

Report on the solution of Oblig 1 in IN2031

Student X (studx@ifi.uio.no)

Student Y (study@ifi.uio.no)

August 28, 2026

1 Introduction

Start with a short descriptive introduction. You are writing for someone else, so the introduction must give the reader an idea of what to expect in the rest of the report.

Always consider who you are writing for. For reports at DIKU you are always writing to explaining your work for other students at your level. Thus, a student should be able to read first the assignment text and then your report to get the full idea of a solution.

This report documents the reference solution of the Oblig1 assignment. It is intended not just to explain the reference solution, but also as an example of a report with good structure and content. You may use it as inspiration for reports you write later during this and other courses. However, do not follow it slavishly: different problems call for different solutions, and different people have different style. If you prefer another style, then stick to it. An idiosyncratic report executed well is superior to a idiomatic report executed poorly.

Also tell the reader where to find things; use specific reference where relevant. You are writing a technical report (not fiction), so no need to surprise the reader with plot twists.

The task was to implement a small programming language. I have not found reason to stray from the functional requirements outlined in the report, and it is to my knowledge implemented completely. For the pedantic, Section 5 notes some theoretical deficiencies.

2 Design

Unless some background knowledge is needed (only if you use things not taught on the course), start by explaining the considerations for your. This should *not* be a direct description of our assignment text; the reader knows this. Instead describe how you read the problem and how you have decided to solve it. What choices have you taken for your implementation? What was not clear and how did you solve this?

The core of the program is ...

3 Implementation

In this part you describe you implementation. Do not describe the all files, procedures and object. Focus on how you get for your design to the implementation. Did you use some interesting data structures? What computer science choices did you take in your implementation. This part should also aid understanding your code, without talking about specifics. Thus, after reading this, I can more easily read your code.

My implementation ...

4 Testing

Tests are to show what your implementation can and cannot do. Like any other scientific experiment tests must be reproducible. Start by explaining the test methodology and how the reader can run them.

I test by extending the handed-out gradle unit test suite, with

Explain the different test cases that you have; with more tests you can group them together. This must also show the reader test coverage. For A0 it is very easy to see based on the below examples, but for larger test suites you would need to argue this explicitly.

Interesting cases include:

- ...
- ...
- ...

4.1 Test Limitations

End by describing what your tests cannot show. This shows that you understand limitations of the methodology and coverage.

Consider not only positive tests, but in some cases negative tests can be used to show what your implementation cannot do.

The automated tests only cover the *happy path*, not any error cases. I have tested trivial cases ...

5 Limitations and Potential Problems

Explain what your implementation cannot and what problems there might be. This should of course be in relation to the assignment text. E.g. writing that the implementation cannot solve the *halting problem* is not useful.

If a string with + is inputted our implementation will.

6 Conclusions

End with a conclusion that (1) explains to what extent you have solved the problem and (2) sums up highlights of the report. This is your final chance to have the readers attention and point to the important parts of your report.

We have implemented a ... I judge the implementation to be ...