CONTRACT ORIENTED PROGRAMMING IN C++

AKA DESIGN BY CONTRACT
BY LOUIS “LUIGI” LANGHOLTZ 9/2023

OVERVIEW

1. Speaker
2. Meetup Group
3. Talk
    - Giving Back
    - Opinion
    - Experience
    - Passion

SOME CONTEXT

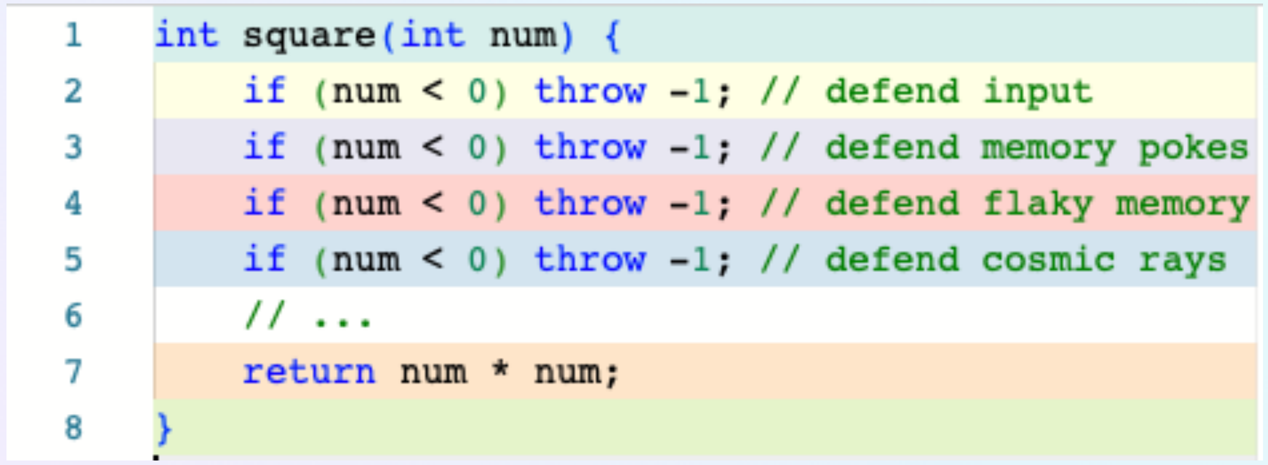
DESIGN & DEVELOPMENT: *ORIENTATIONS* FOR DELIVERING CONCEPTS & SOLUTIONS

- Design by contract (DBC).
- Object oriented design (OOD).
- Domain driven design (DDD).
- Data oriented design (DOD).
- Test driven development (TDD).
- Defensive/offensive programming (defensive design).

FROM DEFENSIVE PROGRAMMING

DEFENSIVE *OBSURDIUM*

- How to test?
- Who’s correct?
- Seen!

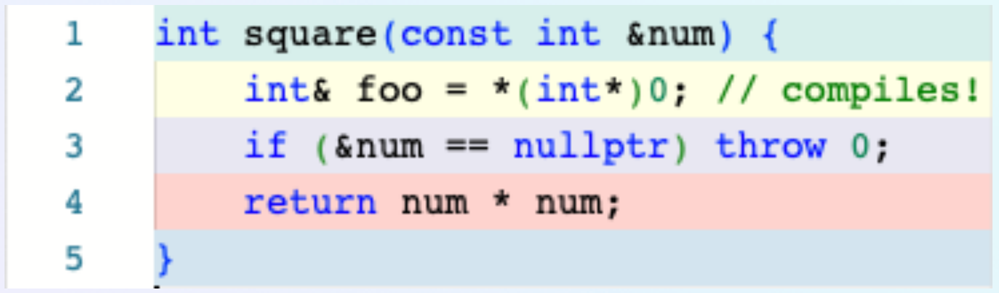


```
int square(int num) {
    if (num < 0) throw -1; // defend input
    if (num < 0) throw -1; // defend memory pokes
    if (num < 0) throw -1; // defend flaky memory
    if (num < 0) throw -1; // defend cosmic rays
    // ...
    return num * num;
}
```

