Project Information:

Our project name is Heap Heap Hooray, and our project is the development of a garbage collector for the MiniJava compiler. This project requires 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.

Our primary goal for Milestone 3 was to implement the mark-and-sweep algorithm, and our secondary goal was to implement the copying algorithm. We completed our primary goal.

Progress Matrix:

Task	**Progress**	**Tyler**	**Trevor**
1. Implementation of root detection	100%	0.0	1.0
2.. Implementation of the Mark Phase algorithm	100%	0.5	0.5
3. Implementation of the Sweep Phase algorithm	100%	0.8	0.2
4. Mark-and-Sweep testing	100%	0.5	0.5

5. Dividing the heap into multiple parts for defragmentation	0%	0.0	0.0
6. Implementation of the Copying Algorithm	0%	0.0	0.0
7. Copying Algorithm testing	0%	0.0	0.0
8. Integration with Reference Counting	100%	0.5	0.5
9. Add parameters to run individual garbage collection algorithms	30%	0.3	0.0
10. Comparing metrics between different GC algorithms	0%	0.0	0.0

Discussion:

1. In order to detect roots, we needed to understand the layout of stack frames on the SPARC architecture, and utilize them to inspect CPU registers, stack locals, and other data that persists throughout the call chain.
2. The mark phase consists of marking all reachable objects. We can iterate through the stack records to determine the reachability of any object. Any object that we see on our stack-walk gets marked, and anything we don't see won't get marked.
3. The sweep phase consists of removing all non-marked objects from the heap. Because all live, heap-allocated objects are kept in the runtime list, we simply iterate through the

runtime list and free any objects that are not marked.

4. We want to be able to use the mark-and-sweep algorithm uniquely and in concurrence with the reference counting algorithm, so we need to write two different types of tests. For the individual tests, we can write them as we would with reference counting, because the only thing that will be freeing memory will be the mark-and-sweep algorithm. When we use it in concurrence, we need to write lots of cyclic references, because the reference counting algorithm won't free them.
5. The runtime relies on the C standard library's malloc() and free() functions to manage memory while interfacing with the operating system. This means that we cannot access the underlying "state" of the heap, and cannot make any guarantees about heap fragmentation. This is a big problem for sweeping GC, and for this reason we will need to develop our own memory pool system.
6. This was a secondary goal which requires #5 to be completed before beginning.
7. This was a secondary goal which required #6 to be completed before beginning.
8. We currently have mark-sweep running in concurrence with reference counting. Anything that is not caught by reference counting will be caught by mark-sweep at any stop-the-world instance.
9. We want to be able to run specific algorithms to ensure that we can collect vital metrics for individual algorithms. The software currently runs both reference counting, and then performs a mark-sweep cycle when there is no more memory available, but this is sub-optimal. We need to decide how we will make the garbage collector configurable.
10. The main difference between the mark-and-sweep and the reference counting is that cyclic references will be resolved, but with the cost of a stop-the-world whenever memory cannot be allocated. We want to be able to compare the different types of algorithms overall, so we want to quantify the vital metrics that we specified earlier.

*Red indicates that a milestone is not completed.

Contributions:

Tyler Gutowski: Wrote the majority of the marking algorithm, and portions of the sweeping algorithm. Developed some new MiniJava test cases for the mark-sweep GC algorithms.

Trevor Schiff: Wrote the stack-walking portion of the marking algorithm, and portions of the sweeping algorithm. Developed some new MiniJava test cases for the mark-sweep GC algorithms.

Next Milestone:

Task	Tyler	Trevor
1. Dividing the heap into multiple parts for defragmentation	0.0	1.0
2. Implementation of the	0.5	0.5

Copying Algorithm		
3. Copying Algorithm testing	0.5	0.5
4. Comparing metrics between different GC algorithms	1.0	0.0
5. Continue adding parameters to run individual garbage collection algorithms	0.8	0.2

Discussion:

1. We recognize that fragmentation is a significant issue in garbage collection, particularly in long-running applications. To address this, we've started working on dividing the heap into multiple regions. This approach aims to localize the fragmentation, making it easier to manage. Each region will handle a specific range of object sizes, allowing us to optimize garbage collection strategies for each size category. However, this raises challenges in managing cross-region references and ensuring efficient memory use without excessive overhead.
2. The copying algorithm is essential for compacting memory and reducing fragmentation. Our implementation involves two primary heap areas, the 'from-space' and 'to-space'. Live objects are copied from the 'from-space' to the 'to-space', which helps in compacting the memory. This process involves updating all pointers to the moved objects, a task that requires precise tracking of object references. We're exploring efficient ways to update these references with minimal performance impact.
3. Rigorous testing of the copying algorithm is crucial. We are focusing on verifying the integrity of object data post-copying, ensuring that all references point to the correct, relocated objects. This includes stress-testing the algorithm with a variety of object graphs, including those with complex interconnections and cyclic references. Our tests also need to assess the performance implications of the algorithm, particularly in terms of memory usage and the time taken for the copying process.
4. A key goal is to compare various garbage collection algorithms, including mark-and-sweep, reference counting, and now the copying algorithm. We aim to measure and analyze critical metrics such as pause time, throughput, memory overhead, and the efficiency of memory usage. This comparison is not only about raw performance numbers but also about understanding the trade-offs of each approach, like how well they handle cyclic references or their behavior in memory-constrained environments. We believe this research will be the main talking-point during the presentations of our board.
5. Enhancing the configurability of our garbage collection system remains a priority. We are working on adding more parameters to allow users to select specific algorithms based on

their needs. Cerain pieces of software will run more efficiently with specific garbage collection algorithms. This includes parameters to control the frequency of garbage collection cycles, set thresholds for memory usage, and enable or disable certain algorithms.

Client Meetings:

See Faculty Advisor Meetings below

Client Feedback:

See Faculty Advisor Feedback below

Faculty Advisor Meetings:

We met with Dr. Stansifer on November 13th to discuss our ideas for Milestone 3, and he provided some feedback regarding heap fragmentation and memory pools, which are important concepts required for copying GC. We did not get to implement these concepts during this milestone; however, they will be our primary goal in the next milestone. We also met with him on November 27th to discuss our progress for this milestone.

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: 11/27/2023

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: _____