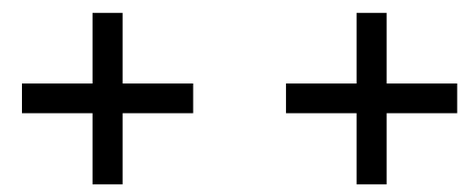
C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton



A 90 minute quiz session
ACCU
April 2012

Here is my development environment:

```
$ myc++ -v
Using built-in specs.
COLLECT_GCC=/Users/oma/mygcc/bin/myg++
COLLECT_LTO_WRAPPER=/Users/oma/mygcc/libexec/gcc/x86_64-apple-darwin10.8.0/4.8.0/lto-
wrapper
Target: x86_64-apple-darwin10.8.0
Configured with: ../configure --prefix=/Users/oma/mygcc --enable-checking=release --with-
gmp=/Users/oma/mygcc --with-mpfr=/Users/oma/mygcc --with-mpc=/Users/oma/mygcc --program-
prefix=my
Thread model: posix
gcc version 4.8.0 20120424 (experimental) (GCC)
$
```

Here is how I compile and run the snippets:

```
$ myc++ --std=c++11 foo.cpp && ./a.out
```

Here is my development environment:

```
$ myc++ -v
Using built-in specs.
COLLECT_GCC=/Users/oma/mygcc/bin/myg++
COLLECT_LTO_WRAPPER=/Users/oma/mygcc/libexec/gcc/x86_64-apple-darwin10.8.0/4.8.0/lto-
wrapper
Target: x86_64-apple-darwin10.8.0
Configured with: ../configure --prefix=/Users/oma/mygcc --enable-checking=release --with-
gmp=/Users/oma/mygcc --with-mpfr=/Users/oma/mygcc --with-mpc=/Users/oma/mygcc --program-
prefix=my
Thread model: posix
gcc version 4.8.0 20120424 (experimental) (GCC)
$
```

Here is how I compile and run the snippets:

```
$ myc++ --std=c++11 foo.cpp && ./a.out
```

The question for all code snippets is:

Here is my development environment:

```
$ myc++ -v
Using built-in specs.
COLLECT_GCC=/Users/oma/mygcc/bin/myg++
COLLECT_LTO_WRAPPER=/Users/oma/mygcc/libexec/gcc/x86_64-apple-darwin10.8.0/4.8.0/lto-
wrapper
Target: x86_64-apple-darwin10.8.0
Configured with: ../configure --prefix=/Users/oma/mygcc --enable-checking=release --with-
gmp=/Users/oma/mygcc --with-mpfr=/Users/oma/mygcc --with-mpc=/Users/oma/mygcc --program-
prefix=my
Thread model: posix
gcc version 4.8.0 20120424 (experimental) (GCC)
$
```

Here is how I compile and run the snippets:

```
$ myc++ --std=c++11 foo.cpp && ./a.out
```

The question for all code snippets is:

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

Here is my development environment:

```
$ myc++ -v
Using built-in specs.
COLLECT_GCC=/Users/oma/mygcc/bin/myg++
COLLECT_LTO_WRAPPER=/Users/oma/mygcc/libexec/gcc/x86_64-apple-darwin10.8.0/4.8.0/lto-
wrapper
Target: x86_64-apple-darwin10.8.0
Configured with: ../configure --prefix=/Users/oma/mygcc --enable-checking=release --with-
gmp=/Users/oma/mygcc --with-mpfr=/Users/oma/mygcc --with-mpc=/Users/oma/mygcc --program-
prefix=my
Thread model: posix
gcc version 4.8.0 20120424 (experimental) (GCC)
$
```

Here is how I compile and run the snippets:

```
$ myc++ --std=c++11 foo.cpp && ./a.out
```

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

Full score is given if you manage to guess:

Here is my development environment:

```
$ myc++ -v
Using built-in specs.
COLLECT_GCC=/Users/oma/mygcc/bin/myg++
COLLECT_LTO_WRAPPER=/Users/oma/mygcc/libexec/gcc/x86_64-apple-darwin10.8.0/4.8.0/lto-
wrapper
Target: x86_64-apple-darwin10.8.0
Configured with: ../configure --prefix=/Users/oma/mygcc --enable-checking=release --with-
gmp=/Users/oma/mygcc --with-mpfr=/Users/oma/mygcc --with-mpc=/Users/oma/mygcc --program-
prefix=my
Thread model: posix
gcc version 4.8.0 20120424 (experimental) (GCC)
$
```

Here is how I compile and run the snippets:

```
$ myc++ --std=c++11 foo.cpp && ./a.out
```

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

Full score is given if you manage to guess:
*Whatever actually 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 in the snippets. The output is always a straightforward sequence of non-whitespace characters.