ABSENT MAJORITY, DEFENSIVE PROGRAMMING FOR BYZANTINE FAILURES LOWERS FAULT TOLERANCE

DEFENSIVE *OBSURDIUM*

- How to test? Compiler may elide the if-statement.
- What does this test?
- Who's correct?



```
int square(const int &num) {
    int& foo = *(int*)0; // compiles!
    if (&num == nullptr) throw 0;
    return num * num;
}
```

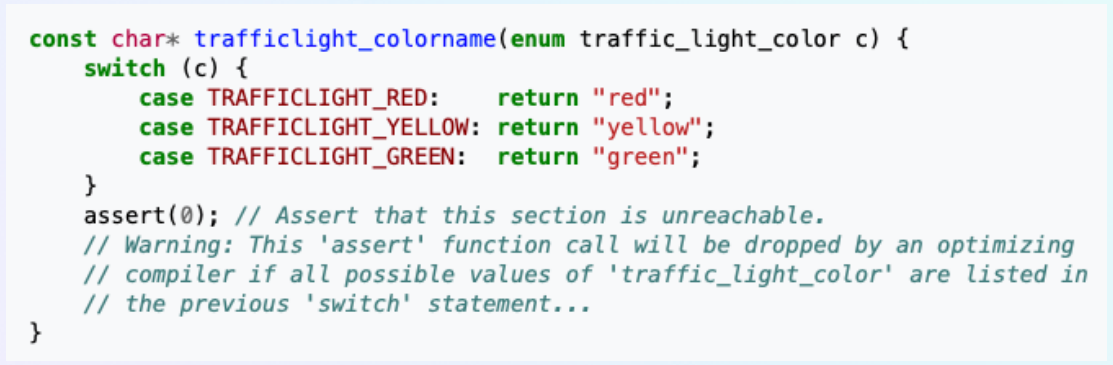
INSANITY IS DOING THE SAME THING OVER AND OVER AGAIN AND EXPECTING DIFFERENT RESULTS

TO OFFENSIVE PROGRAMMING

DEFENSIVE WITHIN REASON

TRUSTING INTERNAL DATA VALIDITY

- “Only errors from outside the program’s control are to be handled”.
- “Data from within the program’s line of defense, are to be trusted”.



```c
const char* trafficlight_colorname(enum traffic_light_color c) {
    switch (c) {
        case TRAFFICLIGHT_RED:    return "red";
        case TRAFFICLIGHT_YELLOW: return "yellow";
        case TRAFFICLIGHT_GREEN:  return "green";
    }
    assert(0); // Assert that this section is unreachable.
    // Warning: This 'assert' function call will be dropped by an optimizing
    // compiler if all possible values of 'traffic_light_color' are listed in
    // the previous 'switch' statement...
}
```

FROM WIKIPEDIA OFFENSIVE PROGRAMMING SECTION OF DEFENSIVE PROGRAMMING PAGE

See also:

- Defensive programming - Friend or Foe?

CONTRACTS

GENERALLY SPEAKING & IN TERMS OF PROGRAMMING…

- Exchange promises between author & user.
- Contracts are to values, what concepts are to types.

DBC like offensive programming, but…

(More) for design, explicitly recognizing:

- <u>Preconditions</u>
- <u>Postconditions</u>
- <u>Invariants</u>

PRECONDITION

“condition or predicate that must always be true just prior to the execution of some section of code or before an operation” - Wikipedia.

BEST AGAINST PROGRAMMER ERRORS THAT OTHERWISE PRODUCE UNDEFINED BEHAVIOR

POSTCONDITION

“condition or predicate that must always be true just after the execution of some section of code or after an operation” - Wikipedia.

POSTCONDITION

SOME EXAMPLES

- Observable state changes.
- An exception safety guarantee: noexcept, strong, basic, none.

INVARIANT

“logical assertion that is always held to be true” - Wikipedia.

