Kilian Henneberger

C++ User Group Aachen

08. März 2023

Wer wird Millionär - C++ Quiz

- Ein C++ Quiz im Stil der TV - Show "Wer wird Millionär"
  - Keine Joker (notfalls gebe ich Hinweise)
  - Kein echtes Preisgeld 😯
- Wir haben ca. 30 Minuten
- Credits an SlideLizard für das PowerPoint Template

Zu den Fragen

- Schwierigkeit der Fragen
  - 1 - 5: zum Einstimmen
  - 6 - 10: sehr fortgeschritten
  - 11 - 14: absurde Herausforderungen
  - 15: Netter Abschluss
- Wir gehen von C++20 aus
- Wir gehen davon aus, dass nötige header und using namespace Anweisungen vorhanden sind
- Noch Fragen?

€ 50

`class`, `default`, `using` und `requires` sind jeweils ein...

A: Scherzwort

B: Schimpfwort

C: Schreckenswort

D: Schlüsselwort

€ 50

`class`, `default`, `using` und `requires` sind jeweils ein...

A: Scherzwort

B: Schimpfwort

C: Schreckenswort

D: Schlüsselwort


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 € 1.000
5 € 500
4 € 300
3 € 200
2 € 100
1 • € 50

€ 100

Welches der folgenden ist kein keyword?

A: `co_await`

B: `co_mpiler`

C: `co_yield`

D: `co_return`

€ 100

Welches der folgenden ist kein keyword?

A: `co_await`

B: `co_mpiler`

C: `co_yield`

D: `co_return`


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 € 1.000
5 € 500
4 € 300
3 € 200
2 • € 100
1 • € 50

€ 200

`std::variant` ist Teil der STL seit...

A: C++11

B: C++14

C: C++17

D: C++20

€ 200

`std::variant` ist Teil der STL seit...

A: C++11

B: C++14

C: C++17

D: C++20


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 € 1.000
5 € 500
4 € 300
3 • € 200
2 • € 100
1 • € 50

€ 300

Wobei handelt es sich nicht um eine C++ Konferenz?

A: C++ Insights

B: C++ Now

C: C++ Russia

D: C++ on Sea

€ 300

Wobei handelt es sich nicht um eine C++ Konferenz?

A: C++ Insights

B: C++ Now

C: C++ Russia

D: C++ on Sea


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 € 1.000
5 € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

€ 500

Den größten Zahlenwert hat...

A: `0b111'111`

B: `'B'`

C: `1 << 6`

D: `0x3D`

€ 500

Den größten Zahlenwert hat...

A: `0b111'111`

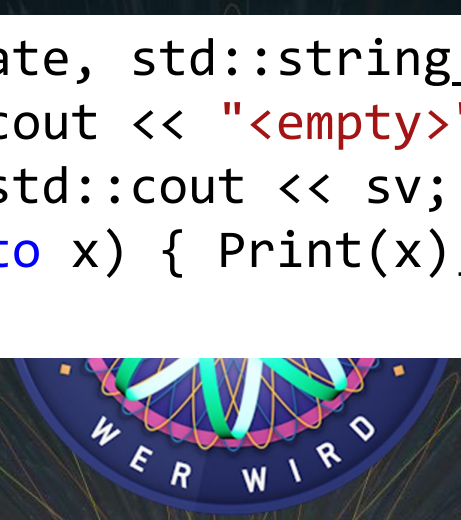
B: `'B'`

C: `1 << 6`

D: `0x3D`


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```cpp
using V = std::variant<std::monostate, std::string_view, double>;
void Print(std::monostate) { std::cout << "<empty>"; }
void Print(std::string_view sv) { std::cout << sv; }
void Print(V v) { std::visit([](auto x) { Print(x); }, v); }
int main() { Print(V{2.0}); }
```



€ 1.000

Das Snippet...

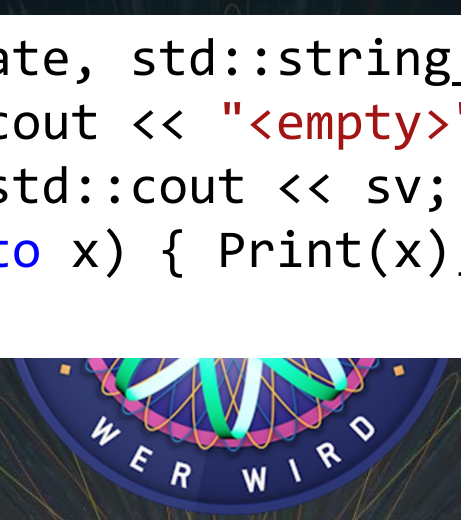
A: ist valide und gibt `<empty>` aus.

B: ist valide und gibt `2.0` aus.

C: compiliert nicht.

D: compiliert, aber führt zu einem Laufzeitfehler.

