

Arithmetic Expressions

- Operations on numerical types
 - Operations:
 - + “addition”
 - “subtraction”
 - * “multiplication”
 - / “division” (different for **int** vs. **double**)
 - % “remainder”
 - Precedence (in order):
 - () highest
 - *, /, %
 - +, - lowest
- Operators in same precedence category evaluated left to right

Type Casting

- Treat one type as another for one operation

```
int x = 3;
```

```
double y;
```

```
y = x / 2;           // y = 1.0
```

```
y = (double)x / 2; // y = 1.5
```

```
y = 5.9;
```

```
x = (int)y;         // x = 5
```

```
x = 7;
```

```
y = x;             // fine: y = 7.0
```

```
x = y;             // error
```

Expression Short-hands

```
int x = 3;
```

```
x = x + 1;
```

```
x += 1;
```

```
x++;
```

```
x = x + 5;
```

```
x += 5;
```

```
x = x - 1;
```

```
x -= 1;
```

```
x--;
```

```
x = x * 3;
```

```
x *= 3;
```

```
x = x / 2;
```

```
x /= 2;
```

Boolean Expressions

- Boolean expression is just a *test* for a condition
 - Essentially, evaluates to **true** or **false**
- Value comparisons:

==	“equals”	(note: not single =)
!=	“not equals”	(cannot say <>)
>	“greater than”	
<	“less than”	
>=	“greater than or equal to”	
<=	“less than or equal to”	

More Boolean Expressions

- Boolean comparisons (in order of precedence):

! “not”

!p if **p** is true, then **!p** is false, and vice versa

&& “and”

p && q only true if **p** and **q** are both true

|| “or”

p || q true if **p** or **q** (or both) are true

```
boolean p = (x != 1) || (x != 2);
```

p is always **true**, you really want:

```
boolean p = (x != 1) && (x != 2);
```

Short Circuit Evaluation

- Stop evaluating boolean expression as soon as we know the answer
- Consider:

```
p = (5 > 3) || (4 <= 2);
```

The test(4 <= 2) is not performed!

- Example of useful case:

```
p = (x != 0) && ((y / x) == 0);
```

Avoid division by 0, since ((y / x) == 0) is not performed