C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes

Sponsored by:

DeveloperFocus LONDON

A 90 minute quiz session
ACCU
April 2013

Here is my development environment: (Mac OS 10.8.2, x86_64)

```
bash-3.2$ myc++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/Users/oma/opt/libexec/gcc/x86_64-apple-darwin12.2.0/4.8.0/lto-wrapper
Target: x86_64-apple-darwin12.2.0
Configured with: ./configure --prefix=/Users/oma/opt --enable-languages=c,c++ --with-
cloog=/Users/oma/opt
Thread model: posix
gcc version 4.8.0 20130210 (experimental) (GCC)
bash-3.2$
```

Here is how I compile and run the snippets:

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

Here is my development environment: (Mac OS 10.8.2, x86_64)

```
bash-3.2$ myc++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/Users/oma/opt/libexec/gcc/x86_64-apple-darwin12.2.0/4.8.0/lto-wrapper
Target: x86_64-apple-darwin12.2.0
Configured with: ./configure --prefix=/Users/oma/opt --enable-languages=c,c++ --with-
cloog=/Users/oma/opt
Thread model: posix
gcc version 4.8.0 20130210 (experimental) (GCC)
bash-3.2$
```

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:
(Mac OS 10.8.2, x86_64)

```
bash-3.2$ myc++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/Users/oma/opt/libexec/gcc/x86_64-apple-darwin12.2.0/4.8.0/lto-wrapper
Target: x86_64-apple-darwin12.2.0
Configured with: ./configure --prefix=/Users/oma/opt --enable-languages=c,c++ --with-
cloog=/Users/oma/opt
Thread model: posix
gcc version 4.8.0 20130210 (experimental) (GCC)
bash-3.2$
```

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:
(Mac OS 10.8.2, x86_64)

```
bash-3.2$ myc++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/Users/oma/opt/libexec/gcc/x86_64-apple-darwin12.2.0/4.8.0/lto-wrapper
Target: x86_64-apple-darwin12.2.0
Configured with: ./configure --prefix=/Users/oma/opt --enable-languages=c,c++ --with-
cloog=/Users/oma/opt
Thread model: posix
gcc version 4.8.0 20130210 (experimental) (GCC)
bash-3.2$
```

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:
(Mac OS 10.8.2, x86_64)

```
bash-3.2$ myc++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/Users/oma/opt/libexec/gcc/x86_64-apple-darwin12.2.0/4.8.0/lto-wrapper
Target: x86_64-apple-darwin12.2.0
Configured with: ./configure --prefix=/Users/oma/opt --enable-languages=c,c++ --with-
cloog=/Users/oma/opt
Thread model: posix
gcc version 4.8.0 20130210 (experimental) (GCC)
bash-3.2$
```

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 few 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 few 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 code. This is just for fun.

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

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

++

Sponsored by:

DeveloperFocus
LONDON

A word from our sponsor:

A word from our sponsor:

A word from our sponsor:

Effective C++11 Programming
Scott Meyers

A word from our sponsor:

Effective C++11 Programming
Scott Meyers

2 day course, Holborn Bars, London
June 17-18, 2013

A word from our sponsor:

Effective C++11 Programming
Scott Meyers

2 day course, Holborn Bars, London
June 17-18, 2013

http://www.developerfocus.com

A word from our sponsor:

Effective C++11 Programming
Scott Meyers

2 day course, Holborn Bars, London
June 17-18, 2013

http://www.developerfocus.com

DeveloperFocus
LONDON

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes

Sponsored by: DeveloperFocus LONDON

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

Team name:

Start bonus	Score	Bonus	Total

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes

Sponsored by: DeveloperFocus LONDON

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

Team name:

The Destructors

Start bonus	Score	Bonus	Total

C++ Pub Quiz

by Olve Maudal, with Lars Gullik Bjønnes

Sponsored by: DeveloperFocus LONDON

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

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

10 points as start bonus
3 points for each correct answer
1 point for no answer
-1 point for an incorrect answer

For some of the answers there are bonus points.

