COP3503

The auto Keyword

auto Keyword Simplifies Declaration

- Syntax for many things in C++ can be at cumbersome.

```
// Creating an iterator... not the nicest code to look at
unordered_map<string, Vehicles>::iterator iter = someContainer.begin();
```

- We often have to repeat data types in our code.

```
SomeClass::NestedClass* somePtr = new SomeClass::NestedClass;
```

- The auto keyword simplifies variable declaration.

```
// Clean, easy to write!
auto iter = vehicles.begin();
auto somePtr = new SomeClass::NestedClass;
```

- With **auto**, the **compiler** determines a variable's type, so you don't have to write it.

How Does `auto` Work?

- Variable type is determined by what's on the **right side** of the assignment operator.
- This is done at **compile time**—it has no effect on how your program runs.
- This only has an effect **once**, during variable definition.

```
// What type of data is example?
// That depends on what someValue is...
auto example = someValue;
```

Is someValue an `int`?
`auto` compiles as `int`.

Is someValue a `vector<float>`?
`auto` compiles as `vector<float>`.

- `auto` **requires** initialization of the variable, or you'll get a compiler error.

```
// What type is this? Hard to infer a type from nothing...
auto wontWork;

auto noProblem = SomeFunction();
auto pi = 3.14f;
auto pointer = new double[100];
```

Good for Libraries Where You Don't Know (or Remember) the Return Details

```
????? someRectangle = projectiles[i]->GetSprite().getGlobalBounds();
```

You've forgotten if `getGlobalBounds` returns: `Rectangle<float>` or `Rectangle<int>`

```
template <typename T>
struct Rectangle
{
    T left;
    T top;
    T width;
    T height;
};
```

- You know to use the Rectangle class, you know what variables and functions it has, etc…
- Choose the wrong one, it's a compiler error that costs you a few seconds…
- But a few seconds, multiplied by this same situation dozens, hundreds of times…

```
// Let the compiler figure it out for you (work smarter, not harder!)
auto someRectangle = projectiles[i]->GetSprite().getGlobalBounds();

if (someRectangle.left < 100 && someRectangle.top > 50)
    // Do something important with this object...
```

The data type, while important… is less important.

This is the part you should care more about.

Maybe Not So Good for Primitive Types…

- The auto keyword also works for primitive types.

```
auto val1 = 5;          // int
auto val2 = '5';        // char
auto val3 = "5";        // const char*
auto val4 = 5.0f;       // float
auto val5 = 5.0;        // double
auto val6 = 5000000000000000;  // long long
```

Do you really **need** the auto keyword? (If so, why?)

Forget how to type `int`, `float`, `char`?

"Too lazy" to type `int` or `char`?
(int actually **saves** you some effort…)

- There are times when you want control over what happens.

```
// Task: Create a small (in memory) variable
// to store the number 50
auto notSmall = 50; // int, 4 bytes
char smaller = 50;  // 1 byte, better!
```

```
// Task: Create a std::string variable
std::string realHero = "Batman";
auto impostor = "Batman"; // const char*
```

`auto` is just another programming tool—they all have a time and a place.

`auto` (or Similar) in Other Languages

- The auto keyword in C++ is just one example of **type inference**.
  - A variable's type is inferred by something else.
- C# has the var keyword (works similarly to `auto`):

```
var message = "Hello, world!"; // string (C# uses strings
var count = 5;   // int
var letter = 'A'; // char
```

- Python doesn't require type declaration at all.

```
someValue = 'Hello, world!'
count = 5
letter = 'A'
```

This concept is sometimes referred to as "Duck Typing".

If it looks like a duck, and quacks like a duck, it must be a duck!

If it looks like a string, ~~and quacks like a string~~, it must be a string!

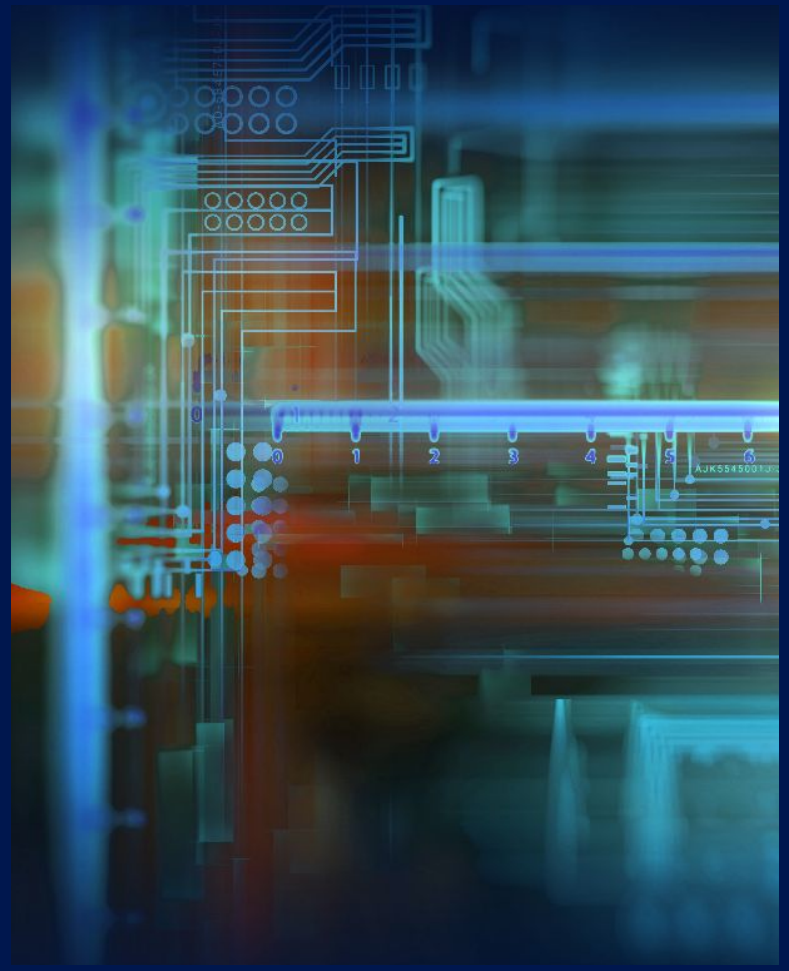
None of these styles or approaches are right or wrong.

You may prefer one over the other as a matter of style or convenience.

Recap

- The `auto` keyword lets the compiler determine data types for variables.
  - Only a compile-time feature, has no effect at runtime.
  - You **must** assign something to a variable in order to use it.
- Great for complex types, or to avoid ambiguity.
  - STL classes (especially iterators!), templates in general.
  - **Small details**:
    Does this function return a pointer, or a const pointer?
    (Auto will figure it out for you #weloveyouauto)
- Maybe not for basic data types (some may disagree).
  - Some languages encourage this for all types.
  - Some languages don't even have types.
- Personal style preference in many situations

Conclusion



Placeholder for the instructor’s welcome message. Video team, please insert the instructor’s video here.

Thank you for watching.