Lightning Talk: Using `auto`

Edwin Kofler

El Camino College Computer Science Club

March 24, 2023

Table of Contents

Motivation

Why learn extra C++ features?

- You will read someone else's code than you write your code
- You may personally prefer using the feature
- Exposure to different features/techniques improves your skills
- May be part of an Interview Question

Feature 1: Placeholder Type Specifier

Before

```
1    int main() {
2        int i = 10;
3    }
```

Feature 1: Placeholder Type Specifier

Before

```
1    int main() {
2        int i = 10;
3    }
```

After

```
1    int main() {
2        auto i = 10; // ?
3    }
```

Feature 1: Placeholder Type Specifier

Before

```
1    int main() {
2        int i = 10;
3    }
```

After

```
1    int main() {
2        auto i = 10; // ?
3    }
```

Use "auto" instead of a type as a "placeholder". Let the C++ compiler deduce (figure out) the type.

What are the types of `i` and `j`?

```
1    int main() {
2        auto i = 1.0; // ?
3        auto j = 1.0F; // ?
4    }
```

Feature 1: Placeholder Type Specifier

Before

```
1    int main() {
2        int i = 10;
3    }
```

After

```
1    int main() {
2        auto i = 10; // ?
3    }
```

Use "auto" instead of a type as a "placeholder". Let the C++ compiler deduce (figure out) the type.

What are the types of `i` and `j`?

```
1    int main() {
2        auto i = 1.0; // ?
3        auto j = 1.0F; // ?
4    }
```

`i` is a `double` and `j` is a `float`.

Feature 1: Knowledge Test

```
1  int some_function() { return 2; };
2
3  int main() {
4      auto example_1 = some_function(); // 1
5
6      auto example_2 = "other text"; // 2
7
8      auto const example_3 = 77; // 3
9
10     int &num = 10;
11     auto example_4 = num; // 4
12 }
```

Feature 1: Knowledge Test

```
1  int some_function() { return 2; };
2
3  int main() {
4      auto example_1 = some_function(); // 1
5
6      auto example_2 = "other text"; // 2
7
8      auto const example_3 = 77; // 3
9
10     int &num = 10;
11     auto example_4 = num; // 4
12 }
```

- example_1 is an int

Feature 1: Knowledge Test

```
1  int some_function() { return 2; };
2
3  int main() {
4      auto example_1 = some_function(); // 1
5
6      auto example_2 = "other text"; // 2
7
8      auto const example_3 = 77; // 3
9
10     int &num = 10;
11     auto example_4 = num; // 4
12 }
```

- example_1 is an int
- example_2 is a char*

Feature 1: Knowledge Test

```
1  int some_function() { return 2; };
2
3  int main() {
4      auto example_1 = some_function(); // 1
5
6      auto example_2 = "other text"; // 2
7
8      auto const example_3 = 77; // 3
9
10     int &num = 10;
11     auto example_4 = num; // 4
12 }
```

- example_1 is an int
- example_2 is a char*
- example_3 is a int const

Feature 1: Knowledge Test

```
1  int some_function() { return 2; };
2
3  int main() {
4      auto example_1 = some_function(); // 1
5
6      auto example_2 = "other text"; // 2
7
8      auto const example_3 = 77; // 3
9
10     int &num = 10;
11     auto example_4 = num; // 4
12 }
```

- example_1 is an int
- example_2 is a char*
- example_3 is a int const
- example_4 is an int

auto "ignores" references

Feature 2: Trailing Return Types

What does this code do?

```
1  #include <iostream>
2
3  int add_two(int num) {
4      int newNum = num + 2;
5      return newNum;
6  }
7
8  int main() {
9      int num1 = 11;
10     int num2 = add_two(num1);
11     std::cout << num2 << '\n';
12 }
```

Feature 2: Trailing Return Types

Before

```
1  int add_two(int num) {
2      int newNum = num + 2;
3      return newNum;
4  }
```

Feature 2: Trailing Return Types

Before

```
1  int add_two(int num) {
2      int newNum = num + 2;
3      return newNum;
4  }
```

After

```
1  auto add_two(int num) -> int {
2      int newNum = num + 2;
3      return newNum;
4  }
```

Feature 2: Full Example

```
1  #include <iostream>
2
3  auto add_two(int num) -> int {
4      int newNum = num + 2;
5      return newNum;
6  }
7
8  auto main() -> int {
9      int num1 = 11;
10     int num2 = add_two(num1);
11     std::cout << num2 << '\n';
12 }
```

Using Both Features

```cpp
#include <iostream>

auto add_one(int num) -> int {
    return num + 1;
}

auto add_fraction(int num) {
    return num + 2.5;
}

int main() {
    auto num1 = add_one(10);
    auto num2 = add_fraction(20);

    std::cout << num1 << '\n';
    std::cout << num2 << '\n';
}
```