```cpp
using V = std::variant<std::monostate, std::string_view, double>;
void Print(std::monostate) { std::cout << "<empty>"; }
void Print(std::string_view sv) { std::cout << sv; }
void Print(V v) { std::visit([](auto x) { Print(x); }, v); }
int main() { Print(V{2.0}); }
```



€ 1.000

Das Snippet...

A: ist valide und gibt `<empty>` aus.

B: ist valide und gibt `2.0` aus.

C: compiliert nicht.

D: compiliert, aber führt zu einem Laufzeitfehler.


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

€ 2.000

Wobei handelt es sich nicht um eine *boost* Bibliothek?

A: Core

B: Describe

C: Monster

D: Stacktrace

€ 2.000

Wobei handelt es sich nicht um eine *boost* Bibliothek?

A: Core

B: Describe

C: Monster

D: Stacktrace


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```cpp
template<auto V> struct End {
    bool operator==(auto pos) const { return *pos == V; }
};
using P = std::pair<int, int>;
std::vector<P> points{ {2, 5}, {4, 6}, {0, 0}, {1, 7} };
std::ranges::subrange sr(points.begin(), End<P(0, 0)>());
fmt::print("{}", sr
    | std::views::elements<0>
    | std::views::reverse);
```

€ 4.000

Was gibt dieses Snippet aus?

A: [5, 2]

B: [4, 6]

C: [0, 4, 2]

D: [4, 2]

```cpp
template<auto V> struct End {
    bool operator==(auto pos) const { return *pos == V; }
};
using P = std::pair<int, int>;
std::vector<P> points{ {2, 5}, {4, 6}, {0, 0}, {1, 7} };
std::ranges::subrange sr(points.begin(), End<P(0, 0)>());
fmt::print("{}", sr
    | std::views::elements<0>
    | std::views::reverse);
```

€ 4.000

Was gibt dieses Snippet aus?

A: [5, 2]

B: [4, 6]

C: [0, 4, 2]

D: [4, 2]


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```cpp
struct X { virtual ~X() = default; };
struct Y {
    virtual ~Y() = default;
    std::string s;
};
bool BX = std::is_nothrow_move_constructible_v<X>;
bool BY = std::is_nothrow_move_constructible_v<Y>;
```

€ 8.000

Welche Aussage stimmt?

A: BX und BY sind beide false.

B: BX ist false und BY ist true.

C: BX ist true und BY ist false.

D: BX und BY sind beide true.

```cpp
struct X { virtual ~X() = default; };
struct Y {
    virtual ~Y() = default;
    std::string s;
};
bool BX = std::is_nothrow_move_constructible_v<X>;
bool BY = std::is_nothrow_move_constructible_v<Y>;
```

€ 8.000

Welche Aussage stimmt?

A: BX und BY sind beide false.

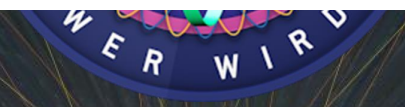
B: BX ist false und BY ist true.

C: BX ist true und BY ist false.

D: BX und BY sind beide true.


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```
struct Deleter : std::default_delete<int> {
    std::string s = "Just a member that makes some
                     constructors potentially throwing.";
};
Deleter d;
int* p = new int();
```



€ 16.000

Welche Art, einen `std::shared_ptr` zu erzeugen, hat möglicherweise undefined behavior?

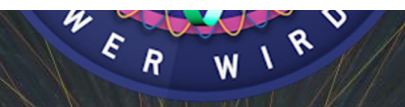
A: `shared_ptr<int>(p)`

B: `shared_ptr<int>(p, d)`

C: `shared_ptr<int>(p, std::ref(d))`

D: `shared_ptr<int>(p, std::move(d))`

```cpp
struct Deleter : std::default_delete<int> {
    std::string s = "Just a member that makes some
                    constructors potentially throwing.";
};
Deleter d;
int* p = new int();
```



€ 16.000

Welche Art, einen `std::shared_ptr` zu erzeugen, hat möglicherweise undefined behavior?

A: `shared_ptr<int>(p)`

B: `shared_ptr<int>(p, d)`

C: `shared_ptr<int>(p, std::ref(d))`

D: `shared_ptr<int>(p, std::move(d))`

15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

€ 32.000

Das einzig gültige literal ist...

