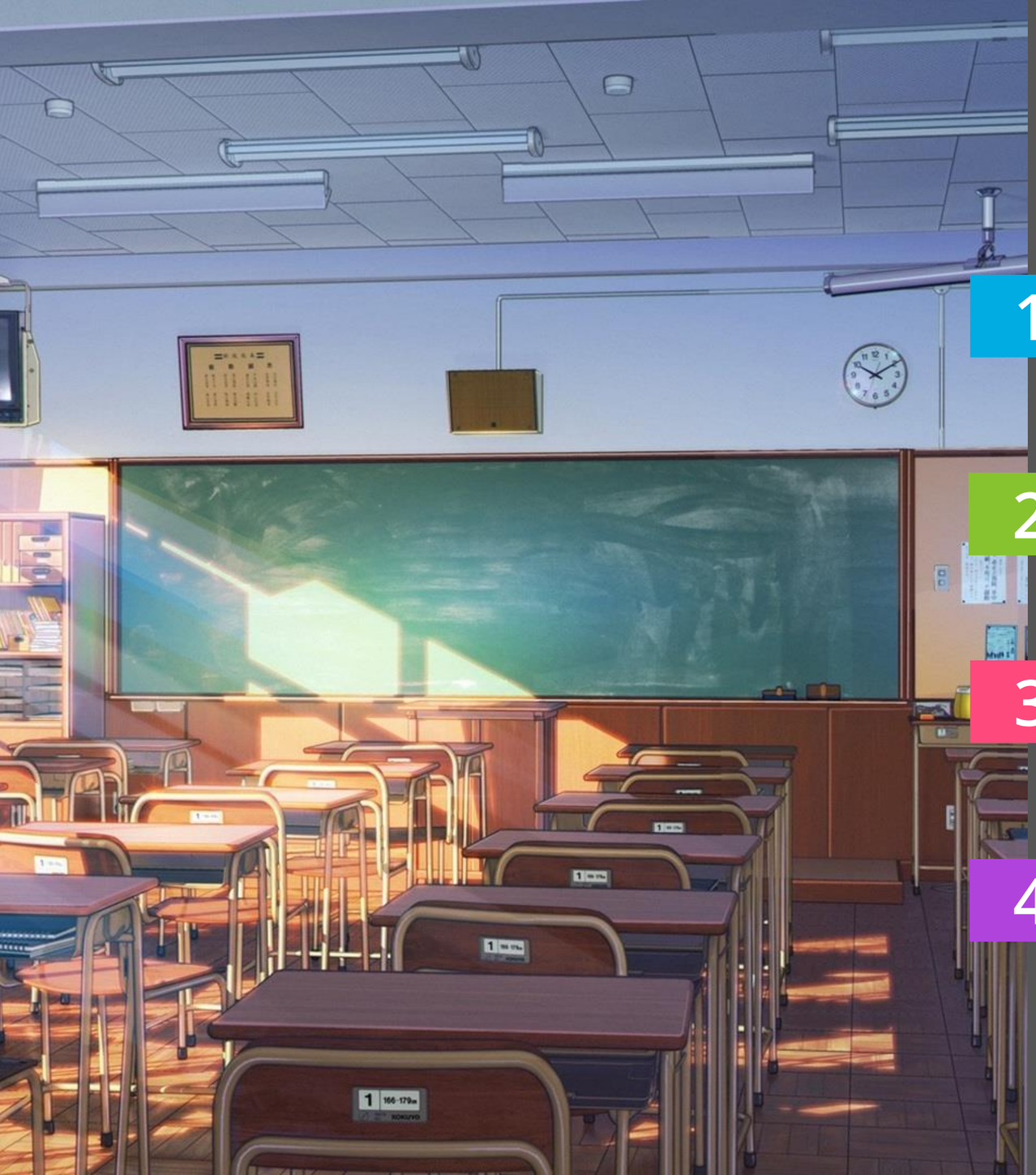
Funções

Aula 04 - Programação orientada à objetos



Professor Daniel Marques

INSTITUTO FEDERAL
Paraíba
Campus Campina Grande



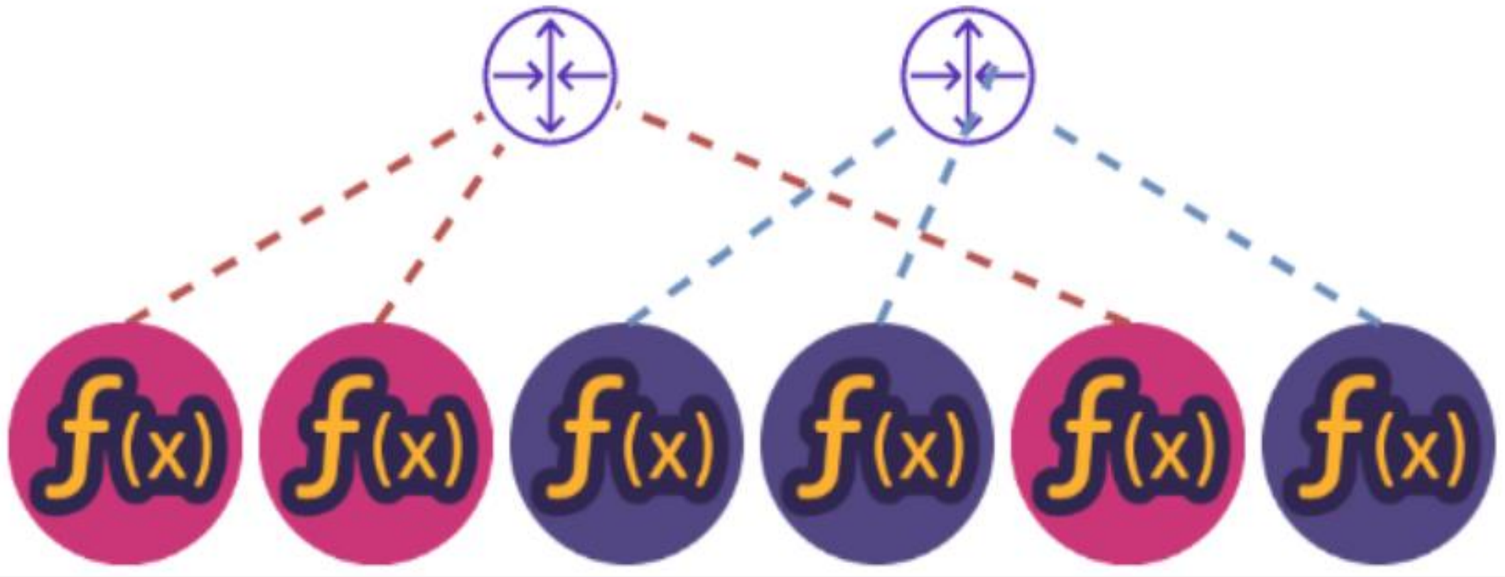
Na aula de hoje

1. Introdução à funções
2. Variáveis locais e estáticas
