Arithmetic Expressions and Datatype Conversions

Principles of Computer Programming I

Spring/Fall 20XX

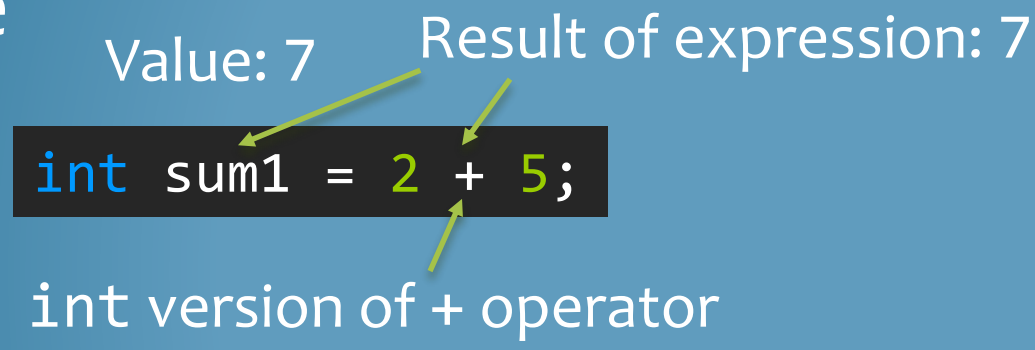
AUGUSTA UNIVERSITY

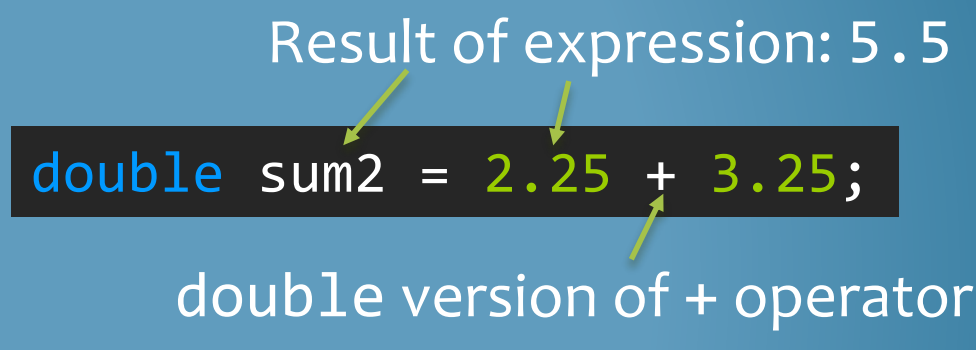
Outline

- Operators and Datatypes
- Implicit and Explicit Conversions with Operators
- Expressions and Result Types
- Order of Operations