INVARIANT

SOME PERSPECTIVES

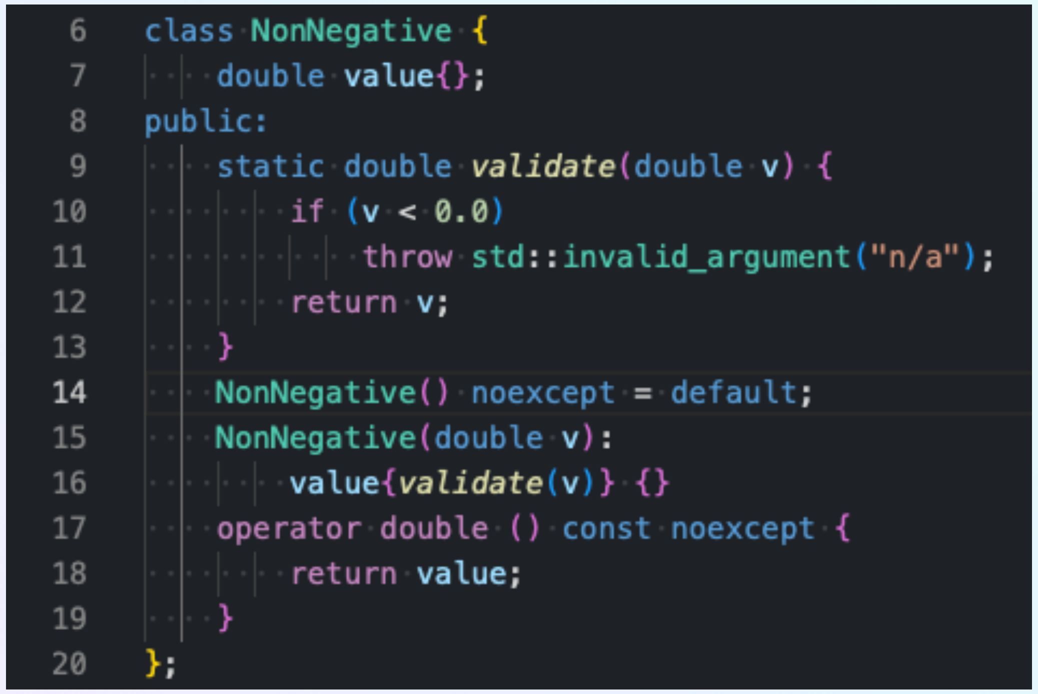
- **Loop.**
- **Function.**
- ***Class.***
- ***Responsibility - SOLID.***
- ***An idea about what’s intended.***

VIOLATION

- Programming issue.
- Not runtime error.
- Fail fast or behavior not specified.

EXAMPLE

- What are the preconditions?
- What are the postconditions?
- What are the invariants?
- Are they enforced?



```cpp
class NonNegative {
    double value{};
public:
    static double validate(double v) {
        if (v < 0.0)
            throw std::invalid_argument("n/a");
        return v;
    }
    NonNegative() noexcept = default;
    NonNegative(double v):
        value{validate(v)} {}
    operator double () const noexcept {
        return value;
    }
};
```

WHY DBC?

INSPIRING DOCUMENTATION?!

INSPIRING TESTING?!

IMPROVING ROBUSTNESS?!

IMPROVING CORRECTNESS?!

First Step Solving Problem Is Recognizing It

NOT JUST SOFTWARE?

ASSERTION: UNDER-RECOGNIZED CONDITIONS IN HARDWARE CAUSES ISSUES TOO LIKE...

- **Spectre.**
- **Meltdown.**
- **Rowhammer.**
- **Broken protection rings.**

IF AI DOESN'T KILL US, NOT COLONIZING SPACE WILL

SPACE INDUSTRY

STATISTICS FROM EXTREMETECH.COM

- Growing faster than workforce.
- $464 billion in January 2023.
- $1 trillion valuation by 2030.
- Bug == *"rapid unscheduled disassembly"!*

FASTER CODE?!