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:

++

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:

++

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.

~ 70 points in total

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.

~ 70 points in total

10 bonus points if you think you might get less than 80% score on this quiz.

Questions

(45 minutes)

```
#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);
}
```

++

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

Start bonus	Score	Bonus	Total

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total

++

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1	0 1 2 3 4 9			
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1	0 1 2 3 4 9	auto, for each loop, list initializer		
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Quiz	Answer	Extra	Score	Bonus
1	0 1 2 3 4 9	auto, for each loop, list initializer	3	
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes and Andrew Sutton

Sponsored by: Cititec

Quiz	Answer	Extra	Score	Bonus
1	0 1 2 3 4 9	auto, for each loop, list initializer	3	0
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

```
#include <iostream>

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

int b(int v) { P(v); return v; }

struct Foo {
    Foo(int a, int b, int c) {}
};

int main()
{
    Foo f = {b(3),b(2),b(1)*b(4)};
    P('-');
    int a[3] = {b(3),b(2),b(1)*b(4)};
}
```

```
#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>
#include <vector>

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

using namespace std;

int main()
{
    vector<bool> a { 1, 1, 1, 1 };
    auto b = a.begin();
    auto c = *b;
    cout << is_same<decltype(c), bool>();

    vector<int> d { 1, 1, 1, 1 };
    auto e = d.begin();
    auto f = *e;
    cout << is_same<decltype(f), int>();
}
```

```
#include <iostream>

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

struct I {
    int v_;
    I(int v) : v_(v) { P('a'); }
    int operator*() { return v_; }
    I operator++() { return ++v_; }
};
bool operator!=(I & lhs, I & rhs) { return *lhs != *rhs; }

struct R {
    int v_;
    R(int v) : v_(v) {}
    I begin() { return 0; }
    I end() { return v_; }
};

int main() {
    for (auto e : R(5))
        P(e);
}
```

#6

```
#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('-');

    g(f(1),f(2));
    P('-');

    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; }

template <typename T>
struct s
{
    template <typename U>
    void f(U&& x) { g(x); }
    void f(T&& x) { g(x); }
    void g(const T& x) { P(3); }
    void g(T& x) { P(4); }
};


int main()
{
    P(0);
    s<int> x;
    x.f(0);
    x.f(0L);
    long y = 0;
    x.f(y);
    P(9);
}
```

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

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

template <typename T>
void swap(T && a, T && b)
{
    P(a); P(b);
    T x = move(a);
    a = move(b);
    b = move(x);
    P(a); P(b);
}

std::string f() { P('f'); return "a"; }
std::string g() { P('g'); return "b"; }

int main()
{
    P(0);
    swap(f(), g());
    P(9);
}
```

```
#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; }

int main()
{
    P(0);
    auto f = []{P(4);};
    auto g = [f](){P(2);f();P(3);};
    P(9);
    g();
}
```

```
#include <iostream>
#include <type_traits>

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

int main()
{
    P('a');
    short x = 0;
    using T = decltype(x);
    using U = decltype((x));
    using V = decltype(+x);
    std::cout << std::is_same<T, short>();
    std::cout << std::is_same<T, int>();
    std::cout << std::is_same<T, short&>();
    std::cout << std::is_same<U, short>();
    std::cout << std::is_same<U, int>();
    std::cout << std::is_same<U, short&>();
    std::cout << std::is_same<V, short>();
    std::cout << std::is_same<V, int>();
    std::cout << std::is_same<V, short&>();
    P('z');
}
```

```
#include <iostream>
#include <type_traits>

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

template <typename T>
using Remove_reference = typename std::remove_reference<T>::type;

template <typename T>
void f(T&& x) {
    std::cout << (std::is_const<Remove_reference<T>>() ? 'C' : 'x');
    std::cout << (std::is_reference<T>() ? 'R' : 'x');
    P('-');
}

int main()
{
    int x = 0;
    const int y = 0;
    int& r = x;
    const int& s = y;
    f(0);
    f(x);
    f(y);
    f(r);
    f(s);
}
```

```
#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 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; }

void foo(char a) {
    P(3);
    P(a);
}

template <typename... A>
void foo(int a, A... args) {
    foo(args...);
    P(a);
}

template <typename... A>
void foo(char a, A... args) {
    P(a);
    foo(args...);
}

int main()
{
    foo('1','2',48,'4','5');
}
```

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.

~ 70 points in total

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.

~ 70 points in total

10 bonus points if you think you might get less than 80% score on this quiz.