3. Parâmetro e retorno do tipo array
4. Sobrecarga de funções

INSTITUTO FEDERAL
Paraíba
Campus Campina Grande

O que são funções?

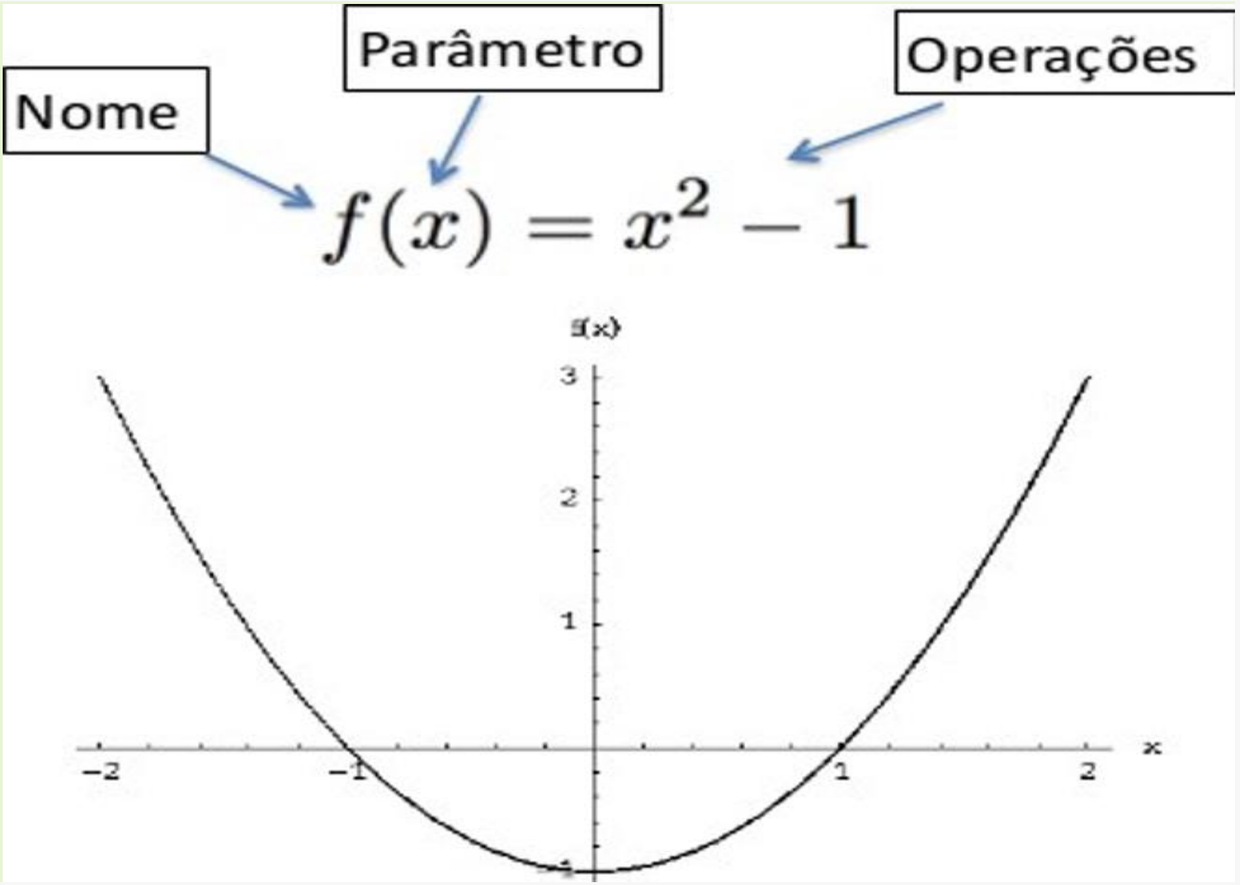
Funções permitem que o mesmo código seja reutilizado várias vezes e implementado somente uma única vez



