C++

Giselle Lopes Ferrari Ronque
ferrari@eletrica.ufpr.br

Construtores

- Atributos não podem ser explicitamente inicializados na declaração da classe – necessidade de métodos de inicialização dos valores dos atributos!
- Para automatizar este processo ⇨ Construtores.
- Construtores – Funções-membro que têm o mesmo nome da classe e são executadas automaticamente e uma única vez! – quando um objeto é criado.

Construtores

- Sintaxe:

class *Nome_Classe*
{
…
*Nome_Classe***(***Lista_Argumentos_inicializ***);**
…
};

- Exemplo:

```
class Ponto{
    public:
        int Lerx();
        void Fixarx(int);
        Ponto(int);             // Protótipo do construtor
    private:
        int x, y;
};
```

Construtores

- Definição do Construtor:

Nome_Classe **::** *Nome_Classe***(***Lista_Argumentos_inicializ***)**
{ …
CORPO DO CONSTRUTOR
…
}

- Exemplo:

```
Ponto::Ponto(int i)                          // não há tipo de retorno!!
{
   x = i;
    // O construtor pode chamar qualquer função-membro
}
```

Construtores

- Arquivo Ponto.h

```
class Ponto{
    public:
        int Lerx();
        void Fixarx(int);
        Ponto(int);
    private:
         int x, y;
};
```

Construtores

- Arquivo Ponto.cpp

```
#include "Ponto.h"

Ponto::Ponto(int i){
   x = i;
}
int Ponto::Lerx(){
   return x;
};
void Ponto::Fixarx (int i){
   x = i;
};
```

Construtores

- Arquivo main.cpp

```
#include <iostream>
#include "Ponto.h“
using namespace std;
int main(int argc, char *argv[])
{
   Ponto P(55);
   cout << "O valor de x de P eh: " << P.Lerx() << endl;
   int i;
   cout << "Entre com o valor de x: " << endl;
   cin >> i;
   P.Fixarx(i);
   cout << "A posicao x do ponto e: " << P.Lerx() << endl;
      system("PAUSE");
   return EXIT_SUCCESS;
}
```

Sobrecarga de Construtores

- Dois códigos distintos ao mesmo Construtor - garantir mais de uma forma de inicialização de parâmetros ao usuário, por exemplo.

- Para tanto ⇨ incluir os diferentes protótipos na declaração da classe e em seguida, definir os diferentes códigos.

Sobrecarga de Construtores

- Arquivo Ponto.h

```
class Ponto{
    public:
        int Lerx();
        void Fixarx(int);
        Ponto();
        Ponto(int);
    private:
         int x, y;
};
```

Sobrecarga de Construtores

- Arquivo Ponto.cpp

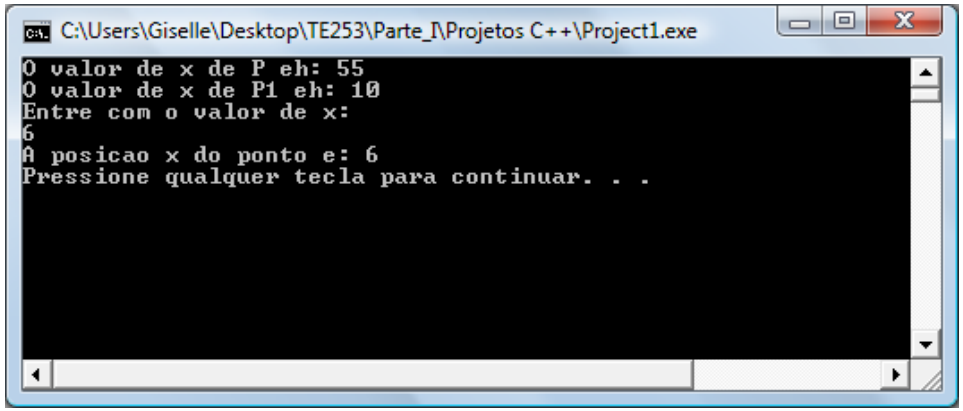
```
#include "Ponto.h"

Ponto::Ponto(){
   x = 10;
   y = 0;
}
Ponto::Ponto(int i){
   x = i;
}
```

```
int Ponto::Lerx(){
   return x;
};
void Ponto::Fixarx (int i){
   x = i;
};
```

Sobrecarga de Construtores

- Arquivo main.cpp

```
#include <iostream>
#include "Ponto.h“
using namespace std;
int main(int argc, char *argv[])
{
    Ponto P(55), P1;
    cout << "O valor de x de P eh: " << P.Lerx()
      << endl;
    cout << "O valor de x de P1 eh: " <<
      P1.Lerx() << endl;
    int i;
    cout << "Entre com o valor de x: " << endl;
    cin >> i;
    P.Fixarx(i);
    cout << "A posicao x do ponto e: " <<
       P.Lerx() << endl;
       system("PAUSE");
    return EXIT_SUCCESS;
}
```



Sobrecarga de Construtores

- Se existe um construtor alternativo, C++ não gerará um construtor por default.
  - Arquivo Ponto.h

```
class Ponto{
    public:
        int Lerx();
        void Fixarx(int);
        Ponto(int);
    private:
        int x, y;
        Ponto();
};
```

Para prevenir que os usuários da classe de criar um objeto sem parâmetros:

1) **Omitir o construtor por default;**
2) **Tornar o construtor privado.**

Destrutores

- São métodos que liberam a memória alocada para o objeto.
- Também utilizam o mesmo nome da classe, mas precedido por um ‘~’ (til).
- Tal como o construtor, o destrutor não pode ter valor de retorno e nem pode ser chamado explicitamente.
- Ao contrário dos construtores, Destrutores não aceitam argumentos e nem podem ser sobrecarregados.

Destrutores

```
class CriaEDestroi
{
    public:
        CriaEDestroi (int);
        ~CriaEDestroi( );
    private:
        int numero;
};

CriaEDestroi::CriaEDestroi(int valor){
    numero = valor;
    cout << “Construtor do objeto ” << numero;
}

// sem tipo e sem argumentos
CriaEDestroi::~CriaEDestroi( ) {
   cout << “ Destrutor do objeto  ” << numero << endl;
}
```

Destrutores

```
int main (){
    {
        CriaEDestroi x (9);
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Exercício 1

- Implementar um programa em C++, usando paradigma de orientação a objetos, que:
  - Crie um construtor que inicialize os dados com zero e outro construtor que inicialize os dados com valor fixo;
  - Crie uma função-membro para solicitar a hora do usuário;

    Hora, minuto e segundo
  - Crie uma função-membro para imprimir a hora no formato; 01:19:49
  - Crie uma função-membro para adicionar dois objetos da classe passados como argumentos;
  - Crie uma função-membro para Subtrair duas horas e retorne o número de segundos entre elas. A função recebe dois objetos da classe passados como argumentos;
  - Os dados da classe devem ser privados;
  - Crie um destrutor.

Exercício 1

class tempo

private:
int hora, minuto, segundo;

public:
tempo(int h, int m, int s);
tempo();
void IncHora();
void PrintHora();
tempo AdicionaHora(tempo t1, tempo t2);
int SubtraiHora(tempo t1, tempo t2);
~tempo();