Project Information:

Our project name is Heap Heap Hooray, and our project will be to develop a garbage collector for the MiniJava compiler. This project will require us to deep-dive into garbage collector implementations, understanding and learning from them to write one that caters to the project's goals.

Our team consists of Tyler Gutowski (tgutowski2020@my.fit.edu), and Trevor Schiff (tschiff2020@my.fit.edu)**.** Our advisor and our client is Dr. Ryan Stansifer (ryan@fit.edu), a compiler researcher who has a deep understanding of the Java compiler.

We met with Dr. Stansifer on Monday, November 27th at 12:00pm to foster a deeper understanding of the project. We discussed our plans for the upcoming semester's milestones.

Key Features and Challenges:

Our primary goal will be to develop a custom garbage collector for the MiniJava compiler. We will also conduct extensive tests to decipher the best-suited implementations for specific algorithms.

Throughout last semester, we have developed a strong understanding of the fundamentals of code compilation and language runtimes, and how to effectively integrate the two. We also investigated three garbage collection algorithms, although we only managed to implement two during the previous semester: refcount and marksweep. We will start off this semester by tackling the third method: the copying algorithm.

Copying GC is more involved than what we previously have done due to its reliance on low-level heap management. A big challenge this semester will be understanding what must be implemented to make copying GC possible. Additionally, this semester will focus more on testing and gathering/interpreting data, which we have not yet done as of this time. Planning such activities will require us to seriously consider which performance metrics are essential for our understanding.

Algorithms and Tools:

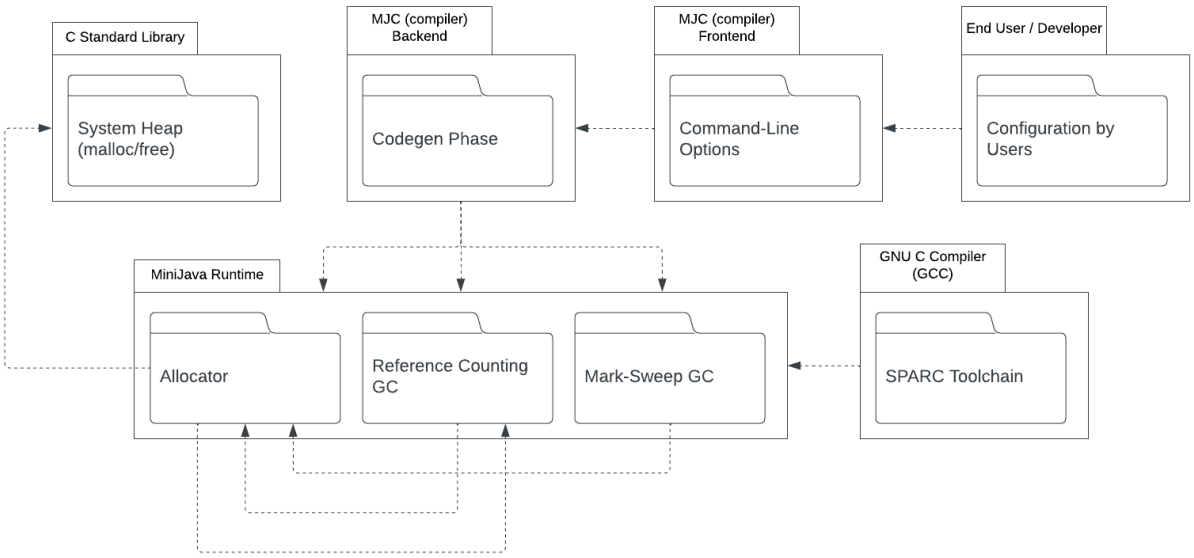
Our MiniJava compiler (which we have named "mjc") is written in Java, and the majority of its code was produced as part of the Compiler Theory course. It does rely on the support library provided by Appel for the course (packaged as "support.jar").

MJC originally had a minimal C runtime to allow for user memory allocation (from the compiled MiniJava code), and this runtime has since been extended to power our garbage collection algorithms. The integration between MJC and the C runtime is somewhat minimal, in that MJC

only places calls to C runtime functions in the assembly code output. The C runtime is compiled using the GNU C Compiler (“GCC”).

Our testing framework involves using the Jabberwocky system (“*Virtual Development Environment in a Box”* by Ian Orzel and Dylan McDougall) to quickly and easily set up a sandbox-like environment for the SPARC architecture, where we can natively run our binaries via the QEMU instance. The Jabberwocky SPARC environment also packages useful tools such as the build of GCC we use: *sparc-linux-gcc*, a cross-compiler specifically targeting the SPARC architecture.

