Week by Week, EDAN70, 2025

Compiler projects

Week 1, Week 2, Week 3, Week 4, Week 5, Week 6, Week 7, January, Initial Report, IPSERC Report Structure, Finding Related Work, Release, Evaluation, Presentation

---

In this course you will implement a compiler-related tool. The work is research-oriented: You will read related articles, do an evaluation of your tool, write up the results in the form of a short research report, and present your results to your course mates. Your code will be open sourced under the BSD license.

If you are doing a project in another area than compilers, check with your supervisor what parts of these instructions apply, if any.

Week 1

- Introductory course meeting, Monday 13-15, Nov 3, 2025, E:2116
- Select project
- Meet with supervisor
    - Discuss project content: motivation, what to implement, how to evaluate, related work
    - Advice about getting started with implementation
    - Advice about writing initial report due next week
    - Concrete plan for next week
- Read about the Initial Report.
- Skim through IPSERC Report Structure
- Start writing initial report with introduction and outline
- Start reading related papers
- Get started on implementation

Week 2

- Continue work on implementation and first version of report
- Read about Finding Related Work
- Meet with supervisor
    - Progress since last meeting
    - Advice about first version of report and finding related work
    - Advice about implementation to create demo due next week
    - Concrete plan for next week
- Email initial report to supervisor by Friday, Nov 14

Week 3

- Continue work on implementation to create demo
- Read about Releases
- Meet with supervisor
    - Progress since last meeting
    - Feedback on initial report
    - Show demo to supervisor
    - Advice about continued development
    - Advice about alpha release
    - Concrete plan for next week
- Email improved initial report to supervisor by Friday, Nov 21

Week 4

- Continue work on implementation and release
- Read about Evaluation
- Meet with supervisor
    - Progress since last meeting
    - Feedback on report
    - Discuss evaluation
    - Show improved demo to supervisor
    - Advice about writing about the solution in the report
    - Concrete plan for next week
- Do alpha release and email info about it to supervisor by Friday, Nov 28

Week 5

- Continue work on implementation and paper
- Read IPSERC Report Structure in detail.
- Meet with supervisor
    - Progress since last meeting
    - Feedback on first release
    - Discuss more related work
    - Discuss presentation
    - Concrete plan for next week
- Email middle version of report, including solution description, by Friday, Dec 5

Week 6

- Continue work on implementation and paper
- Read about Presentation
- Work on draft of slides for presentation. Bring initial draft of slides to meeting.
- Meet with supervisor
    - Progress since last meeting
    - Feedback on new version of report
    - Discuss draft of presentation slides
    - Concrete plan for next week
- Email improved draft of slides to supervisor by Friday, Dec 12

Week 7

- Continue work on implementation and presentation
- Present your work at one of the seminar sessions. Listen to other presentations at the session.
- Email final version of slides to supervisor by Friday, Dec 19

January 2024

- Finalize report and release. Make sure to follow all the instructions.
- Email final report and info about release to supervisor by Wednesday, Jan 14, 2026

Initial Report

Getting started

You should get started early with writing. Already at the second meeting you should have an initial report with

- Introduction section, somewhat fleshed out (but at most one page) with
    - Motivation (why is the work interesting/useful)?
    - Problem description (what is the problem you intend to solve)?
- Solution section, with an outline of how you plan to solve the problem
- Evaluation section, with an outline of how you plan to evaluate your solution to the problem
- Related work section, with an outline of existing alternative solutions to the problem, and existing solutions to related problems.
- Conclusion section, empty for now.

- References. The most important ones should be cited already in the introduction section.

As your project progresses, you update and flesh out the text in the report. In the solution and evaluation sections, you replace your plans with what you actually did..

Advice on writing

Format

- Your final report should be written in LaTeX.
- Use the LaTeX template (ReportTemplate.zip) that you can download at https://bitbucket.org/edan70/reporttemplate/downloads/
- For references and citations, use BibTeX
- Your paper should be submitted as a pdf by email to your supervisor.

General

- 3-6 pages for single author. 4-8 pages for pair authors. No less. No more.
- Your audience are engineers that have taken a compilers course, but not necessarily the one at Lund University.

Writing style

- The text should be “fluid”: easy to read and understand.
- Use active rather than passive voice.
- Use clear and precise language. Give good examples. Avoid jargon. Avoid claiming too much. Use words and expressions at your own level of English: avoid the use of fancy words or expressions that are not part of your normal vocabulary.

