Modern C++ tips & tricks

- Kamil Szatkowski, kamil.szatkowski@nokia.com
- Łukasz Ziobroń, lukasz.ziobron@nokia.com

- 2015-10-24

NOKIA

About authors

Kamil Szatkowski

- PWr graduate, W8
- C++ software engineer - Nokia, LTE, CPlane
- Code Reviewer
- Ocassional trainer (Modern C++, Memory management in C++)
- Lecturer (AMPPZ)
- Blogger (netrix.org.pl)

Łukasz Ziobroń

- PWr graduate, W4
- C++ software engineer - Nokia, LTE, LOM
- Scrum Master
- Ocassional trainer (Modern C++, Memory management in C++)
- Lecturer (code::dive 2015, AMPPZ)
- Blogger (ziobron.net)

Introduction to C++11 standard

C++ standarization history

- 1998 – first ISO C++ standard
- 2003 - TC1 (“Technical Corrigendum 1”) published as (“C++03”). Bug fixes for C++98
- 2005 - “Technical Report 1” published
- 2011 – ratified C++0x -> C++11
- 2013 – full version of C++14 draft
- 2014 - C++14 published (minor revision)
- 2017? – big standard modification planned as C++17

Introduction to C++14 standard
Compilers support

C++11 support

- Selected features from C++11 - gcc4.3, clang2.9
- Full support - gcc4.6, clang3.3
- Compiler flag:
  - -std=c++0x
  - -std=c++11 – since gcc4.7, clang3.3
- More details:
  - http:://gcc.gnu.org/projects/cxx0x.html
  - http://clang.llvm.org/cxx_status.html

C++14 support

- Basic functionality - gcc4.9, clang3.3
- Full support – gcc5, clang3.4
- Compiler flag:
  - -std=c++1y
  - -std=c++14
- More details:
  - http:://gcc.gnu.org/projects/cxx1y.html
  - http://clang.llvm.org/cxx_status.html

Question 1

Which of these keywords were introduced in C++11 standard?

A. `alignas`

B. `alignof`

C. `char16_t`

D. `char32_t`

E. `constexpr`

F. `decltype`

G. `noexcept`

H. `nullptr`

I. `static_assert`

J. `thread_local`

Question 2

Passing empty pointer to `bar` function:

```
bar(nullptr);
```

Which of its overloaded version will be called?

```
A.  void bar(int);
B.  void bar(void*);
C.  void bar(nullptr_t);
```

Question 3

What type is variable `something`**?**

```
auto something = {1, 2, 3, 4, 5};
```

A. `std::vector<int>`
B. `std::initializer_list<int>`
C. `std::array<int, 5>`
D. `int[]`

Question 4

What will be the result of the following code?

```
template<class T>
void foo(T v) {
   std::cout << v << std::endl;
}

foo({1,2,3,4,5});
```

Question 5

What will be the result of the following code?

```
template<class T>
void foo(T v) {
   std::cout << v << std::endl;
}

auto v = {1,2,3,4,5};
foo(v);
```

Question 6

Will it compile?

```
int array[] = { 1, 2, 5.5 };
```

Question 7

Are following lines of code correct?

```
std::array<int, 3> a1{1, 2, 3};
std::array<int, 3> a2 = {1, 2, 3};
```

Question 8

What will happen?

```
int main()
{
    int item();
    item = 5;
    std::cout << item;
}
```

Question 9

What type is g1?

```
Gadget items[10];
auto g1 = items;
```

Question 10

What is wrong with this code?

```
std::map<int, std::string> m;
// ... filling m ...
for(std::pair<int, std::string> const& elem: m)
    cout << elem.first << " -> " << elem.second << endl;
```

Question 11

Will it work or not?

```
auto get_name(int id)
{
    if (id == 1)
        return "Gadget"s;
    else if (id == 2)
        return "SuperGadget"s;
    return string("Unknown");
}
```

Question 12

Following code is provided:

```
struct Foo {
 void talk() const& {std::cout << „talk const&” << std::endl; }
 void talk() const&& {std::cout << „talk const&&” << std::endl;}
 void talk() & { std::cout << „talk &” << std::endl; }
 void talk() && { std::cout << „talk &&” << std::endl; }
};

decltype(auto) declFoo() {
  Foo f;
  return f;
}
decltype(auto) declBar() {
  Foo f;
  return(f);
}
```

```
declFoo().talk();
declBar().talk();
```

What will happen?