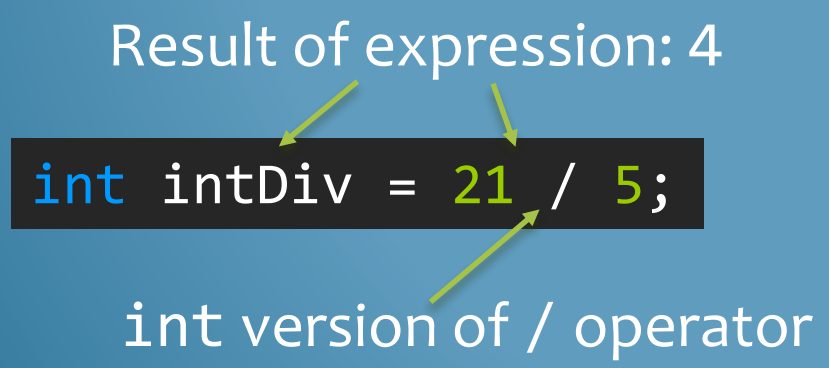
Integer vs. Fractional Arithmetic

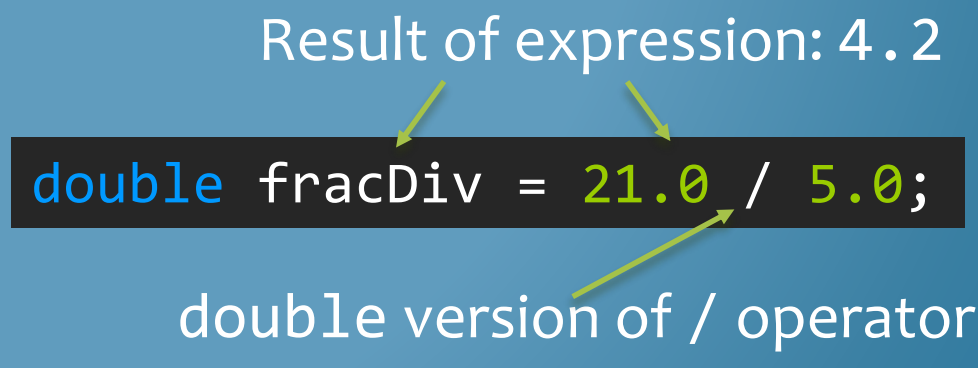
- Math operators (+, -, *, /, %) are defined separately for each type





- The `int` / operator does *integer division* – remainder is dropped





Operators For Each Type

- Binary operators, like +, -, *, /, are really 2-input functions
- Result type of function depends on input types:

```
int + int       ⟶ int
float + float   ⟶ float
double + double ⟶ double
decimal + decimal ⟶ decimal
```

- Action taken (function code) also depends on input types:

`int / int` ⟶ integer division ⟶ `int`

`double / double` ⟶ floating-point division ⟶ `double`

Operator Selection

- Easy case: Both operands are the same type: use that operator

```
float floatDiv = 30.0f / 7.0f;
```

`float / float` operator, produces 4.285714

```
double doubleDiv = 30.0 / 7.0;
```

`double / double` operator, produces 4.2857142857142857

- What if the operands are different types?

`int / int` or `double / double`?

