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 client is Dr. Ryan Stansifer (ryan@fit.edu), a compiler researcher who has a deep understanding of the MiniJava compiler and Java runtime.

This milestone we made considerable progress towards implementing the generational garbage collection algorithm. Additionally, we fixed the glaring problem in our copying GC implementation: the updating of existing references. Lastly, we also rewrote nearly all of the C runtime code to be more easily extendable, testable, and decouple core parts of the system. This was necessary to do with how our project scope adjusted over time.

Progress Matrix:

Task	**Progress**	**Tyler**	**Trevor**
1. Fix “copying” GC method	100%	0.3	0.7
2. Implement “generational” GC method	85%	0.5	0.5
3. Write and execute tests for “generational” GC method	70%	0.5	0.5
4. Conduct evaluation and analyze results	50%	0.0	1.0

5. Create poster for Senior Design Showcase	75%	0.8	0.2
6. Create ebook page for Senior Design Showcase	0%	-	-

Discussion:

1. *Fix "copying" GC method:* In the previous milestone, we implemented a form of the "copying" garbage collection algorithm. Evidently we did not write satisfactory test programs, because we did not notice that object references were not updated as part of the GC cycle. When a copying GC cycle occurs, live objects are copied to a new heap. As a result, references to those objects become "dangling" references and must be updated to reflect the new addresses. Thankfully, Dr. Chan pointed this out during our presentation, allowing us to catch this bug early before it cascaded into bigger problems during testing.

2. *Implement "generational" GC method*: The generational method of garbage collection is notably similar to that of the copying method. They both require separate heaps, as copying of objects and "flipping" of heaps (swapping the handles) must occur to ensure memory defragmentation. More specifically, generational garbage collection extends copying GC by categorizing objects into one of many "generations". These generations represent objects grouped by age, or the amount of garbage collection cycles which they have survived. By grouping objects in this way, the "stop-the-world" moments seen in these heavier-impact GC algorithms can be reduced.

Our implementation is very similar to our copying implementation, where we use multiple chunks of memory (referred to in previous milestones as "slabs") to represent generations into which the objects can be separated. At the moment we use two generations: generation "zero" and generation "one". There are still some issues to resolve, but the implementation is very close to completion.

When a GC cycle must occur to free memory, the lowest generation of objects is first to be attempted to be collected. All surviving objects move on to the next generation. If the *next* generation runs out of memory, the same process occurs there, and so on.

3. *Write and execute tests for "generational" GC method*: We do not yet have very many test cases for the generational method, as we spent the large majority of our time refactoring and redesigning existing systems. However, we do believe that this was time well spent and that the codebase is now considerably easier to add features to.

4. *Conduct evaluation and analyze results*: Currently we have our own internal test suites for the compiler and garbage collector, but regretfully we have not yet set up infrastructure for obtaining specific metrics regarding these tests. There is still some time left though, and we think addressing the code quality issues during this milestone was a better idea and will help us down the road.

5. *Create poster for Senior Design Showcase*: The poster has been mostly completed, and the only thing missing is the information about the test results and the conclusion. Our next milestone will focus on collecting data in order to have something interesting for the showcase, so the poster depends heavily on that data. The poster shows the algorithms we implemented, what they're used for, then it gives a brief overview of our design philosophy.

6. *Create ebook page for Senior Design Showcase*: No progress done on the ebook page.

Contributions:

Tyler Gutowski: I helped Trevor with rewriting the majority of our code, and also worked on generational garbage collection. As stated earlier, the implementation is not yet perfect, but I believe we have gotten the difficult parts out of the way. I also helped design test cases for both the generational method and some of our older implementations. Unfortunately we did not get around to collecting runtime metrics yet, but we have nearly everything we need to do so. Lastly, I created the majority of the Senior Design Showcase materials: our poster and e-book page.

Trevor Schiff: I spent the majority of my time tracking down bugs in the existing implementations. Specifically, Dr. Chan pointed out (after the previous milestone) that we had forgotten to update all existing references when copying objects between heaps. With this milestone's refactor of the MJC codebase, I was able to come up with a solution I am very happy with. It involves building a map of pointers to convert between the old/new heap addresses. By performing the same marking process seen in many GC methods, we can isolate all references to live objects and then overwrite them. This fixes all existing references and the programs work great. I also helped Tyler with some of the generational GC implementation while we redesigned the code to be more modular and less coupled. Some of the GC methods were previously very tangled together, and I can thankfully say that isn't a problem anymore.

Next Milestone:

Task	**Trevor**	**Tyler**
Finish generational GC implementation	70	30
Conduct evaluation and analyze results	50	50
Create user/developer manual	50	50
Create demo video	30	70

Discussion:

Finish generational GC implementation: We need to iron out any issues still remaining in the implementation and also allow for the number of generations to be configurable. This will be a very important setting for our testing.

Conduct evaluation and analyze results: The most important part of this project is what we can conclude/takeaway. As such, it is very important for us to analyze the data we collect, draw conclusions from said data, and evaluate the success of our project. Considering our lack of progress so far, we need to buckle down for this last milestone and make sure this gets done.

Create user/developer manual: We need to create a manual for users/developers who may be interested in using MJC to test and gather their own results regarding GC algorithms. We need to document how to use the compiler and its configurations, among other things. The developer manual will need to go more in depth with the system architecture, and implementation details.

Create demo video: As our last piece of Senior Design Showcase material, we will need to create a demo video showcasing our project. This will be difficult to do well, considering our software is a command-line tool.

Client Meetings:

See Faculty Advisor Meetings below

Client Feedback:

See Faculty Advisor Feedback below

Faculty Advisor Meetings:

In email.

Faculty Advisor Feedback:

In email.

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: _____

Evaluation by Faculty Advisor:

Tyler Gutowski	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Trevor Schiff	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Signature: _____ Date: _____