Programação Orientada a Objetos

Aula 06 — Overload de operações e Polimorfismo

Hugo Marcondes

Departamento Acadêmico de Eletrônica
DAELN

hugo.marcondes@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Câmpus Florianópolis

Notes

C++ Overloading

- C++ permite a sobrecarga (overloading) de funções e operadores
  - Function Overloading
  - Operator Overloading
- Uma declaração sobrecarregada é nada menos que que declaração com mesmo nome, dentro de um mesmo escopo (namespace ou class), contudo, com parâmetros e argumentos diferentes, e implementações diferentes.
- Ao chamar uma função ou operator sobrecarregado, o compilador irá determinar qual é a definição mais apropriada através da comparação dos tipos dos argumentos da chamada utilizada.

Notes

Sobrecarga de Funções

- Definição da mesma função no mesmo escopo
  - Diferem pelo número e tipo dos parâmetros da função
  - O tipo de retorno não é considerado!

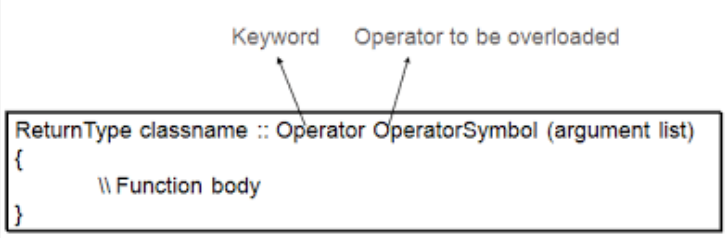
```
1  class printData  {
2      public:
3          void print(int i) {
4          cout << "Print int: " << i << endl;
5          }
6
7          void print(double  f) {
8          cout << "Print float: " << f << endl;
9          }
10
11         void print(char* c) {
12         cout << "Print character: " << c << endl;
13         }
14 };
```

```
1  int main(void) {
2      printData pd;
3
4      // Call print to print integer
5      pd.print(5);
6
7      // Call print to print float
8      pd.print(500.263);
9
10     // Call print to print character
11     pd.print("Hello C++");
12
13     return 0;
14 }
```


Notes

Sobrecarga de Operadores

- Você pode realizar a sobrecarga da maioria dos operadores disponíveis em C++
  - Uso em tipos (classes) definidos pelo usuário
- Operadores sobrecarregados são funções que possuem o modificador “operator” seguido do símbolo do operador sobrecarregado

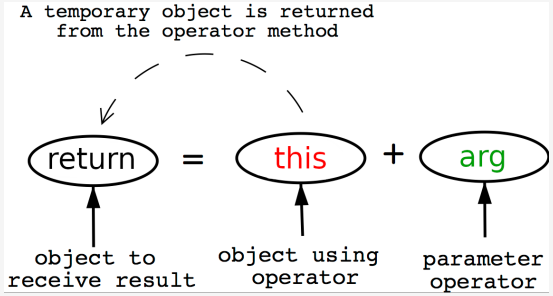


Notes

Sobrecarga de Operadores

- A maioria dos operadores sobrecarregados podem ser definidos como uma **função membro da classe**, ou como uma função não membro.

```
Box operator+(const Box &arg);
```

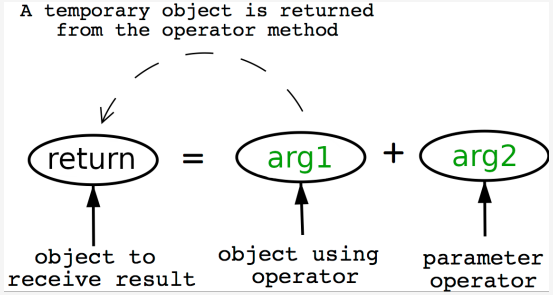


Notes

Sobrecarga de Operadores

- A maioria dos operadores sobrecarregados podem ser definidos como uma função membro da classe, ou como uma **função não membro**.

```
Box operator+(const Box &arg1, const Box &arg2);
```



Notes

Sobrecarga de Operadores

```
1  class Box {
2  public:
3      Box(double len, double bre, double hei) {
4          this->length = len;
5          this->breadth = bre;
6          this->height = hei;
7      }
8
9      double getVolume(void) {
10             return length * breadth * height;
11     }
12
13     Box operator+(const Box& b) {
14         Box box;
15         box.length = this->length + b.length;
16         box.breadth = this->breadth + b.breadth;
17         box.height = this->height + b.height;
18         return box;
19     }
20
21 private:
22     double length;      // Length of a box
23     double breadth;     // Breadth of a box
24     double height;      // Height of a box
25 };
```

```
1  int main() {
2      Box Box1(6.0, 7.0, 5.0);
3      Box Box2(12.0, 13.0, 10.0);
4      Box Box3;
5
6      cout << "Volume of Box1 : ";
7      cout << Box1.getVolume() << endl;
8
9      cout << "Volume of Box2 : ";
10     cout << Box2.getVolume() << endl;
11
12     cout << "Volume of Box3 : ";
13     cout << Box3.getVolume() << endl;
14
15     cout << "Box3 = Box1 + Box2" << endl;
16     Box3 = Box1 + Box2;
17
18     cout << "Volume of Box3 : ";
19     cout << Box3.getVolume() <<endl;
20
21     return 0;
22 }
23
```