Vantagens:

- baixo acoplamento, devido a quebra de um problema em subproblemas
- Facilidade na localização de erros

Funções



Função matemática

Função em C++

```
12
13  double funcaoMatematica(double x) {
14      double calculo = (x*x-1);
15      return calculo;
16  }
17
18
19
```

Funções e seu formato

Uma função é um módulo, um bloco de código.

O código de uma função é executado ao chamarmos a função

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6      <tipo de retorno> nome(<parametros>) {
7          \\ bloco de comandos
8          return <valor>;
9      }
10
11     return 0;
12
13 }
```

Funções e seu formato

Uma função pode receber zero ou mais argumentos e retornar (geralmente) um valor

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      <tipo de retorno> nome(<parametros>) {
6          \\ bloco de comandos
7          return <valor>;
8      }
9
10     return 0;
11 }
12
13 int soma(int x, int y){
14     int somatorio = x + y;
15     return somatorio;
16 }
```

Usamos o tipo ***void*** para indicar que função não retorna valor

Exemplo

Programa para calcular o fatorial de um número através de uma função

Crio a função fatorial, recebendo um parâmetro (número inteiro) e retorno uma variável do tipo inteira

```
1  #include <iostream>
2
3  int fatorial(int numero) {
4
5      int fatorial = 1;     //produto é uma variável local
6      while (numero > 1) {
7          fatorial = fatorial * numero;
8          numero = numero - 1;
9      }
10     return fatorial;
11 }
12
13 int main() {
14     int valor = 6;
15     std::cout << "Fatorial de " << valor << " é " <<fatorial(valor);
16     return 0;
17 }
```

O que aconteceria se eu colocasse o main( ) antes da função fatorial( ) ?

Parâmetro x Argumento

Argumento são os inicializadores dos parâmetros

Parâmetros são as variáveis definidas no cabeçalho da função

```
1  #include <iostream>
2
3  int fatorial(int numero) {
4
5      int fatorial = 1;     //produto é uma variável local
6      while (numero > 1) {
7          fatorial = fatorial * numero;
8          numero = numero - 1;
9      }
10     return fatorial;
11 }
12
13 int main() {
14     int valor = 6;
15     std::cout << "Fatorial de " << valor << " é " <<fatorial(valor);
16     return 0;
17 }
```

- int numero : Aqui temos o parâmetro numero, do tipo int (linha 3)
- fatorial(5): Aqui temos a invocação da função, estamos passando o argumento valor para inicializar (linha 15)

Parâmetro x Argumento

Argumentos devem ser da mesma quantidade e do mesmo tipo dos parâmetros.

```
fat("5");      //erro: tipo incompatível
fat();         //erro: a função espera um argumento
fat(5,6);      //erro: mais argumentos que o esperado
fat(3.14)      //ok: porém argumento será convertido para int
```

Parâmetro x Argumento

```
int f1() {/* */}                    //ok: f1 não recebe argumentos
int f2(void) {/* */}                //ok: f2 não recebe argumentos
int f3(int v1, v2) {/* */}          //erro: cada parâmetro deve especificar seu tipo
int f4(int v1, int v2) {/* */}      //ok
```

Variáveis locais

Todos os parâmetros e as variáveis definidas dentro da função possuem escopo local, ou seja, são visíveis apenas dentro da função

```cpp
#include <iostream>

int fatorial(int numero) {

    int fatorial = 1;    //produto é uma variável local
    while (numero > 1) {
        fatorial = fatorial * numero;
        numero = numero - 1;
    }
    return fatorial;
}

int main() {
    int valor = 6;
    std::cout << "Fatorial de " << valor << " é " <<fatorial(valor);
    return 0;
}
```

