STS
Software
Technology
Systems
TUHH

Compiler Construction
Summer Term 23
April 20
Lübke/Schupp

Frontend

Deadline : 06.07.23 08:00 AM (2023-07-06T08:00:00+02:00)

How to complete a project successfully?

Follow the rules as described in the Lecture!

GitLab at TUHH[1] is used as the submission system in this course. Your GitLab credentials are the same as for StudIP. If you cannot log into GitLab with your StudIP login and password, contact the teaching assistant immediately. All your code must be pushed to your group repository provided to you by the teaching assistant. You can update your solution until the deadline. The latest commit before the deadline is considered by default. If you want your late submission to be graded for penalized maximal number of possible points, the **whole** group must send an email to the teaching assistant **before** the interview day.

How to get additional information?

You are encouraged to discuss past and present project sheets with the teaching assistants. Either approach the teaching assistant during the exercise session, or visit us during the weekly office hours. We are also available through e-mail or on the StudIP forum. We try to reply as quickly as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

[1] https://collaborating.tuhh.de/

Frontend for . . .

In the compiler construction course you are learning how to build a grammar, parser, type checker, and code generator (in summary: a compiler) by the example of a subset of C/C++. For the final project, it is time apply this knowledge to build a frontend for a language of your choice (from a selection made by us). In other words, you're repeating the first two project phases (grammar/parser, type checker), but for another language.

This task sheet is published early, alongside the first project phase, mainly for two reasons:

- There will be three exercise sessions dedicated to this task throughout the semester that require a little preparation (see below).
- You can start the thought process for the final task while you're working on the counterparts for the C/C++ subset.

You can choose from the following languages; most of them are general-purpose languages, but some are designed to do one specific thing safer, easier, faster, or in another way "better" than other languages.

- **Nix**[2] The only non-general-purpose, but also the only pure functional language on this list. It has been created specifically for use with the Nix package manager[3], used in NixOS[4]. The language is leveraged to describe reproducible package builds in a *declarative* way. The built packages are stored in an *immutable* package store in a specific way, "essentially eliminating dependency hell"[3]. In consequence, a NixOS installation can be rolled back to a previous state at any time, and updates can never break the system (there are no inconsistent states during or in between updates).
- **Rust**[5]: A language designed to, among other things, overcome the shortcomings of C/C++, especially in regard to the use of pointers. It includes a so-called *borrow checker* that can guarantee validity of all memory accesses, thus eliminating the must prominent security and safety issues with C/C++. It is used in all kinds of different areas, from embedded to web development (by compiling to WebAssembly[6]).
- **Go**[7]: A language designed especially for modern-day concurrent programming on networked multicore systems. This is enabled by so-called *goroutines* and *channels* through which those can communicate. It is

[2] https://nixos.wiki/wiki/Overview_of_the_Nix_Language
[3] https://nixos.wiki/wiki/Nix_package_manager
[4] https://nixos.org/
[5] https://www.rust-lang.org/
[6] https://webassembly.org/
[7] https://go.dev/

often used for backends in (micro) web services but can also be compiled to WebAssembly.

- **Typescript**[8]: A language designed to add a static type system to Javascript, thus enabling more thorough static analysis and largely preventing runtime type errors. It is compiled to Javascript and is mostly used for frontend web development, but can also be used for server-side applications and, of course, scripting.
- **Kotlin**[9]: A language designed to overcome certain shortcomings of Java and Scala that runs in the Java Virtual Machine (JVM). It incorporates many modern language features from different programming paradigms but still has comparatively short compilation times. It has superseded Java as the recommended programming language for Android Apps, but iOS development is possible as well as virtually everything else.
- **Java**[10]: The oldest language in this list that forms the base for the many other languages that run in the JVM. Its main programming paradigm is object oriented programming; all code is encapsulated in *classes* (yet there are basic, non-object data types like `int`). It is also one of the most widely-spread languages.

Tasks

Task 1

a) Read this task sheet entirely.

b) Choose a programming language from the list above. At most 3 groups may work on the same language, so send your choice and at least one alternative to ole.luebke@tuhh.de by **April 26, 8 AM**.

c) Define a subset of your chosen language that fulfills the requirements of the other tasks. Briefly present your choice in the exercise session on **June 15** to get approval. To support your explanations with examples, please include

- 10 legal test programs.
- 10 syntactically illegal test programs, highlighting syntactic problems you plan to catch.
- 5 semantically illegal test programs, highlighting semantic problems you plan to catch.

d) Present the state of your work in class (lecture) on **June 29 (mandatory)**
Your presentation should

- last approximately 7 minutes.

[8] https://www.typescriptlang.org/
[9] https://kotlinlang.org/
[10] https://dev.java/

- describe the chosen language subset with examples of
  - legal programs
  - syntactically illegal programs
  - semantically illegal programs
- demonstrate the current state of your work:
  - show what works, and what doesn't (yet)
  - include a live demo (if possible)

Please also see the preparation tasks below.

Task 2 Provide a grammar in LBNF (for use with BNFC) for relevant features of the language you chose. The size of the grammar should be comparable to the first project phase (not counting identifiers and expressions).

Task 3 Provide a type checker for relevant features of the language you chose. The type checker should check at least two language features that are not covered in the second project phase.

Preparation tasks

The following tasks in Table 1 should help you prepare for your presentation and your final submission of this project. Each of those tasks is connected to one exercise session that starts with a short recapitulation of the current topic, and consists of a small preparation task you're supposed to do at home, and an in-class task that you work on during the session, with the teaching assistant available for help.

Submission

Submit your solution via GitLab[11]. If you have any problems, contact the teaching assistant.

Your submission should include the grammar, frontend source code, and your test programs (at least 10 legal, 10 syntactically illegal, 5 semantically illegal, cf. above). Additionally, you need to provide a way to build the compiler frontend (e.g., using `make`). Finally, include a little documentation in an appropriate format (e.g., a README file in plain text or Markdown[12]; binary file types are not accepted (pdf, doc(x), odt, ...)). The documentation should include information on:

- How to build and execute the compiler frontend.
- Your chosen language subset and test cases:
  - Which parts of the language are supported, and why?
  - Which test case tests what?

[11] https://collaborating.tuhh.de
[12] https://www.markdownguide.org/basic-syntax/

Table 1: Preparation tasks

Topic	Date	Preparation	In-class
Syntax	27.04.23	Tasks 1 a), b). Bring a grammar specification or comparable documentation for your chosen language.	Start working on the grammar subset and related test programs.
Semantics	25.05.23	Tasks 1 a), b). Bring documentation of your chosen language's type system	If necessary, refine the grammar subset, and start working on the semantics-related test programs.
Approval	15.06.23	Task 1 a), b), c).	You present and get feedback on your chosen language subset. Refine your language subset and/or continue working on your implementation.
Feedback	29.06.23	Task 1.	You get feedback on your presentation & current state, and we discuss the remaining steps to be taken in the last week of the project.

Grading

You can earn 35 points in this project.

- 5 points are awarded for your language proposal (the presentation).
- 13 points are awarded for the grammar/parser, including the related test cases.
- 13 points are awarded for the type checker, including the related test cases.
- 4 points are awarded for the build system and documentation.

As with the other tasks, make sure that someone else can understand what you did, by producing *readable* code.