Federal University of Santa Catarina
Real-Time Operating Systems

Introduction to Object-Oriented Programming with C++

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br
http://www.lisha.ufsc.br/Giovani

Objectives

- Present composition
- Constructor's member initializer list
- The this pointer
- How to allocate/deallocate dynamic memory in C++
- Static class members

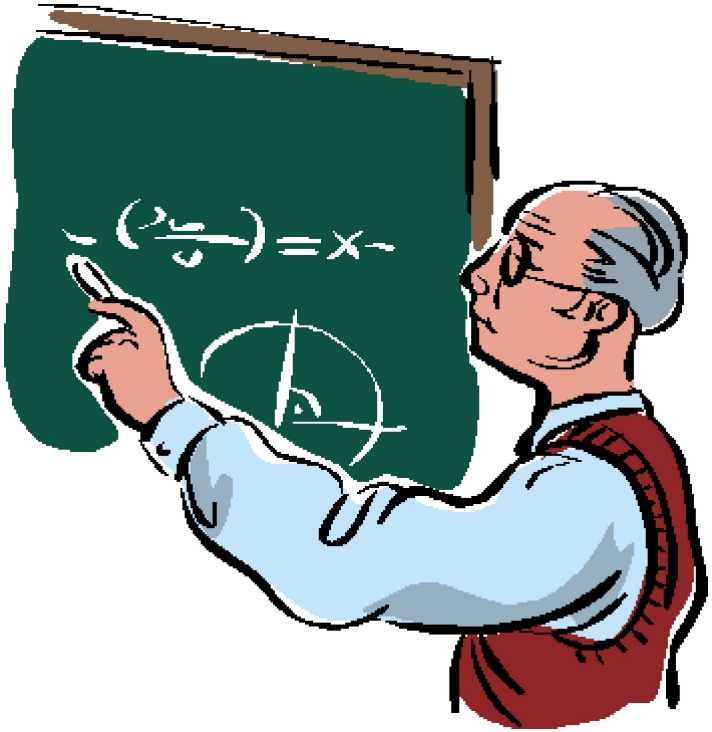
What you need to know to follow

- 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++
  - How to use constructors and destructors

What you will learn

- Declare and initialize objects inside a class
- To use the this pointer
- To allocate and deallocate dynamic memory in C++
  - How to create an object using dynamic memory
- To declare and use static class members

Let's get started



Composition: Objects as members of Classes

- Sometimes referred to as a has-a relationship
- A class can have objects of other class as members
- Ex: An AlarmClock object needs to know when it's suppose to sound its alarm
  - Include a Time object as a member of the AlarmClock
- This is called composition

Constructor's member Initializer List

- Member initializers pass arguments from the constructor to the object-member constructor
- The object-member constructor are created according to the order in which they are declared
  - Not in the order they are used in the member initializer list
- If a member initializer is not called, the standard constructor of the object is used

Example (1)

```
1  // Figura 10.12: Employee.h
2  // Definição da classe Employee.
3  // Funções-membro definidas em Employee.cpp.
4  #ifndef EMPLOYEE_H
5  #define EMPLOYEE_H
6
7  #include "Date.h" // inclui definição da classe Date
8
9  class Employee
10 {
11 public:
12    Employee( const char * const, const char * const,
13       const Date &, const Date & );
14    void print() const;
15    ~Employee(); // fornecida para confirmar a ordem de destruição
16 private:
17    char firstName[ 25 ];
18    char lastName[ 25 ];
19    const Date birthDate; // composição: objeto-membro
20    const Date hireDate; // composição: objeto-membro
21 }; // fim da classe Employee
22
23 #endif
```

Parâmetros a serem passados por meio de inicializadores de membro ao construtor para a classe **`Date`**.

Objetos **`const`** da classe **`Date`** como membros.

Example (2)

```
1  // Figura 10.13: Employee.cpp
2  // Definições de função-membro da classe Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // protótipos strlen e strncpy
8  using std::strlen;
9  using std::strncpy;
10
11 #include "Employee.h" // definição da classe Employee
12 #include "Date.h" // definição da classe Date
13
14 // construtor usa lista de inicializadores de membro para passar valores de inicializadores
15 // para construtores dos objetos-membro birthDate e hireDate
16 // [Nota: Isso invoca o chamado 'construtor de cópia padrão' que o
17 // compilador C++ fornece implicitamente.]
18 Employee::Employee( const char * const first, const char * const last,
19    const Date &dateOfBirth, const Date &dateOfHire )
20    : birthDate( dateOfBirth ), // inicializa birthDate
21      hireDate( dateOfHire ) // inicializa hireDate
22 {
23    // copia primeiro para firstName e certifica-se de que ele se ajusta
24    int length = strlen( first );
25    length = ( length < 25 ? length : 24 );
26    strncpy( firstName, first, length );
27    firstName[ length ] = '\0';
```

Inicializadores de membro que passam argumentos ao construtor implícito de cópia padrão **`Date`**.

