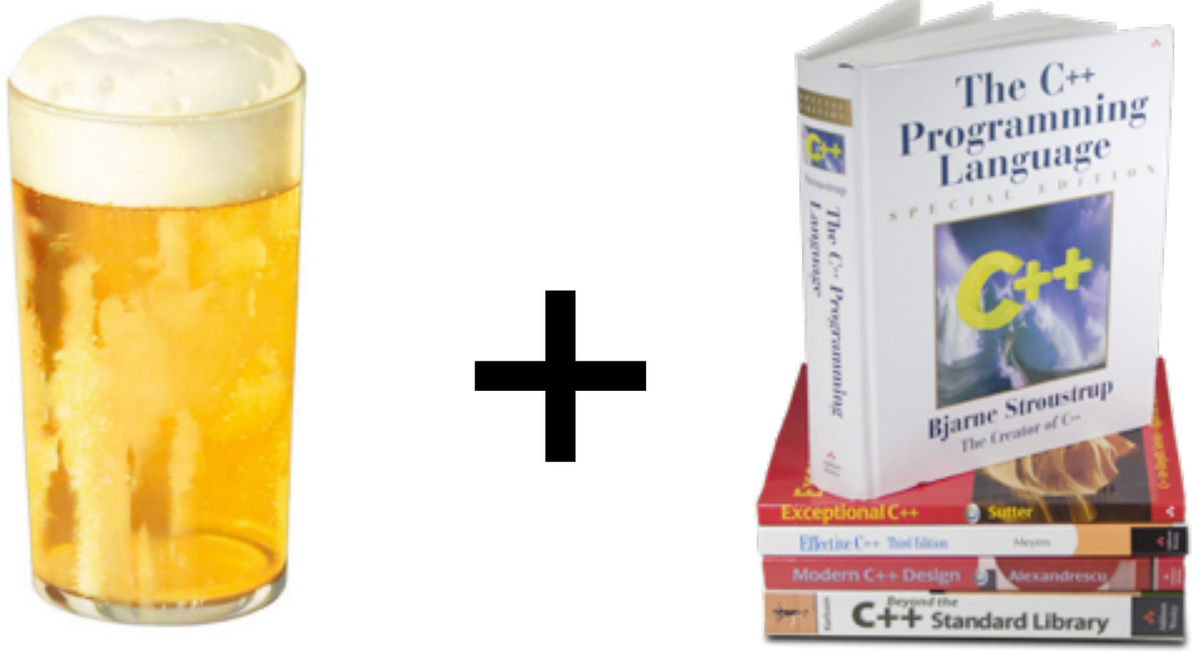
C++ Pub Quiz

by Olve Maudal



A 90 minute quiz session
Oslo C++ Users Group Meetup
October 2011

Here is my development environment:

```
oma@ubuntu:~/quiz$ uname -a
Linux ubuntu 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:40:53 UTC 2011 i686 GNU/Linux
oma@ubuntu:~/quiz$ gcc --version
gcc (GCC) 4.7.0 20111012 (experimental)
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Here is how I compile and run the snippets:

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

Here is my development environment:

```
oma@ubuntu:~/quiz$ uname -a
Linux ubuntu 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:40:53 UTC 2011 i686 GNU/Linux
oma@ubuntu:~/quiz$ gcc --version
gcc (GCC) 4.7.0 20111012 (experimental)
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Here is how I compile and run the snippets:

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

The question for all code snippets is:

Here is my development environment:

```
oma@ubuntu:~/quiz$ uname -a
Linux ubuntu 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:40:53 UTC 2011 i686 GNU/Linux
oma@ubuntu:~/quiz$ gcc --version
gcc (GCC) 4.7.0 20111012 (experimental)
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Here is how I compile and run the snippets:

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

The question for all code snippets is:

*What will happen on **my** machine?*

Here is my development environment:

```
oma@ubuntu:~/quiz$ uname -a
Linux ubuntu 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:40:53 UTC 2011 i686 GNU/Linux
oma@ubuntu:~/quiz$ gcc --version
gcc (GCC) 4.7.0 20111012 (experimental)
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Here is how I compile and run the snippets:

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

The question for all code snippets is:
*What will happen on **my** machine?*

Full score is given if you manage to guess:

Here is my development environment:

```
oma@ubuntu:~/quiz$ uname -a
Linux ubuntu 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:40:53 UTC 2011 i686 GNU/Linux
oma@ubuntu:~/quiz$ gcc --version
gcc (GCC) 4.7.0 20111012 (experimental)
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Here is how I compile and run the snippets:

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

The question for all code snippets is:
*What will happen on **my** machine?*

Full score is given if you manage to guess:
*Whatever happens on **my** machine!*

There are no trick questions here, most/all of the code snippets do produce the expected result and should be quite easy if you really understand 🍺++

There are no trick questions here, most/all of the code snippets do produce the expected result and should be quite easy if you really understand 🍺++

PS: All the code snipppets do indeed compile, link and run on ***my*** machine. There are no missing semicolons or syntax errors.

Disclaimer: the code snippets here are all crap examples of how to write C++. This is just for fun.

Disclaimer: the code snippets here are all crap examples of how to write C++. This is just for fun.

Remember, this is **not** about C++, nor G++, it is about:

Disclaimer: the code snippets here are all crap examples of how to write C++. This is just for fun.

Remember, this is **not** about C++, nor G++, it is about:

++

Questions

(45 minutes)

#1

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x * 2; }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int main()
{
    int a[]{1,2,3,4};
    P(0);
    for (auto x : a)
        P(x);
    P(9);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct Y {
    Y(int) { P(0); }
    Y(Y &) { P(1); }
    Y() { P(2); }
    ~Y() { P(3); }
    void operator=(const Y &) { P(4); }
};

struct X {
    Y v;
    X(int val) { P(5); v = val; P(6); }
    ~X() { P(7); }
};

int main()
{
    P(8);
    X(9);
    P('a');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

int main()
{
    P(0);
    X a(1);
    X b(2);
    X c(3);
    P(4);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

static X a(1);
X b(2);

int main()
{
    X e(5);
    static X f(6);
}
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

#7

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

class X {
    int value;
public:
    X() : value(42) { P('a'); }
    ~X() { P('b'); }
    X(const X & f) : value(f.value) { P('c'); }
    X & operator=(const X & f) { P('d'); value = f.value; return *this; }
    X operator++(int) { P('e'); X old(*this); ++*this; return old;}
    X & operator++() { P('f'); value += 4; return *this; }
};

int main() {
    X f1; P('-');
    ++f1; P('-');
    f1++; P('-');
    X f2 = f1; P('-');
    f2 = f1; P('-');
}
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

void f(double) { P(1); }
void f(float) { P(2); }
void f(std::string) { P(3); }
void f(const std::string &) { P(4); }
void f(const char *) { P(5); }
void f(char *) { P(6); }

int main() {
    f(1.9);
    f("hello");
}
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

#define MAX(a,b) a > b ? a : b
#define MIN(a,b) (((a)<(b))?(a):(b))

int main() {
    int a=2, b=3, c=0;
    c = 1 + MAX(a,b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int value;
    X(int v) : value(v) { P('X'); P(v); }
    ~X() { P('x'); }
    X(const X & x) : value(x.value) { P(2); }
    X operator+(const X & x) { P(3); return this->value + x.value; }
};

int main() {
    X a(4);
    P('-');
    P( (a + a).value );
    P('-');
    X b = (a + a);
    P('-');
    P(b.value);
    P('-');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    X(char v) { P('X'); P(v); }
    ~X() { P('x'); }
};

void foo(X x) {
    P('F');
}

int main() {
    foo(65);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
    ~A() { P(1); }
};

struct B : A {
    B() { P(2); }
    ~B() { P(3); }
};

struct C {
    C() { P(4); }
    virtual ~C() { P(5); }
};

struct D : C {
    D() { P(6); }
    ~D() { P(7); }
};

int main() {
    A * a = new B;
    delete a;
    P('-');
    C * c = new D;
    delete c;
}
```