A: `0x.ep2`

B: `0815`

C: `u32'\"'`

D: `-0b'11`

€ 32.000

Das einzig gültige literal ist...

A: `0x.ep2`

B: `0815`

C: `u32'\"'`

D: `-0b'11`

15 € 1 MILLION
14 € 500.000
13 € 125.000
12 € 64.000
11 • € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

€ 64.000

Welches Snippet compiliert nicht?

A: `views::reverse ("A1")`

B: `ranges::reverse ("B2")`

C: `views::reverse (string("C3"))`

D: `ranges::reverse (string("D4"))`

€ 64.000

Welches Snippet compiliert nicht?

A:
```
views::reverse
("A1")
```

B:
```
ranges::reverse
("B2")
```

C:
```
views::reverse
(string("C3"))
```

D:
```
ranges::reverse
(string("D4"))
```


15 € 1 MILLION
14 € 500.000
13 € 125.000
12 • € 64.000
11 • € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```
struct PingPong {
    std::atomic_bool isRunning = true;
    std::jthread ping{[this]{
        while (isRunning) { /*do your ping thing*/ }
    }};
    std::jthread pong{[this]{
        while (isRunning) { /*do your pong thong*/ }
    }};
    ~PingPong() { isRunning = false; }
};
```

€ 125.000

Eine Variable der Klasse `PingPong` zu erzeugen, ...

A: kann zu einer exception und einem dead lock führen.

B: scheitert, da sie keinen default constructor hat.

C: ist aufgrund eines Compilerfehlers innerhalb der Klasse nicht möglich.

D: klappt und ist garantiert exception safe.

```
struct PingPong {
    std::atomic_bool isRunning = true;
    std::jthread ping{[this]{
        while (isRunning) { /*do your ping thing*/ }
    }};
    std::jthread pong{[this]{
        while (isRunning) { /*do your pong thong*/ }
    }};
    ~PingPong() { isRunning = false; }
};
```

€ 125.000

Eine Variable der Klasse `PingPong` zu erzeugen, ...

A: kann zu einer exception und einem dead lock führen.

B: scheitert, da sie keinen default constructor hat.

C: ist aufgrund eines Compilerfehlers innerhalb der Klasse nicht möglich.

D: klappt und ist garantiert exception safe.


15 € 1 MILLION
14 € 500.000
13 • € 125.000
12 • € 64.000
11 • € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```cpp
class Impl;
auto P = new                 std::unique_ptr<Impl> ();
auto Q = new std::tuple<std::unique_ptr<Impl>>();
```

€ 500.000

Welche Aussage über die Initialisierungen von P und Q stimmt?

A: Keine der beiden compiliert.

B: Nur die von P compiliert.

C: Nur die von Q compiliert.

D: Beide compilieren.

```cpp
class Impl;
auto P = new                 std::unique_ptr<Impl> ();
auto Q = new std::tuple<std::unique_ptr<Impl>>();
```

€ 500.000

Welche Aussage über die Initialisierungen von P und Q stimmt?

A: Keine der beiden compiliert.

B: Nur die von P compiliert.

C: Nur die von Q compiliert.

D: Beide compilieren.

15 € 1 MILLION
14 • € 500.000
13 • € 125.000
12 • € 64.000
11 • € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

```cpp
void Schedule() {
    year_month_weekday today = Wednesday[2]/March/2023;
    year nextYear = today.year() + years{1};
    for (auto date = today + months{2};
        date.year() != nextYear; date += months{2})
    {
        year_month_day ymd{date};
        std::cout << ymd << "\n";
    }
}
```

€ 1 MILLION

Die Funktion `Schedule` gibt was aus?

A:

B:

C:

D:

```
void Schedule() {
    year_month_weekday today = Wednesday[2]/March/2023;
    year nextYear = today.year() + years{1};
    for (auto date = today + months{2};
        date.year() != nextYear; date += months{2})
    {
        year_month_day ymd{date};
        std::cout << ymd << "\n";
    }
}
```

€ 1 MILLION

Die Funktion `Schedule` gibt was aus?

Die kommenden Termine unserer Aachener User Group:

2023-05-10
2023-07-12
2023-09-13
2023-11-08

15 • € 1 MILLION
14 • € 500.000
13 • € 125.000
12 • € 64.000
11 • € 32.000
10 • € 16.000
9 • € 8.000
8 • € 4.000
7 • € 2.000
6 • € 1.000
5 • € 500
4 • € 300
3 • € 200
2 • € 100
1 • € 50

€ 1 MILLION
GRATULATION!