A cada chamada, os parâmetros e variáveis/objetos são criados e destruídos

Variáveis estáticas

É possível definir uma variável local que não seja destruída após o término da execução da função. Para isso, definimos a variável como ***static***

```
1  #include <iostream>
2  using namespace std;
3
4  int call_me(void) {
5      static int v = 0;
6      return v = v+1;
7  }
8
9  int main() {
0      std::cout << call_me() << std::endl;
1      std::cout << call_me() << std::endl;
2      return 0;
3  }
```

- Variáveis locais static são inicializadas na primeira execução e permanecem “vivas” até o término do programa.
- Contudo o escopo continua sendo local.

Parâmetro array

Uma função pode receber um ou mais arrays como argumento

```cpp
#include <iostream>
using namespace std;

void printarArraySoma(int array1[5], int array2[10]){
    int arraySoma[10];

    for(int i = 0; i < 5; i++){
        arraySoma[i] = array1[i] + array2[i];
        std::cout << "Array Soma, posição " << i << " = " << arraySoma[i] << std::endl;
    }
}

int main() {
    int lista1[5] = {10,20,30,40,50};
    int lista2[10] = {1,2,3,4,5,6,7,8,9,10};
    printarArraySoma(lista1, lista2);

    return 0;
}
```

Função que retorna um array

Precisamos utilizar

#include <array>

E no início da função, devemos utilizar a seguinte nomenclatura vista abaixo

```
array<tipo,tamanho> nomeDaFunção( ){
        //bloco de código
}
```

```
1  #include <iostream>
2  #include <array>
3  using namespace std;
4
5  array<int,5> printarArraySoma(int array1[5], int array2[5]){
6      std::array<int,5> arraySoma;
7
8      for(int i = 0; i < 5; i++){
9          arraySoma[i] = array1[i] + array2[i];
10         std::cout << "Array Soma, posição " << i << " = " << arraySoma[i] << std::endl;
11     }
12
13     return arraySoma;
14 }
15
16 int main() {
17     int lista1[5] = {10,20,30,40,50};
18     int lista2[5] = {1,2,3,4,5};
19     printarArraySoma(lista1, lista2);
20
21     return 0;
22 }
```

Sobrecarga de funções

Quando temos várias funções com mesmo nome, mas diferentes parâmetros

```
1  #include <iostream>
2  #include <string>
3
4  using std::string;
5
6  int print(int x) {
7      return x;
8  }
9
10 char print(char c) {
11     return c;
12 }
13
14 int main() {
15     std::cout << print(100) << std::endl;
16     std::cout << print('c') <<  std::endl;
17     return 0;
18 }
```

Sobrecarga de funções

O compilador encontra a função que melhor casa com os argumentos passados

```
int func(int, int);
double func(double, double);

func(10, 20);         //ok: invoca func(int, int)
func(10.0, 20.0);     //ok: invoca func(double, double)
```

Sobrecarga de funções

Caso não haja uma função que case com os argumentos passados, o compilador gera uma mensagem de erro

```
int func(int, int);
double func(double, double);

func(10);          //erro: sem casamento
func(10.0);        //erro: idem
func('c');         //erro: idem
```

Sobrecarga de funções

Caso haja mais de uma função possível, o compilador gera uma mensagem de erro pela ambiguidade

```
int func(int, int);
double func(double, double);

func(10, 20.0);      //erro: ambiguidade
func(10.0, 20);      //erro: ambiguidade
```

Sobrecarga de funções

Argumentos default (padrão):

Quando definimos um valor padrão para um parâmetro

```
void func(int x = 0, double y = 0.0) {/**/}

func();         //ok: x = 0 e y = 0.0
func(10);       //ok: x = 10 e y = 0.0
func(10,5.0); //ok: x = 10 e y = 5.0
func(,5.0);    //erro: não podemos omitir argumentos à esquerda
func(5.0);     //ok: mas x = 5.0 e y = 0.0
```

Sobrecarga de funções

Argumentos default (padrão):

Parâmetros com valor padrão devem ser considerados da direita para esquerda

```
void func(int x = 0, double y = 0.0) {/**/} //ok
void func(int x, double y = 0.0) {/**/}     //ok
void func(int x = 0, double y) {/**/}       //erro
```

Dúvidas???

Referências

1. DA COSTA, Anderson Fabiano F. **Fundamentos de C++**. Instituto Federal da Paraíba. 2022.

2. OLIVEIRA, Victor A. P. **Fundamentos de C++**. Instituto Federal da Paraíba. 2022.

3. HORSTMANN, C. **Conceitos de Computação com o Essencial de C++**, 3ª edição, Bookman, 2005

4.

5.