#15

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
};

struct B : A {
    B() { P(2); }
};

struct C : virtual A {
    C() { P(4); }
};

struct D : B, C {
    D() { P(6); }
};

struct E : C, B{
    E() { P(8); }
};

int main() {
    D d;
    P('-');
    E e;
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X { X() { P('0'); } ~X() { P(1); } };

int main() {
    try {
        P(5);
        throw X();
        P(6);
    } catch (X e) {
        P(7);
    }
    P(8);
}
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int j = 3;

int main()
{
    P(j);
    [&](){j++; P(j);}();
    P(j);
    [=](){j++; P(j);}();
    P(j);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }
template<typename T> void foo(T *) { P(2); }
template<typename T> void foo(T &&) { P(4); }
template<> void foo(int&) { P(5); }

int main()
{
    int a = 65;
    foo(a);
    foo(&a);
    char b = 'a';
    foo(b);
    foo(&b);
}
```

Answers

3 points for correct answer

1 point if only a minor mistake

0 point if the answer is not correct

For some of the answers there are bonus points.

Bonus point if you think you might get less than 100% score on this quiz.

#1

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x * 2; }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x * 2; }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x * 2; }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12326
```

#1

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x * 2; }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
}
```

Bonus point if you discussed
evaluation order

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12326
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int main()
{
    int a[]{1,2,3,4};
    P(0);
    for (auto x : a)
        P(x);
    P(9);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int main()
{
    int a[]{1,2,3,4};
    P(0);
    for (auto x : a)
        P(x);
    P(9);
}
```

```
$ g++ --std=c++0x -Wall -O foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int main()
{
    int a[]{1,2,3,4};
    P(0);
    for (auto x : a)
        P(x);
    P(9);
}
```

```
$ g++ --std=c++0x -Wall -O foo.cpp && ./a.out
012349
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct Y {
    Y(int) { P(0); }
    Y(Y &) { P(1); }
    Y() { P(2); }
    ~Y() { P(3); }
    void operator=(const Y &) { P(4); }
};

struct X {
    Y v;
    X(int val) { P(5); v = val; P(6); }
    ~X() { P(7); }
};

int main()
{
    P(8);
    X(9);
    P('a');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct Y {
    Y(int) { P(0); }
    Y(Y &) { P(1); }
    Y() { P(2); }
    ~Y() { P(3); }
    void operator=(const Y &) { P(4); }
};

struct X {
    Y v;
    X(int val) { P(5); v = val; P(6); }
    ~X() { P(7); }
};

int main()
{
    P(8);
    X(9);
    P('a');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct Y {
    Y(int) { P(0); }
    Y(Y &) { P(1); }
    Y() { P(2); }
    ~Y() { P(3); }
    void operator=(const Y &) { P(4); }
};

struct X {
    Y v;
    X(int val) { P(5); v = val; P(6); }
    ~X() { P(7); }
};

int main()
{
    P(8);
    X(9);
    P('a');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
825043673a
```

#3

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct Y {
    Y(int) { P(0); }
    Y(Y &) { P(1); }
    Y() { P(2); }
    ~Y() { P(3); }
    void operator=(const Y &) { P(4); }
};

struct X {
    Y v;
    X(int val) { P(5); v = val; P(6); }
    ~X() { P(7); }
};