Questions

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

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

Sponsored by: DeveloperFocus LONDON

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

Team name:

The Destructors

Start bonus	Score	Bonus	Total
10			

#2

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

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

void foo() { static X c = 42; }

static X a(1);

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

X b(2);
```

#3

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

struct E { ~E() { P(3); } } e;

template <typename T, size_t s>
auto a(T (&t)[s]) -> decltype(std::begin(t))
{
    P(0); return std::begin(t);
}

template <typename T>
auto a(T && t) -> decltype(std::begin(t))
{
    P(2); return std::begin(t);
}

template <typename T>
void a(T const &) { P(1); }

int main()
{
    int r[1];
    a(r);
    a(e);
    a(a(std::string()));
}
```

#4

```
#include <iostream>

int main()
{
    {
        unsigned int i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        short j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned long i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned int i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    foo() : counter_(0) { P(0); }
    foo(const foo & other) : counter_(other.counter_) { P(1); }
    ~foo() { P(2); }
    foo & operator++() { P(3); ++counter_; return *this; }
    foo operator++(int) { P(4);
        foo tmp(*this);
        ++counter_;
        return tmp;
    }
    operator int() const { P(9); return counter_; };
private:
    int counter_;
};

int main()
{
    foo f; P(5); f++; P(8); P(f);
}
```

#6

```
#include <iostream>
#include <cmath>

void foo(double a, double b)
{
    if (a < b)
        std::cout << "0";
    else if (a > b)
        std::cout << "1";
    else if (a == b)
        std::cout << "2";
    else
        std::cout << "3";
}

int main(void)
{
    try {
        foo(1/0.0, -1/0.0);
        foo(42, std::sqrt(-1));
        foo(-1/0.0, 1/0.0);
        foo(std::sqrt(-1), std::sqrt(-1));
    } catch(...) {
        std::cout << "4";
    }
}
```

```
#include <iostream>
#include <future>

template <typename T> void P(T const & t) { std::cout << t; }

std::mutex mymutex;

int print(char ch)
{
    P(ch);
    std::lock_guard<std::mutex> mylock(mymutex);
    P(2);
    return 21;
}

int main() {
    P(0);
    auto f1 = std::async (std::launch::deferred, print, '4');
    auto f2 = std::async (std::launch::deferred, print, '5');
    std::cout << f1.get() + f2.get();
    P(9);
}
```

#8

```
#include <iostream>
#include <climits>

template <typename T> void P(T const & t) { std::cout << t; }

void foo(int) { P(1); }
void foo(long) { P(2); }
void foo(float) { P(3); }
void foo(double) { P(4); }
void foo(const char *) { P(5); }
void foo(std::string &) { P(6); }
void foo(short) { P(7); }

int main() {
    short s = 3;
    foo(3.14);
    foo(-INT_MAX);
    foo(-2147483648);
    foo(-2147483647-1);
    foo("foo");
    foo(s);
    foo(s+s);
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
   foo() { P(0); }
   ~foo() { P(9); }
   foo(foo const &) { P(2); }
   foo(foo const &&) { P(3); };
   foo operator=(foo const &) { P(4);  return *this; }
   foo & operator=(foo const &&) { P(5); return *this; };
};

int main()
{
   foo f{foo()};
   foo g{f = (f = foo(f))};
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    virtual void f(int) = 0;
};

class bar : public foo {
public:
    void f(int final) override final { P(override); P(final); }
private:
    int override = 4;
};

int main()
{
    P(2);
    bar b;
    b.f(3);
    P(9);
}
```

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) { P(1); }
    explicit operator double () const { P(2); return size_; }
    double size() const { P(3); return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    P(4); return FontSize(fs);
}

int main() {
    FontSize fs2 = FontSize(5.2);
    P(double(fs2));
    FontSize fs3 = 5.5_pt;
    P(fs3.size());
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

int foo(int * x, int * y)
{
    *x = 1;
    *y = 2;
    return *x + *y;
}

char str[] = "6789";

int main()
{
    P(0);
    int a = 3;
    int b = 4;
    int c = foo(&a,&a);
    P(c);
    (c == a+b ? a : b) = 5;
    P(a);
    P(b);
    int i = 0;
    str[i] = i[str+2];
    std::cout << str;
}
```

```
#include <iostream>
#include <sys/types.h>
#include <memory>
#include <functional>
#include <sys/socket.h>
#include <cassert>
#include <stdlib.h>

struct O
{
    operator int * () { return 0; }
    int operator * () { return 1; }
    template<typename T>
    operator T * () { return 0; }
    template<class... T>
    int operator()(T... t) { return bind( **this, *this, t * t ... ); }
};

using namespace std;

int main()
{
    std::cout << 0-((bind( O(), 0))(::bind))-0;
}
```

Answers

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

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 const & t) { std::cout << t; }

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

