

Introduction to Object-Oriented Programming with C++

Prof. Dr. Giovanni Gracioli

`giovani@lisha.ufsc.br`

`http://www.lisha.ufsc.br/Giovani`

Objectives

- Defining and declaring methods
 - header and implementation files
- Class scope and accessing class members
- Default constructor parameter values
- Destructors
- Default memberwise assignment

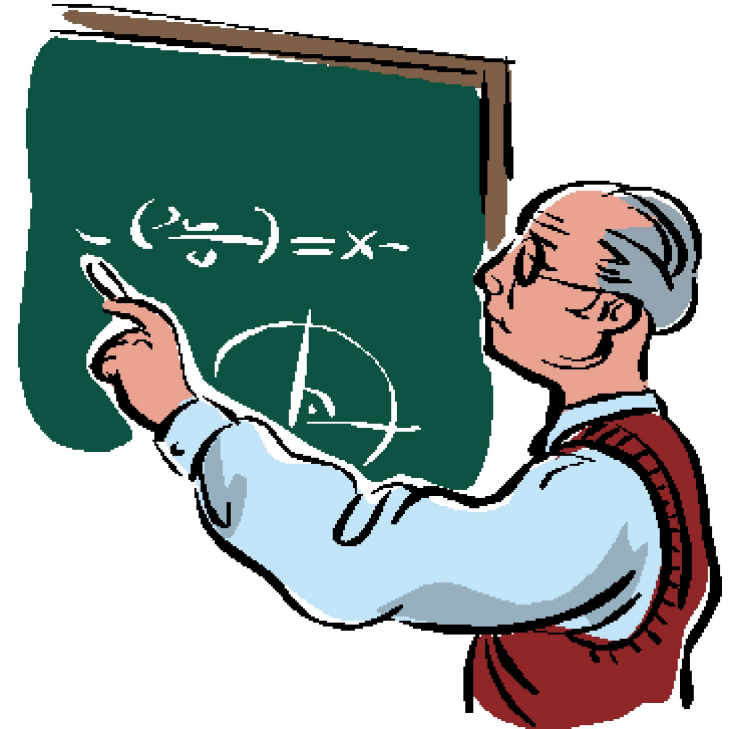
What you need to know to follow

- Basics C++ skills
 - Concepts of class and object
 - How to declare a class in C++
 - How to create an object in C++
 - How to use an object in C++

What you will learn

- How to separate method definition from its implementation in different files
- How to access an object using its name or a pointer to an object
- How to declare constructors with default arguments
- How to implement destructors
- Default memberwise assignment

Let's get started



Declaring and defining a method

- A method can be declared inside a class, but defined outside this class definition
 - It still keeps the class scope
 - It is only known by other class members
- Method defined inside the class (the way you have learned so far)
 - C++ compiler tries to put all method calls inline into the final code

Example (1)

```
1 // Fig. 9.1: Time.h
2 // Time class definition.
3 // Member functions are defined in Time.cpp
4
5 // prevent multiple inclusions of header
6 #ifndef TIME_H
7 #define TIME_H
8
9 // Time class definition
10 class Time
11 {
12 public:
13     Time(); // constructor
14     void setTime( int, int, int ); // set hour, minute and second
15     void printUniversal(); // print time in universal-time format
16     void printStandard(); // print time in standard-time format
17 private:
18     int hour; // 0 - 23 (24-hour clock format)
19     int minute; // 0 - 59
20     int second; // 0 - 59
21 }; // end class Time
22
23 #endif
```

Example (2)

```
1 // Fig. 9.2: Time.cpp
2 // Member-function definitions for class Time.
3 #include <iostream>
4 #include <iomanip>
5 #include <stdexcept> // for invalid_argument exception class
6 #include "Time.h" // include definition of class Time from Time.h
7
8 using namespace std;
9
10 // Time constructor initializes each data member to zero.
11 Time::Time()
12 {
13     hour = minute = second = 0;
14 } // end Time constructor
```


Class scope and Accessing class member (1)

- The class scope contains
 - Attributes and methods
- Class members can be accessed by all methods
- Outside the class scope
 - Class members defined as **public** are referenced by a handle
 - name of an object
 - reference to an object
 - pointer to an object
- **Methods can be overloaded**

Class scope and Accessing class member (2)

- Dot member selection operator (.)
 - used with the object's name or by a reference to an object
- Arrow member selection operator (->)
 - used with a pointer to an object