CONTRACT?

```cpp
#include <stdexcept>

double process(double v) {
    if (v < 0.0) {
        throw std::exception();
    }
    return v * v;
}
```

CONTRACT?

```cpp
#include <cassert>
#include <stdexcept>

double process(double v) noexcept {
    assert(v >= 0);
    return v * v;
}
```

SOME TOOLS

Doxygen

Assert

Throw expression

Unit testing

Class Types

C++ Contracts?

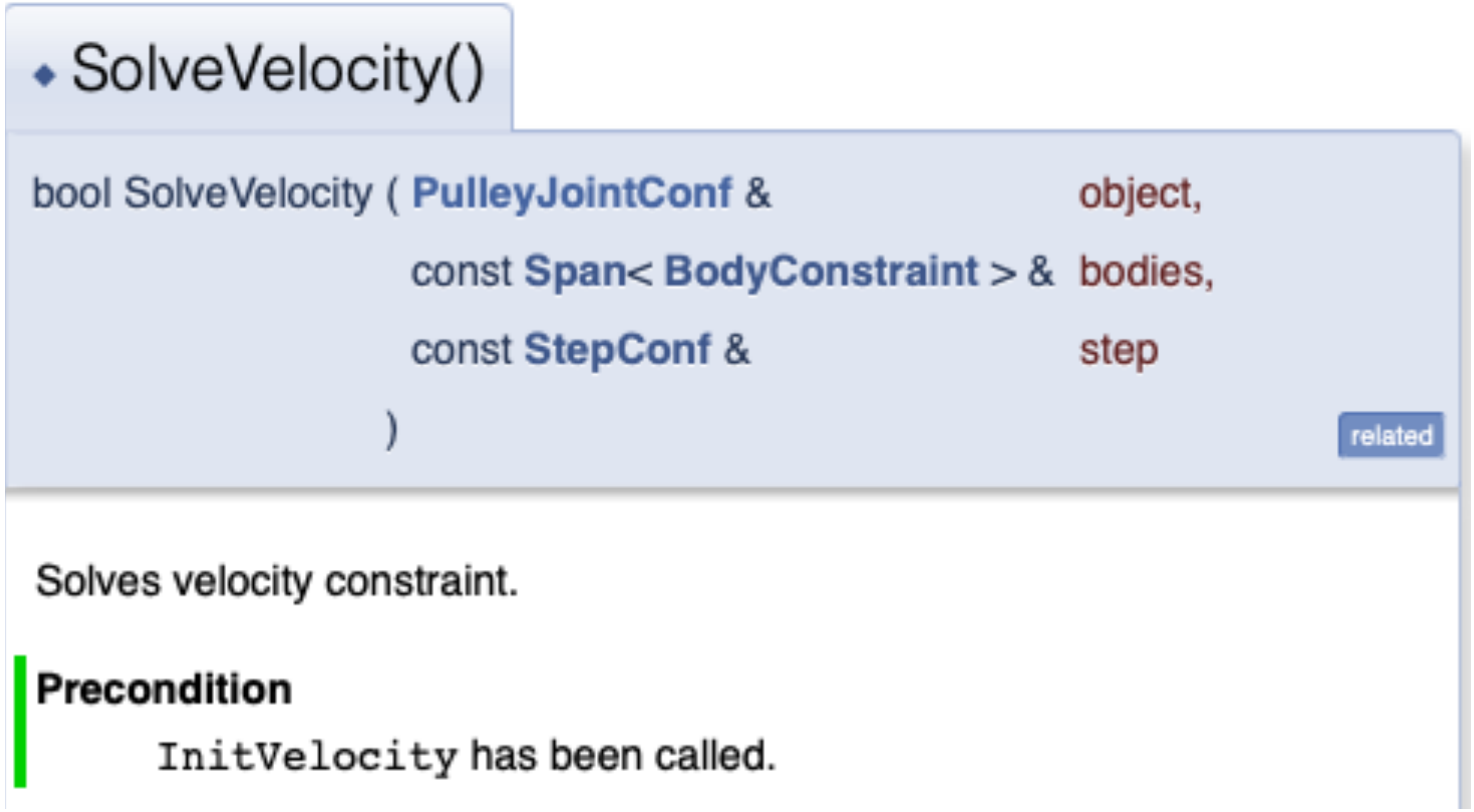
TOOL: DOXYGEN

	Doxygen (via “\” or “@“)	Code

Preconditions	pre	`/// @pre @v is greater-than or equal-to 0.`
		`double sqrt(double v) noexcept;`

