**References**

- **Reference**: An alias for another variable
  - **Alias**: Another name that is bound to the aliased variable
  - Mutating a reference is mutating the aliased variable
  - Introduced in C++ as part of the language

```cpp
int main(int argc, char** argv) {
    int x = 5, y = 10;
    int& z = x;
    z += 1;
    x += 1;
    z = y;
    z += 1;
    return EXIT_SUCCESS;
}
```

**Pass-By-Reference**

- C++ allows you to use real **pass-by-reference**
  - Client passes in an argument with normal syntax
  - Function uses reference parameters with normal syntax
  - Modifying a reference parameter modifies the caller's argument!

```cpp
void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
}
```

**Poll**

- What will happen when we run this?
  - **A.** Output "(1,2,3)"
  - **B.** Output "(3,2,3)"
  - **C.** Compiler error about arguments to foo (in main)
  - **D.** Compiler error about body of foo
  - **E.** We're lost...

```cpp
int main(int argc, char** argv) {
    int a = 1;
    int b = 2;
    int& c = a;
    foo(a, b, c);
    std::cout << "a: " << a << " b: " << b << std::endl;
    return EXIT_SUCCESS;
}
```

**const and Pointers**

- Pointers can change data in two different contexts:
  1. You can change the value of the pointer
  2. You can change the thing the pointer points to (via dereference)

- **const** can be used to prevent either/both of these behaviors!
  - **const** next to pointer name means you can't change the value of the pointer
  - **const** next to data type pointed to means you can't use this pointer to change the thing being pointed to

- **Tip**: read variable declaration from right-to-left
Polling Question

What will happen when we try to compile and run?

A. Output "(2, 4, 0)"
B. Output "(2, 4, 3)"
C. Compiler error about arguments to foo (in main)
D. Compiler error about body of foo
E. We’re lost...

```cpp
void foo(int* const x,
         int& y, int z) {
  *x += 1;
  y *= 2;
  z -= 3;
}

int main(int argc, char** argv) {
  const int a = 1;
  int b = 2, c = 3;
  foo(&a, b, c);
  std::cout << "(" << a << ", " << b << ", " << c << ")" << std::endl;
  return EXIT_SUCCESS;
}
```

When to Use References?

- A stylistic choice, not mandated by the C++ language
- Google C++ style guide suggests:
  - Input parameters:
    - Either use values (for primitive types like `int` or small structs/objects)
      - Avoid making unnecessary copies
    - Or use `const` references (for complex struct/object instances)
  - Output parameters:
    - Use `const` pointers
      - Unchangeable pointers referencing changeable data
  - Ordering:
    - List input parameters first, then output parameters last

```cpp
class Integer {
  public:
    Integer(int x) : x_(x) {}
    int GetValue() const { return x_; }
    void SetValue(int x) const { x_ = x; }
  private:
    int x_;}

int main(int argc, char** argv) {
  const Integer i(1);
  i.SetValue(i.GetValue() + 350);
  std::cout << i.GetValue() << std::endl;
  return EXIT_SUCCESS;
}
```