Class inheritance: is-a

Victor Eijkhout and Charlie Dey

spring 2017

General case, special case

You can have classes where an object of one class is a special case of the other class. You declare that as

```
class General {
protected: // note!
 int g;
public:
 void general_method() {};
};
class Special : public General {
public:
  void special_method() { g = ... };
};
```

Inheritance: derived classes

'Derived' class `Special` 'inherits' methods and data from 'base class' `General`:

```
int main() {
  Special special_object;
  special_object.general_method();
```

Data needs to be `protected`, not `private`, to be inheritable.

Constructors

When you run the special case constructor, usually the general case needs to run too. By default the 'default constructor', but:

```
class General {
public:
  General( double x,double y ) {};
};
class Special : public General {
public:
  Special( double x ) : General(x,x+1) {};
};
```

Exercise 1

Take your code where a `Rectangle` was defined from one point, width, and height.

Make a class `Square` that inherits from `Rectangle`. It should have the function `area` defined, inherited from `Rectangle`.

First ask yourself: what should the constructor of a `Square` look like?

Exercise 2

Revisit the `LinearFunction` class. Add methods `slope` and `intercept`.

Now generalize `LinearFunction` to `StraightLine` class. These two are almost the same except for vertical lines. The `slope` and `intercept` do not apply to vertical lines, so design `StraightLine` so that it stores the defining points internally. Let `LinearFunction` inherit.

More

- Multiple inheritance: an X is-a A, but also is-a B.
  This mechanism is somewhat dangerous.
- Virtual base class: you don't actually define a function in the base class, you only say 'any derived class has to define this function'.

Back to prime numbers

Exercise 3

Write a class `primesequence` that contains the members of the structure, and the functions `nextprime`, `isprime`. The function `nextprime` does not need the structure as argument, because the structure members are in the class, and therefore global to that function.

Your main program should look as follows:

```
primesequence sequence;
while (sequence.numberfound<nprimes) {
  int number = sequence.nextprime();
  cout << "Number " << number << " is prime" << endl;
}
```

to remind you. . .

Exercise 4

Rewrite the exercise that found a predetermined number of primes, putting the `number_of_primes_found` and `last_number_tested` variables in a structure. Your main program should now look like:

```
struct primesequence sequence;
while (sequence.number_of_primes_found<nprimes) {
  int number = nextprime(sequence);
  cout << "Number " << number << " is prime" << endl;
}
```

and to see if you really understand this. . .

Exercise 5

The *Goldbach conjecture* says that every even number, from 4 on, is the sum of two primes $p + q$. Write a program to test this for the even numbers up to 20 million.

Make an outer loop over the even numbers. In each iteration, make a `primesequence` object to generate p values. Then, for each p, make a second `primesequence` object to generate q values, and test with these.

For each even number, print out how it is the sum of two primes. If multiple possibilities exist, only print the first one you find.