Composition of paragraphs

To make the text fluid and easy to understand, you need to understand how to structure sentences and paragraphs. Here are some pointers:

- William Strunk has excellent advice for writing in his book from 1918: William Strunk, The Elements of Style, available online in the Gutenberg project: https://www.gutenberg.org/files/37134/37134-h/37134-h.htm
  In particular, look at chapter III: Elementary Principles of Composition. It has advice about topic sentences, using the active voice, using specific and concrete language, etc.
- Lund University has a MOOC in English on academic writing. The following page discusses paragraphs with topic sentences, supporting sentences and concluding/transition sentences: https://www.awelu.lu.se/writing/writing-stage/structuring-the-text/structure-within-paragraphs/

- Lund University also has a MOOC in Swedish on academic writing. You can e.g., look at the following short videos about structuring paragraphs: “Att skapa flyt i texten”
  - Del 1: https://www.youtube.com/watch?v=-NCDpG-pfEs

    - Del 2: https://www.youtube.com/watch?v=PM91-o3n_cw
    - Del 3: https://www.youtube.com/watch?v=M0V3kQP9O60
    - Del 4: https://www.youtube.com/watch?v=FzR-edHFYoY

  You can find the whole playlist for the course “Akademiskt skrivande” at https://www.youtube.com/playlist?list=PLeVxAnFsasIoRqR6SgakSiqGKxy0yjKbx

Plagiarism issues

- Never copy text from wikipedia or any other source. All your text must be written by you, using your own words, and without using so called *patchwriting*. See https://www.awelu.lu.se/academic-integrity/plagiarism/different-kinds-of-plagiarism/
- You are encouraged to improve your text by using a spell checker and/or grammar checking tool. Examples of tools supporting this include Word, Google Docs, and Grammarly.
- Some of these tools may include more advanced features generative AI features that suggest rephrasing or restructuring your text, or where you generate text from prompts. You are **not** allowed to use those features.

IPSERC report structure

Research articles in different fields have different structure and style. In this course you are writing a CS engineering paper. These papers typically follow an IPSERC structure (Introduction (with Problem), Solution, Evaluation, Related work, Conclusions). This structure is different from the typical IMRaD structure (Introduction, Methods, Results, and Discussion) used in natural science, and that you may have learnt about elsewhere.

To understand how to write IPSERC papers, we recommend to:

- Study the structure of research articles in computer science where the main contribution is a new tool, method, or idea. You will likely reference a number of such papers.
- Look at the structure of previous EDAN70 student papers on compiler projects. You find them at https://fileadmin.cs.lth.se/courses/edan70/CompilerProjects/current/.
- Look through Simon Peyton Jones talk on How to Write a Great Research Paper: https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/

Your final report should have title, authors, abstract, introduction section, optional motivating example section, optional background section, 1-3 solution sections, evaluation section, related work section, conclusion section, and references. These are detailed below.

The title

- Short yet specific. You should get an idea of what the paper is about by reading the title. Max 10 words. Five is better.

Authors, etc

- Authors should appear below the title. As affiliation you can write, e.g., D21, Lund University, Sweden. See the report template. The template will automatically include information about EDAN70 and about the date.

Abstract

- 100-200 words. Describe what you have done and why it is interesting.

Introduction

- Should not be more than one page.
- Should describe the problem you are attempting to solve, and motivate it, i.e., describe why it is interesting. So you describe the same thing as in the abstract but in a little more detail.
- In doing this, important related work should be briefly mentioned and cited, to put your work in context.
- Should summarize the results.

Motivating Example (optional)

- It might be useful to have a section in the beginning that describes a motivating example. This can be a great way to introduce the problem, and you can then in subsequent sections describe how you have solved the problem.

Background (optional)

- You might need a section on background theory or systems to explain what existing work that you build your solution on. Try to explain just the things that the reader needs to know to understand your solution in later sections. I.e., this section should not be a general tutorial. Don't forget to include references to the existing work that you describe.
- You don't need to call this section "Background". Hopefully you can find a more specific title.

Solution

- Describe your own work in typically 1-3 sections, i.e., the solution to the problem. In these sections you can give an overall description of what your system can do, and give interesting concrete examples. You can also describe how you have implemented the system, giving an overview of the implementation parts, and discuss interesting details of the implementation.
- Give the sections meaningful titles that relate to the content. “Solution” is not a good title. Look at other research papers to understand how to do this.

