Inhaltsverzeichnis

[1. Introduction: Project Objective 2](#_Toc97833979)

[2. Fundamentals 2](#_Toc97833980)

[2.1. Theoretical Fundamentals 2](#_Toc97833981)

[2.1.1. Typing 2](#_Toc97833982)

[2.1.2. Structural Subtyping 2](#_Toc97833983)

[2.2. Technical Fundamentals and Decisions 2](#_Toc97833984)

[2.2.1. Language, Framework and Dependencies 2](#_Toc97833985)

[2.2.2. Architectural Decisions 2](#_Toc97833986)

[3. Solution Implementation 2](#_Toc97833987)

[3.1. User Input via “TypeConstructionKitComponent” 2](#_Toc97833988)

[4. Outlook on future work 3](#_Toc97833989)

[4.1. Replayable History for TypingTree and SubtypingGraph 3](#_Toc97833990)

[4.2. Persisting User Input in Localstorage 3](#_Toc97833991)

[4.3. User Input via code editor 3](#_Toc97833992)

# Introduction: Project Objective

* Goal of Typing in general (e.g.: manage memory, select correct assembler instruction, avoid certain run-time errors); The purpose of types (http://www.cse.chalmers.se/edu/year/2011/course/TIN321/lectures/proglang-07.html)
* Goal of Tool: Webtool to enable visually accompanied learning/understanding of Typing and Structural Subtyping

# Fundamentals

## Theoretical Fundamentals

### Typing

### Structural Subtyping

## Technical Fundamentals and Decisions

### Language, Framework and Dependencies

* Angular (Component-based)
* TypeScript (detail about structural subtyping)
* RxJS (Subject)

### Architectural Decisions

* Client only
* High-level UML Class Diagram about most important classes (AbstractType, AstNode, TypeEnvironment)

# Solution Implementation

* Main focus of tool: perform algorithms on arbitrary complex types. Where do they come from?

## User Input via “TypeConstructionKitComponent”

* First approach integrating library ‘cparse’ failed due to incapability of parsing:
  + Nested structs
  + Higher order functions (Functions taking functions as parameters)
  + Recursive type definitions
* Solution:
  + UI providing base types as atomic building blocks
  + Type constructor templates as toolset for consecutive type construction
  + Output:
    - Larger set of types
    - Feature of generating typdefs and declarations on built types (printed in readonly code-editor)
* Implementation:
  + Lifecycle-hooks of AbstractTypeBubble as a framework enabling extension (provide UML Class Diagram!)
  + States CONSTRUCTION and IDLE
  + AliasPlaceholder as workaround for recursive typedefs (provide example use case)
* Further Consequence
  + Computing valid C-Code from an arbitrary Type not trivial, e.g.

typedef char\* (\*(\*x(int\* (\*(()[]))))[5])();

* + Webservice cdecl.org cannot be used due to lack of support for structs; also workaround not possible for cases where it holds: “Some non-struct type is pointing on a struct” (e.g. [complex pointer/array combo] to pointer to struct) 🡪 toEnglish() would contain struct TODO: Really a problem?
  + However, webservice still useful for testing --> Test driven development

## Typing

## Structural Subtyping

# Outlook on future work

## Replayable History for TypingTree and SubtypingGraph

## Persisting User Input in Localstorage

* Implement Serialization/ Deserialization of AbstractType

## User Input via code editor

* Implement or look for suitable C-Code parser
* Implement Adapter mapping that output to