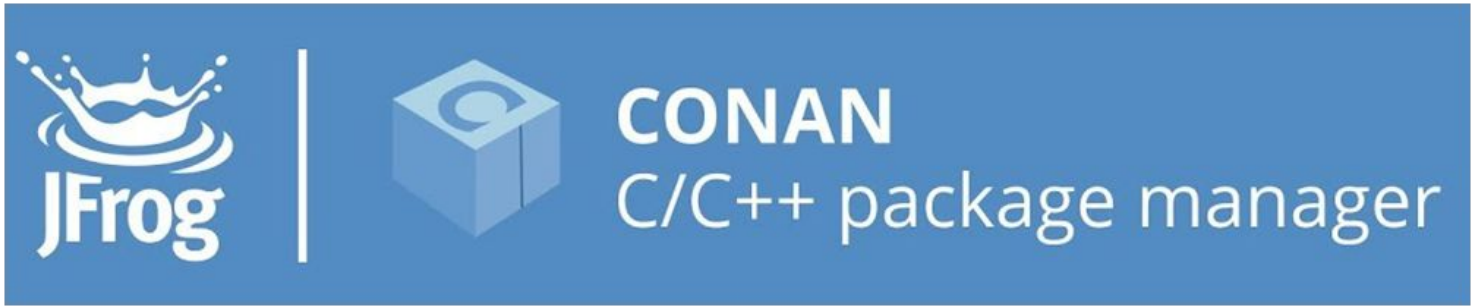
Almost Always Auto

Madrid C/C++ Meetup
25-Oct-18

About me

Diego Rodriguez-Losada

@diegorlosada



C++11

```
using namespace std::literals;

int main() {
    auto x = 42; // int
    auto y = 42.0f; // float
    auto z = 42ul; // unsigned long
    auto s1 = "42"s; // C++14, std::string
    auto t1 = 42ns; // C++14, std::nano_seconds
```

Always Almost Auto

Herb Sutter:

https://herbsutter.com/2013/08/12/gotw-94-solution-aaa-style-almost-always-auto/

Declaring (left to right)

```
auto i = 0, j = 2, k = 3;                  //OK
auto a = 2, b = "Hello World", c = 3.2; //SYNTAX ERROR
auto a = 2, &b = a;                        //OK
auto c;                                    //SYNTAX ERROR

//heap
auto v2 = std::make_unique<Employee>("Pepe");
//stack
auto v = Employee{ "Name" };
```

Beware of references!

```
int x = 1;
int& y = x;
auto z = y;
z++;
std::cout << x << "\n";
```

```
int x = 1;
int& y = x;
auto& z = y;
z++;
std::cout << x << "\n";
```

With decltype

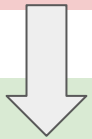
```
int x = 1;
int& y = x;
decltype(auto) z = y;
z++;
std::cout << x << "\n";
```

const

```
int x = 1;
const int& y = x;
auto& z = y;
z++; //SYNTAX ERROR
```

From iterators to range-fors

```
auto v = std::vector<int>{ 1, 2, 3, 4 };
for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
       std::cout << *it << " ";
```



```
for (auto value : v)
   std::cout << value << " ";
for (auto& value : v)
   value++;
for (const auto& value : v)
   std::cout << value << " ";
```

Function return

```
auto add(int a, int b){
    return a + b;
}

int main(){
    auto r = add(2, 3);
}
```

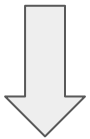
Function params (not VS2017)

```
auto add(auto a, auto b){
    return a + b;
}

int main(){
    auto r = add(2, 3);
    auto r2 = add(2.1, 3.2);
}
```

Lambda arguments

```
[] (int a, int b) {return a + b;}
```



```
[] (auto a, auto b) {return a + b;}
```

Lambda arguments

```
template <typename T>
auto mycall(T t){
    return t(2.1, 3.2) + t(1, 2);
};

std::cout << mycall([](auto a, auto b) {return a + b;}) ;
std::cout << mycall([](auto a, auto b) {return int(a + b);}) ;
```

Lambdas types

```
auto v = std::vector<int>{ 1, 2, 3, 4 };
auto inc = 3;
//Works without capture
void (*myout)(int&) = [](int& a) {std::cout << a << " "; };
std::for_each(v.begin(), v.end(), myout);
std::cout << "\n";
//Better
auto mylamb = [=](int& a) {a += inc; };
std::for_each(v.begin(), v.end(), mylamb);
```

Structure Bindings

```
auto f() { //  -> std::tuple <int, double> opcional!
    return std::make_tuple(1, 123.4);
}

auto [a, b] = f(); // vs int a; double b; std::tie(a, b) = f();
std::cout << a << ", " << b << "\n";
```

For-each & structure binding

```
std::map<int, char> m = { {1,'a'},{2,'b'} ,{3,'c'} };
for (const auto& [k, v] : m)
    std::cout << k << ":" << v << "\n";
```

Readability yes/no?

```
auto readable_function() {
    auto v = GetString();
    for (auto c : v)
        std::cout << c;
    return v.length();
}
```