int main()
{
    P(8);
    X(9);
    P('a');
}
```

Bonus point if you discussed the strange **signature** of the assignment operator

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
825043673a
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

int main()
{
    P(0);
    X a(1);
    X b(2);
    X c(3);
    P(4);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

int main()
{
    P(0);
    X a(1);
    X b(2);
    X c(3);
    P(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

int main()
{
    P(0);
    X a(1);
    X b(2);
    X c(3);
    P(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
01234321
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

static X a(1);
X b(2);

int main()
{
    X e(5);
    static X f(6);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

static X a(1);
X b(2);

int main()
{
    X e(5);
    static X f(6);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int v;
    X(int val) : v(val) { P(v); }
    ~X() { P(v); }
};

static X a(1);
X b(2);

int main()
{
    X e(5);
    static X f(6);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12565621
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp:6:16: warning: deprecated conversion from string
constant to ‘char*’ [-Wwrite-strings]
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp:6:16: warning: deprecated conversion from string
constant to ‘char*’ [-Wwrite-strings]
Foo Bar
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp:6:16: warning: deprecated conversion from string
constant to ‘char*’ [-Wwrite-strings]
Foo Bar
Zoo Bar
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp:6:16: warning: deprecated conversion from string
constant to ‘char*’ [-Wwrite-strings]
Foo Bar
Zoo Bar
/bin/bash: line 1: 16973 Segmentation fault      ./a.out
```

```
#include <iostream>

int main()
{
    char a[] = "Foo";
    char * b = "Bar";

    std::cout << a << " " << b << std::endl;

    a[0] = 'Z';
    std::cout << a << " " << b << std::endl;

    b[0] = 'C';
    std::cout << a << " " << b << std::endl;
}
```

Bonus point if you discussed the deprecated **string conversion**

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp:6:16: warning: deprecated conversion from string
constant to ‘char*’ [-Wwrite-strings]
Foo Bar
Zoo Bar
/bin/bash: line 1: 16973 Segmentation fault      ./a.out
```

#7

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
foo.cpp:21:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
foo.cpp:21:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
1237
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
foo.cpp:21:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
1237
2112
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
foo.cpp:21:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
1237
2112
0023
```

#7

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int f(int x) { P(x); return x; }

void g(int a, int b) { P(a); P(b); }

int main()
{
    int a = f(1) + f(2) * f(3);
    P(a);
    P('\n');

    g(f(1),f(2));
    P('\n');

    int v[4] = {};
    int i=2;
    v[i] = i++;
    v[i] = i++;
    for (int j : v)
        P(j);

}
```

Bonus point if you discussed
sequence points

Another bonus point if you discussed
unspecified behaviour vs
undefined behaviour

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:20:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
foo.cpp:21:15: warning: operation on ‘i’ may be undefined [-Wsequence-point]
1237
2112
0023
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

class X {
    int value;
public:
    X() : value(42) { P('a'); }
    ~X() { P('b'); }
    X(const X & f) : value(f.value) { P('c'); }
    X & operator=(const X & f) { P('d'); value = f.value; return *this; }
    X operator++(int) { P('e'); X old(*this); ++*this; return old;}
    X & operator++() { P('f'); value += 4; return *this; }
};

int main() {
    X f1; P('-');
    ++f1; P('-');
    f1++; P('-');
    X f2 = f1; P('-');
    f2 = f1; P('-');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

class X {
    int value;
public:
    X() : value(42) { P('a'); }
    ~X() { P('b'); }
    X(const X & f) : value(f.value) { P('c'); }
    X & operator=(const X & f) { P('d'); value = f.value; return *this; }
    X operator++(int) { P('e'); X old(*this); ++*this; return old;}
    X & operator++() { P('f'); value += 4; return *this; }
};

int main() {
    X f1; P('-');
    ++f1; P('-');
    f1++; P('-');
    X f2 = f1; P('-');
    f2 = f1; P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

class X {
    int value;
public:
    X() : value(42) { P('a'); }
    ~X() { P('b'); }
    X(const X & f) : value(f.value) { P('c'); }
    X & operator=(const X & f) { P('d'); value = f.value; return *this; }
    X operator++(int) { P('e'); X old(*this); ++*this; return old;}
    X & operator++() { P('f'); value += 4; return *this; }
};

int main() {
    X f1; P('-');
    ++f1; P('-');
    f1++; P('-');
    X f2 = f1; P('-');
    f2 = f1; P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
a-f-ecfb-c-d-bb
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

class X {
    int value;
public:
    X() : value(42) { P('a'); }
    ~X() { P('b'); }
    X(const X & f) : value(f.value) { P('c'); }
    X & operator=(const X & f) { P('d'); value = f.value; return *this; }
    X operator++(int) { P('e'); X old(*this); ++*this; return old;}
    X & operator++() { P('f'); value += 4; return *this; }
};

int main() {
    X f1; P('-');
    ++f1; P('-');
    f1++; P('-');
    X f2 = f1; P('-');
    f2 = f1; P('-');
}
```

