Type safety

Ryuichi OHORI

Graduate School of Mathematical Sciences, The University of Tokyo

Keisan Sugaku I

Using functions in mathematics

- If you write $f(t, x)$ for $f(x, t)$, it's confusing.
- However, the reader has a chance to correct you.

Using functions in programming

- Many programming languages support multivariate functions.
- If you code $f(t, x)$ for $f(x, t)$, compilers interpret it as-is.

Example

```
auto _f(int precision, int dimension)
{...}

void main(){
   int dimension = 4;
   int precision = 32;
   _f(dimension, precision);
   // a bug that compiles!
}
```

Solution: Use different types

Though you code $f(t, x)$ for $f(x, t)$, it doesn't compile if x and t has different types.

Example

```
struct Prec{immutable int p;}
struct Dim{immutable int d;}
auto f(Prec p, Dim d){
  return _f(p.p, d.d);
}
void main(){
  f(Prec(32), Dim(4)); // works
  // f(Dim(4), Prec(32));
  // doesn't compile
}
```

Problem

- It's tiring, boring and bug-prone to write a lot of conventional codes.
- Use some metaprogramming features (e.g., templates and mixins in D) or library implementations (e.g. std.typecons.Typedef in D).

Remarks

- Use concrete names for types, functions, variables and arguments.
- Interpreted language (e.g., Python) usually supports named parameters.