

# CS 343S Project: Language Design and Implementation

## 1 Deliverables and Due Dates

Below are short descriptions and deadlines for each component of the term project. The following sections provide more details on the expected contents of the deliverables. The deadlines and requirements are tentative; we will announce any changes in lecture and on Ed.

Component	Deadline
Brainstorming	Tuesday, April 22nd, 11:59 PM
Proposal	Tuesday, April 29th, 11:59 PM
Interim Project Presentations	(In-class) May 13th and 15th
Presentation	In-class May 29th and June 3rd
Final Implementation	Friday, June 6th, 11:59 PM

Table 1: Project deadlines

## 2 Summary

The deliverables are designed to guide your project direction and allow you to receive feedback at multiple points in time. That being said, you are not restricted to prior deliverables (i.e. you can write a proposal for an idea that you did not submit for brainstorming, and you can change the language sketch from your proposal while implementing the language). **There are no teams for final projects.** You are expected to implement your own project, and will receive feedback both from the staff and from a small group of your peers. You are welcome to implement your project in *any language you prefer*, you are not required to use the Python tools we will teach in the first few weeks of class.

## 3 Brainstorming

Your submission should include 3-5 short problem descriptions that you believe might be eloquently solved by a domain-specific language. Each problem description do not need to be longer than a paragraph (though you are welcome to write more), and should describe a **problem domain**, as well as at least one concrete example of a problem in that domain. We provide two sample brainstorming examples below:

### 3.1 Brainstorming Example 0

*Fixed-point arithmetic [1] is commonly used in high-performance digital signal processing codes. Programmers using fixed-point arithmetic must painstakingly track the range of possible values and precision of each variable. This is in stark contrast to the more-commonly used floating-point arithmetic, where the*

number type itself automatically controls precision. A language for fixed-point arithmetic could allow the compiler to perform this task automatically, alleviating programmer cognitive burden. Consider a simple 8-bit multiply-add:

```
uint8_t a, b, c;  
? d = a * b + c;
```

What is the correct output type for this operation?  $a * b$  can overflow, so should be widened to 16 bits, and adding  $c$  requires another bit of precision. Therefore,  $d$  needs to be at least 17 bits. A correct program looks like:

```
uint8_t a, b, c;  
uint32_t d = (uint16_t)a * (uint16_t)b + (uint32_t)c;
```

A DSL for fixed-point arithmetic would allow the user to write just the mathematical expression, and the compiler would figure out types and when to insert widening. Alternatively, a DSL might help the programmer reason about approximate fixed-point computations. For instance, by deducing or allowing the programmer to specify operations which should intentionally lose some precision, and then bound the impact of that precision loss.

## 3.2 Brainstorming Example 1

One common game for children (and adults) is assembling LEGOs. In this, someone takes a set of small, simple pieces and follows either instructions or inspiration in order to assemble them into a larger, complex structure. Can a DSL be used to express LEGO assembly instructions? The DSL might be executed to produce a virtual representation of the final assembly, or to produce a rendering of visual instructions that a human could follow to build the assembly.

Consider the problem of describing this assembly. Perhaps it could be described by a program that formalizes the following pseudo-code:

```
pin p1, p2;  
long l1, l2;  
short s1, s2;  
flat f1, f2;  
insert p1 into hole 2 of l1;  
insert p2 into hole -2 of l1;  
insert p1 into hold 2 of l2 and p2 into hold -2 of l2;  
check that l1 and l2 are adjacent, with shape 15(?)x2x1;  
call this group b;  
// ..continue..
```

In the case that a program produces a virtual assembly, that assembly could be (simplistically) rendered with pygame.

The basic nouns in this language appear to bricks (the things an assembled) and locations (used to specify what connects to what). The basic verbs are to connect bricks and to compute locations from bricks.

## 4 Proposal

Your proposal will involve choosing one specific problem domain, and providing a language sketch for your DSL. Your submission should be a 2-3 page report

providing a sketch of your language, and multiple examples of programs in the language, highlighting more examples in the problem domain than in the brainstorming submission.

## 5 Interim Project Presentations

You will demonstrate some working components of your language implementation to a small group of your peers and the teaching staff. You are expected to have a working, end-to-end demo of some part of your language at this point. It doesn't have to do everything you promised in the proposal, but it should do something. This is a live (in-class) presentation. Please prepare enough content (slides, live programming, etc) for 5 minutes, followed by 3 minutes of Q&A. This is a good opportunity to refine the sketch from your proposal, discuss any possible issues you have encountered, and get feedback on your design. You will be expected to pay attention to your peers' presentations, and provide feedback to them as well.

## 6 Final Submission

Your final submission contains three parts:

1. Link to a repository containing your language implementation.
2. A language tutorial walking a user through writing programs in your language, in Markdown (as is standard in many open-source project repos).
3. A report (4-8 pages) detailing your language and your implementation of that language.

Your tutorial should not just involve writing the equivalent "Hello World" program (though it should include this, if such an example exists), but should also provide examples of *each* operator in your language, and a walkthrough of a sufficiently complicated program.

Your report should provide a thorough high-level description of the language and implementation, with figures detailing any necessary compiler details. You should describe which problems in your chosen domain are expressible, and which *are not*. Your report should clearly show how problems in your chosen domain map to language constructs in your DSL. Additionally, you should include a short related works section (if applicable), *and evaluation*, whether that is a user report, performance numbers, or otherwise.

## References

- [1] Wikipedia. Fixed-point arithmetic. [https://en.wikipedia.org/wiki/Fixed-point\\_arithmetic](https://en.wikipedia.org/wiki/Fixed-point_arithmetic), 2024. [Online; accessed 11-February-2024].