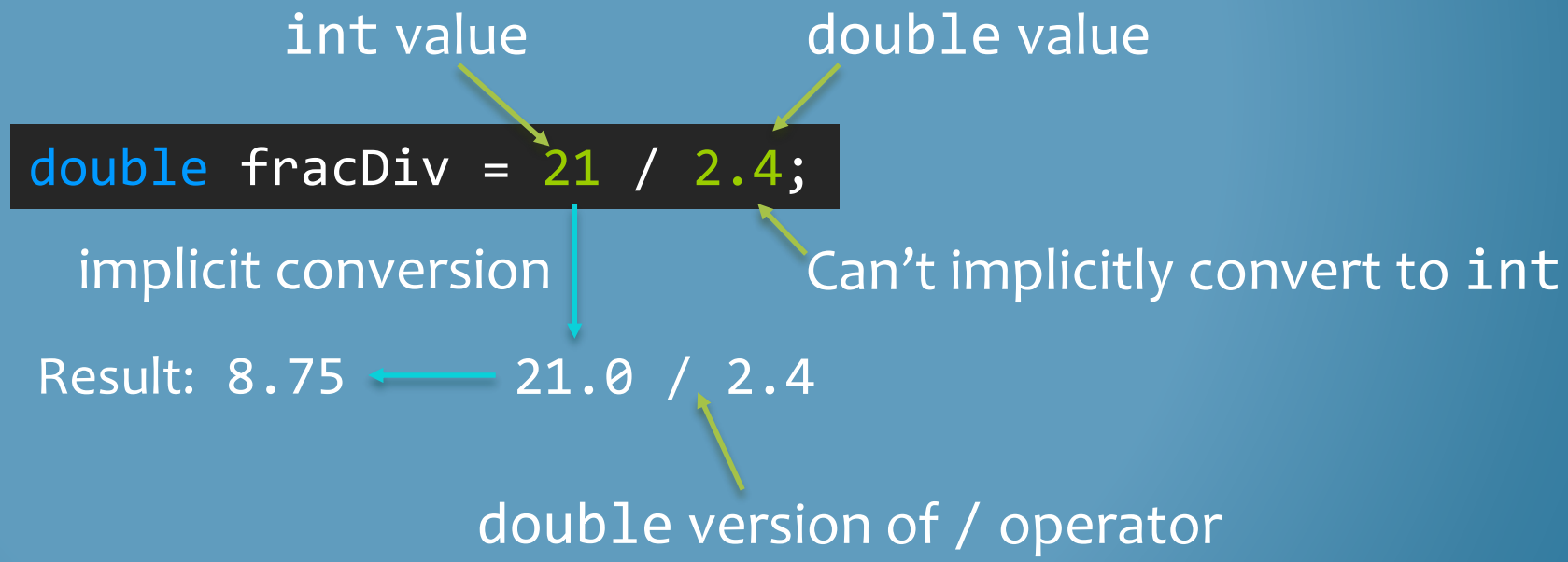
```
double fracDiv = 21 / 2.4;
```

`int` value

`double` value

Implicit Conversions in Math

- If operands are not the same type, C# will try implicit conversion
- Less-precise operand gets converted to more-precise type



Explicit Conversions in Math

- If implicit conversion fails, you will get a compile error

Can't implicitly convert to `double`

Can't implicitly convert to `decimal`

```
double badMath = 3.75m * 2.66;
```

Error! Cannot apply operator * to `double` and `decimal`

- Use **explicit** conversion (casting) to make one type match the other

`double * double` operator

```
double goodMath = (double) 3.75m * 2.66;
```

Convert to `3.75`

Casting to Prompt Conversion

- What if you don't want integer division?

```
int numCookies = 21;
int numPeople = 6;
double share = numCookies / numPeople;
```

int version of / operator

result: 3

- Make one argument a double using casting:

becomes a double value

implicitly converted to double

```
double share = (double) numCookies / numPeople;
```

result: 3.5

double version of / operator

Casting in Math Expressions

- Casting may be **necessary** to make types match:

```
double a = 35.0;
decimal b = 0.5m;
decimal result = (decimal) a * b;
```

decimal * decimal operator

double can't implicitly convert to decimal

- Casting may be **desirable** to change which operator runs:

```
int numCookies = 21;
int numPeople = 6;
double share = (double) numCookies / numPeople;
```