Example (3)

```
1  // Figura 10.14: fig10_14.cpp
2  // Demonstrando composição -- um objeto com objetos-membro.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // definição da classe Employee
8
9  int main()
10 {
11    Date birth( 7, 24, 1949 );
12    Date hire( 3, 12, 1988 );
13    Employee manager( "Bob", "Blue", birth, hire );
14
15    cout << endl;
16    manager.print();
17
18    cout << "\nTest Date constructor with invalid values:\n";
19    Date lastDayOff( 14, 35, 1994 ); // mês e dia inválidos
20    cout << endl;
21    return 0;
22 } // fim de main
```

Using the this pointer

- How do methods know which object's data members to manipulate?
- Every object has access to its own address through a pointer called this
- The this pointer is not part of the object itself
- It is passed by the compiler as an implicit argument to each of the object's non-static member functions
- Objects use the this pointer implicitly (as done until now) or explicitly to reference their data members or member functions

Example (1)

```
1  // Figura 10.17: fig10_17.cpp
2  // Utilizando o ponteiro this para referenciar membros de objeto.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  class Test
8  {
9  public:
10    Test( int = 0 ); // construtor-padrão
11    void print() const;
12 private:
13    int x;
14 }; // fim da classe Test
15
16 // construtor
17 Test::Test( int value )
18    : x( value ) // inicializa x como value
19 {
20    // corpo vazio
21 } // fim do construtor Test
22
23 // imprime x utilizando ponteiros this implícito e explícito;
24 // os parênteses em torno de *this são requeridos
25 void Test::print() const
```

Example (2)

```
26  {
27     // utiliza implicitamente o ponteiro this para acessar o membro x
28     cout << "        x = " << x;
29
30     // utiliza explicitamente o ponteiro this e o operador seta
31     // para acessar o membro x
32     cout << "\n  this->x = " << this->x;
33
34     // utiliza explicitamente o ponteiro this desreferenciado e
35     // o operador ponto para acessar o membro x
36     cout << "\n(*this).x = " << ( *this ).x << endl;
37  } // fim da função print
38
39  int main()
40  {
41     Test testObject( 12 ); // instancia e inicializa testObject
42
43     testObject.print();
44     return 0;
45  } // fim de main
```

```
        x = 12
  this->x = 12
(*this).x = 12
```

Usando implicitamente o ponteiro **`this`** para acessar o membro **`x.`**

Usando explicitamente o ponteiro **`this`** para acessar o membro **`x.`**

Usando o ponteiro **`this`** desreferenciado e o operador ponto.

new and delete operators

- These operators offers support for dynamic memory allocation/deallocation
- Operator new
  - Allocates memory for an object
  - It calls the constructor to initialize the object
  - Returns a pointer to the memory space
  - It can be used to allocate memory for any fundamental type or class
- Operator delete
  - Destroys an object previously allocated with new
  - Calls the object's destructor

Examples

```
double *ptr = new double( 3.14159 );

Time *timePtr = new Time( 12, 45, 0 );

int *gradesArray = new int[ 10 ];

delete [] gradesArray;
```

Static class members (1)

- In certain cases, only one copy of a variable should be shared by all objects of a class
- A static attribute or static data member is used for these and other reasons
- Such variable represents “class-wide” information
- Although they look global variables, they have class scope
- Can be declared as public, private, or protected

Static class members (2)

- A fundamental-type static data member is initialized by default to 0
- Static attributes must be defined at global namespace scope (i.e., outside the body of the class definition) and can be initialized only in those definitions
- A class's static members exist even when no object of that class exist
- To access a public static class member when no object of the class exist, simply prefix the class name and the scope resolution operator (::) to the name of the data member
- The same applies to member functions

Example (1)

```
1  // Figura 10.21: Employee.h
2  // Definição da classe Employee.
3  #ifndef EMPLOYEE_H
4  #define EMPLOYEE_H
5
6  class Employee
7  {
8  public:
9     Employee( const char * const, const char * const ); // construtor
10    ~Employee(); // destrutor
11    const char *getFirstName() const; // retorna o nome
12    const char *getLastName() const; // retorna o sobrenome
13
14    // função-membro static
15    static int getCount(); // retorna número de objetos instanciados
16 private:
17    char *firstName;
18    char *lastName;
19
20    // dados static
21    static int count; // número de objetos instanciados
22 }; // fim da classe Employee
23
24 #endif
```

Protótipo de função para a função-membro **`static`**.

`O` membro de dados **`static`** não perde de vista o número de objetos **`Employee`** que existe atualmente.