```
012349
```

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

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

void foo() { static X c = 42; }

static X a(1);

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

X b(2);
```

#2

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

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

void foo() { static X c = 42; }

static X a(1);

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

X b(2);
```

```
12425656 4221
```

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

struct E { ~E() { P(3); } } e;

template <typename T, size_t s>
auto a(T (&t)[s]) -> decltype(std::begin(t))
{
    P(0); return std::begin(t);
}

template <typename T>
auto a(T && t) -> decltype(std::begin(t))
{
    P(2); return std::begin(t);
}

template <typename T>
void a(T const &) { P(1); }

int main()
{
    int r[1];
    a(r);
    a(e);
    a(a(std::string()));
}
```

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

struct E { ~E() { P(3); } } e;

template <typename T, size_t s>
auto a(T (&t)[s]) -> decltype(std::begin(t))
{
    P(0); return std::begin(t);
}

template <typename T>
auto a(T && t) -> decltype(std::begin(t))
{
    P(2); return std::begin(t);
}

template <typename T>
void a(T const &) { P(1); }

int main()
{
    int r[1];
    a(r);
    a(e);
    a(a(std::string()));
}
```

```
01213
```

#4

```
#include <iostream>

int main()
{
    {
        unsigned int i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        short j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned long i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned int i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
}
```

```
#include <iostream>

int main()
{
    {
        unsigned int i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        short j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned long i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned int i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
}
```

```
73733
```

```
#include <iostream>

int main()
{
    {
        unsigned int i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        short j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned long i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned int i = 1;
        long j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
    {
        unsigned short i = 1;
        int j = -1;
        std::cout << ( i > j ? "3" : "7" );
    }
}
```

1 bonus point if your team had a good discussion about integer promotion rules.

```
73733
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    foo() : counter_(0) { P(0); }
    foo(const foo & other) : counter_(other.counter_) { P(1); }
    ~foo() { P(2); }
    foo & operator++() { P(3); ++counter_; return *this; }
    foo operator++(int) { P(4);
        foo tmp(*this);
        ++counter_;
        return tmp;
    }
    operator int() const { P(9); return counter_; };
private:
    int counter_;
};

int main()
{
    foo f; P(5); f++; P(8); P(f);
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    foo() : counter_(0) { P(0); }
    foo(const foo & other) : counter_(other.counter_) { P(1); }
    ~foo() { P(2); }
    foo & operator++() { P(3); ++counter_; return *this; }
    foo operator++(int) { P(4);
        foo tmp(*this);
        ++counter_;
        return tmp;
    }
    operator int() const { P(9); return counter_; };
private:
    int counter_;
};

int main()
{
    foo f; P(5); f++; P(8); P(f);
}
```

```
054128912
```

#6

```cpp
#include <iostream>
#include <cmath>

void foo(double a, double b)
{
    if (a < b)
        std::cout << "0";
    else if (a > b)
        std::cout << "1";
    else if (a == b)
        std::cout << "2";
    else
        std::cout << "3";
}

int main(void)
{
    try {
        foo(1/0.0, -1/0.0);
        foo(42, std::sqrt(-1));
        foo(-1/0.0, 1/0.0);
        foo(std::sqrt(-1), std::sqrt(-1));
    } catch(...) {
        std::cout << "4";
    }
}
```

#6

```
#include <iostream>
#include <cmath>

void foo(double a, double b)
{
    if (a < b)
        std::cout << "0";
    else if (a > b)
        std::cout << "1";
    else if (a == b)
        std::cout << "2";
    else
        std::cout << "3";
}

int main(void)
{
    try {
        foo(1/0.0, -1/0.0);
        foo(42, std::sqrt(-1));
        foo(-1/0.0, 1/0.0);
        foo(std::sqrt(-1), std::sqrt(-1));
    } catch(...) {
        std::cout << "4";
    }
}
```