Bonus point if you discussed
return value optimization (RVO)

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
a-f-ecfb-c-d-bb
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

void f(double) { P(1); }
void f(float) { P(2); }
void f(std::string) { P(3); }
void f(const std::string &) { P(4); }
void f(const char *) { P(5); }
void f(char *) { P(6); }

int main() {
    f(1.9);
    f("hello");
}
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

void f(double) { P(1); }
void f(float) { P(2); }
void f(std::string) { P(3); }
void f(const std::string &) { P(4); }
void f(const char *) { P(5); }
void f(char *) { P(6); }

int main() {
    f(1.9);
    f("hello");
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

void f(double) { P(1); }
void f(float) { P(2); }
void f(std::string) { P(3); }
void f(const std::string &) { P(4); }
void f(const char *) { P(5); }
void f(char *) { P(6); }

int main() {
    f(1.9);
    f("hello");
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
15
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

#define MAX(a,b) a > b ? a : b
#define MIN(a,b) (((a)<(b))?(a):(b))

int main() {
    int a=2, b=3, c=0;
    c = 1 + MAX(a,b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
}
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

#define MAX(a,b) a > b ? a : b
#define MIN(a,b) (((a)<(b))?(a):(b))

int main() {
    int a=2, b=3, c=0;
    c = 1 + MAX(a,b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

#define MAX(a,b) a > b ? a : b
#define MIN(a,b) (((a)<(b))?(a):(b))

int main() {
    int a=2, b=3, c=0;
    c = 1 + MAX(a,b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
233-444-566-
```

```
#include <iostream>
#include <string>

template<typename T> void P(T x) { std::cout << x; }

#define MAX(a,b) a > b ? a : b
#define MIN(a,b) (((a)<(b))?(a):(b))

int main() {
    int a=2, b=3, c=0;
    c = 1 + MAX(a,b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
    c = MIN(++a,++b);
    P(a); P(b); P(c); P('-');
}
```

Bonus point if you discussed if the use of MIN macro introduced a **sequence point violation**

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
233-444-566-
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In constructor ‘C::C()’:
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In constructor ‘C::C()’:
foo.cpp:17:7: warning: ‘C::b’ will be initialized after [-Wreorder]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In constructor ‘C::C()’:
foo.cpp:17:7: warning: ‘C::b’ will be initialized after [-Wreorder]
foo.cpp:16:7: warning:   ‘A C::a’ [-Wreorder]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In constructor ‘C::C()’:
foo.cpp:17:7: warning: ‘C::b’ will be initialized after [-Wreorder]
foo.cpp:16:7: warning:   ‘A C::a’ [-Wreorder]
foo.cpp:18:5: warning:   when initialized here [-Wreorder]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P('A'); }
    ~A() { P('a'); }
};

struct B {
    B() { P('B'); }
    ~B() { P('b'); }
};

struct  C {
    A a;
    B b;
    C() : b(), a() {}
    C(int) {}
};

int main() {
    C c1;
    C c2(4);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In constructor ‘C::C()’:
foo.cpp:17:7: warning: ‘C::b’ will be initialized after [-Wreorder]
foo.cpp:16:7: warning:   ‘A C::a’ [-Wreorder]
foo.cpp:18:5: warning:   when initialized here [-Wreorder]
ABABbaba
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int value;
    X(int v) : value(v) { P('X'); P(v); }
    ~X() { P('x'); }
    X(const X & x) : value(x.value) { P(2); }
    X operator+(const X & x) { P(3); return this->value + x.value; }
};

int main() {
    X a(4);
    P('-');
    P( (a + a).value );
    P('-');
    X b = (a + a);
    P('-');
    P(b.value);
    P('-');
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int value;
    X(int v) : value(v) { P('X'); P(v); }
    ~X() { P('x'); }
    X(const X & x) : value(x.value) { P(2); }
    X operator+(const X & x) { P(3); return this->value + x.value; }
};

int main() {
    X a(4);
    P('-');
    P( (a + a).value );
    P('-');
    X b = (a + a);
    P('-');
    P(b.value);
    P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    int value;
    X(int v) : value(v) { P('X'); P(v); }
    ~X() { P('x'); }
    X(const X & x) : value(x.value) { P(2); }
    X operator+(const X & x) { P(3); return this->value + x.value; }
};

int main() {
    X a(4);
    P('-');
    P( (a + a).value );
    P('-');
    X b = (a + a);
    P('-');
    P(b.value);
    P('-');
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
X4-3X88x-3X8-8-xx
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    X(char v) { P('X'); P(v); }
    ~X() { P('x'); }
};

void foo(X x) {
    P('F');
}

int main() {
    foo(65);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    X(char v) { P('X'); P(v); }
    ~X() { P('x'); }
};

void foo(X x) {
    P('F');
}

int main() {
    foo(65);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    X(char v) { P('X'); P(v); }
    ~X() { P('x'); }
};

void foo(X x) {
    P('F');
}

int main() {
    foo(65);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
XAFx
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X {
    X(char v) { P('X'); P(v); }
    ~X() { P('x'); }
};

void foo(X x) {
    P('F');
}

int main() {
    foo(65);
}
```