Evaluation

- Describe what you have evaluated, how you did the measurements, what the results are, and what conclusions you draw from these results.

- Include details about your experimental setup, like what tools you used for the measurements, how many times you repeated measurements, etc.
- If you want to, you can use the IMRaD structure within the evaluation section:
    - *Introduction*: explain what is important to evaluate about the solution and why. You can formulate explicit research questions and hypotheses if you like.
    - *Method:* explain what research methods are used to prove/disprove hypotheses, or answering the research questions. Explain your experimental setup for the evaluation.
    - *Results and Discussion*: what were the answers found, and what are the implications of the answers.

Related work

- Discuss how your work relates to what others have done. This puts your work in context so that others can more easily understand it. The goal is to both point out similarities and differences to other work.
- Examples of related work are both work that has tried to solve the same problem in other ways, but also work that has tried to solve other similar problems.
- You will often have a brief discussion of key related work already in the introduction section. In this separate section that comes after the solution and evaluation sections, you have the possibility of comparing in more detail.
- Make sure to give proper references to all the related work you discuss.

Conclusion and Future Work

- Briefly sum up what you have done and what the results were.
- Discuss the results, and how they can be useful.
- Discuss ideas for future work.

Acknowledgments

- If you got help from individuals, thank them briefly.

References

- There should be **at least five references to books or to peer-reviewed articles in conferences or journals**.
- The sources should be **properly referenced**, including authors, title, conference or journal, publisher, and year.
- You may include the "doi" (digital object identifier) in order to link to the publication, if there is one.
- You can have references to web pages, blog posts, etc. But these don’t count as important as research articles, since they are not peer reviewed. If you cite such sources, don’t just give the web page (it will likely change in the future). Give the title, and preferably also the authors or organization behind the web page, and preferably also the year when it was written. Also include a date for when you accessed the page.
- If you provide a link in a reference, always use original sources of the information, i.e., not secondary sources like wikipedia, google books, or similar.

- If you are citing the web place of a system, it is better to write it as a footnote instead of as a reference. References go to written material. For systems, it is a good idea to try to find an article describing the system that can be cited as well.
- Don't cite wikipedia. It changes, and it is secondary information. However, wikipedia is great for finding information, and there are often interesting references to real sources there. Those might be interesting to cite.
- Never cite an article that you don't have access to – you need to read enough of the article so that you know exactly what you are citing.

Finding Related Work

Finding articles

Try to find research articles about your subject. The easiest way is to use Google Scholar (https://scholar.google.se). If you find an interesting article, look at what other articles it cites. You can see in Google Scholar what other articles cite the interesting article. If you have difficulties finding good articles, discuss with your supervisor for more ideas for what to search for.

Accessing articles

To access the pdf version of a research article (full text), you often will have to be on the university network, or login using your STiL account.

For books, you can use the LTH library (https://www.lth.se/bibliotek/). They can help you with loans from other libraries as well.

Referencing articles

You should reference relevant articles in your report. Look at other research papers for how to write citations and references. Using BibTeX is highly recommended. DBLP has high quality bibtex for research articles.

Releases

Your final version of the project should be submitted as a tagged release of your source code repository.

- **README**: There should be a README text file explaining how to build, test, and run the product, and explaining where to find things, installation instructions, etc. It is recommended to write the README as a proper Markdown document (https://en.wikipedia.org/wiki/Markdown).
- **Tests**: There should be an automated test suite with suitable test cases.
- **Examples**: There should be examples, and instructions for how to run them.

- **LICENSE**: There should be a LICENSE text file, explaining the source code license used (2-clause BSD). See example below.
- **Credits**: If your repository includes files that were copied from others and possibly modified, the README file should include information about this, and also about what licenses those files have. See example below.
- **Tagging**: The final release should be tagged FinalA. If you need to fix something and submit a new version, you should tag that one FinalB, and so on. Tag earlier releases with Alpha1, Alpha2, ...

First Release. Should be tagged with Alpha1. You should have done a clean checkout of it and verified that the tool can be built and run according to your README instructions.

Intermediate releases. Improve your release and notify your supervisor. Make sure there is always an up-to-date README.

Final release. Should be tagged with FinalA, and should be available when you submit your final report in January.

