More Objects

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: September 28, 2025

Interaction between objects

1. Methods that create a new object

Code:

```
// geom/pointscale.cpp
class Point {
    /* ... */
  Point scale( float a ) {
    Point scaledpoint( x*a, y*a );
    return scaledpoint;
  };
    /* ... */
  println("p1 to origin {:.5}",
          p1.dist_to_origin());
  Point p2 = p1.scale(2.);
  println("p2 to origin {:.5}",
          p2.dist_to_origin());
```

Output:

```
p1 to origin 2.2361
p2 to origin 4.4721
```

2. Anonymous objects

Create a point by scaling another point:

```
new_point = old_point.scale(2.81);
```

Two ways of handling the `return` statement of the `scale` method:

Naive:

```
// geom/pointscale.cpp
Point Point::scale( float a ) {
  Point scaledpoint =
    Point( x*a, y*a );
  return scaledpoint;
};
```

Concise:

```
// geom/pointscale.cpp
Point Point::scale( float a ) {
  return Point( x*a, y*a );
};
```

'move semantics' and 'copy elision':
compiler is pretty good at avoiding copies

Exercise 1

Write a method `halfway` that, given two `Point` objects `p,q`, construct the `Point` halfway, that is, $(p+q)/2$:

```
Point p(1,2.2), q(3.4,5.6);
Point h = p.halfway(q);
```

You can write this function directly, or you could write functions `Add` and `Scale` and combine these.
(Later you will learn about operator overloading.)

How would you print out a `Point` to make sure you compute the halfway point correctly?

3. Using the default constructor

No constructor explicitly defined;
You recognize the default constructor in the main by the fact that an object is defined without any parameters.

Code:

```
// object/default.cpp
class IamOne {
private:
  int i=1;
public:
  void print() {
    println( "{}",i );
  };
};
    /* ... */
  IamOne one;
  one.print();
```

Output:

```
1
```

4. Default constructor

Refer to `Point` definition above.
Consider this code that looks like variable declaration, but for objects:

```
Point p1(1.5, 2.3);
Point p2;
p2 = p1.scaleby(3.1);
```

Compiling gives an error (g++; different for intel):

```
pointdefault.cpp: In function 'int main()':
pointdefault.cpp:32:21: error: no matching function for call to
                'Point::Point()'
```

5. Default constructor

The problem is with *p2*:

```
Point p1(1.5, 2.3);
Point p2;
```

- *p1* is created with your explicitly given constructor;
- *p2* uses the default constructor:

```
Point() {};
```

- default constructor is there by default, unless you define another constructor.
- you can re-introduce the default constructor:

```
// geom/pointdefault.cpp
Point() = default;
Point( float x,float y )
  : x(x),y(y) {};
```

(but often you can avoid needing it)

6. Other way

State that the default constructor exists with the `default` keyword:

```
// object/default.cpp
Point() = default;
Point( double x,double y )
  : x(x),y(y) {};
```

State that there should be no default constructor with the `delete` keyword:

```
Point() = delete;
```

Exercise 2

Make a class *LinearFunction* with a constructor:

```
LinearFunction( Point input_p1,Point input_p2 );
```

and a member function

```
float evaluate_at( float x );
```

which you can use as:

```
LinearFunction line(p1,p2);
cout << "Value at 4.0: " << line.evaluate_at(4.0) << endl;
```

7. Classes for abstract objects

Objects can model fairly abstract things:

Code:

```
// object/stream.cpp
class Stream {
private:
  int last_result{0};
public:
  int next() {
    return last_result++; };
};

int main() {
  Stream ints;
  println( "Next: {}",
    ints.next() );
  println( "Next: {}",
    ints.next() );
  println( "Next: {}",
    ints.next() );
```

Output:

```
Next: 0
Next: 1
Next: 2
```

8. Preliminary to the following exercise

A prime number generator has:
an API of just one function: `nextprime`

To support this it needs to store:
an integer `last_prime_found`

Programming Project Exercise 3

Write a class *primegenerator* that contains:

- Methods *number_of_primes_found* and *nextprime*;
- Also write a function *isprime* that does not need to be in the class.

Your main program should look as follows:

```cpp
// primes/6primesbyclass.cpp
cin >> nprimes;
primegenerator sequence;
while (sequence.number_of_primes_found()<nprimes) {
  int number = sequence.nextprime();
  cout << "Number " << number << " is prime" << '\n';
}
```

Programming Project Exercise 4

Write a program to test the Goldbach conjecture for the even numbers up to a bound that you read in.

First formulate the quantor structure of this statement, then translate that top-down to code, using the generator you developed above.

1. Make an outer loop over the even numbers e.
2. For each e, generate all primes p.
3. From $p + q = e$, it follows that $q = e - p$ is prime: test if that q is prime.

For each even number `e` then print `e,p,q`, for instance:

```
The number 10 is 3+7
```

If multiple possibilities exist, only print the first one you find.

9. A Goldbach corollary

The Goldbach conjecture says that every even number $2n$ (starting at 4), is the sum of two primes $p+q$:

$$2n = p + q.$$

Equivalently, every number n is equidistant from two primes:

$$n = \frac{p+q}{2} \qquad \text{or} \qquad q - n = n - p.$$

In particular this holds for each prime number:

$$\forall_{r\,\mathrm{prime}} \exists_{p,q\,\mathrm{prime}} \colon r = (p+q)/2 \text{ is prime.}$$

We now have the statement that each prime number is the average of two other prime numbers.

Programming Project Exercise 5

Write a program that tests this. You need at least one loop that tests all primes r; for each r you then need to find the primes p, q that are equidistant to it.

Use your prime generator. Do you use two generators for this, or is one enough? Do you need three, for p, q, r?

For each r value, when the program finds the p, q values, print the p, q, r triple and move on to the next r.