C++ Pub Quiz ACCU 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

++

Sponsored by: Bloomberg

A 90 minute quiz session at ACCU
Terrace Bar, Marriott Hotel, Bristol
1600-1730, April 21, 2016

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

The question for all code snippets is:

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

Full score is given if you manage to guess:

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!*

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!*

Hint: if you don't know, just write 42 as your answer

"My machine" is a plain, up-to-date, MacBook Pro running latest gcc-6 installed via MacPorts

```
sizeof(short) == 2, sizeof(int) == 4, sizeof(long) == 8, sizeof(char *) == 8
```

All examples are built like this:

```
g++ -std=c++1z -w foo.cpp
```

No warnings and no errors. An executable is created that runs just fine and when executed it writes a sequence of alphanumeric characters to standard out.

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:

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:

Start bonus	Score	Bonus	Total

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by:
Bloomberg

#	Answer	Notes	Score	Bonus
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

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

For many of the questions there are bonus points.

Questions

```
#include <iostream>

template <typename T> void P(const 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 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0	0 1 2 3 4 9			
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by:

Bloomberg

#	Answer	Notes	Score	Bonus
0	0 1 2 3 4 9	auto ref, for(x:a)		
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```cpp
#include <iostream>
#include <type_traits>
#include <utility>

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
   return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
   std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
   foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
#include <iostream>

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

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

Everything
Louder
Than
Everything
Else

ACCU

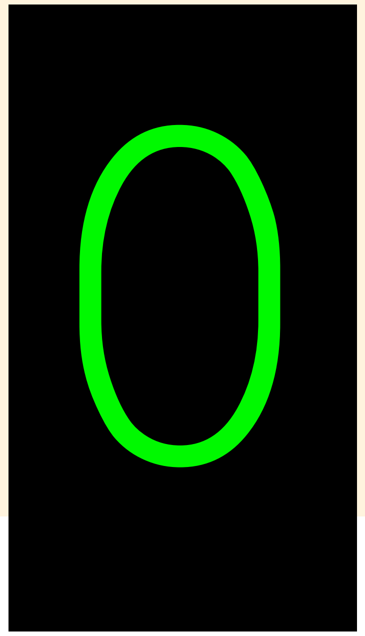
BUS LANE

Cocktails

Answers

```cpp
#include <iostream>

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

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



```cpp
#include <iostream>

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

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

```
01
```

```
#include <iostream>

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

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

```
012
```

```
#include <iostream>

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

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

```
0123
```

```
#include <iostream>

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

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

```
01234
```

#0

```cpp
#include <iostream>

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

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

```
012349
```

#0

```
#include <iostream>

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

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

```
012349
```

1 bonus point if the use of "auto ref" was discussed in your group

#0

```cpp
#include <iostream>

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

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

```
012349
```

1 bonus point if the use of "auto ref" was discussed in your group

1 bonus points if `for(x : a)` was discussed in your group

#0

```
#include <iostream>

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

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

```
012349
```

1 bonus point if the use of "auto ref" was discussed in your group

1 bonus points if `for(x : a)` was discussed in your group

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by:
Bloomberg

#	Answer	Notes	Score	Bonus
0	0 1 2 3 4 9	auto ref, for(x:a)		
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0	0 1 2 3 4 9	auto ref, for(x:a)	3	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

++

C++ Pub Quiz 2016

Olve Maudal, feat Lars Gullik Bjønnes

Sponsored by: Bloomberg

#	Answer	Notes	Score	Bonus
0	0 1 2 3 4 9	auto ref, for(x:a)	3	1 + 1
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Team name:
The Destructors

Start bonus	Score	Bonus	Total
10			

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

#1

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

1

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
12
```

#I

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
123
```

#I

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
1234
```

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
12345
```

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
123456
```

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
1234567
```

#1

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
1234567
```

1 bonus point if someone in yout team plans to attend the Bloomberg reception tonight!

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

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

void f(const void*) { P(7); }
void f(const std::string&) { P(8); }

int main() {
    int i;
    for (i=1; i<3; ++i)
        P(i);
    for (; i<7; i++)
        P(i);
    unsigned int j = 42;
    if (j > -1)
        P(7);
    f("Bloomberg");
}
```

```
1234567
```

1 bonus point if someone in yout team plans to attend the Bloomberg reception tonight!