Example license file

Your repository should include a file LICENSE that explains that you have the copyright, but that the source code is distributed under the 2-clause BSD license. Here is how you write that file (where NN1 and NN2 are your names):

```
Copyright (c) 2025, NN1 and NN2
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Example of referring to the LICENSE file in the README file

It is a good idea to write in the README file what license you use for your code. Here is how you write that in the README file:

```
License
-------
This repository is covered by the license BSD 2-clause, see file LICENSE.
```

Example of giving credit to other software

If your repository includes files that you have copied from other sources, and possibly modified, you need to give credit in the README file. Here is an example of how you can write:

```
Credits
-------
The following files in the repository come from:

- include/fmi2/: Header files from the FMI2 standard, licensed under BSD 2-clause.
- include/fmusdk2/: FMU SDK from QTronic, licensed under BSD 2-clause.
- build_fmu: based on script from a port of FMU SDK by Christopher Brooks, licensed under BSD
2-clause.
```

Evaluation

You can evaluate your results in many different ways. As always, look at other research papers for how they do evaluations. Here are some resources for helping you do a good evaluation.

Performance evaluation

If you run your results in Java or similar managed languages, you should typically differentiate between start-up and steady-state performance, depending on if you measure from the start of the program, or after the JVM has run long enough to optimize the code. Here are some papers to read about this:

- Andy Georges, Dries Buytaert and Lieven Eeckhout: Statistically Rigorous Java Performance Evaluation. OOPSLA '07. (https://dl.acm.org/doi/10.1145/1297027.1297033) Read sections 1-4, but skip

sections 3.4 and 3.5. In particular, read 4.1 and 4.2 for startup or steady state measurements.
- S. M. Blackburn et al.Wake up and smell the coffee: evaluation methodology for the 21st century. Communications of the ACM, 51(8):83-89, 2008. (https://dl.acm.org/doi/10.1145/1378704.1378723) You can skim this article.

Confidence intervals

When you measure performance, you should normally measure several times, and take the average and compute confidence intervals. Here is some advice on how to do this: https://bitbucket.org/edan70/course-material/wiki/Confidence%20interval

Source lines of code

One way to measure how much effort it is to implement something is to measure the number of lines of code. It is of course a crude measure, so it is usually only the order of magnitude that is interesting here. Typically, you measure only the source lines of code (SLOC), i.e., code without blank lines and without comments. A tool that can be used for this, and that handles many programming languages, is cloc, see https://github.com/AlDanial/cloc .

Usability evaluation

Large user studies are beyond the scope of this course, and are only relevant when you have a fairly mature tool. But a crude measure is to make a small study and see if someone other than yourself can use your tool for a given task. One very useful variant is to do short simple “Think aloud” studies. Here, you get feedback from how users understand your tool, which is very useful during the development. Some resources on usability evaluation:

- Xavier Ferré, Natalia Juristo, Helmut Windl, Larry Constantine: Usability Basics for Software Developers IEEE SOFTWARE January/February 2001. (https://doi.org/10.1109/52.903160)
- Books on Usability, for example Jacob Nielsen: Usability Engineering. AP Professional, 1993. See also his website with articles on Thinking Aloud and 10 Usability Heuristics for User Interface Design:
    - https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/
    - https://www.nngroup.com/articles/ten-usability-heuristics/

Proof of concept examples

A good way to evaluate your tool is to show that it works for a number of interesting examples. There are different ways to do this, and often several of these are used in a tool evaluation:

- Systematic test case examples. This shows some kind of completeness of the tool. Explain in what way your examples are systematic.
- Small compelling examples. This serves the purpose of explaining the most interesting aspects of what the tool can do.
- Realistic example that is useful, and of reasonable size.
- Very large example, showing that the tool can handle large examples.

- Examples constructed by others. If your tool can handle examples created by others, this gives more credibility than if you only show it can be used on examples created by yourself. For example, if you are analyzing Java code, run your tool on existing Java repositories. Discuss with your supervisor which such repositories might be suitable.

Presentation

- Prepare for a 15 minutes talk.
- Your talk will be followed by a 5 minute question session.
- Use around 10 slides.
- Your presentation should include motivation, examples of what your prototype can do, and results from your evaluation. If you can show a quick demo too, that is always nice.
- For two-author projects, both should contribute equally to the presentation. Preferably switch a bit back and forth between who is talking.
- We expect an interesting and well prepared presentation.

Simon Peyton Jones has given a good talk on how to do research presentations in computer science. See
https://www.microsoft.com/en-us/research/academic-program/give-great-research-talk/

Before your talk, get feedback from your supervisor on your slides.