Postconditions	post	`/// @post <code>get_handler()</code> returns @p handler given.`
		`void set_handler(handler_type handler);`

Invariants	invariant	`/// @invariant Value always non-negative.`
		`class NonNegative {`



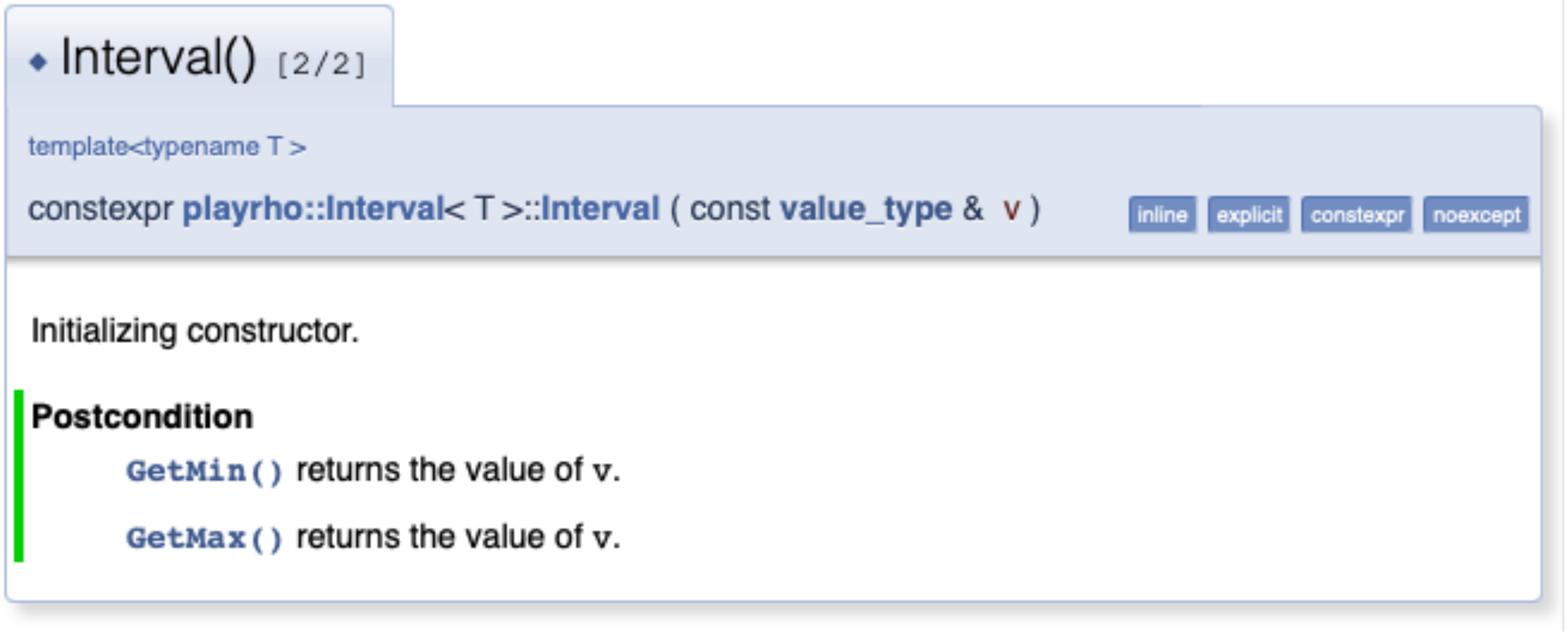
◆ SolveVelocity()

```
bool SolveVelocity ( PulleyJointConf &                 object,
                     const Span< BodyConstraint > &    bodies,
                     const StepConf &                  step
                   )
```

Solves velocity constraint.

