

III.Sobrecarga

Sobrecarga de Métodos
Sobrecarga de Constructores

Sobrecarga de Métodos

- É a possibilidade de existência de múltiplas versões do mesmo método dentro da mesma classe, desde que possuam assinaturas diferentes.
 - as listas de tipos de seus parâmetros formais
- Possibilita a construção de classes versáteis, na qual a realização de certas operações admitam varias formas.

Sobrecarga de Métodos

- Diferentes números de parâmetros
 - int **soma**(int a) { retorna a+a; }
 - int **soma**(int a, int b) { return a+b; }
 - int **soma**(int a, int b, int c) { return a+b+c; }

Sobrecarga de Métodos

- Diferentes números de parâmetros
 - `int soma(int a) { retorna a+a; }`
 - `int soma(int a, int b) { return a+b; }`
 - `int soma(int a, int b, int c) { return a+b+c; }`
- Diferentes tipos de parâmetros
 - `int soma(int a, int b) { return a+b; }`
 - `double soma(int a, double b) { return a+b; }`
 - `double soma(double a, double b) { return a+b; }`

Sobrecarga de Construtores

- Da mesma forma que métodos, construtores podem ser sobrecarregados.

```
Contador() { this->num=0; }
```

```
Contador(int num) { this->num = num; }
```

```
int main() {
```

```
    Contador c1, c2(9);
```

```
    ...
```

VALOR DEFAULT

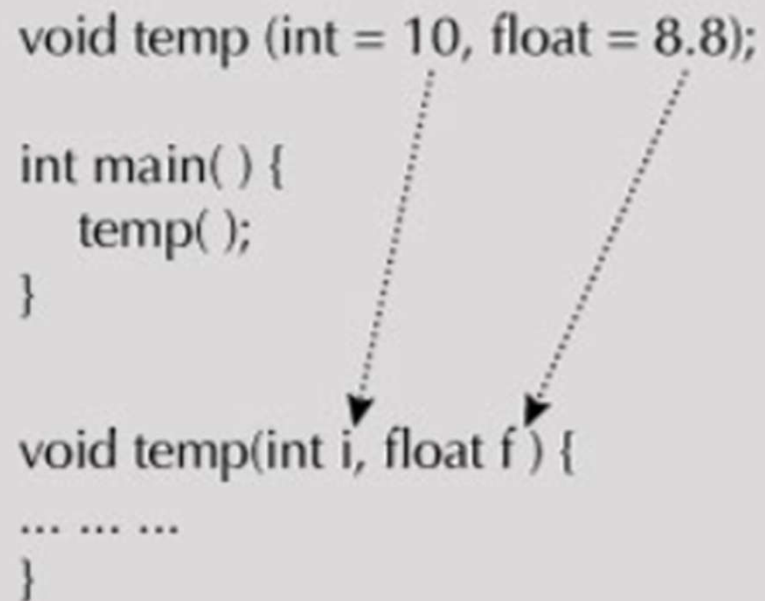
Valor Default – Argumentos

- Em C++, você pode prover valores default (padrão) para parâmetros de funções.
- Se o argumento não for passado para a função, então o valor padrão é usado.

Valor Default

Sem argumentos

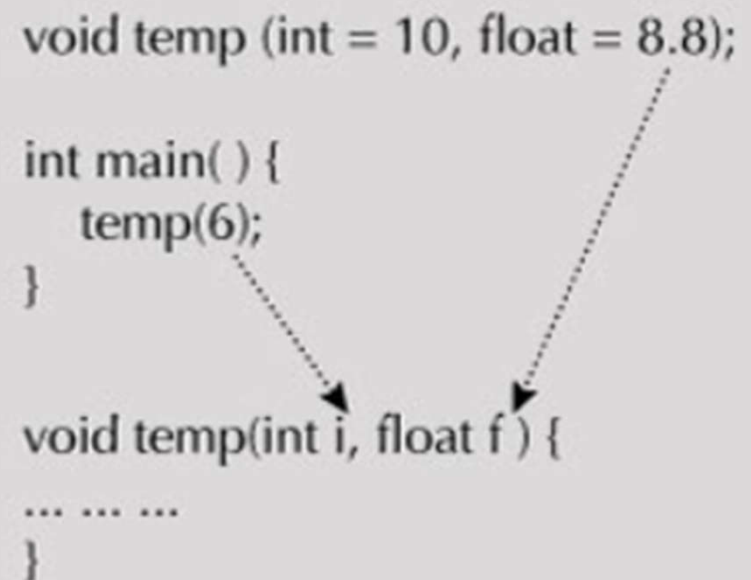
```
void temp (int = 10, float = 8.8);  
  
int main( ) {  
    temp( );  
}  
  
void temp(int i, float f) {  
    ... ..  
}
```



