

6 Algorithmic Journeys with Concepts

Taras Shevchenko

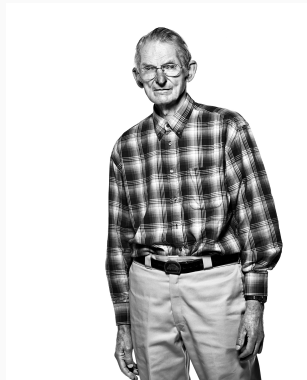
Rails Reactor / Giphy

Table of contents

1. Programming with concepts
2. Conclusion

The Software Industry is Not Industrialized

Software components (routines), to be widely applicable to different machines and users, should be available in families arranged according to precision, robustness, generality and time-space performance.



A Familiar Example. Douglas McIlroy about sin

Dimensions along which we wish to have variability:

1. precision, for which perhaps ten different approximating functions might suffice
2. floating vs fixed computation
3. argument ranges $0 - \pi/2$, $0 - 2\pi$, also $-\pi/2 - \pi/2$, $-\pi - \pi$, $-big - +big$
4. robustness - ranging from no argument validation through signaling of complete loss of significance, to signaling of specified range violations

Periodic Table

ПЕРИОДИЧЕСКАЯ СИСТЕМА ЭЛЕМЕНТОВ Д. И. МЕНДЕЛЕЕВА

1	H																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

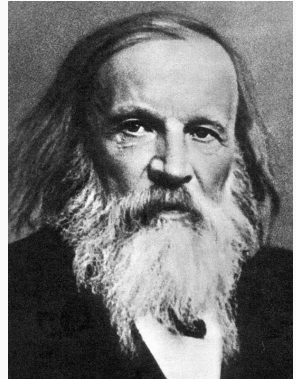


Figure 1: A Russian periodic table based on Dmitri Mendeleev's original table of 1869.

Species Plantarum

Lists every species of plant known at the time, classified into genera. It is the first work to consistently apply binomial names and was the starting point for the naming of plants.



Elements

1. Definitions
2. Postulates
3. Common notions



Common Notions

1. Things which are equal to the same thing are also equal to one other.
2. If equals be added to equals, the wholes are equal.
3. If equals be subtracted from equals, the remainders are equal.
4. Things which coincide with one another are equal to one another.
5. The whole is greater than the part.

Basic idea

The essence of generic programming lies in the idea of concepts. A concept is a way of describing a family of related object types.

Natural Science	Mathematics	Programming	Programming Examples
genus	theory	concept	Integral, Character
species	model	type or class	uint8_t, char
individual	element	instance	01000001(65, 'A')

1. Datum
2. Value
3. Value type
4. Object
5. Object type

Definition

A **datum** is a sequence of bits.

Example

01000001 is an example of a datum.

Definition

A **value** is a **datum** together with its interpretation.

Example

The **datum** 01000001 might have the interpretation of the integer 65, or the character “A”.

Explanation

Every **value** must be associated with a **datum** in memory; there is no way to refer to disembodied **values** in modern programming languages.

Definition

A **value type** is a set of values sharing a common interpretation.

Definition

An **object** is a collection of bits in memory that contain a **value** of a given **value type**.

Explanation

An **object** is immutable if the value never changes, and mutable otherwise. An object is unrestricted if it can contain any **value** of its **value type**.

Definition

An **object type** is a uniform method of storing and retrieving **values** of a given **value type** from a particular **object** when given its address.

Programming with concepts

Operation

1. Copy construction
2. Assignment
3. Equality
4. Destruction

Semantic

$$\forall a \forall b \forall c : T \ a(b) \implies (b = c \implies a = c)$$

$$\forall a \forall b \forall c : a \leftarrow b \implies (b = c \implies a = c)$$

$$\forall f \in \text{RegularFunction} : a = b \implies f(a) = f(b)$$

Operation

1. Copy construction
2. Assignment
3. Destruction

Semantic

$$\forall a \forall b \forall c : T \ a(b) \implies (b = c \implies a = c)$$

$$\forall a \forall b \forall c : a \leftarrow b \implies (b = c \implies a = c)$$

$$\forall f \in \text{RegularFunction} : a = b \implies f(a) = f(b)$$

FunctionalProcedure(F) \triangleq F is a regular procedure defined on regular types : replacing its inputs with equal objects results in equal output objects.