implicitly converted to double

share: 3.5

explicitly convert from int to double

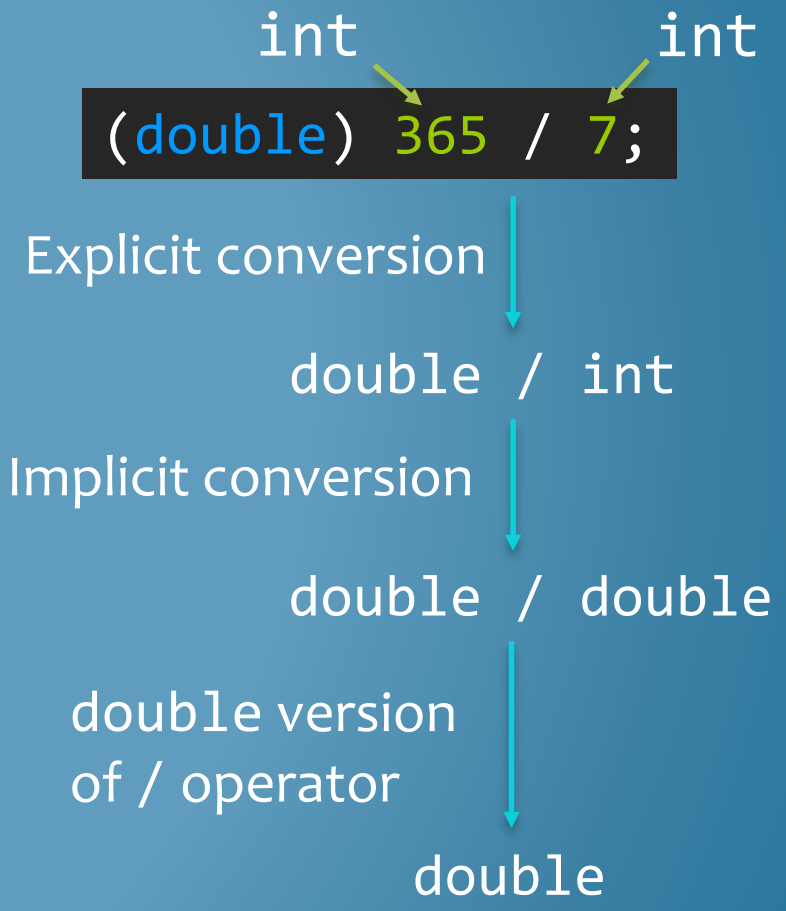
double / double operator

Outline

- Operators and Datatypes
- Implicit and Explicit Conversions with Operators
- **Expressions and Result Types**
- Order of Operations

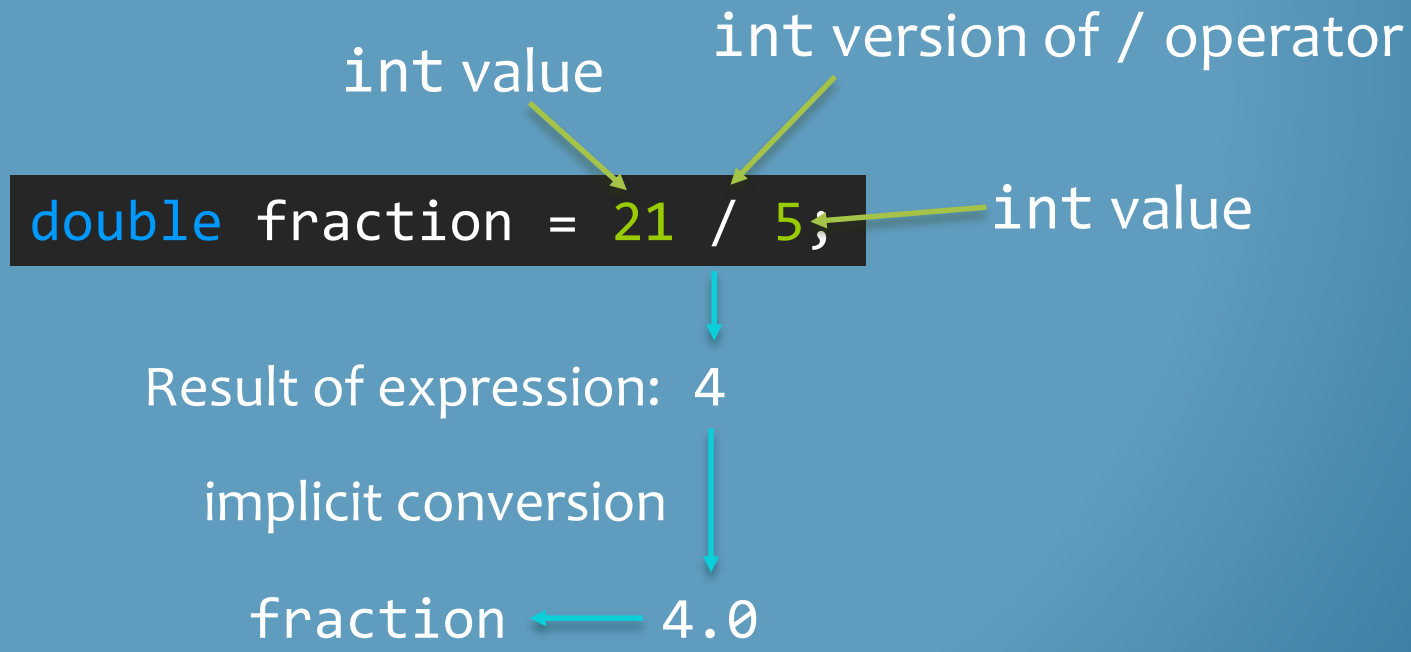
Determining Result Type

- Each expression in C# has a datatype: the type of its result
- Defined by which operator version is used (e.g. `int * int` → `int`)
- To determine expression type:
  - Do explicit conversions on operands
  - Do implicit conversions on operands
  - Pick matching operator version