Question 13

What will happen?

```
struct AAAAA {
  static const int VALUE = 213;
};
std::cout << AAAAA::VALUE << std::endl;
```

Question 14

What will happen?

```
struct AAAAA {
  static const int VALUE = 213;
};
std::cout << &AAAAA::VALUE << std::endl;
```

Question 15

Will following code compile?

```
constexpr int max(int a, int b) {
  if(a > b)
     return a;
  else
     return b;
}
```

Question 16

Is this code correct?

```
struct A
{
    void bar() const final {}
};
struct B : A
{
    void bar() const {}
};
```

Question 17

Which of `override` specifiers are used correctly?

```
struct A {
    virtual void foo() = 0;
    void dd() {}
};
struct B : A {
    void foo() override {}
    void bar() override {}
    void dd() override {}
}
```

Question 18

Is there a difference?

```
auto x = std::make_shared<std::string>("hello, world!");
std::shared_ptr<std::string> y{new std::string("hello, world!")};
```

Question 19

Consider following code (C++14):

```
auto a = std::make_shared<int>(1);
auto b = std::make_unique<int>(2);
template<class A>
void foo(A a) { /**/ }
```

Which line will not compile?

A. `foo(a);`

B. `foo(b);`

C. `foo(std::move(a));`

D. `foo(std::move(b));`

Question 20

Which line will not compile?

```
1 [](){};
2 []{};
3 {};
4 []();
5 [];
6 ();
7 [](){}();
```

Question 21

What will happen?

```
struct Sth {
    int* a = new int(2);
    int b = 3;
    Sth() { std::cout << [&](){ return *a + b; }() << std::endl; }
};
int main() {
    Sth s;
}
```

Question 22

What will happen?

```
struct Sth {
    int* a = new int(2);
    int b = 3;
    Sth() { std::cout << [=](){ return *a + b; }() << std::endl; }
};
int main() {
    Sth s;
}
```

Question 23

What will happen?

```
struct Sth {
    int* a = new int(2);
    int b = 3;
    Sth() { std::cout << [&a,b](){ return *a + b; }() << std::endl; }
};
int main() {
    Sth s;
}
```

Question 24

How to capture `std::unique_ptr` in lambda expression?

Question 25

How to capture `std::unique_ptr` in lambda expression?

How to do it in C++11?

Question 26

Will following code compile?

```
struct Foo {
  void talk()
  {}
  void talk() &&
  {}
};
```

Question 27

Can free function (not from class) be marked with `delete` keyword?

Question 28

Following code is provided:

```
struct Foo {
  virtual void call(unsigned a) {}
  void call(int a) = delete;
};
struct Bar : Foo {
};

Bar b;
b.call(1u); // A
b.call(1);  // B
Foo & f = b;
f.call(1u); // C
f.call(1);  // D
```

Which of `call` method calls will be blocked by the compiler?

Question 29

Following code is provided:

```
struct Foo {
  virtual void call(unsigned a) {}
  void call(int a) = delete;
};
struct Bar : Foo {
  void call(int a) = delete;
};
Bar b;
b.call(1u); // A
b.call(1);  // B
Foo & f = b;
f.call(1u); // C
f.call(1);  // D
```

What will happen?

Question 30

Following code is provided:

```
struct Foo {
  virtual void call(unsigned a) {}
  virtual void call(int a) final = delete;
};
struct Bar : Foo {
  void call(int a) = delete;
};
Bar b;
b.call(1u); // A
b.call(1);  // B
Foo & f = b;
f.call(1u); // C
f.call(1);  // D
```

What will happen?