Example (1)

```
1 // Figura 9.4: fig09_04.cpp
2 // Demonstrando os operadores de acesso ao membro de classe com . e ->
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 // definição da classe Count
8 class Count
9 {
10 public: // dados public são perigosos
11     // configura o valor do membro de dados private x
12     void setX( int value )
13     {
14         x = value;
15     } // fim da função setX
16
17     // imprime o valor do membro de dados private x
18     void print()
19     {
20         cout << x << endl;
21     } // fim da função print
22
23 private:
24     int x;
25 }; // fim da classe Count
```

Example (2)

Usando o operador de seleção de membro ponto com um objeto.

```
26
27 int main()
28 {
29     Count counter; // cria objeto counter
30     Count *counterPtr = &counter; // cria ponteiro para counter
31     Count &counterRef = counter; // criar referência para counter
32
33     cout << "Set x to 1 and print using the object's name: ";
34     counter.setX( 1 ); // configura membro de dados x como 1
35     counter.print(); // chama função-membro print
36
37     cout << "Set x to 2 and print using a reference to an object: ";
38     counterRef.setX( 2 ); // configura membro de dados x como 2
39     counterRef.print(); // chama função-membro print
40
41     cout << "Set x to 3 and print using a pointer to an object: ";
42     counterPtr->setX( 3 ); // configura membro de dados x como 3
43     counterPtr->print(); // chama função-membro print
44     return 0;
45 } // fim de main
```

Usando o operador de seleção de membro ponto com uma referência.

Usando o operador de seleção de membro seta com um ponteiro.

```
Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

Constructor with default arguments

- The constructors can specify **default arguments**
 - They can initialize data members in a consistent state
 - even if no values are passed to the method

Example (1)

```
1 // Figura 9.8: Time.h
2 // Declaração da classe Time.
3 // Funções-membro definidas em Time.cpp.
4
5 // impede múltiplas inclusões de arquivo de cabeçalho
6 #ifndef TIME_H
7 #define TIME_H
8
9 // Definição de tipo de dados abstrato Time
10 class Time
11 {
12 public:
13     Time( int = 0, int = 0, int = 0 ); // construtor-padrão
14
15     // funções set
16     void setTime( int, int, int ); // configura hour, minute, second
17     void setHour( int ); // configura hour (depois da validação)
18     void setMinute( int ); // configura minute (depois da validação)
19     void setSecond( int ); // configura second (depois da validação)
```


Example (2)

```
1 // Figura 9.9: Time.cpp
2 // Definições de função-membro para a classe Time.
3 #include <iostream>
4 using std::cout;
5
6 #include <iomanip>
7 using std::setfill;
8 using std::setw;
9
10 #include "Time.h" // inclui a definição da classe Time a partir de Time.h
11
12 // Construtor de Time inicializa cada membro de dados como zero;
13 // assegura que os objetos Time iniciem em um estado consistente
14 Time::Time( int hr, int min, int sec )
15 {
16     setTime( hr, min, sec ); // valida e configura time
17 } // fim do construtor de Time
18
19 // configura novo valor de Time utilizando a hora universal; assegura que
20 // os dados permaneçam consistentes configurando valores inválidos como zero
21 void Time::setTime( int h, int m, int s )
22 {
23     setHour( h ); // configura campo private hour
24     setMinute( m ); // configura campo private minute
25     setSecond( s ); // configura campo private second
26 } // fim da função setTime
27
28 // configura valor de hour
29 void Time::setHour( int h )
```

Example (3)

```
1 // Figura 9.10: fig09_10.cpp
2 // Demonstrando um construtor-padrão para a classe Time.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "Time.h" // inclui a definição da classe Time a partir de Time.h
8
9 int main()
10 {
11     Time t1; // todos os argumentos convertidos para sua configuração-padrão
12     Time t2( 2 ); // hour especificada; minute e second convertidos para o padrão
13     Time t3( 21, 34 ); // hour e minute especificados; second convertido para o padrão
14     Time t4( 12, 25, 42 ); // hour, minute e second especificados
15     Time t5( 27, 74, 99 ); // valores inválidos especificados
16
17     cout << "Constructed with:\n\nt1: all arguments defaulted\n ";
18     t1.printUniversal(); // 00:00:00
19     cout << "\n ";
20     t1.printStandard(); // 12:00:00 AM
21
22     cout << "\n\nt2: hour specified; minute and second defaulted\n ";
23     t2.printUniversal(); // 02:00:00
24     cout << "\n ";
25     t2.printStandard(); // 2:00:00 AM
```


Destructors (1)

- As a constructor, a destructor is a special method
- Its name begins with the til character (~) followed by the class name
 - Example: ~Time()
- It is implicitly called when the object is destroyed
 - Ex: when a function ends
- The memory used by the destroyed object can be reused

Destructors (2)

- Does not **receive** nor **return** any values
- A class can only have **one destructor**
 - Overloading of destructor is not allowed
- If the programmer does not implement a destructor, the compiler will offer a **standard empty destructor**

Example (1)