Bonus point if you discussed the **explicit** specifier for constructors

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
XAFx
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
    ~A() { P(1); }
};

struct B : A {
    B() { P(2); }
    ~B() { P(3); }
};

struct C {
    C() { P(4); }
    virtual ~C() { P(5); }
};

struct D : C {
    D() { P(6); }
    ~D() { P(7); }
};

int main() {
    A * a = new B;
    delete a;
    P('-');
    C * c = new D;
    delete c;
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
    ~A() { P(1); }
};

struct B : A {
    B() { P(2); }
    ~B() { P(3); }
};

struct C {
    C() { P(4); }
    virtual ~C() { P(5); }
};

struct D : C {
    D() { P(6); }
    ~D() { P(7); }
};

int main() {
    A * a = new B;
    delete a;
    P('-');
    C * c = new D;
    delete c;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
    ~A() { P(1); }
};

struct B : A {
    B() { P(2); }
    ~B() { P(3); }
};

struct C {
    C() { P(4); }
    virtual ~C() { P(5); }
};

struct D : C {
    D() { P(6); }
    ~D() { P(7); }
};

int main() {
    A * a = new B;
    delete a;
    P('-');
    C * c = new D;
    delete c;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
021-4675
```

#15

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
};

struct B : A {
    B() { P(2); }
};

struct C : virtual A {
    C() { P(4); }
};

struct D : B, C {
    D() { P(6); }
};

struct E : C, B{
    E() { P(8); }
};

int main() {
    D d;
    P('-');
    E e;
}
```

#15

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
};

struct B : A {
    B() { P(2); }
};

struct C : virtual A {
    C() { P(4); }
};

struct D : B, C {
    D() { P(6); }
};

struct E : C, B{
    E() { P(8); }
};

int main() {
    D d;
    P('-');
    E e;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

#15

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
};

struct B : A {
    B() { P(2); }
};

struct C : virtual A {
    C() { P(4); }
};

struct D : B, C {
    D() { P(6); }
};

struct E : C, B{
    E() { P(8); }
};

int main() {
    D d;
    P('-');
    E e;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
00246-04028
```

#15

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct A {
    A() { P(0); }
};

struct B : A {
    B() { P(2); }
};

struct C : virtual A {
    C() { P(4); }
};

struct D : B, C {
    D() { P(6); }
};

struct E : C, B{
    E() { P(8); }
};

int main() {
    D d;
    P('-');
    E e;
}
```

Bonus point if you have not been using **virtual inheritance** in production code.