```
#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);
}
```

```
012349
```

```
#include <iostream>

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

int b(int v) { P(v); return v; }

struct Foo {
    Foo(int a, int b, int c) {}
};

int main()
{
    Foo f = {b(3),b(2),b(1)*b(4)};
    P('-');
    int a[3] = {b(3),b(2),b(1)*b(4)};
}
```

#2

```
#include <iostream>

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

int b(int v) { P(v); return v; }

struct Foo {
    Foo(int a, int b, int c) {}
};

int main()
{
    Foo f = {b(3),b(2),b(1)*b(4)};
    P('-');
    int a[3] = {b(3),b(2),b(1)*b(4)};
}
```

```
1423-3214
```

```
#include <iostream>

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

int b(int v) { P(v); return v; }

struct Foo {
    Foo(int a, int b, int c) {}
};

int main()
{
    Foo f = {b(3),b(2),b(1)*b(4)};
    P('-');
    int a[3] = {b(3),b(2),b(1)*b(4)};
}
```

Bonus point if your team discussed unspecified evaluation order. Another bonus point if you discussed the use of explicit specifier on multi-argument constructors.

```
1423-3214
```

```
#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);
}
```

```
507118
```

#3

```
#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 your team discussed problems around catching exceptions by copy.

```
507118
```

```
#include <iostream>
#include <vector>

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

using namespace std;

int main()
{
    vector<bool> a { 1, 1, 1, 1 };
    auto b = a.begin();
    auto c = *b;
    cout << is_same<decltype(c), bool>();

    vector<int> d { 1, 1, 1, 1 };
    auto e = d.begin();
    auto f = *e;
    cout << is_same<decltype(f), int>();
}
```

```
#include <iostream>
#include <vector>

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

using namespace std;

int main()
{
    vector<bool> a { 1, 1, 1, 1 };
    auto b = a.begin();
    auto c = *b;
    cout << is_same<decltype(c), bool>();

    vector<int> d { 1, 1, 1, 1 };
    auto e = d.begin();
    auto f = *e;
    cout << is_same<decltype(f), int>();
}
```

```
01
```

#4

```
#include <iostream>
#include <vector>

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

using namespace std;

int main()
{
    vector<bool> a { 1, 1, 1, 1 };
    auto b = a.begin();
    auto c = *b;
    cout << is_same<decltype(c), bool>();

    vector<int> d { 1, 1, 1, 1 };
    auto e = d.begin();
    auto f = *e;
    cout << is_same<decltype(f), int>();
}
```

Bonus point if your team discussed implementation defined type determination.

```
01
```

```
#include <iostream>

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

struct I {
    int v_;
    I(int v) : v_(v) { P('a'); }
    int operator*() { return v_; }
    I operator++() { return ++v_; }
};
bool operator!=(I & lhs, I & rhs) { return *lhs != *rhs; }

struct R {
    int v_;
    R(int v) : v_(v) {}
    I begin() { return 0; }
    I end() { return v_; }
};

int main() {
    for (auto e : R(5))
        P(e);
}
```

```
#include <iostream>

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

struct I {
    int v_;
    I(int v) : v_(v) { P('a'); }
    int operator*() { return v_; }
    I operator++() { return ++v_; }
};
bool operator!=(I & lhs, I & rhs) { return *lhs != *rhs; }

struct R {
    int v_;
    R(int v) : v_(v) {}
    I begin() { return 0; }
    I end() { return v_; }
};

int main() {
    for (auto e : R(5))
        P(e);
}
```

```
aa0a1a2a3a4a
```

#6

```
#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('-');

    g(f(1),f(2));
    P('-');

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

#6

```
#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('-');

    g(f(1),f(2));
    P('-');

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

```
1237-2112-0002
```

#6

```
#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('-');
    g(f(1),f(2));
    P('-');

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

1237-2112-0002

Bonus point if your team discussed evaluation order and sequence points / sequencing. Another bonus point if you discussed the unspecified behavior vs undefined behaviour.