```
1 // Figura 9.11: CreateAndDestroy.h
2 // Definição da classe CreateAndDestroy.
3 // Funções-membro definidas em CreateAndDestroy.cpp.
4 #include <string>
5 using std::string;
6
7 #ifndef CREATE_H
8 #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13     CreateAndDestroy( int, string ); // construtor
14     ~CreateAndDestroy(); // destrutor
15 private:
16     int objectID; // Número de ID do objeto
17     string message; // mensagem descrevendo o objeto
18 }; // fim da classe CreateAndDestroy
19
20 #endif
```

Protótipo para o destrutor.


Example (2)

```
1 // Figura 9.12: CreateAndDestroy.cpp
2 // Definições de função-membro da classe CreateAndDestroy.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "CreateAndDestroy.h"// inclui a definição da classe CreateAndDestroy
8
9 // construtor
10 CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
11 {
12     objectID = ID; // configura o número de ID do objeto
13     message = messageString; // configura mensagem descritiva do objeto
14
15     cout << "Object " << objectID << "   constructor runs   "
16         << message << endl;
17 } // fim do construtor CreateAndDestroy
18
19 // destrutor
20 CreateAndDestroy::~CreateAndDestroy()
21 {
22     // gera saída de nova linha para certos objetos; ajuda a legibilidade
23     cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
24
25     cout << "Object " << objectID << "   destructor runs   "
26         << message << endl;
27 } // fim do destrutor ~CreateAndDestroy
```

Definindo o destrutor da classe.

Example (3)

```
1 // Figura 9.13: fig09_13.cpp
2 // Demonstrando a ordem em que construtores e
3 // destrutores são chamados.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "CreateAndDestroy.h" // inclui a definição da classe CreateAndDestroy
9
10 void create( void ); // protótipo
11
12 CreateAndDestroy first( 1, "(global before main)" ); // objeto global
13
14 int main()
15 {
```



Objeto criado fora de **main**.

Default memberwise assignment

- The assignment operator (=) can be used to assign an object to another object of the same type
- Each attribute of the object on the right of the assignment operator is assigned to the same attribute in the object on the left side
- **Caution:** If the attribute is a pointer, this can cause serious problems

Example

```
1 // Figura 9.19: fig09_19.cpp
2 // Demonstrando que os objetos de classe podem ser atribuídos
3 // um ao outro utilizando atribuição-padrão de membro a membro.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "Date.h" // inclui a definição da classe Date a partir de Date.h
9
10 int main()
11 {
12     Date date1( 7, 4, 2004 );
13     Date date2; // date2 assume padrão de 1/1/2000
14
15     cout << "date1 = ";
16     date1.print();
17     cout << "\ndate2 = ";
18     date2.print();
19
20     date2 = date1; // atribuição-padrão de membro a membro
21
22     cout << "\n\nAfter default memberwise assignment, date2 = ";
23     date2.print();
24     cout << endl;
25     return 0;
26 } // fim de main
```

A atribuição membro a membro atribui membros de dados de **date1** a **date2**.

date2 agora armazena a mesma data como **date1**.

date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004

Review

- We can declare a method in a header file and define the method in a cc file
 - All methods in a header files will be inlined
- dot operator (.) and arrow (->) operator to access objects
- Constructors with default arguments
 - This can also be applied to any method
- Destructors
- Default memberwise assignment

Tasks

- Read chapter 9 of the text book (C++ how to program 8th edition)
- Lists of exercises are at moodle

References

- Paul Deitel e Harvey Deitel, C++: como programar, 5a edição, Ed. Prentice Hall Brasil, 2006.
- Paul Deitel e Harvey Deitel, C++: how to program, 8th edition, Ed. Prentice Hall, 2012.