System Design:



Evaluation:

Our evaluation will not be based on how well we performed our task as a whole, but how well each garbage collection method works with specific types of algorithms. It is important to specify which applications worked the best, and which applications had shortcomings, as this helps to demonstrate proficiency with the software we made. The system should always be reliable, and the majority of the algorithms are expected to have identical inputs and outputs. We will be writing test suites to test different “genres” of algorithms with different types of garbage collection methods, then determine the amount of time it takes for each algorithm to run while using varying amounts of memory.

Progress Summary:

Module/feature	% Complete	To Do
Reference Counting	100	N/A
Mark-Sweep	100	N/A
Copying	0	Design and implement
Generational	0	Design and implement
Final Bugfixes & Test Suites	0	Design and implement
Compiler flags/options for end-user	0	Design and implement

Milestones:

A pathway comprising of three significant milestones has been devised for this semester. Each milestone contains tasks aimed at gradually building up the project up to a fully functional system.

Milestone 4 (Feb 19):

1. Implement “copying” GC method
2. Write and execute tests for “copying” GC method
3. Allow GC configuration when compiling MiniJava programs
4. Run tests and gather metrics across the three GC methods (Refcount, Marksweep, Copying)

Milestone 5 (Mar 18):

1. Implement “generational” GC method
2. Write and execute tests for “generational” GC method
3. Conduct evaluation and analyze results
4. Create poster and ebook page for Senior Design Showcase

Milestone 6 (Apr 15):

1. Write test suites
2. Make debugging less verbose

3. Fix minor bugs before showcase
4. Test/demo of the entire system
5. Conduct evaluation and analyze results
6. Create user & developer manual(s)
7. Create demo video

Task Distribution Matrix (Milestone 4):

The task matrix delineates the division of responsibilities between the team members, ensuring a balanced workload and efficient progress.

Task	Trevor	Tyler
Implement heap system	0.8	0.2
Implement copying garbage collection	0.6	0.4
Design compiler flag/configuration user interface	0.2	0.8
Implement compiler flag/configuration functionality	0.8	0.2
Implement compiler flag for setting garbage collection method	1.0	0.0
Run tests across refcount/marksweep/copying garbage collection methods	0.0	1.0
Gather metrics for refcount/marksweep/copying garbage collection methods	0.0	1.0

Task Descriptions (Milestone 4):

Implement heap system: Our runtime currently relies on the C standard library for memory allocation operations (malloc/free). To implement copying garbage collection we need more

control over the heap, specifically for the defragmentation step of the GC. We cannot achieve this with malloc/free, so we will need to write a minimal wrapper over these functions.

Implement copying garbage collection: Our next garbage collection target is the “copying” method, which involves managing two independent heaps. During a GC cycle, all live allocations are copied over to the other (of the two) heaps, defragmenting the new heap in the process and leaving all unreachable allocations in the old heap (which can in turn be freed/overwritten).

Design compiler flag/configuration user interface: For testing purposes we need to add compiler flags and other methods of configuration. The codebase was not prepared for this throughout the Compiler Theory course so this will require a considerable amount of new system design. It should be user-friendly so that more flags can be added in the future for the end-user, such as configuring the garbage collector.

Implement compiler flag/configuration: After designing this extension of the compiler it will need to be implemented in the Java codebase, and potentially also in the C runtime if any features need to occur at or during MiniJava program execution.

Implement compiler flag for setting garbage collection method: Among other things, it will be very useful to have the ability to compile programs for a specific garbage collection method. This will greatly help testing, but will also be useful to enforce the optimal GC method for a given program.

Run tests across refcount/marksweep/copying garbage collection methods: Tests are the most important part of this project, and will give us the data which powers our conclusions. We have written many MiniJava test cases during compiler development, but we have not yet ran tests to collect more detailed data, such as the memory/speed cost of the garbage collector and its configurations.

Gather metrics for refcount/marksweep/copying garbage collection methods: After running tests we will need to compile them in a useful and intuitive way, such that we can draw meaningful conclusions from the data and understand how to optimize program execution by configuring the garbage collector.

Approval from Faculty Advisor:

"I have discussed with the team and approve this project plan. I will evaluate the progress and assign a grade for each of the three milestones." Signature:
_____ Date: _____