The diagram illustrates the use of default arguments in a C++ function. The function signature `void temp (int = 10, float = 8.8);` is shown at the top. Below it, the `main` function calls `temp();` without any arguments. At the bottom, the function definition `void temp(int i, float f) { }` is shown. Two dotted arrows originate from the default values `10` and `8.8` in the function signature and point to the parameters `i` and `f` in the function definition, respectively, indicating that these default values are used when no arguments are provided.

Primeiro argumento passado

```
void temp (int = 10, float = 8.8);  
  
int main( ) {  
    temp(6);  
}  
  
void temp(int i, float f) {  
    ... ..  
}
```



The diagram illustrates the use of default arguments in a C++ function. The function signature `void temp (int = 10, float = 8.8);` is shown at the top. Below it, the `main` function calls `temp(6);` with one argument. At the bottom, the function definition `void temp(int i, float f) { }` is shown. Two dotted arrows originate from the default values `10` and `8.8` in the function signature and point to the parameters `i` and `f` in the function definition, respectively. Additionally, a dotted arrow originates from the argument `6` in the `main` function call and points to the parameter `i` in the function definition, indicating that the value `6` is passed for `i`, while the default values are used for `f`.


Valor Default

Todos os argumentos passados

```
void temp (int = 10, float = 8.8);
```

```
int main( ) {  
    temp(6, -2.3 );  
}
```

```
void temp(int i, float f ) {  
    ... ..  
}
```

The diagram consists of two dashed arrows. One arrow originates from the number '6' in the function call 'temp(6, -2.3);' within the 'main' function and points to the parameter 'i' in the function definition 'void temp(int i, float f)'. The second arrow originates from the value '-2.3' in the same function call and points to the parameter 'f' in the function definition.

Segundo argumento passado

- IMPOSSÍVEL

Problema na Sobrecarga

- Quando tem um parâmetro com valor default
- Dá ambiguidade
 - Ajusta(int a)
 - Ajusta(int a, int b = 9)
- São o mesmo para o compilador

EXERCÍCIOS

Exercício 1

- Para a classe Caixa, crie 3 formas para o método **ajustaMedidas()**, com:
 - 2 parâmetros: peso e empilhamento
 - 3 parâmetros: altura, largura e profundidade
 - 5 parâmetros: todos eles

```
double altura, largura, profundidade;  
double peso;  
int empilhamento;
```

Classe Caixa

```
class Caixa {  
    double altura, largura, profundidade;  
    double peso;  
    int empilhamento;  
  
    double area();  
    double volume();  
    void mostra();  
};
```

Exercício 2

- Para a classe Data, crie três construtores diferentes:
 - um com 3 parâmetros: dia, mês e ano
 - um sem parâmetros: que inicia com a data de hoje
 - um com 2 parâmetros: mês e ano
 - o dia passa a ser o primeiro dia do mês

Exercício 3

- Crie uma classe chamada Ponto.
 - com 2 atributos: X e Y
- Crie dois construtores para esta classe:
 - um, sem parâmetros, que inicializa o ponto com 0,0
 - outro com 2 parâmetros
- Crie dois métodos chamados distancia:
 - um que recebe dois valores reais: X e Y
 - um que recebe outro Ponto

Classe Ponto

```
class Ponto
{
    float x;
    float y;

    public:

    Ponto();
    Ponto(float a, float b);
    void mostra(void);
    void move(float dx, float dy);
    float distancia(float x, float y);
    float distancia(Ponto outro);
};
```