Overview. The goal of this project is to dive deeper into language design and implementation by developing a significant extension to MyPL. The project will be graded on: (1) the ***quality of your work***; (2) the ***completeness of your work***; (3) the ***quality of your tests and documentation***; and (4) the ***difficulty of your project***. The project is worth 60 points and includes a proposal, a project specification, and a demo presentation (video and slides).

Step 1 (due by March 7th). The first step is to identify *two* extensions you are interested in exploring.[1] The following is a non-exhaustive list of possible project ideas. You are not required to select one of the following—each is just an idea of a possible project that you may be interested in or may help give you ideas in terms of picking your own direction. You should also look at existing program languages for inspiration for possible features to add to MyPL. The projects are generally grouped by difficulty below. However, note that difficulty can also depend on the thoroughness and scope of your design, testing, and overall implementation as well.

(*Significantly Difficult*) The following projects are more involved and generally require additional thinking about design decisions as well as some independent research into how each would work. Note these would score significantly higher on the "level of difficulty" project criteria.

- Compilation to another VM-based interpreter (e.g., JVM or CPython bytecode, WASM, LLVM IR)
- A binary-based bytecode for our VM (extending what we've done in class)
- A completely separate language syntax design that "compiles" to our VM
- Compilation to assembly language (e.g., to ARM or x86).
- Fully functional implementation of a garbage collector
- Full featured module/import system (e.g., like "import" in Python and Java)
- Concurrency/multi-threading support
- Performance optimizations (pick a set of optimizations, implement and benchmark them). Note that that there are various kinds of optimizations possible, but many standard optimizations are more applicable to compilers than to VM-based interpreters. Optimizations that focus on reducing the amount of generated code and/or the overhead of function calls will generally lead to speed improvements.

(*More Challenging*) The following projects are less challenging than those above, but still require thought about design issues and/or involve some implementation challenges and research. These projects would generally score on the higher end of the "level of difficulty" evaluation criteria.

[1] Because each student in the class will need to work on a different extension, you must provide a first and a second choice. Assuming neither of your choices have already been assigned, you will receive your first choice, followed by your second choice (if the first choice is taken), or else be notified to select an additional alternative.

- Classes (with public/private member variables and methods)
- Pointers and/or pass by reference
- Function or operator overloading
- Struct type subclassing (inheritance)
- Other more advanced language features such as named parameters and default values, decorators, generators, higher-order functions, closures, pattern matching, reflection, and so on.

Less Challenging: These projects are more straightforward and are generally easier to implement than those above. These projects would generally score on the mid-to-low end of the "level of difficulty" evaluation criteria.

- Transcompilation to another language (e.g., C, C++, or Python).
- Additional collection types (e.g., dictionaries, lists, data frames).
- List comprehensions.
- Exception handling (e.g., try-catch blocks).
- Additional "common" constructs (e.g., switch statements, for each loops).
- Constant variables and function args.
- Proper handling of associativity and precedence.
- Collections of new operations and operators.
- File I/O.
- Delete operation.

Again, you are free to choose something different than what is listed above. The difficulty level of your project will generally be based on the scope of your project, the degree of modifications and/or additions needed to implement the extension, the complexity of the extension itself, the amount of research you had to do to implement the project, and the amount of testing and quality assurance you perform.

Submitting your Choices. You must submit your first and second choice preferences in class (as a hard-copy print out) on or before the due date. For your first and second choices, you must state each extension and provide a short description of what you plan to tackle (the scope). This description should contain a list of the "features" of the extension (e.g., what a user would be able to do, what capabilities will be added to MyPL, etc.). Note that these are just your initial thoughts to help me understand what you are thinking in terms of the project. Note that a special "project" repository will be made available in GitHub for you to store and submit your project source code and documentation.

Step 2 (due by March 31st). For this step, you must hand in a hard-copy specification (plan of attack) for your extension and what you have already accomplished. The details of the specification will differ depending on your project. At a minimum, the specification should include the scope of your extension, examples of how a user would use your extension (e.g., what the new syntax is), a detailed plan for how you will test your extension, an initial design (e.g., what will need to be modified in the MyPL implementation, what will need to be added, etc.), and a weekly plan with milestones for completing the project. Note that you will have five weeks to finish the project from this point.

Step 4 (due by Friday, May 2nd.) Your project must be fully completed on or before the Friday of the last week of classes of the semester. All of your source code must be pushed to your repository as well as the slides you used for your final project presentation. Your project presentation must consist of a "narrated" video that consists of:

- Your name, the course, the semester, and a brief description of the extension you implemented with examples.
- A demo of the extension "in action", in which you must: show how the extension would be used in a "real world" setting, give a walk-through of the (non unit) tests you developed (e.g., mypl example programs), and show your extension working over the tests.
- A discussion of any additional tests you developed (but didn't include in the demo). Also describe the scope as well as "interesting" examples of the unit tests you created.
- A description of the parts of the MyPL pipeline that you modified and at a high level what the modifications were.
- What was successfully completed, what if anything wasn't completed (compared to your specification in Step 2), and what you would do next if you had additional time.

Your video should be *at most* 12 minutes in length and must be made accessible (i.e., either by including it into your repository or by including a link to the video). Note that a large portion of your grade will be based on your demo. It is your responsibility to ensure I can access the video and that the video correctly presents the material. Finally, you must create a README.md file for your GitHub repository that, at a minimum, contains a link to your video presentation (or where I should find it) and any additional instructions necessary to run your extension. All materials must also be available in your GitHub repository, including your slides, your test programs, your unit tests, and all source code needed to run your extension.

Grading Rubric

The following information describes how points will be allocated for your final project. Note that the project is worth a maximum of 60 points.