Implicitly Converting the Result

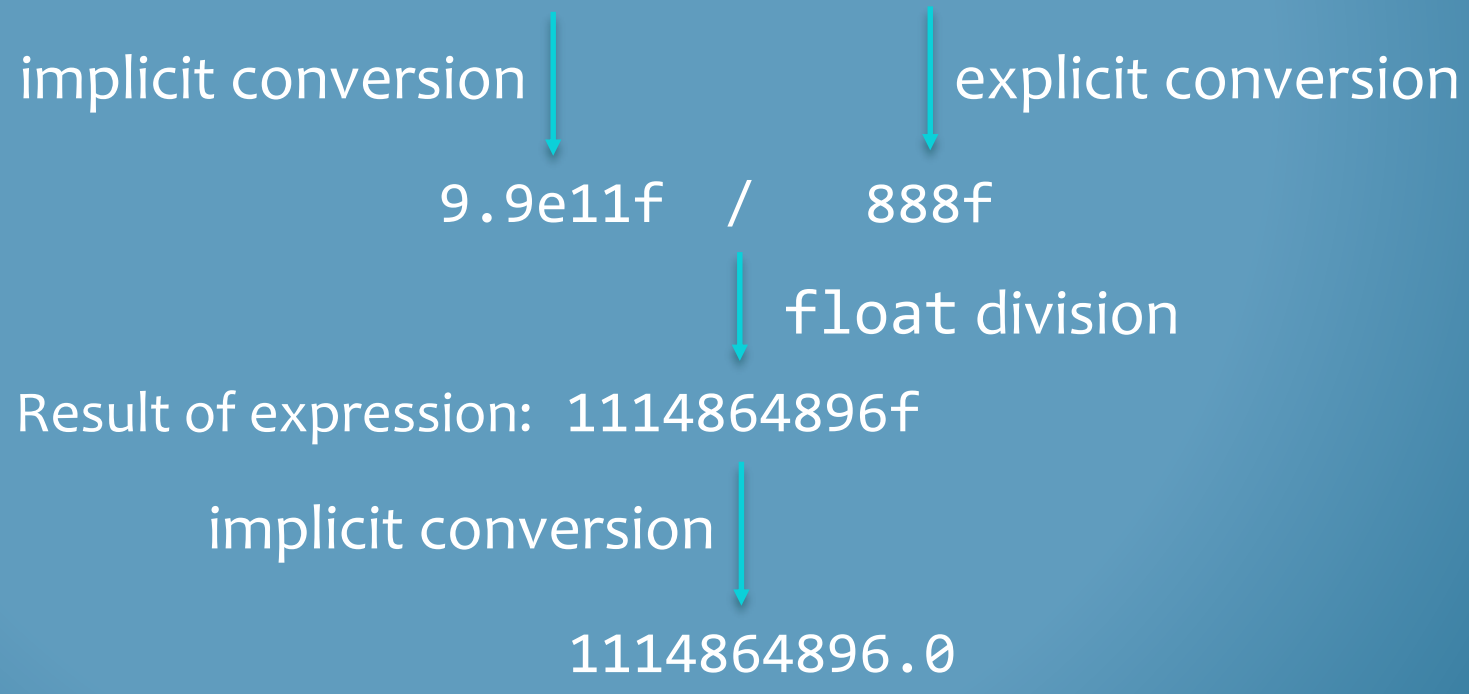
- Expressions can only be assigned to variables of matching type
- Implicit conversion will be attempted if type does not match
  - This happens *after* computing the result