Notes

Operadores sobrecarregáveis

Operator Category	Operators
Arithmetic	+, -, *, /, %
Bit-Wise	&, |, ~, ^
Logical	&&, ||, !
Relational	<, >, ==, !=, <=, >=
Assignment	=
Arithmetic assignment	+=, -=, *=, /=, %=, &=, |=, ^=
Shift	>>, <<, >>=, <<=
Unary	++, —
Subscripting	[ ]
Function call	( )
Dereferencing	->
Unary sign prefix	+, -
Allocate and free	new, delete

Table 9.1 C++ Overlodable Operators

Notes

Exemplo: Output e Input Streams

```
class Box {
   public:
      ...
      Box operator+(const Box& b) {
         Box box;
         box.length = this->length + b.length;
         box.breadth = this->breadth + b.breadth;
         box.height = this->height + b.height;
         return box;
      }

      friend ostream &operator<<( ostream &output, const Box& b ) {
         output << "Box(" << b.length << "," << b.breadth << "," << b.height << ")";
         return output;
      }

      friend istream &operator>>( istream  &input, Box &b ) {
         cout << "Length ? ";
         input >> b.length;
         cout << "Breadth ? ";
         input >> b.breadth;
         cout << "Height ? ";
         input >> b.height;
         return input;
      }
      ...
};
```

Notes

Exemplo: Output e Input Streams

```
int main() {

   Box Box1;
   Box Box2;
   Box Box3;

   cout << "We have 3 null boxes: " << Box1 << ", " << Box2 << ", " << Box3 << endl;

   cout << "Box1\n";
   cin  >> Box1;
   cout << "Box2\n";
   cin  >> Box2;

   cout << "Volume of " << Box1 << " " << Box1.getVolume() <<endl;
   cout << "Volume of " << Box2 << " " << Box2.getVolume() <<endl;
   cout << "Volume of " << Box3 << " " << Box3.getVolume() <<endl;

   cout << "Box3 = Box1 + Box2" << endl;
   Box3 = Box1 + Box2;

   cout << "Volume of " << Box3 << " " << Box3.getVolume() <<endl;

   return 0;

}
```

Notes

Polimorfismo

Notes

O que é Polimorfismo?

- O termo “polimorfismo” vem do grego e significa “muitas formas”.
- Em programação orientada a objetos, permite que objetos de diferentes classes sejam tratados de forma uniforme, compartilhando uma interface comum.

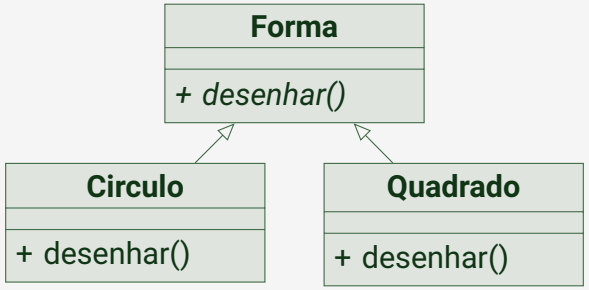
- **Tipos de Polimorfismo:**
  - Polimorfismo em tempo de compilação
    - Sobrecarga de métodos e operadores.
  - Polimorfismo em tempo de execução
    - Baseado em *herança*.
    - Uso de métodos virtuais e ponteiros ou referências.
    - A decisão de qual método chamar ocorre em tempo de execução.

Notes

Exemplo no Mundo Real

Exemplo: Formas Geométricas

- Imagine diferentes formas como círculo, quadrado e triângulo.
- Todas podem ser desenhadas (`desenhar()`), mas o comportamento específico depende da forma.
- Isso é o polimorfismo em ação: um único método (`desenhar()`) com implementações diferentes.



Notes

Exemplo: Formas Geométricas

```cpp
class Forma {
public:
    virtual void desenhar() const = 0; // Método virtual puro
    virtual ~Forma() {}
};

class Circulo : public Forma {
public:
    void desenhar() const override {
        cout << "Desenhando um círculo." << endl;
    }
};

class Quadrado : public Forma {
public:
    void desenhar() const override {
        cout << "Desenhando um quadrado." << endl;
    }
};

Forma* forma = new Circulo();
forma->desenhar(); // Saída: "Desenhando um círculo"
delete forma;
forma = new Quadrado();
forma->desenhar(); // Saída: "Desenhando um quadrado."
delete forma;
```


Notes

Polimorfismo

Benefícios

- **Flexibilidade:** Código pode lidar com diferentes tipos de objetos de maneira uniforme.
- **Extensibilidade:** É fácil adicionar novos comportamentos sem modificar código existente.
- **Redução de código repetitivo:** Reutilização de métodos gerais.

Aplicações Práticas

- Sistemas de pagamento (cartão, boleto, Pix, etc.).
- Modelagem de interfaces gráficas (botões, caixas de texto, sliders).
- Jogos (diferentes tipos de personagens, inimigos ou objetos interativos).

Notes

Conclusão

- Polimorfismo é uma das ferramentas mais poderosas da Programação Orientada a Objetos.
- Permite criar sistemas flexíveis, escaláveis e reutilizáveis.
- Combinado com herança e encapsulamento, é essencial para o design de software moderno.

Notes

That’s all folks!

INSTITUTO FEDERAL
Santa Catarina
Câmpus Florianópolis

Notes

Notes