```
1303
```

```cpp
#include <iostream>
#include <future>

template <typename T> void P(T const & t) { std::cout << t; }

std::mutex mymutex;

int print(char ch)
{
    P(ch);
    std::lock_guard<std::mutex> mylock(mymutex);
    P(2);
    return 21;
}

int main() {
    P(0);
    auto f1 = std::async (std::launch::deferred, print, '4');
    auto f2 = std::async (std::launch::deferred, print, '5');
    std::cout << f1.get() + f2.get();
    P(9);
}
```

```
#include <iostream>
#include <future>

template <typename T> void P(T const & t) { std::cout << t; }

std::mutex mymutex;

int print(char ch)
{
    P(ch);
    std::lock_guard<std::mutex> mylock(mymutex);
    P(2);
    return 21;
}

int main() {
    P(0);
    auto f1 = std::async (std::launch::deferred, print, '4');
    auto f2 = std::async (std::launch::deferred, print, '5');
    std::cout << f1.get() + f2.get();
    P(9);
}
```

```
04252429
```

```
#include <iostream>
#include <future>

template <typename T> void P(T const & t) { std::cout << t; }

std::mutex mymutex;

int print(char ch)
{
    P(ch);
    std::lock_guard<std::mutex> mylock(mymutex);
    P(2);
    return 21;
}

int main() {
    P(0);
    auto f1 = std::async (std::launch::deferred, print, '4');
    auto f2 = std::async (std::launch::deferred, print, '5');
    std::cout << f1.get() + f2.get();
    P(9);
}
```

3 bonus point if you discussed unspecified evaluation order.

```
04252429
```

```
#include <iostream>
#include <climits>

template <typename T> void P(T const & t) { std::cout << t; }

void foo(int) { P(1); }
void foo(long) { P(2); }
void foo(float) { P(3); }
void foo(double) { P(4); }
void foo(const char *) { P(5); }
void foo(std::string &) { P(6); }
void foo(short) { P(7); }

int main() {
    short s = 3;
    foo(3.14);
    foo(-INT_MAX);
    foo(-2147483648);
    foo(-2147483647-1);
    foo("foo");
    foo(s);
    foo(s+s);
}
```

```cpp
#include <iostream>
#include <climits>

template <typename T> void P(T const & t) { std::cout << t; }

void foo(int) { P(1); }
void foo(long) { P(2); }
void foo(float) { P(3); }
void foo(double) { P(4); }
void foo(const char *) { P(5); }
void foo(std::string &) { P(6); }
void foo(short) { P(7); }

int main() {
    short s = 3;
    foo(3.14);
    foo(-INT_MAX);
    foo(-2147483648);
    foo(-2147483647-1);
    foo("foo");
    foo(s);
    foo(s+s);
}
```

```
4121571
```

#9

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
   foo() { P(0); }
   ~foo() { P(9); }
   foo(foo const &) { P(2); }
   foo(foo const &&) { P(3); };
   foo operator=(foo const &) { P(4);  return *this; }
   foo & operator=(foo const &&) { P(5); return *this; };
};

int main()
{
   foo f{foo()};
   foo g{f = (f = foo(f))};
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
   foo() { P(0); }
   ~foo() { P(9); }
   foo(foo const &) { P(2); }
   foo(foo const &&) { P(3); };
   foo operator=(foo const &) { P(4);  return *this; }
   foo & operator=(foo const &&) { P(5); return *this; };
};

int main()
{
   foo f{foo()};
   foo g{f = (f = foo(f))};
}
```

```
02542999
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    virtual void f(int) = 0;
};

class bar : public foo {
public:
    void f(int final) override final { P(override); P(final); }
private:
    int override = 4;
};

int main()
{
    P(2);
    bar b;
    b.f(3);
    P(9);
}
```

#10

```cpp
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    virtual void f(int) = 0;
};

class bar : public foo {
public:
    void f(int final) override final { P(override); P(final); }
private:
    int override = 4;
};

int main()
{
    P(2);
    bar b;
    b.f(3);
    P(9);
}
```

```
2439
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class foo {
public:
    virtual void f(int) = 0;
};

class bar : public foo {
public:
    void f(int final) override final { P(override); P(final); }
private:
    int override = 4;
};

int main()
{
    P(2);
    bar b;
    b.f(3);
    P(9);
}
```

```
2439
```

Bonus point if your team discussed the difference between a keyword and an identifier with special meanings.