```
#include <iostream>

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

template <typename T>
struct s
{
    template <typename U>
    void f(U&& x) { g(x); }
    void f(T&& x) { g(x); }
    void g(const T& x) { P(3); }
    void g(T& x) { P(4); }
};


int main()
{
    P(0);
    s<int> x;
    x.f(0);
    x.f(0L);
    long y = 0;
    x.f(y);
    P(9);
}
```

```
#include <iostream>

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

template <typename T>
struct s
{
    template <typename U>
    void f(U&& x) { g(x); }
    void f(T&& x) { g(x); }
    void g(const T& x) { P(3); }
    void g(T& x) { P(4); }
};


int main()
{
    P(0);
    s<int> x;
    x.f(0);
    x.f(0L);
    long y = 0;
    x.f(y);
    P(9);
}
```

```
04339
```

```
#include <iostream>

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

template <typename T>
struct s
{
    template <typename U>
    void f(U&& x) { g(x); }
    void f(T&& x) { g(x); }
    void g(const T& x) { P(3); }
    void g(T& x) { P(4); }
};


int main()
{
    P(0);
    s<int> x;
    x.f(0);
    x.f(0L);
    long y = 0;
    x.f(y);
    P(9);
}
```

```
04339
```

Bonus point if your team discussed template argument deduction

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

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

template <typename T>
void swap(T && a, T && b)
{
    P(a); P(b);
    T x = move(a);
    a = move(b);
    b = move(x);
    P(a); P(b);
}

std::string f() { P('f'); return "a"; }
std::string g() { P('g'); return "b"; }

int main()
{
    P(0);
    swap(f(), g());
    P(9);
}
```

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

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

template <typename T>
void swap(T && a, T && b)
{
    P(a); P(b);
    T x = move(a);
    a = move(b);
    b = move(x);
    P(a); P(b);
}

std::string f() { P('f'); return "a"; }
std::string g() { P('g'); return "b"; }

int main()
{
    P(0);
    swap(f(), g());
    P(9);
}
```

0gfabba9

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

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

template <typename T>
void swap(T && a, T && b)
{
    P(a); P(b);
    T x = move(a);
    a = move(b);
    b = move(x);
    P(a); P(b);
}

std::string f() { P('f'); return "a"; }
std::string g() { P('g'); return "b"; }

int main()
{
    P(0);
    swap(f(), g());
    P(9);
}
```

```
0gfabba9
```

Bonus point if your team discussed standard swap and library design issues.

```
#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('-');
}
```

a-f-ecfb-c-d-bb

#9

```
#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 your team discussed return value optimization. Another if you discussed preincrement vs postincrement.

```
a-f-ecfb-c-d-bb
```

```
#include <iostream>

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

int main()
{
    P(0);
    auto f = []{P(4);};
    auto g = [f](){P(2);f();P(3);};
    P(9);
    g();
}
```

```
#include <iostream>

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

int main()
{
    P(0);
    auto f = []{P(4);};
    auto g = [f](){P(2);f();P(3);};
    P(9);
    g();
}
```

```
09243
```

#10

```
#include <iostream>

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

int main()
{
    P(0);
    auto f = []{P(4);};
    auto g = [f](){P(2);f();P(3);};
    P(9);
    g();
}
```

09243

Bonus point if your team discussed the possibility of mutual recursive lambdas

```
#include <iostream>
#include <type_traits>

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

int main()
{
    P('a');
    short x = 0;
    using T = decltype(x);
    using U = decltype((x));
    using V = decltype(+x);
    std::cout << std::is_same<T, short>();
    std::cout << std::is_same<T, int>();
    std::cout << std::is_same<T, short&>();
    std::cout << std::is_same<U, short>();
    std::cout << std::is_same<U, int>();
    std::cout << std::is_same<U, short&>();
    std::cout << std::is_same<V, short>();
    std::cout << std::is_same<V, int>();
    std::cout << std::is_same<V, short&>();
    P('z');
}
```

