```
 1 class Date {
 2     int d, m, y;
 3 public:
 4     Date(int day, int month, int year) {
 5         d = day;
 6         m = month;
 7         y = year;
 8     }
 9     int day() {
10         return d;
11     }
12     int month() {
13         return this->m;
14     }
15     int year() {
16         return y;
17     }
18 };
```

Please criticize this code.

Concrete types (or value types)

There are several things to comment here:

- probably not a good idea to define several variables on line 2
- not using member initializer properly
- possible confusion about day, day(), and d
- a const after the argument list indicates that these functions do no modify the state of a Date object (query method)
- a non-const methods indicates that the state of an object changes (modifier method)
- how to handle invalid Date?
- do we need copy constructor and assignment operator?
- the use of this-> on line 13 is not necessary, but some prefer to write this-> for clearity.
- it often makes sence to place data members last to empasize the funtions providing the public user interface [p234]

foo.hpp

```
1  #include <iostream>
2
3  class Foo {
4      int value;
5      int read_counter;
6  public:
7      Foo(int v) : value(v), read_counter(0) { }
8      int value();
9      void debug_print_read_counter();
10 };
```

foo.cpp

```
1  #include "foo.hpp"
2
3  int Foo::value() const {
4      read_counter++;
5      return value;
6  }
7
8  void Foo::debug_print_read_counter() {
9      std::cerr << "read_counter = "
10               << read_counter << std::endl;
11 }
```

main.cpp

```
1  #include "foo.hpp"
2
3  int main() {
4      Foo f(42);
5      f.debug_print_read_counter();
6      f.value();
7      f.value();
8      f.debug_print_read_counter();
9  }
```

```
g++ foo.cpp main.cpp && ./a.out
```

This code does not compile and it smells bad. How can you fix it? Please criticize.

mutable, name collisions

```
- iosfwd
- conflicting redeclaration of Foo::value() and Foo::value
- consider underscore postfixing of member variables
- declaration on foo.hpp:8 and definition on foo.cpp:3 does not match
- logical constness (p241). consider making read_counter mutable
- debug_print_read_counter should be const, indicating that it is a query
- there is a cohesion problem with Foo.cpp
```

Once fixed, it might print:

```
read_counter = 0
read_counter = 2
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct B {
 6     B() { p('B'); }
 7     ~B() { p('b'); }
 8 };
 9
10 class A {
11     B * v;
12 public:
13     A() { p('A'); v = new B[3]; }
14     ~A() { p('a'); delete v; }
15 };
16
17 int main() {
18     A a1;
19     A a2 = a1;
20     A a3;
21     a3 = a1;
22 }
```

What might happen if you try to compile, link and run this program? Please criticize.

new[]/delete[], the rule of three On my machine I get:

```
g++ -Wall scratch.cpp && ./a.out
ABBBABBBababab
```

- if you new an array you must delete an array
- the rule of three, if you need a destructor, you probably need a copy constructor and copy assignment operator
- deleting something twice is a serious error; the behaviour is undefined and most likely disastrous
- should the destructor of A be virtual?

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct A {
 6     A() { p('A'); }
 7     ~A() { p('a'); }
 8 };
 9
10 struct B {
11     B() { p('B'); }
12     ~B() { p('b'); }
13 };
14
15 class C {
16     A a;
17     B b;
18 public:
19     C() : b(), a() {}
20     C(int) {}
21 };
22
23 int main() {
24     C c1;
25     C c2(4);
26 }
```

what might happen if you try to compile, link and run this program?

initialization of objects

On my machine I get:

```
g++ -Wall scratch.cpp && ./a.out
scratch.cpp: In constructor 'C::C()':
scratch.cpp:17: warning: 'C::b' will be initialized after
scratch.cpp:16: warning:   'A C::a'
scratch.cpp:19: warning:   when initialized here
ABABbaba
```

Without -Wall, no warning is printed.

If a member constructor needs no argument, the member need not to be mentioned in the member initializer list. what about native type members?

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct A {
 6     int value;
 7     A() { p(0); }
 8     A(int v) : value(v) { p(1); }
 9     ~A() { p(2); }
10     A(const A & a) { p(3); }
11     void operator=(const A & a) { p(4); }
12 };
13
14 struct B {
15     A a;
16     B(int v) { p('B'); a=42; p('b'); }
17 };
18
19 struct C {
20     A a;
21     C(int v) : a(v) { p('C'); p('c'); }
22 };
23
24 int main() {
25     p('-');
26     B b(42);
27     p('-');
28     C c(42);
29     p('-');
30 }
```

What might happen if you try to compile, link and run this program?

using member initializer

```
g++ -Wall scratch.cpp && ./a.out
-0B142b-1Cc-22
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct X {
 6     int a;
 7     int b;
 8     X() : a(1), b(2) {}
 9 };
10
11 struct Y {
12     int c;
13     int d;
14     Y() : c(3), d(4) {}
15 };
16
17 int main() {
18     X x;
19     new (&x) Y;
20     p(x.a);
21     p(x.b);
22 }
```

what might happen if you try to compile, link and run this program?

placement

```
g++ -Wall scratch.cpp && ./a.out
34
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct X {
 6     int value;
 7     X(int v) : value(v) { p('X'); p(v); }
 8     ~X() { p('x'); }
 9     X(const X & x) : value(x.value) { p(2); }
10     X operator+(const X & x) { p(3); return this->value + x.value; }
11 };
12
13 int main() {
14     X a(4);
15     p('-');
16     p( (a + a).value );
17     p('-');
18     X b = (a + a);
19     p('-');
20     p(b.value);
21     p('-');
22 }
```

what might happen if you try to compile, link and run this program?

temporary objects

```
g++ -Wall scratch.cpp && ./a.out
X4-3X88x-3X8-8-xx
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct X {
 6     X(int v) { p('X'); p(v); }
 7     ~X() { p('x'); }
 8 };
 9
10 void foo(X x) {
11     p('F');
12 }
13
14 int main() {
15     foo(4);
16 }
```

what might happen if you try to compile, link and run this program?

implicit conversion

```
g++ -Wall scratch.cpp && ./a.out
X4Fx
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct X {
 6     int value;
 7     X(int v) : value(v) { }
 8     operator int() const { return value; }
 9 };
10
11 void foo(int v) {
12     p(v);
13 }
14
15 int main() {
16     X x = 42;
17     foo(x);
18 }
```

what might happen if you try to compile, link and run this program?

conversion operator

```
g++ -Wall scratch.cpp && ./a.out
42
```

What happens if you put explicit in front of line 7? What would you then need to write on line 16?

```
 1 #include <iostream>
 2 #include <vector>
 3
 4 class X {
 5     int value;
 6 public:
 7     X(int v) : value(v) {}
 8     void operator()(int & i) const {
 9         i += value;
10     }
11 };
12
13 void p(int & i) {
14     std::cout << i << std::endl;
15 }
16
17 int main() {
18     std::vector<int> v;
19     v.push_back(1);
20     v.push_back(2);
21     v.push_back(3);
22     for_each(v.begin(), v.end(), X(42));
23     for_each(v.begin(), v.end(), p);
24 }
```

what might happen if you try to compile, link and run this program?

functor

```
g++ -Wall scratch.cpp && ./a.out
43
44
45
```