#2

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

#2

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

0

#2

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

01

#2

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
012
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124
```

```cpp
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
01246
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
012460
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601 8
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
012460181
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
012460181 3
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813 7
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813 74
```

```cpp
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813748
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813748 1
```

#2

```cpp
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
012460181374813
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813748133
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
01246018137481333
```

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

1 point if "delegating constructor" was mentioned
1 point if "return value optimization" was mentioned
1 point if the "explicit" keyword was discussed

```
0124601813748133 3
```

#2

```
#include <iostream>

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

struct I
{
    I(int v) : val((P(0),v)) { P(1); }
    I() : I(0) { P(2); }
    ~I() { P(3); }
    I(const I & other) { P(4); this->val = other.val; }
    I operator=(const I & other) { P(5); this->val = other.val; return *this; }
    I operator++() { P(6); ++val; return val; }
    I operator++(int) { P(7); I tmp(*this); ++val; return tmp; }
    operator int() const { P(8); return val; }
    int val;
};

int main()
{
    I v[1];
    for (auto i : v) {
        P(int(++i));
        P(int(i++));
    }
}
```

```
0124601813748l333
```

1 point if "delegating constructor" was mentioned
1 point if "return value optimization" was mentioned
1 point if the "explicit" keyword was discussed

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

2

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
23
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
233
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
2334
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
23344
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
233444
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
2334445
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
2334456
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
233444566
```

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

template <typename T> void P(const 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);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
    c = MIN(++a,++b);
    P(a); P(b); P(c);
}
```

```
233444566
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

#4

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

```
1
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

```
13
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

```
133
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

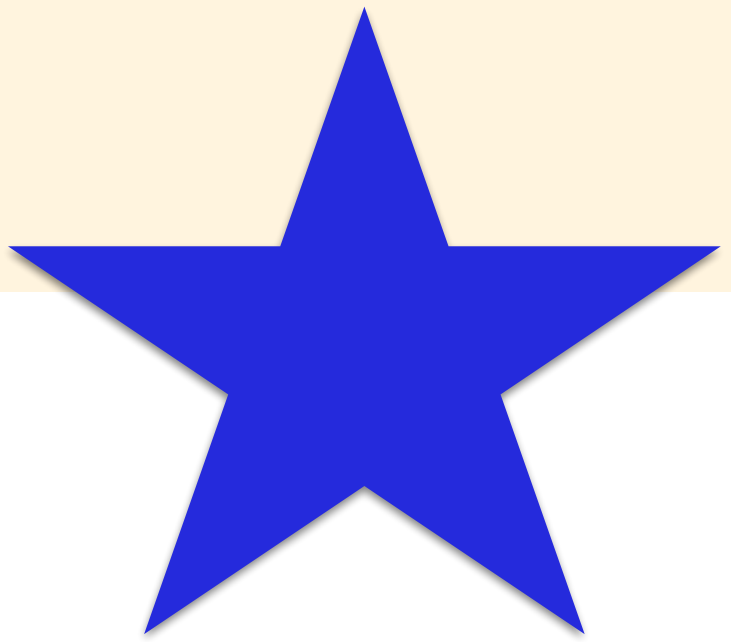
1 bouns point if the new rules for auto deduction from braced-init-list was discussed

```
133
```

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

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

int main()
{
    int i = 3;
    auto a = {i};
    auto b{i};
    decltype(auto) c{i};
    std::vector<int> v1(a);
    std::vector<int> v2(b);
    std::vector<int> v3(c);
    P(v1.size());
    P(v2.size());
    P(v3.size());
}
```

1 bouns point if the new rules for auto deduction from braced-init-list was discussed

```
133
```



•

#5

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a5
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a5b
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a5b6
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a5b6c
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

```
1a2b3c4a5b6c1
```

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

I point if the word "capture" was mentioned in you discussions

```
1a2b3c4a5b6c1
```

#5

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

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

void foo(const auto & str, auto & mylambda)
{
    for (auto ch : str)
        mylambda(ch);
}

int main()
{
    int num = 1;
    std::string str("abc");
    auto f = [=](char ch) mutable { P(num++); P(ch); };
    foo(str, f);
    foo(str, f);
    P(num);
}
```

I point if the word “capture” was mentioned in you discussions

```
1a2b3c4a5b6c1
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

#6

```cpp
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

1

#6

```cpp
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
la
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
lak
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
lako
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
lakos
```

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
lakose
```

#6

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

```
lakosed
```

#6

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

2 points if the origin and meaning of the adjective "lakosed" was discussed.