```
#include <iostream>
#include <type_traits>

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

int main()
{
    P('a');
    short x = 0;
    using T = decltype(x);
    using U = decltype((x));
    using V = decltype(+x);
    std::cout << std::is_same<T, short>();
    std::cout << std::is_same<T, int>();
    std::cout << std::is_same<T, short&>();
    std::cout << std::is_same<U, short>();
    std::cout << std::is_same<U, int>();
    std::cout << std::is_same<U, short&>();
    std::cout << std::is_same<V, short>();
    std::cout << std::is_same<V, int>();
    std::cout << std::is_same<V, short&>();
    P('z');
}
```

```
a100001010z
```

```
#include <iostream>
#include <type_traits>

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

int main()
{
    P('a');
    short x = 0;
    using T = decltype(x);
    using U = decltype((x));
    using V = decltype(+x);
    std::cout << std::is_same<T, short>();
    std::cout << std::is_same<T, int>();
    std::cout << std::is_same<T, short&>();
    std::cout << std::is_same<U, short>();
    std::cout << std::is_same<U, int>();
    std::cout << std::is_same<U, short&>();
    std::cout << std::is_same<V, short>();
    std::cout << std::is_same<V, int>();
    std::cout << std::is_same<V, short&>();
    P('z');
}
```

```
a100001010z
```

Bonus point if your team discussed *why* +x is an int, and smallest matching integer type

```
#include <iostream>
#include <type_traits>

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

template <typename T>
using Remove_reference = typename std::remove_reference<T>::type;

template <typename T>
void f(T&& x) {
    std::cout << (std::is_const<Remove_reference<T>>() ? 'C' : 'x');
    std::cout << (std::is_reference<T>() ? 'R' : 'x');
    P('-');
}

int main()
{
    int x = 0;
    const int y = 0;
    int& r = x;
    const int& s = y;
    f(0);
    f(x);
    f(y);
    f(r);
    f(s);
}
```

```
#include <iostream>
#include <type_traits>

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

template <typename T>
using Remove_reference = typename std::remove_reference<T>::type;

template <typename T>
void f(T&& x) {
    std::cout << (std::is_const<Remove_reference<T>>() ? 'C' : 'x');
    std::cout << (std::is_reference<T>() ? 'R' : 'x');
    P('-');
}

int main()
{
    int x = 0;
    const int y = 0;
    int& r = x;
    const int& s = y;
    f(0);
    f(x);
    f(y);
    f(r);
    f(s);
}
```

```
xx-xR-CR-xR-CR-
```

#12

```
#include <iostream>
#include <type_traits>

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

template <typename T>
using Remove_reference = typename std::remove_reference<T>::type;

template <typename T>
void f(T&& x) {
    std::cout << (std::is_const<Remove_reference<T>>() ? 'C' : 'x');
    std::cout << (std::is_reference<T>() ? 'R' : 'x');
    P('-');
}

int main()
{
    int x = 0;
    const int y = 0;
    int& r = x;
    const int& s = y;
    f(0);
    f(x);
    f(y);
    f(r);
    f(s);
}
```

```
xx-xR-CR-xR-CR-
```

Bonus point if your team discussed alias templates and template metafunctions

```
#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);
}
```

```
12565621
```

```
#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 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;
}
```

```
00246-04028
```

```
#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;
}
```

```
00246-04028
```

Bonus point if you tend to avoid writing code like this.

```
#include <iostream>

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

void foo(char a) {
    P(3);
    P(a);
}

template <typename... A>
void foo(int a, A... args) {
    foo(args...);
    P(a);
}

template <typename... A>
void foo(char a, A... args) {
    P(a);
    foo(args...);
}

int main()
{
    foo('1','2',48,'4','5');
}
```

```
#include <iostream>

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

void foo(char a) {
    P(3);
    P(a);
}

template <typename... A>
void foo(int a, A... args) {
    foo(args...);
    P(a);
}

template <typename... A>
void foo(char a, A... args) {
    P(a);
    foo(args...);
}

int main()
{
    foo('1','2',48,'4','5');
}
```

```
12355248
```

Max score is 10 + 3x15 + ~15 = around 70 points

>60 points = have you cheated?
>40 points = exceptionally good results
>20 points = you certainly understand 🍺++
>10 points = your team has a great potential
<10 points = your team is probably a bit too confident