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
00246-04028
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X { X() { P('0'); } ~X() { P(1); } };

int main() {
    try {
        P(5);
        throw X();
        P(6);
    } catch (X e) {
        P(7);
    }
    P(8);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X { X() { P('0'); } ~X() { P(1); } };

int main() {
    try {
        P(5);
        throw X();
        P(6);
    } catch (X e) {
        P(7);
    }
    P(8);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X { X() { P('0'); } ~X() { P(1); } };

int main() {
    try {
        P(5);
        throw X();
        P(6);
    } catch (X e) {
        P(7);
    }
    P(8);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
507118
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

struct X { X() { P('0'); } ~X() { P(1); } };

int main() {
    try {
        P(5);
        throw X();
        P(6);
    } catch (X e) {
        P(7);
    }
    P(8);
}
```

Bonus point if you discussed the problems of **catching execptions by copy**.

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
507118
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12
12
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12
12
16
```

```
#include <iostream>

struct A {
    int x;
    char y;
    int z;
};

struct B : A {
    void foo() {}
};

struct C : A {
    virtual void foo() {}
};

struct D : A {
    virtual void foo() {}
    virtual void bar() {}
};

int main() {
    std::cout << sizeof(A) << std::endl;
    std::cout << sizeof(B) << std::endl;
    std::cout << sizeof(C) << std::endl;
    std::cout << sizeof(D) << std::endl;
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
12
12
16
16
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:17:10: warning: unused variable ‘h’ [-Wunused-variable]
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

void foo(std::initializer_list<int> numbers, void (*func)(int))
{
    for (int n : numbers)
        func(n);
}

int main()
{
    foo({1,2,3,4},[](int x){P(x+2);});
    P('-');
    auto f = [](){P('f'); return 0;};
    auto g = [](){P('g'); return 1;};
    auto h = [](char c){P(c); return 2;}('h');
    auto j = f() && g();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
foo.cpp: In function ‘int main()’:
foo.cpp:17:10: warning: unused variable ‘h’ [-Wunused-variable]
3456-hf0
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int j = 3;

int main()
{
    P(j);
    [&](){j++; P(j);}();
    P(j);
    [=](){j++; P(j);}();
    P(j);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int j = 3;

int main()
{
    P(j);
    [&](){j++; P(j);}();
    P(j);
    [=](){j++; P(j);}();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int j = 3;

int main()
{
    P(j);
    [&](){j++; P(j);}();
    P(j);
    [=](){j++; P(j);}();
    P(j);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
34455
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }

int j = 3;

int main()
{
    P(j);
    [&](){j++; P(j);}();
    P(j);
    [=](){j++; P(j);}();
    P(j);
}
```

Bonus point if you used the word **capture** when discussing this code snippet.

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
34455
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }
template<typename T> void foo(T *) { P(2); }
template<typename T> void foo(T &&) { P(4); }
template<> void foo(int&) { P(5); }

int main()
{
    int a = 65;
    foo(a);
    foo(&a);
    char b = 'a';
    foo(b);
    foo(&b);
}
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }
template<typename T> void foo(T *) { P(2); }
template<typename T> void foo(T &&) { P(4); }
template<> void foo(int&) { P(5); }

int main()
{
    int a = 65;
    foo(a);
    foo(&a);
    char b = 'a';
    foo(b);
    foo(&b);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
```

```
#include <iostream>

template<typename T> void P(T x) { std::cout << x; }
template<typename T> void foo(T *) { P(2); }
template<typename T> void foo(T &&) { P(4); }
template<> void foo(int&) { P(5); }

int main()
{
    int a = 65;
    foo(a);
    foo(&a);
    char b = 'a';
    foo(b);
    foo(&b);
}
```

```
$ g++ --std=c++0x -O -Wall foo.cpp && ./a.out
5242
```

Max score is 3x20 + 12 = 72 points

>60 points = not sure if I believe you
>50 points = exceptionally good results
>35 points = very, very good
>20 points = you certainly know something about 🍺++
>10 points = you need to study more
<1 point = you are probably a bit too confident