lakosed

```
#include <iostream>
#include <algorithm>
using namespace std::literals;

std::string str;

void foo(double) { str += 'd'; }
void foo(const std::string &) { str += 'e'; }
void foo(const void *) { str += 's'; }
void foo(int) { str += 'l'; }
void foo(unsigned int) { str += 'f'; }
void foo(short) { str += 'o'; }
void foo(std::nullptr_t) { str += 'a'; }
void foo(float) { str += 'b'; }
void foo(long) { str += 'k'; }

int main() {
    short s = 3;
    foo(s);
    foo(+s);
    foo(1.0);
    foo("hello");
    foo(nullptr);
    foo(NULL);
    foo("hello"s);
    for (int i = 0; i < 3862; ++i)
        std::next_permutation(str.begin(), str.end());
    std::cout << str;
}
```

2 points if the origin and meaning of the adjective "lakosed" was discussed.

```
lakosed
```

Lakosed (*adjective*) To become more drunk and go to bed far later than you had planned because you were intercepted by someone who bought you a whole load of drinks as you were thinking of retiring for the evening.

```
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```
1
```

```
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```
10
```

```
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

10

1 point if you had a discussion about trigraphs in your group.

#7

```cpp
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```
10
```

I point if you had a discussion about trigraphs in your group.

I point if someone in your group quoted something along the lines of:
"I'll use regular expressions." Now you have two problems.

#7

```cpp
#include <iostream>
#include <regex>

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

int main()
{
    std::string myrubbish("*!??<!1711=!??>?&");
    std::regex myregex("(.*)([0-9]+)(.*)");
    std::smatch mymatch;
    std::regex_match(myrubbish, mymatch, myregex);
    P(mymatch[2]);
    P(0);
}
```

```
10
```

1 point if you had a discussion about trigraphs in your group.

1 point if someone in your group quoted something along the lines of:
“I'll use regular expressions.” Now you have two problems.

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
   return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
   std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
   foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

```
1
```

```cpp
#include <iostream>
#include <type_traits>
#include <utility>

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

```
10
```

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
   return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
   std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
   foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

```
101
```

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
     bar(2,3,1,5,4.5);
}
```

```
1010
```

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
     bar(2,3,1,5,4.5);
}
```

```
10102
```

#8

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

```
101024
```

I bonus point for discussing pretty printing of tuples.

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
     bar(2,3,1,5,4.5);
}
```

```
101024
```

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

template<typename ...Args>
typename std::common_type<Args...>::type sum(Args && ...args)
{
    return (args + ...);
}

template<size_t ...Is>
void foo(std::index_sequence<Is...>)
{
    std::cout << sum(Is...) << (1 << ... << Is);
}

template<typename ...T>
void bar(T && ...)
{
    foo(std::make_index_sequence<sizeof...(T)>());
}

int main()
{
    bar(2,3,1,5,4.5);
}
```

1 bonus point for discussing pretty printing of tuples.

```
101024
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

2

#9

```cpp
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
24
```

```cpp
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
241
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
2412
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

```
24124
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

I point if alignment and padding was discussed in your group

```
24124
```

```
#include <iostream>

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

struct A { int a; char b; int c; char * d; };

class B {
public:
};

class C : B {
public:
    A a;
    int get_value() { return a.a; }
};


int main()
{
    P(sizeof(A));
    P(sizeof(B));
    P(sizeof(C));
}
```

1 point if alignment and padding was discussed in your group

```
24124
```

```
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```cpp
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```
1
```

#10

```cpp
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

12

```cpp
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```
122
```

```
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```
1223
```

```
#include <iostream>

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

struct Foo {
  void bar() & { P(1); }
  void bar() const & { P(3); }
  void bar() && { P(2); }
  void bar() const && { P(4); }
};

int main()
{
   Foo f;
   f.bar();
   Foo().bar();
   std::move(f).bar();
   [=]{ f.bar(); }();
}
```

```
1223
```

.

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

4

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
42
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
421
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
4213
```

```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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
42132
```

```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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
421323
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
4213231
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
42132314
```

```
#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 = 1; }

int main() {
    X e(2);
    foo();
    static X f(3);
    foo();
}

X b(4);
```

```
42132314
```

```
#include <iostream>

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

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

```
#include <iostream>

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

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

4

```
#include <iostream>

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

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

```
42
```

```
#include <iostream>

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

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

```
42
```

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; }

struct override {
    virtual void f(int) = 0;
};

struct final final : override {
    void f(int final) final override { P(override); P(final); }
    int override = 4;
};

int main()
{
    final f;
    f.f(2);
}
```

```
42
```

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

The winners of the C++ Pub Quiz ACCU 2016 (with an amazing score of 48)

Team 42

++

If you are into C++ you should definitely visit:
isocpp.org

If you enjoy C++ quiz in general, then have a go at:
cppquiz.org

If you like these slides and want to find more, have a look at:
olvemaudal.com/talks

And finally, if you are curious about the sponsor for this particular event:
techatbloomberg.com