Lec 3: Integer Arithmetic

CS220: Programming Principles

Sang Kil Cha

In-Class Activity #02

Preparation

We are going to use the same git repository as before. Just in case you don't have it, clone the repository using the following command.

1. Clone the repository to your machine.
   - `git clone https://github.com/KAIST-CS220/CS220-Main.git`
2. Move in to the directory `CS220-Main/Activities`
   - `cd CS220-Main`
   - `cd Activities`

Exercise

Modify the `max` function. The function should take in four integers as input, and returns as output the largest number among the given numbers.

Attendance Check

Note:

1. This slide appears at random time during the class.
2. This link is only valid for a few minutes.
3. We don't accept late responses.

Integer Types

Various Primitive Types in F#

`int64`, `uint64`, `int32`, `uint32`, `int16`, `uint16`, `uint8`, `int8`.

But, no `int128`! Why?

Suffix for Number Literals

To represent a number literal, we use suffix to represent its type.

Type	Suffix	Example
`int` / `int32`	no suffix / l	`1` / `1l`
`uint32`	u	`1u`
`int64`	L	`1L`
`uint64`	UL	`1UL`
`int16`	s	`1s`
`uint16`	us	`1us`
`int8` / `sbyte`	y	`1y`
`uint8` / `byte`	y	`1uy`

Signed vs. Unsigned

Unsigned integers cannot represent "negative numbers".

```
1u - 42u = ?
```

Integer Overflow

Unit of Computation

In a modern desktop machine, we use a 64-bit CPU, which means that the basic computation unit of it is 64 bits (8 bytes). Most instructions in our CPU can handle only up to 64-bit numbers.

What’s the Implication?

18446744073709551615 + 1 = 0

What's the Implication?

18446744073709551615 + 1 = 0

Why? $2^{64} - 1 = 18446744073709551615$.

Overflow By Examples

```
18446744073709551615UL + 1UL // 0UL
2147483647 + 1 // -2147483648
0u - 1u // 4294967295u
```

Detecting Overflows?

We can always detect overflows by subdividing normal and abnormal cases.

```
let z = x - y // example: subtract operation
if z <= x then // normal
else // abnormal (overflow)
```

Representing Big Numbers?

We can “emulate” big numbers with 64-bit/32-bit operations using data abstraction![1]

```
type UInt128 = uint64 * uint64
```

[1] Using a tuple. You will learn this later.

Big vs. Normal Integers

Big integer (arbitrary-precision) arithmetic is slower, and uses more memory. Therefore, we prefer to use normal integers in most cases.

Why? You need to understand the distinction between register and memory. Take CS230 (System Programming), etc.

Big Numbers

There is a built-in data type `bigint` for representing big numbers. We use ‘I’ as a suffix for big numbers.

```
1I + 2I // 3I
1I - 2I // -1I
999999999999999999999999999999999999999999999999999I // valid
bigint 42 // bigint is a conversion function
bigint 42L = 42I // true
```

Floating Point Types?

Use `float` for double-precision floating-point numbers, and `float32` for single-precision floating-point numbers.

Quiz #1

- The problem is publicly available at `https://github.com/KAIST-CS220/Quiz1`.
- This will be auto-graded (unlike the previous in-class activities).
- You can even see all the tests: `https://github.com/KAIST-CS220/Quiz1/blob/main/Tests/Tests.fs`.
- The main purpose of this quiz is to get used to the GitHub classroom environment, which will be used for future assignments.
- First, you should accept the assignment invitation.
- Then you wait for a minute or two until your own private repository is created.
- Finally, you can clone your own repository and start working on the quiz.

Conclusion

Further Readings

- https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html

Question?