Objects and classes, advanced topics

Victor Eijkhout, Susan Lindsey

Fall 2023

last formatted: October 24, 2023

Operator overloading

1. Operator overloading

Syntax:

```
<returntype> operator<op>( <argument> ) { <definition> }
```

For instance:

Code:

```
1  // geom/pointscale.cpp
2  Point Point::operator*(double f) {
3      return Point(f*x,f*y);
4  };
5      /* ... */
6    cout << "p1 to origin "
7         << p1.dist_to_origin() <<
         '\n';
8    Point scale2r = p1*2.;
9    cout << "scaled right: "
10        << scale2r.dist_to_origin()
       << '\n';
11   // ILLEGAL Point scale2l = 2.*p1;
```

Output:

```
p1 to origin 2.23607
scaled right: 4.47214
```

Can also:

Exercise 1

Revisit exercise **??** and replace the `add` and `scale` functions by overloaded operators.

Hint: for the `add` function you may need '`this`'.

2. Constructors and contained classes

Finally, if a class contains objects of another class,

```
1 class Inner {
2 public:
3   Inner(int i) { /* ... */ }
4 };
5 class Outer {
6 private:
7   Inner contained;
8 public:
9 };
```

3. When are contained objects created?

```
Outer( int n ) {
  contained = Inner(n);
};
```

1. This first calls the default constructor
2. then calls the *Inner(n)* constructor,
3. then copies the result over the *contained* member.

```
Outer( int n )
  : contained(Inner(n)) {
    /* ... */
};
```

1. This creates the *Inner(n)* object,
2. placed it in the *contained* member,
3. does the rest of the constructor, if any.

Internal access

4. Direct alteration of internals

Return a reference to a private member:

```
1 class Point {
2 private:
3   double x,y;
4 public:
5   double &x_component() { return x; };
6 };
7 int main() {
8   Point v;
9   v.x_component() = 3.1;
10 }
```

Only define this if you need to be able to alter the internal entity.

5. Reference to internals

Returning a reference saves you on copying.
Prevent unwanted changes by using a 'const reference'.

```
1  class Grid {
2  private:
3    vector<Point> thepoints;
4  public:
5    const vector<Point> &points() const {
6      return thepoints; };
7  };
8  int main() {
9    Grid grid;
10   cout << grid.points()[0];
11   // grid.points()[0] = whatever ILLEGAL
12 }
```

6. Access gone wrong

We make a class for points on the unit circle

```
1  // object/unit.cpp
2  class UnitCirclePoint {
3  private:
4    float x,y;
5  public:
6    UnitCirclePoint(float x) {
7      setx(x); };
8    void setx(float newx) {
9      x = newx; y = sqrt(1-x*x);
10   };
```

You don't want to be able to change just one of x, y!
In general: enforce invariants on the members.

7. Const functions

A function can be marked as const:
it does not alter class data,
only changes are through return and parameters

8. ‘this’ pointer to the current object

```
1 class Myclass {
2 private:
3   int myint;
4 public:
5   Myclass(int myint) {
6     this->myint = myint; // `this' redundant!
7   };
8 };
```

9. 'this' use

You don't often need the `this` pointer. Example: you need to call a function inside a method that needs the object as argument)

```
1 /* forward definition: */ class someclass;
2 void somefunction(const someclass &c) {
3   /* ... */ }
4 class someclass {
5 // method:
6 void somemethod() {
7   somefunction(*this);
8 };
```

(Rare use of dereference star)

Headers

10. C headers plusplus

You know how to use `.h` files in C.

Classes in C++ need some extra syntax.

11. Data members in proto

Data members, even private ones, need to be in the header file:

```
1 class something {
2 private:
3   int localvar;
4 public:
5   // declaration:
6   double somedo(vector);
7 };
```

Implementation file:

```
1 // definition
2 double something::somedo(vector v) {
3     .... something with v ....
4     .... something with localvar ....
5 };
```

12. Static class members

A static member acts as if it's shared between all objects.
(Note: C++17 syntax)

Code:

```
1 // link/static17.cpp
2 class myclass {
3 private:
4   static inline int count=0;
5 public:
6   myclass() { ++count; };
7   int create_count() {
8     return count; };
9 };
10    /* ... */
11  myclass obj1,obj2;
12  cout << "I have defined "
13       << obj1.create_count()
14       << " objects" << '\n';
```

Output:

```
I have defined 2 objects
```

13. Static class members, C++11 syntax

```
1 // link/static.cpp
2 class myclass {
3 private:
4   static int count;
5 public:
6   myclass() { ++count; };
7   int create_count() { return count; };
8 };
9     /* ... */
10 // in main program
11 int myclass::count=0;
```