C++: Classes

Fundamentals

Giulia Meneghetti

Department of Electrical Engineering
Linköping University

May 11, 2015

Classes and Objects

- A **class** is a set of **objects** that share structure (variables) and behaviours (function).
- An **object** is an instantiation of a class.
- Definition of a class:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} object_names;
```

class versus **struct**

- A class can be defined with the **struct** keyword: its members are **public** by default.

```
struct person
{
  string name;
  int age;

};
```

- A class defined with the **class** keyword has **private** members by default.

```
class person
{
  public:
    string name;
    int age;
};
```

Access Specifiers

These keywords determine how the members of the class can be accessed:

- **private** members are accessible only from within other class members or "friends". (object-structure related data)
- **protected** members are accessible from within the same class (or from their "friends"), but also from members of their derived classes. (inheritance)
- **public** members are accessible where the object is visible. (manipulate the object)

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area (void);
} rect;
```

Example I

```
1  // classes example
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      void set_values (int,int);
9      int area() {return width*height;}
10 };
11
12 void Rectangle::set_values (int x, int y) {
13   width = x;
14   height = y;
15 }
16
17 int main () {
18   Rectangle rect;
19   rect.set_values (3,4);
20   cout << "area: " << rect.area();
21   return 0;
22 }
```

Exmaple I

```
1  // classes example
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      void set_values (int,int);
9      int area() {return width*height;}
10 };
11
12 void Rectangle::set_values (int x, int y) {
13   width = x;
14   height = y;
15 }
16
17 int main () {
18   Rectangle rect;
19   rect.set_values (3,4);
20   cout << "area: " << rect.area();
21   return 0;
22 }
```

Constructor

- The constructor initializes member variables or allocate storage.
- Constructors are only executed once, when a new object of that class is created.
- A constructor can also be overloaded with different versions taking different parameters.
- More recently, C++ introduced the possibility of constructors to be called using uniform initialization (use braces {} instead of parentheses ()).

```
1  // overloading class constructors
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      Rectangle ();
9      Rectangle (int x, int y) : width(x), height(y) {};
10     ~Rectangle();
11     int area (void) {return (width*height);}
12 };
13
14 Rectangle::Rectangle () {
15   width = 5;
16   height = 6;
17 }
18
19 Rectangle::~Rectangle () {
20   cout << "Rectangle destroied" << endl;
21 }
22
23
24 int main () {
25   Rectangle rect (3,4);
26   Rectangle rectb;
27   Rectangle rectc {2,6};
28   cout << "rect area: " << rect.area() << endl;
29   cout << "rectb area: " << rectb.area() << endl;
30   cout << "rectc area: " << rect.area() << endl;
31   return 0;
32 }
```

Destructor

- A destructor frees up the memory taken up by the private variables of the class.
- Anything else?

```
1  // overloading class constructors
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      Rectangle ();
9      Rectangle (int x, int y) : width(x), height(y) {};
10     ~Rectangle();
11     int area (void) {return (width*height);}
12 };
13
14 Rectangle::Rectangle () {
15   width = 5;
16   height = 6;
17 }
18
19 Rectangle::~Rectangle () {
20   cout << "Rectangle destroied" << endl;
21 }
22
23
24 int main () {
25   Rectangle rect (3,4);
26   Rectangle rectb;
27   Rectangle rectc {2,6};
28   cout << "rect area: " << rect.area() << endl;
29   cout << "rectb area: " << rectb.area() << endl;
30   cout << "rectc area: " << rect.area() << endl;
31   return 0;
32 }
```

Pointers to classes - Exmaple II

```
13 int main() {
14   Rectangle obj (3, 4);
15   Rectangle * foo, * bar, * baz;
16   foo = &obj;
17   bar = new Rectangle (5, 6);
18   baz = new Rectangle[2] { {2,5}, {3,6} };
19   cout << "obj's area: " << obj.area() << '\n';
20   cout << "*foo's area: " << foo->area() << '\n';
21   cout << "*bar's area: " << bar->area() << '\n';
22   cout << "baz[0]'s area:" << baz[0].area() << '\n';
23   cout << "baz[1]'s area:" << baz[1].area() << '\n';
24   delete bar;
25   delete[] baz;
26   return 0;
27 }
```

expression	can be read as
`*x`	pointed to by `x`
`&x`	address of `x`
`x.y`	member `y` of object `x`
`x->y`	member `y` of object pointed to by `x`
`(*x).y`	member `y` of object pointed to by `x` (equivalent to the previous one)
`x[0]`	first object pointed to by `x`
`x[1]`	second object pointed to by `x`
`x[n]`	(`n+1`)th object pointed to by `x`

