05 Numbers and Integer Precision

May 12, 2022

1 Numbers and Integer Precision

Like you learned in the last module, when doing math with integers, you only get integers as a result, no decimals or fractions. The numbers are `truncated`, which just means that the remainder is cut off. You can find the remainder with `%`, the remainder operator. The remainder is the left over amount from a division problem.

[1]:
```
int a = 7;
int b = 4;
int c = 3;
int d = (a + b) / c;
int e = (a + b) % c;
Console.WriteLine($"quotient: {d}");
Console.WriteLine($"remainder: {e}");
```

```
quotient: 3
remainder: 2
```

What is this saying? Well when you take 11 and divide it by 3, there are 3 3s that fit into 11, with two leftover, or remaining. That's why 3 is the quotient, and 2 is the remainder

1.1 Minimum and Maximum Integer Size

Because of how integers are structured in coding, there is a limit to their size.

[4]:
```
int max = int.MaxValue;
int min = int.MinValue;
Console.WriteLine($"The range of integers is {min} to {max}");
```

```
The range of integers is -2147483648 to 2147483647
```

That's still a pretty big range! > But what happens if you try to go beyond?

[3]:
```
int what = max + 3;
Console.WriteLine($"An example of overflow: {what}");
```

```
An example of overflow: -2147483646
```

That number, which should be really big, is now close to the minimum! This is because an `overflow` "wraps," going back to the minimum and then continuing to count.

1.2 Doubles: Precision and Size

Doubles are another form of numbers. They can hold and answer in floating point. > Repeat the same code from the beginning, and see the difference a double makes.

[6]:
```
double a = 7;
double b = 4;
double c = 3;
double d = (a + b) / c;
Console.WriteLine(d);
```

```
3.6666666666666665
```

[7]:
```
double a = 19;
double b = 23;
double c = 8;
double d = (a + b) / c;
Console.WriteLine(d);
```

```
5.25
```

Find out the range of doubles:

[8]:
```
double max = double.MaxValue;
double min = double.MinValue;
Console.WriteLine($"The range of double is {min} to {max}");
```

```
The range of double is -1.7976931348623157E+308 to 1.7976931348623157E+308
```

That's pretty big! Much larger than integers.

Of course, doubles aren't perfect. They also have rounding errors. > Check out this rounding:

[9]:
```
double third = 1.0 / 3.0;
Console.WriteLine(third);
```

```
0.3333333333333333
```

Technically, 1/3 converted to decimal should be 3 repeating infinitely, but that isn't practical in coding. It's good to be aware of though, if you're working in extremely precise variables.

[ ]:
```

```