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) { P(1); }
    explicit operator double () const { P(2); return size_; }
    double size() const { P(3); return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    P(4); return FontSize(fs);
}

int main() {
    FontSize fs2 = FontSize(5.2);
    P(double(fs2));
    FontSize fs3 = 5.5_pt;
    P(fs3.size());
}
```

#11

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) { P(1); }
    explicit operator double () const { P(2); return size_; }
    double size() const { P(3); return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    P(4); return FontSize(fs);
}

int main() {
    FontSize fs2 = FontSize(5.2);
    P(double(fs2));
    FontSize fs3 = 5.5_pt;
    P(fs3.size());
}
```

```
125.24135.5
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) { P(1); }
    explicit operator double () const { P(2); return size_; }
    double size() const { P(3); return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    P(4); return FontSize(fs);
}

int main() {
    FontSize fs2 = FontSize(5.2);
    P(double(fs2));
    FontSize fs3 = 5.5_pt;
    P(fs3.size());
}
```

```
125.24135.5
```

Bonus point if the term
"Domain Specific Language"
was mentioned or discussed.

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

int foo(int * x, int * y)
{
    *x = 1;
    *y = 2;
    return *x + *y;
}

char str[] = "6789";

int main()
{
    P(0);
    int a = 3;
    int b = 4;
    int c = foo(&a,&a);
    P(c);
    (c == a+b ? a : b) = 5;
    P(a);
    P(b);
    int i = 0;
    str[i] = i[str+2];
    std::cout << str;
}
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

int foo(int * x, int * y)
{
    *x = 1;
    *y = 2;
    return *x + *y;
}

char str[] = "6789";

int main()
{
    P(0);
    int a = 3;
    int b = 4;
    int c = foo(&a,&a);
    P(c);
    (c == a+b ? a : b) = 5;
    P(a);
    P(b);
    int i = 0;
    str[i] = i[str+2];
    std::cout << str;
}
```

```
04258789
```

```
#include <iostream>

template <typename T> void P(T const & t) { std::cout << t; }

int foo(int * x, int * y)
{
    *x = 1;
    *y = 2;
    return *x + *y;
}

char str[] = "6789";

int main()
{
    P(0);
    int a = 3;
    int b = 4;
    int c = foo(&a,&a);
    P(c);
    (c == a+b ? a : b) = 5;
    P(a);
    P(b);
    int i = 0;
    str[i] = i[str+2];
    std::cout << str;
}
```

1 bonus point for mentioning aliasing
1 bonus point if rvalues vs lvalues was discussed
1 bonus point if the restrict keyword was discussed

```
04258789
```

```
#include <iostream>
#include <sys/types.h>
#include <memory>
#include <functional>
#include <sys/socket.h>
#include <cassert>
#include <stdlib.h>

struct O
{
    operator int * () { return 0; }
    int operator * () { return 1; }
    template<typename T>
    operator T * () { return 0; }
    template<class... T>
    int operator()(T... t) { return bind( **this, *this, t * t ... ); }
};

using namespace std;

int main()
{
    std::cout << 0-((bind( O(), 0))(::bind))-0;
}
```

```
#include <iostream>
#include <sys/types.h>
#include <memory>
#include <functional>
#include <sys/socket.h>
#include <cassert>
#include <stdlib.h>

struct O
{
    operator int * () { return 0; }
    int operator * () { return 1; }
    template<typename T>
    operator T * () { return 0; }
    template<class... T>
    int operator()(T... t) { return bind( **this, *this, t * t ... ); }
};

using namespace std;

int main()
{
    std::cout << 0-((bind( O(), 0))(::bind))-0;
}
```

```
1
```

```
#include <iostream>
#include <sys/types.h>
#include <memory>
#include <functional>
#include <sys/socket.h>
#include <cassert>
#include <stdlib.h>

struct O
{
    operator int * () { return 0; }
    int operator * () { return 1; }
    template<typename T>
    operator T * () { return 0; }
    template<class... T>
    int operator()(T... t) { return bind( **this, *this, t * t ... ); }
};

using namespace std;

int main()
{
    std::cout << 0-((bind( O(), 0))(::bind))-0;
}
```

```
1
```

1 Bonus point if Jonathan Wakely was mentioned in your discussion.

++

Sponsored by:

DeveloperFocus
LONDON