```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class A { };
 6 class B : public A { };
 7 class C : public A { };
 8
 9 void foo() {
10     p("1");
11     throw C();
12 }
13
14 int main() {
15     try {
16         p("2");
17         foo();
18         p("3");
19     } catch(B) {
20         p("4");
21     } catch(C) {
22         p("5");
23     } catch(A) {
24         p("6");
25     }
26     p("7");
27 }
```

What might happen if you try to compile this code?

§14.2, Grouping of Exceptions

```
g++ scratch.cpp && ./a.out
2157
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class A {
 6 protected:
 7     void foo() { p("foo"); };
 8 };
 9
10 class B : virtual public A {
11 public:
12     void bar() { p("bar"); };
13 };
14
15 int main() {
16     B b;
17     b.foo();
18 }
```

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

§15.3.2.2, Using-Declarations and Access Control

I get:

```
g++ -Wall scratch.cpp && ./a.out
scratch.cpp: In function 'int main()':
scratch.cpp:7: error: 'void A::foo()' is protected
scratch.cpp:17: error: within this context
```

How con you make this code compile and print "foo"? Eg, by adding "using A::foo;" in the public part of B.

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class A {
 6 public:
 7     virtual void boo() { p("A"); };
 8 };
 9
10 class B : public A {
11 public:
12     virtual void boo() { p("B"); };
13 };
14
15 int main() {
16     try {
17         throw B();
18     } catch( A a ) {
19         a.boo();
20     }
21 }
```

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

§14.2, Derived Exceptions

```
g++ -Wall scratch.cpp && ./a.out
scratch.cpp:5: warning: 'class A' has virtual functions but non-virtual destructor
scratch.cpp:10: warning: 'class B' has virtual functions but non-virtual destructor
A
```

How to make this code write "B"? Catch by reference

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class A { };
 6
 7 int main() {
 8     p("0");
 9     try {
10         p("1");
11         try {
12             p("2");
13             throw A();
14             p("3");
15         } catch(...) {
16             p("4");
17             throw;
18             p("5");
19         }
20     } catch(A a) {
21         p("6");
22         throw;
23         p("7");
24     }
25     p("8");
26 }
```

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

14.3.1 Re-Throw

```
g++ -Wall scratch.cpp && ./a.out
terminate called after throwing an instance of 'A'
01246
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class B {
 6 public:
 7     B() { p("B"); }
 8     ~B() { p("b"); }
 9 };
10
11 class A {
12     B * v;
13 public:
14     A() { p("A"); v = new B[3]; throw "x"; }
15     ~A() { p("a"); delete[] v; }
16 };
17
18 int main() {
19     try {
20         p("0");
21         A a;
22         p("1");
23     } catch( const char * s ) {
24         p(s);
25     }
26 }
```

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

§14.4.1, RAII

```
0ABBBx
```

What if you remove the throw on line 14?

```
0ABBB1abbb
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 class B {
 6 public:
 7     B() { p("B"); throw "x"; }
 8     ~B() { p("b"); }
 9 };
10
11 class A {
12     B b;
13 public:
14     A() try : b() { p("A"); } catch(...) { p("y"); }
15     ~A() { p("a"); }
16 };
17
18 int main() {
19     try {
20         p("0");
21         A a;
22         p("1");
23     } catch( const char * s ) {
24         p(s);
25     }
26 }
```

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

§14.4.6.1, Exceptions and Member Initialization

```
g++ -Wall scratch.cpp && ./a.out
0Byx
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 void foo() throw() {
 6     throw "x";
 7 };
 8
 9 int main() {
10     try {
11         p("0");
12         foo();
13         p("1");
14     } catch( const char * s ) {
15         p(s);
16     }
17 }
```

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

§14.6, Exception Specification

```
g++ -Wall scratch.cpp && ./a.out
terminate called after throwing an instance of 'char const*'
0
```

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct B {
 6     void foo(int) { p(1); }
 7 };
 8
 9 struct C : B {
10     void foo(float) { p(2); }
11 };
12
13 int main() {
14     C c;
15     c.foo(42);
16 }
```

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

§15.2.2, Inheritance and Using-Declarations

```
g++ -Wall scratch.cpp && ./a.out
2
```

How to make this print 1? Add "using B::foo" after line 9.

```
 1 #include <iostream>
 2
 3 template<typename T> void p(T x) { std::cout << x; }
 4
 5 struct A {
 6     A() { p('A'); }
 7 };
 8
 9 struct B : A {
10     B() { p('B'); }
11 };
12
13 struct C : virtual A {
14     C() { p('C'); }
15 };
16
17 struct D : B, C {
18     D() { p('D'); }
19 };
20
21 int main() {
22     D d;
23 }
```

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

§15.2.4, Virtual Base Classes

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

What if we add "virtual" on line 9? ABCD What if we remove "virtual" on line 13? ABACD

```
 1 #include <iostream>
 2 #include <typeinfo>
 3
 4 template<typename T> void p(T x) { std::cout << x; }
 5
 6 struct A {
 7     virtual void foo() { p("A"); }
 8 };
 9
10 struct B : A {
11     virtual void foo() { p("B"); }
12 };
13
14 struct C : A {
15     virtual void foo() { p("C"); }
16 };
17
18 int main() {
19     A * b = new B;
20     A * c = new C;
21     b = c;
22     b->foo();
23     b = dynamic_cast<B*>(c);
24     b->foo();
25 }
```

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

§x.x, title

I got:

```
g++ -Wall scratch.cpp && ./a.out
scratch.cpp:6: warning: 'struct A' has virtual functions but non-virtual destructor
scratch.cpp:10: warning: 'struct B' has virtual functions but non-virtual destructor
scratch.cpp:14: warning: 'struct C' has virtual functions but non-virtual destructor

Compilation exited abnormally with code 138 at Fri Oct 26 02:14:06
```

Why does it crash? Because b is 0 on line 24.

I did expect that it printed a "C", but std is not flushed so it is stuck in the buffer. Ass a std::flush to line 4 and it should print "C" before crashing.

Consider line 23 which gives b = 0. Is there a useful way of using dynamic cast like this? To determine if some object is of a particular type.

What if we replace line 23 with a static cast or just plain old cast? Then you get "CC"

What if you remove "virtual" on line 7,11,15? Then you get a compile error:

```
g++ -Wall scratch.cpp && ./a.out
scratch.cpp: In function 'int main()':
scratch.cpp:23: error: cannot dynamic_cast 'c' (of type 'struct A*') to type 'struct B*'
                (source type is not polymorphic)
```

That is quite handy.

```
 1 #include <iostream>
 2 #include <typeinfo>
 3
 4 struct Foo {
 5 };
 6
 7 struct Bar : Foo {
 8 };
 9
10 struct Gaz : Foo {
11 };
12
13 int main() {
14     Foo * b = new Bar;
15     Foo * c = new Gaz;
16     std::cout << typeid(*b).name();
17     std::cout << typeid(*c).name();
18 }
```

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

§15.4.4, Typeid and Extended Type Info

I get:

```
g++ -Wall scratch.cpp && ./a.out
3Foo3Foo
```

How can you get this code write out "3Bar3Gaz"? Add a dummy virtual function to Foo.