$$\begin{aligned} \text{UnaryFunction}(F) &\triangleq \text{FunctionalProcedure}(F) \wedge \text{Arity}(F) = 1 \\ &\wedge \text{Domain} : \text{UnaryFunction} \rightarrow \text{Regular} \\ &F \mapsto \text{InputType}(F, 0) \end{aligned}$$

$$\begin{aligned} \text{HomogeneousFunction}(F) &\triangleq \text{FunctionalProcedure}(F) \wedge \text{Arity}(F) > 0 \\ &\wedge (\forall i, j \in \mathbb{N})(i, j < \text{Arity}(F)) \implies (\text{InputType}(F, i) = \text{InputType}(F, j)) \\ &\wedge \text{Domain} : \text{HomogeneousFunction} \rightarrow \text{Regular} \\ &F \implies \text{InputType}(F, 0) \end{aligned}$$

$$\text{Predicate}(P) \triangleq \text{FunctionalProcedure}(F) \wedge \text{Codomain}(P) = \text{bool}$$

$$\text{HomogeneousPredicate}(P) \triangleq \text{Predicate}(P) \wedge \text{HomogeneousFunction}(P)$$

$$\text{Relation}(R) \triangleq \text{HomogeneousPredicate}(R) \wedge \text{Arity}(R) = 2$$

$$\text{TotallyOrdered}(T) \triangleq \text{Regular}(T) \wedge <: T \times T \rightarrow \text{bool} \wedge \text{total_ordering}(<)$$

property(*R* : *Relation*)

total_ordering : *R*

$r \mapsto \text{transitive}(r) \wedge (\forall a, b \in \text{Domain}(R))$ exactly one of following holds :

$r(a, b)$, $r(b, a)$, or $a = b$

Journey 1

1. min
2. max

```
int min(int x, int y) {  
    if (y < x) {  
        return y;  
    }  
    return x;  
}
```

```
int min(int x, int y) {  
    if (y < x) {  
        return y;  
    }  
    return x;  
}
```

```
double min(double x, double y) {  
    if (y < x) {  
        return y;  
    }  
    return x;  
}
```



```
template<typename T>
T min(T x, T y) {
    if (y < x) {
        return y;
    }
    return x;
}
```

Dealing with large objects

```
template<typename T>
const T& min(const T& x, const T& y) {
    if (y < x) {
        return y;
    }
    return x;
}
```

```
template<typename T, typename P>
const T& min(const T& x, const T& y, P pred) {
    if (pred(y, x)) {
        return y;
    }
    return x;
}
```

Journey #1

```
struct employee {  
    std::string full_name;  
    int64_t salary;  
};  
  
void usage() {  
    employee e0{"Bjarne Stroustrup", 9999999ll};  
    employee e1{"Alex Stepanov", 9999999ll};  
    min(e0, e1, [](const auto& x, const auto& y) {  
        return x.salary < y.salary;  
    });  
}
```

```
template<typename T, typename P>
T& min(T& x, T& y, P pred) {
    if (pred(y, x)) {
        return y;
    }
    return x;
}
```

```
template<Regular T, Relation r>
const T& min(const T& x, const T& y, Relation r) {
    if (r(y, x)) { return y; }
    return x;
}
```

```
template<Regular T, Relation r>
T& min(T& x, T& y, Relation r) {
    if (r(y, x)) { return y; }
    return x;
}
```

```
template<TotallyOrdered T>
const T& min(const T& x, const T& y) {
    return min(x, y, std::less<T>());
}
```

```
template<TotallyOrdered T>
T& min(T& x, T& y) {
    return min(x, y, std::less<T>());
}
```


1. unique
2. unique_count

1. frequencies
2. transform_subgroups
3. squash_subgroups

1. split
2. transform_splits

1. `remove_if`
2. `partition_semistable`

More examples of concepts

1. Regular Type
2. Semiregular Type
3. Functional Procedure
4. Homogeneous Function
5. Homogeneous Predicate
6. Semiring
7. Sequence
8. Totally Ordered
9. Input Iterator
10. Forward Iterator
11. Bidirectional Iterator

Properties

1. Associative
2. Distributive
3. Transitive
4. Semiregular Type
5. Functional Procedure

1. Transformation-action duality
2. Operation-accumulation procedure duality
3. Memory adaptivity
4. Reduction to constrained subproblem

Conclusion

Conclusion

1. Concreteness costs
2. Abstracting algorithms to their most general setting without losing efficiency
3. Know your algorithms