Result Type and Implicit Conversion

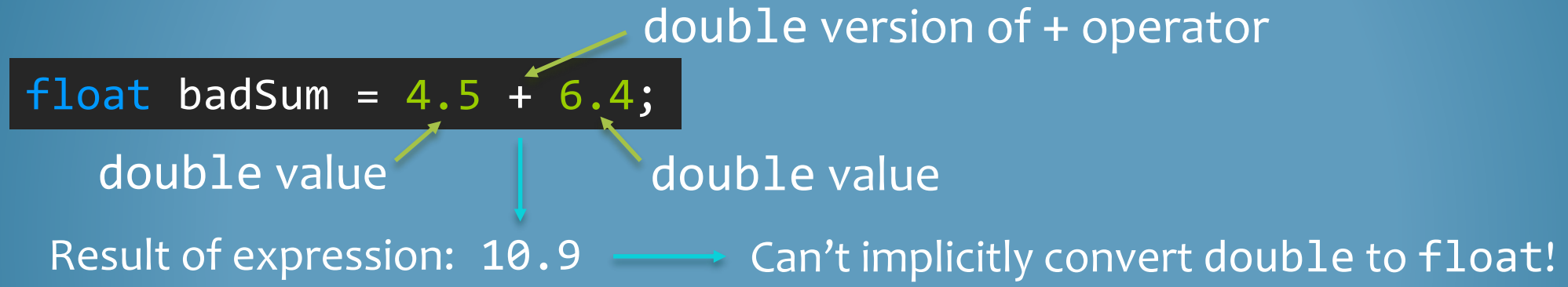
- A more complex example:

```
long bigNumber = 99000000000;
double result = bigNumber / (float) 888;
```

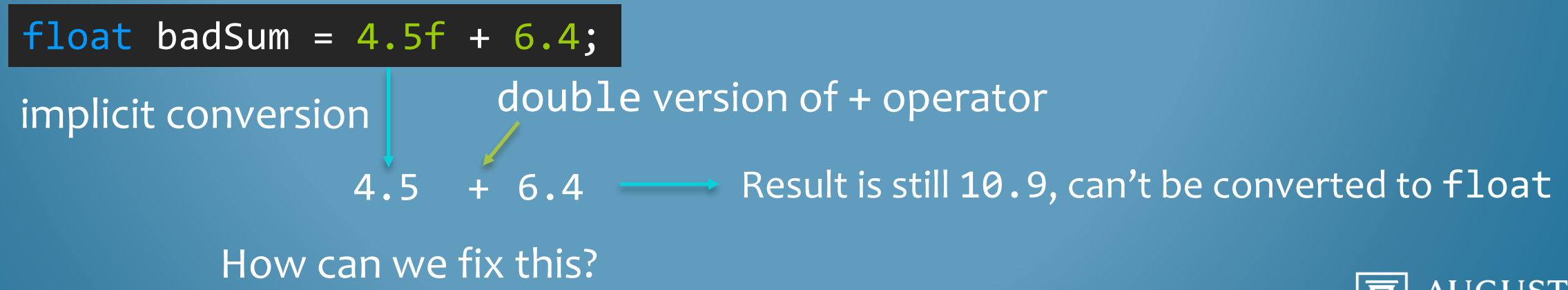


Assignment Can Still Fail

- Result type of expression can't always be implicitly converted



- Implicit conversion within expression happens first



How can we fix this?

Outline

- Operators and Datatypes
- Implicit and Explicit Conversions with Operators
- Expressions and Result Types
- **Order of Operations**

Order of Operations

- C# math operators follow standard PEMDAS order, left-to-right

```
int x = 4 + 10 * 3 - 21 / 2 - (3 + 3);
```

4 + 30 - 10 - 6 → 18

- Cast operator is **higher priority** than binary (math) operators

```
double share = (double) numCookies / numPeople;
```