Overloading operators

Here is a list of all the operators that can be overloaded:

Overloadable operators												
+	-	*	/	=	<	>	+=	-=	*=	/=	<<	>>
<<=	>>=	==	!=	<=	>=	++	--	%	&	^	!	|
~	&=	^=	|=	&&	||	%=	[]	()	,	->*	->	new
delete		new[]		delete[]								

Overloading operators - Example III

```
1  // overloading operators example
2  #include <iostream>
3  using namespace std;
4
5  class CVector {
6    public:
7      int x,y;
8      CVector () {};
9      CVector (int a,int b) : x(a), y(b) {}
10     CVector operator + (const CVector&);
11 };
12
13 CVector CVector::operator+ (const CVector& param) {
14   CVector temp;
15   temp.x = x + param.x;
16   temp.y = y + param.y;
17   return temp;
18 }
19
20 int main () {
21   CVector foo (3,1);
22   CVector bar (1,2);
23   CVector result;
24   result = foo + bar;
25   cout << result.x << ',' << result.y << '\n';
26   return 0;
27 }
```

The keyword **this** - Example IV

The keyword **this** represents a pointer to the object whose member function is being executed.

```
1  // example on this
2  #include <iostream>
3  using namespace std;
4
5  class Dummy {
6    public:
7      bool isitme (Dummy& param);
8  };
9
10 bool Dummy::isitme (Dummy& param)
11 {
12   if (&param == this) return true;
13   else return false;
14 }
15
16 int main () {
17   Dummy a;
18   Dummy* b = &a;
19   if ( b->isitme(a) )
20     cout << "yes, &a is b\n";
21   return 0;
22 }
```

static member

- **static** members can be either data or functions and can exist only one instance of these members.
- A static member is shared by all objects of the class. It is typically used in an object counter. What is the output of the example?

```
1  // static members in classes
2  #include <iostream>
3  using namespace std;
4
5  class Dummy {
6    public:
7      static int n;
8      Dummy () { n++; };
9  };
10
11 int Dummy::n=0;
12
13 int main () {
14   Dummy a;
15   Dummy b[5];
16   cout << a.n << '\n';
17   Dummy * c = new Dummy;
18   cout << Dummy::n << '\n';
19   delete c;
20   return 0;
21 }
```

const member

- The access to a const members from outside the class is restricted to **read-only**.
- The constructor is still allowed to initialize and modify static members.
- The member functions of a const object can only be called if they are themselves specified as const members.

```
1  // constructor on const object
2  #include <iostream>
3  using namespace std;
4
5  class MyClass {
6    public:
7      int x;
8      MyClass(int val) : x(val) {}
9      int get() {return x;}
10 };
11
12 int main() {
13   const MyClass foo(10);
14 // foo.x = 20;            // not valid: x cannot be modified
15   cout << foo.x << '\n';  // ok: data member x can be read
16   return 0;
17 }
```

Overload **const**ness

- A class may have two member functions with identical signatures except that one is **const** and the other is **not-const**.
- In this case, the const version is called only when the object is itself const, and the non-const version is called when the object is itself non-const.

```
1  // overloading members on constness
2  #include <iostream>
3  using namespace std;
4
5  class MyClass {
6      int x;
7    public:
8      MyClass(int val) : x(val) {}
9      const int& get() const {return x;}
10     int& get() {return x;}
11 };
12
13 int main() {
14   MyClass foo (10);
15   const MyClass bar (20);
16   foo.get() = 15;          // ok: get() returns int&
17 // bar.get() = 25;         // not valid: get() returns const int&
18   cout << foo.get() << '\n';
19   cout << bar.get() << '\n';
20
21   return 0;
22 }
```

Reference

This presentation is based on the C++ Tutorial - Classes I and II :
`http://www.cplusplus.com/doc/tutorial/classes/`

Next:

Classes (templates and namespaces)

Linköping University

expanding reality