Example (2)

```
1  // Figura 10.22: Employee.cpp
2  // Definições de função-membro da classe Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // protótipos de strlen e strcpy
8  using std::strlen;
9  using std::strcpy;
10
11 #include "Employee.h" // definição da classe Employee
12
13 // define e inicializa o membro de dados static no escopo de arquivo
14 int Employee::count = 0;
15
16 // define a função-membro static que retorna o número de
17 // objetos Employee instanciados (static declarado em Employee.h)
18 int Employee::getCount()
19 {
20    return count;
21 } // fim da função static getCount
22
```

O membro de dados **`static`** é definido e inicializado no escopo de arquivo, no arquivo **`.cpp`**.

A função-membro **`static`** pode acessar apenas dados **`static`**, porque a função poderia ser chamada somente quando não houvesse nenhum objeto.

Example (3)

```
23  // o construtor aloca dinamicamente espaço para o nome e o sobrenome e
24  // usa strcpy para copiar o nome e o sobrenome para o objeto
25  Employee::Employee( const char * const first, const char * const last )
26  {
27     firstName = new char[ strlen( first ) + 1 ];
28     strcpy( firstName, first );
29
30     lastName = new char[ strlen( last ) + 1 ];
31     strcpy( lastName, last );
32
33     count++; // incrementa contagem estática de empregados
34
35     cout << "Employee constructor for " << firstName
36        << ' ' << lastName << " called." << endl;
37  } // fim do construtor Employee
38
39  // o destrutor desaloca memória dinamicamente alocada
40  Employee::~Employee()
41  {
42     cout << "~Employee() called for " << firstName
43        << ' ' << lastName << endl;
44
45     delete [] firstName; // libera memória
46     delete [] lastName; // libera memória
47
48     count--; // decrementa contagem estática de empregados
49  } // fim do destrutor ~Employee
```

Arrays **`char`** dinamicamente alocados.

A função-membro não-**`static`** (isto é, construtor) pode modificar os membros de dados **`static`** da classe.

Desalocando a memória reservada a arrays.

Example (4)

```
1  // Figura 10.23: fig10_23.cpp
2  // Driver para testar a classe Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // Definição da classe Employee
8
9  int main()
10 {
11    // utiliza o nome da classe e o operador de resolução de escopo binário para
12    // acessar a função static number getCount
13    cout << "Number of employees before instantiation of any objects is "
14       << Employee::getCount() << endl; // utiliza o nome da classe
15
16    // utiliza new para criar dinamicamente dois novos Employees
17    // operador new também chama o construtor do objeto
18    Employee *e1Ptr = new Employee( "Susan", "Baker" );
19    Employee *e2Ptr = new Employee( "Robert", "Jones" );
20
21    // chama getCount no primeiro objeto Employee
22    cout << "Number of employees after objects are instantiated is "
23       << e1Ptr->getCount();
24
25    cout << "\n\nEmployee 1: "
26       << e1Ptr->getFirstName() << " " << e1Ptr->getLastName()
27       << "\nEmployee 2: "
28       << e2Ptr->getFirstName() << " " << e2Ptr->getLastName() << "\n\n";
```

Chamando a função-membro **`static`** por meio do nome de classe e do operador binário de resolução de escopo.

Criando dinamicamente **`Employee`**s com **`new.`**

Chamando uma função-membro **`static`** por meio de um ponteiro para um objeto da classe.

Example (5)

Resumo

fig10_23.cpp (2 de 2)

Liberando memória para a qual um ponteiro aponta.

```
29
30    delete e1Ptr; // desaloca memória
31    e1Ptr = 0; // desconecta o ponteiro do espaço de armazenamento livre
32    delete e2Ptr; // desaloca memória
33    e2Ptr = 0; // desconecta o ponteiro do espaço de armazenamento livre
34
35    // não existe nenhum objeto, portanto chama a função-membro static getCount
36    // utilizando o nome da classe e o operador de resolução de escopo binário
37    cout << "Number of employees after objects are deleted is "
38       << Employee::getCount() << endl;
39    return 0;
40 } // fim de main
```

Desconectando um ponteiro de qualquer espaço na memória.

```
Number of employees before instantiation of any objects is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after objects are instantiated is 2

Employee 1: Susan Baker
Employee 2: Robert Jones

~Employee() called for Susan Baker
~Employee() called for Robert Jones
Number of employees after objects are deleted is 0
```

Review

- A class can have objects of other classes (composition)
- Constructor's member initializer list
  - It calls the specific constructors of the member-objects
  - Order in which the member-objects were declared
- New and delete operators
- Static class members

Tasks

- Read chapter 10 of the text book (C++ how to program 8th edition)

- Exercises of chapter 10
  - 10.3, 10.4, 10.5, 10.6, 10.7, 10.9, 10.10, 10.11

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.