1. Cast `int` to `double`
2. Implicitly convert to `double` so operands match types
3. Execute `/` operator

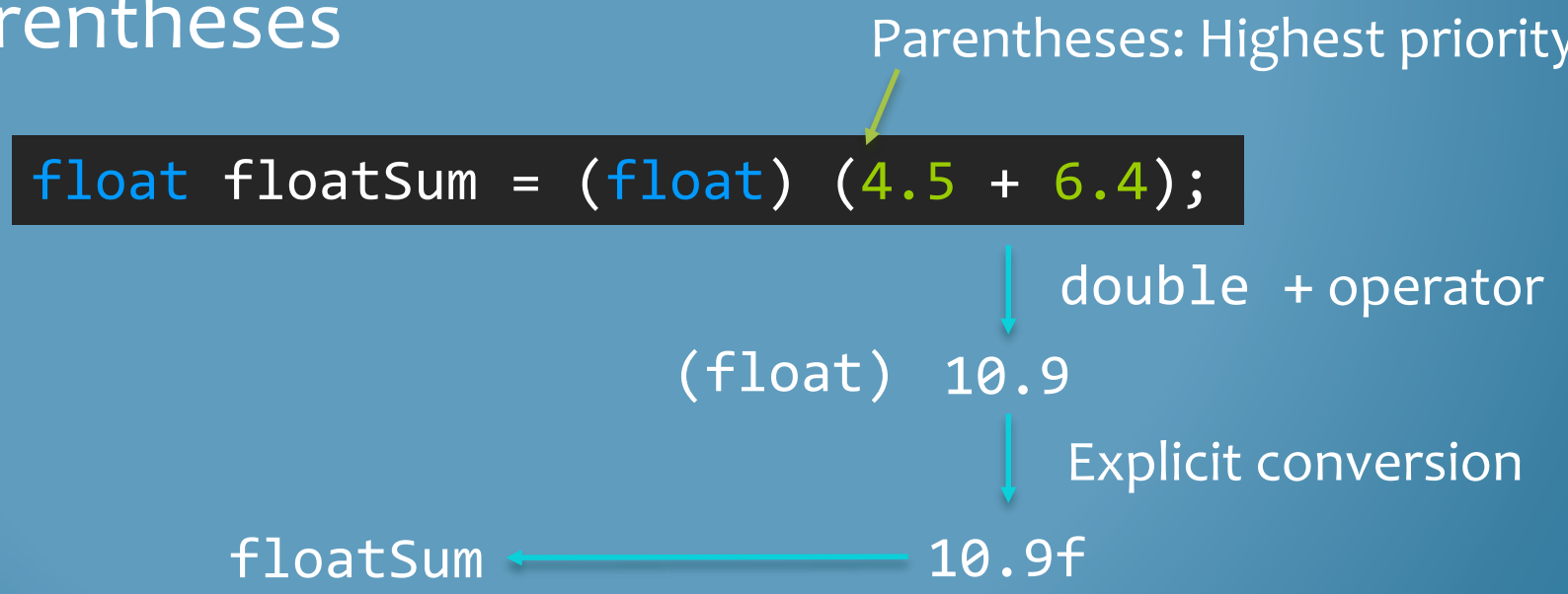
Casting an Expression

- What if we want to cast the result of an expression?

This does not work:

```
float floatSum = (float) 4.5 + 6.4;
```

- Answer: Use parentheses



Conversions in Arithmetic

- What will each of these results be? What is the difference?

```
int a = 5, b = 4;
double result;
result = a / b;
result = (double) a / b;
result = a / (double) b;
result = (double) a / (double) b;
result = (double) (a / b);
```

Summary: Math in C#

Process for evaluating a C# math expression:

1. Determine types of operands, after applying casts. Do they match?
2. If types do not match, can one be implicitly converted to the other? If not, error!
3. Determine which operator type will be used based on operand types. This determines the result type (`int / int` → `int`)
4. Compute result, and apply any "outside" casts
5. If result is assigned to a variable, does its type match the variable?
6. If result type does not match variable type, can it be implicitly converted to the variable's type? If not, error!