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 implemented the copying garbage collection algorithm, which requires us to fragment the heap into multiple “sub-heaps”. This was accomplished through our “memory slab” system discussed later.

Progress Matrix:

Task	Progress	Trevor	Tyler
1. Implement memory slab system	100%	0.7	0.3
2. Implement copying garbage collection	100%	0.4	0.6
3. Design compiler flag/configuration user interface	100%	0.2	0.8
4. Implement compiler flag/configuration functionality	100%	0.8	0.2

5. Implement compiler flag for setting garbage collection method	100%	1.0	0.0
6. Run tests across refcount/marksweep/copying garbage collection methods	0%	0.0	1.0
7. Gather metrics for refcount/marksweep/copying garbage collection methods	0%	0.0	1.0

Discussion:

Implement heap system: Our runtime relies on the C standard library for memory allocation operations (malloc/free). To implement copying garbage collection we needed more control over the heap, specifically for the defragmentation step of the GC. We cannot achieve this with malloc/free, so we needed to write a minimal wrapper over these functions.

We designed what we call memory “slabs”, which represent a chunk of memory that is known to be contiguous and that we have total control over. This is imperative to copying garbage collection because we need to perform memory copies between the to/from slabs (referred to as to/from “heaps” in copying GC algorithms).

Implement copying garbage collection: Our next garbage collection target was 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).

Through our memory slab system, this was relatively easy to accomplish. First, we determine “live” allocations through the exact same code we wrote for the mark-sweep GC. This is because it is exactly the “mark” phase of mark-sweep. Next, all marked allocations are copied from the “from” slab, to the “to” slab. In the process, this defragments the allocations as the “to” slab is

cleared before the copying occurs. Lastly, we free everything that still exists in the “from” slab, resulting in the “to” slab containing all memory that is said to be non-garbage. The “to” and “from” slabs must be swapped before the next garbage collection cycle.

Design compiler flag/configuration user interface: For testing purposes we needed to add compiler flags and other methods of configuration. The codebase was not prepared for this throughout the Compiler Theory course, so this required a considerable amount of new system design.

We modeled the design after modern compilers such as GCC, where you specify the compiler flags first and then the name of the source file. Below is an example of how our compiler (mjc) can be invoked at the command line:

```
java -jar compile.jar --verbose --gc=copying
```

Implement compiler flag/configuration: We implemented this functionality mostly in the compiler itself, as the runtime does not need to be aware of most compiler options. As mentioned above, these options can be configured at the command line when invoking the compiler. For settings which have an effect at runtime (such as the selected GC method), these settings are baked into the executable and the C runtime will see them.

Implement compiler flag for setting garbage collection method: It is very useful to have the ability to compile programs for a specific garbage collection method. This greatly simplified our testing workflow, and is why we wanted to tackle this over the course of this milestone. The garbage collection method is selected through the **--gc** flag at the command line and the choice is baked into the executable, where the C runtime will toggle its features based on the method. Below is our documentation on this compiler flag:

```
--gc=<type>      Set <type> as the garbage collection
method in the main function.
(None|Refcount|MarkSweep|Copying|Generational)
```

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. Unfortunately, the slab system was more complex than we had anticipated, and

we did not have enough time to partition for tests. However, we remain optimistic that we can get much testing done before the next milestone, as our next target algorithm (generational GC) relies heavily on what we have implemented here. We will have much more time in the future to perform testing.

Gather metrics for refcount/marksweep/copying garbage collection methods: After running tests we 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. As with the testing task above, we unfortunately did not have enough time to address this task.

Contributions:

Tyler Gutowski: Trevor and I had about an equal split of the copying GC implementation, but I focused on implementing the algorithm while he worked on the memory management tools we needed. I also extended the compiler user-interface for necessities such as compiler flags. These flags have proven to be very useful for testing our code quickly.

Trevor Schiff: I worked on the lower-level parts of this milestone, mostly the memory slab system. I wrote most of the code responsible for slabs and integrated them with the copying GC, while Tyler focused more on the actual copying algorithm. I also implemented the compiler flags with the specifications Tyler drafted.

Next Milestone:

Task	**Trevor**	**Tyler**
Implement “generational” GC method	50	50
Write and execute tests for “generational” GC method	50	50
Conduct evaluation and analyze results	50	50

Create poster and ebook page for Senior Design Showcase	50	50

Discussion:

Implement "generational" GC method: As stated earlier, the "generational" method of garbage collection is our next target. It relies heavily on copying GC, so our work during this milestone will be very useful. We expect generational GC to be a lighter workload than copying GC as it will be implemented at a higher level.

Write and execute tests for "generational" GC method: Our generational GC will need to be thoroughly tested as with all other GC methods, both to ensure correctness and to gather useful metrics that we will need to draw any meaningful conclusions.

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.

Create poster and ebook page for Senior Design Showcase: As we only have a few months and milestones remaining in this project, we will need to start preparing for the Senior Design Showcase. The first task we will need to complete is creating our display material: specifically, our poster and ebook page.

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: _____