Precondition

`InitVelocity` has been called.


◆ Interval() [2/2]

template<typename T >

```
constexpr playrho::Interval< T >::Interval ( const value_type & v )
```

inline explicit constexpr noexcept

Initializing constructor.

Postcondition

`GetMin()` returns the value of `v`.

`GetMax()` returns the value of `v`.

Detailed Description

A "body" physical entity.

A rigid body entity having associated properties like position, velocity, acceleration, and mass.

Invariant

Only bodies that allow sleeping, can be put to sleep.

Only "speedable" bodies can be awake.

Only "speedable" bodies can have non-zero velocities.

Only "accelerable" bodies can have non-zero accelerations.

Only "accelerable" bodies can have non-zero "under-active" times.

The body's transformation is always the body's sweep position one's linear position and the unit vector of the body's sweep position one's angular position.

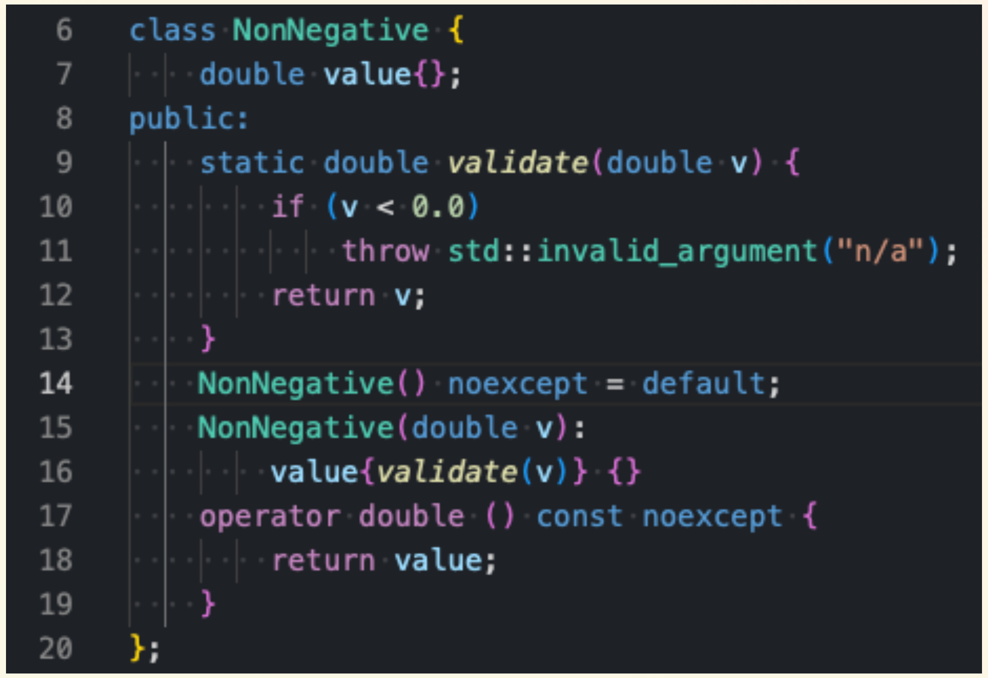
TOOL: CLASS TYPES

BASICS

- Available pre C++-contracts.
- Enforce condition on construction.
- Self documenting.
- Recognize relationships.
- DRY.

EXAMPLE

FROM BEFORE, PERHAPS FOR SQUARE ROOT FUNCTION...



```cpp
class NonNegative {
    double value{};
public:
    static double validate(double v) {
        if (v < 0.0)
            throw std::invalid_argument("n/a");
        return v;
    }
    NonNegative() noexcept = default;
    NonNegative(double v):
        value{validate(v)} {}
    operator double () const noexcept {
        return value;
    }
};
```

ADVANCED

- *Partial functions* become *total functions*?
  - Functions with preconditions are *partial functions*.
- Denotational semantics (domains)?
- Correct **by-design**?!

QUESTIONS?