- **Steps 1 and 2 (5 points).** Steps 1 and 2 will each be worth a total of 2 points. An additional point will be awarded at the end of the project if both check-ins are turned in by their respective due dates.

- **Project Completeness (15 points).** You will receive up to 15 points based on whether and to what extent the project was fully completed. This includes whether all parts implemented work correctly and that all necessary parts were implemented. Note that if you have not fully implemented the base MyPL system (i.e., HW 1–6), points will still be deducted for not fully completing the extension if it relies on these base portions being finished.

- **Presentation and Slides (10 points).** You will receive up to 10 points for the completeness and overall quality of your presentation. Completeness here means that you covered all parts asked for with respect to your video presentation. Note that the points in this category are predicated on the successful completion of your project.

- **Difficulty of Extension (10 points).** You will receive up to 10 points based on the overall difficulty of the project. As mentioned above, the difficulty will be based on a number of factors. Similar to the presentation, the points in this category are predicated on successful completion of your project.

- **Overall Quality of Extension (10 points).** You will receive up to 10 points based on the overall quality of your implementation. Overall quality will be based on easement of the code you write (including comments, coding style, etc.) as well as from the demonstration of your extension as part of your presentation.

- **Scope and Quality of Testing (10 points).** You will receive up to 10 points for the scope and quality of your tests, which includes the overall coverage of your tests.

Optional Extra-Credit Opportunity *Multiple Due Dates*

Overview. You can earn up to 30 points (approximately one homework assignment) by completing *one* of the options below. Note that *you must complete your semester project to receive the optional extra credit project points.* In other words, the extra credit is not an alternative to doing the project.

Option 1. This course scratches the surface of programming language design and theory. This project option allows you to go deeper into specific topics that you find interesting and/or would like to learn more about. For this option, you must:

(a). Find two research papers of interest (see below). These can be on similar or different topics related to programming language design and implementation.

(b). Read the papers and take detailed notes on each. Your notes should be organized by the sections of the paper, should include definitions and important ideas (in your own words), and any other items that you find useful to write down to remember as you read the paper. In addition, you must include observations on the work including what you find confusing, interesting, and general questions that you have, as part of your notes. Note that reading a research paper is not like reading a novel (i.e., reading in one pass without studying and interacting with the content). Instead it is an active process, that requires multiple passes and more engaged reading and writing to effectively work through the material. For example, in the first pass, you might read the title, section titles, then figures and captions to get a general sense of paper topics, overall organization, maybe scope of results. In the second pass, you would read the abstract and the introduction to get a sense of what problem is being solved, how the authors go about solving the problem, and what the results of the work are (broadly). In the third pass, you would do a detailed read through each section, taking notes (described above) as you go.

(c). In addition, as part of your notes (in a separate section), describe in detail how the work relates to what we have covered in class and/or through homework assignments (as applicable).

(d). Create a presentation (slides and a video) covering the main ideas of the papers. The target audience for the presentation would be the students in our class (along with myself), with the goal of making it clear what the main ideas are in each paper, their relation to programming language design and implementation broadly, and their relation to the ideas we have covered in class. As part of your presentation, you must include for each paper the main technical ideas and contributions described in a way that is clear and easy to follow. You should aim for a presentation that is *at most* 20 minutes long. Thus, around 10 minutes per paper. The actual length will depend on the nature of the papers you select and how well you are able to distill the work down into the major ideas and contributions (not an easy task!).

Here are some programming-language specific conferences to help you find relevant research papers. Note that each link below is to the conference program, which lists the papers by general topic area (with links to the abstracts and the papers themselves). These conferences have a mix of papers on PL theory, systems, and practical aspects (e.g., benchmarking).

- The Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) Conference: 2024, 2023, 2022.
- The International Conference on Software Language Engineering (SLE): 2024, 2023, 2022
- The Symposium on New Ideas in Programming and Reflections on Software (Onward!): 2024, 2023, 2022
- The ACM SIGPLAN Symposium on Principles of Programming (POPL): 2024, 2023, 2022.
- The ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI): 2024, 2023, 2022

Option 2. The second extra-credit option is to learn a new programming language, implement a significant program(s) in the language, and create a basic tutorial presentation (slides and video). The language you select must be one you are not already familiar with and be significantly different than those discussed in class or that you already know. Your example program(s) must be non-trivial and demonstrate the different and/or innovative constructs and features of the language. For this option, you must:

(a). Select the language you plan to study and identify the novel features / constructs of the language.

(b). Find resources for learning the language and define one or more example programs to implement.

(c). Use the resources to learn the language and implement the example programs.

(d). Create a presentation (slides and a video) that: (i) briefly describes the language you selected and why you selected it; (ii) describes (using examples) the novel features and constructs of the language; (iii) gives a live demo of the example program(s); (iv) shows how the novel features were used to implement the example programs (e.g., via snippets of relevant code); and (v) summarizes your thoughts and opinions about the language. You should aim for a presentation that is *at most* 15-20 minutes long. Note that you won't be graded on the length of the presentation—shorter is better as long as you cover items i–v above.

Step 1: Proposal (due on or before April 4th). If you are planning to do an extra credit option, you must submit a proposal. For option 1, your proposal must list the two papers you plan to read and review. Include a full reference to the paper (authors, paper title, conference, and year). For option 2, your proposal must include the language you plan to learn and the example program(s) you plan on implementing.

Step 2: Submission (due by the end of Finals Week). Similar to the project, a GitHub repository will be created for submitting your extra credit work. For option 1, you must include your notes, slides, and presentation. For option 2, you must include all source code for your example program(s), instructions for running your programs, your slides, and your presentation.