Name: _____

Today's Date: _____

Today's Goals

- Explain the difference between promotion and conversion.
- Identify when conversion will happen.
- Identify which version of an overloaded function will be called.
- Write test programs to check C++ behavior.

Today's Question(s)

Why does the compiler need to know the *type* of every variable?

Lingering Questions

Which of these will cause a compiler error?

```
int w = "bessie";

int x = 42;
x = "bessie";

double y = 3.0;

double z = 3;
```

Promotion and Conversion (primitive types)

- **Conversion**:
- **Promotion**:

Exercises

What generalization can we extract from what happens?

Conversion and user-defined types

Which of these will work?

```
void printCow(Cow cow) {...}

SomeType d;
string s = "hello";

printCow(d);
printCow(2);
printCow('c');
printCow("s");
```

Exercise

Suppose we have a Person class with constructors

```
Person::Person(const string & name)
Person::Person(const int age)
```

and a Cow class with a member function

```
Cow::addRider(const Person & rider)
```

What happens if bessie is a Cow and we say

```
string s = "Chris";

bessie.addRider(20);
bessie.addRider(Person(s));
bessie.addRider(s);
bessie.addRider("Chris");
```

Restrictions on type conversion

Why doesn't C++ transform

```
"Bessie".feedWith(GRASS, 33);
```

to

```
Cow("Bessie").feedWith(GRASS, 33);
```

?

Choosing between functions

The compiler ranks each argument match as follows

1.
2.
3.
4.

Chosen function must be both

- ▶
- ▶

Extra Practice to Test Your Understanding

Which function defintion (if any) will the compiler choose?

Function declarations

```
foo(int, int);
foo(double, double);
foo(float, double);
```

Function calls

```
foo(42, 54)
foo(3.14, 2.71)
foo(3.14f, 2.71)
foo(42, 2.71)
```

Built-in Operators

When we write the expression

```
a + c;
```

C++ translates this code to

```
a.operator+(c);
```

or to

```
operator+(a, c);
```

Think about: Why allow both?

Overloadable operators

You can overload the following binary operators:

```
+ - * / % ^
& | ~ ! , =
< > <=  >=  ++  --
<<  >>  ==  !=  &&  ||
+=  -=  /=  %=  ^=  &=
|=  *=  <<= >>= []  ()
->  ->* new new []  delete  delete []
=         [ ]         ->
```

The last row of operators can only be overloaded as member functions.

Templates

When doing conversions, the compiler looks for a non-templated operation first. If the program still doesn't type-check, then it will look for a templated operation.

Exercise: What will each of these print?

```
int main() {
    cout << 3 + 3 << endl;
    cout << 3 + 3.14 << endl;
    cout << 3.14 + 3 << endl;
    cout << 3.14 + 3.14 << endl;
}
```

```
int main() {
    int iVal = 3.14 + 3;
    cout << iVal << endl;
}
```

```
int main() {
    int i =  3.49;
    int j =  3.50;
    int k =  3.51;
    int m = -3.49;
    int n = -3.50;
    int o = -3.51;

    cout << i << endl;
    cout << j << endl;
    cout << k << endl;
    cout << m << endl;
    cout << n << endl;
    cout << o << endl;
}
```

```
int main() {
    float fVal1 = 1 / 2;
    int iVal2 = 1.75 + 1 / 2;

    cout << fVal1 << endl;
    cout << iVal2 << endl;
}
```

```
int main() {
    int negative1 = -1;
    size_t positive1 = 1;
    if (negative1 > positive1) {
        cout << "-1 > 1" << endl;
    } else {
        cout << "-1 <= 1" << endl;
    }
}
```

```
int main() {
    int p    = 'a';
    char q   = p + 3;
    int r    = 4.25 * 100;
    size_t s = -1;
    size_t t = s + 1;
    int u    = s - 1;

    cout << p << endl;
    cout << q << endl;
    cout << r << endl;
    cout << s << endl;
    cout << t << endl;
    cout << u << endl;
}
```

Extra Practice: What will each of these print?

```
void printInt(int i) {
    cout << "The int is " << i << endl;
}

int main() {
    double d = 1.2;
    printInt(d);
    printInt(2);
    printInt('c');
    printInt("hi");
}
```

```
void square(int i) {
    cout << "This int squared is " << i * i << endl;
}

void square(double x) {
    cout << "This double squared is " << x * x << endl;
}

int main() {
    square(1);
    square(1.2);
    square(3/4);
    square(3.4f);
}
```

```
void product(int i0, int i1) {
    cout << "This int product is " << i0 * i1 << endl;
}

void product(double d0, double d1) {
    cout << "This double product is " << d0 * d1 << endl;
}

int main() {
    product(1, 1);
    product(1.2, 3.4);
    product(1, 3.4);
    product(2, 'c');
    product(3.4f, 1.0);
    product(3.4f, 1);
}
```