Heap Heap Hooray: Improving Memory Management

Tyler Gutowski, Trevor Schiff,
Dr. Ryan Stansifer (client)

Team

- Tyler Gutowski (member)
- Trevor Schiff (member)
- Dr. Ryan Stansifer (faculty advisor, client)

Goals

- Primary goal:
  - Develop runtime garbage collector (GC)
  - Integrate with MiniJava compiler developed as part of Compiler Theory course
    - “MiniJava” refers to a simple, but non-trivial subset of Java
- Secondary goal:
  - Determine optimal garbage collection configuration based on algorithms exhibited in source code
    - Explore memory and execution overhead

Motivation

- MiniJava runtime does not offer automatic memory management
  - GC is not a required part of the Compiler Theory course
  - “New” operator exists, but user is responsible for lifetime of allocation
- MiniJava is a subset of Java, so memory cannot be manually freed
  - No “delete” operator exists
  - Without GC, all heap allocations are permanent
  - Losing reference means losing memory block forever

Key Features

- Automated memory management in MiniJava
  - “Garbage collection”
- No effort required by the user
  - GC will be part of compiler runtime
- Verbose debugging and graphics
  - Current GCs are very abstracted

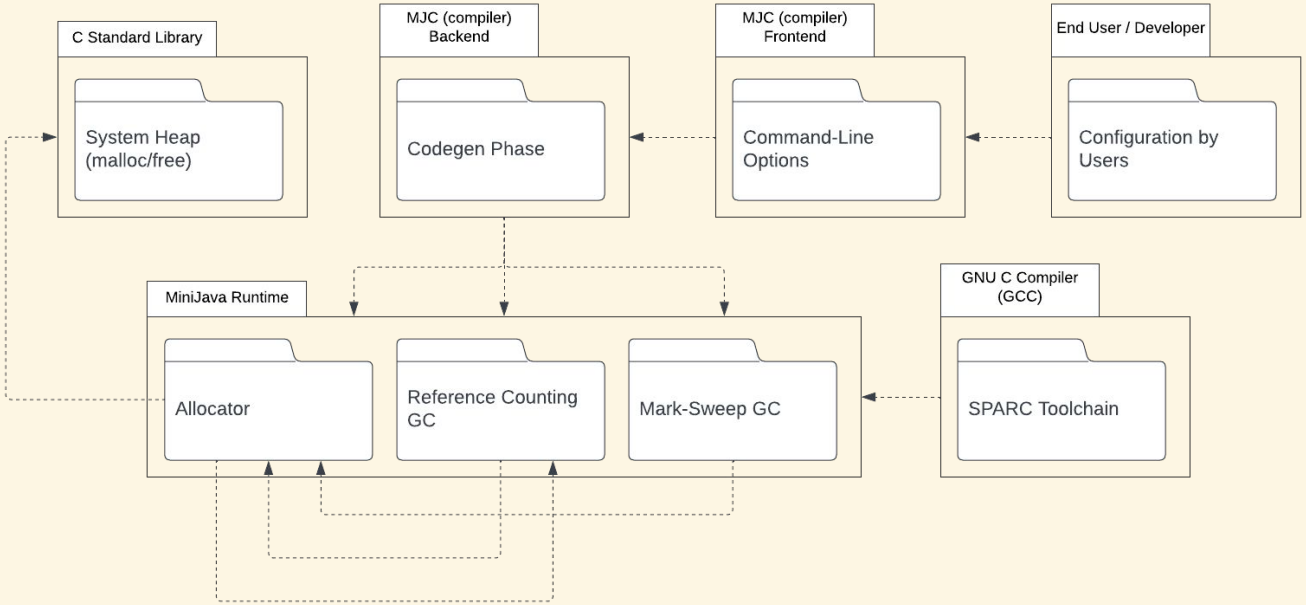
Technical Challenges

- ~~Understand and implement GC algorithm(s)~~
  - ~~Reference Counting~~
  - ~~Mark-Sweep~~
- ~~Learn how to integrate GC with MiniJava runtime~~
- Understand requirements for implementation of “copying” GC
  - More involved with the heap than previous algorithms
- Determine data/algorithm set for GC performance testing
  - Consider which metrics are essential for understanding

Algorithms and Tools

- MiniJava compiler (named “mjc”)
  - Developed during Compiler Theory course
  - Extended with garbage collection throughout this project
  - Written in Java
- Compiler runtime support
  - Runtime support for MiniJava programs
  - Handles memory allocation, garbage collection
  - Written in C
- Jabberwocky (“*Virtual Development Environment in a Box*”)
  - SPARC environment similar to a Docker container
  - Contains useful SPARC tools such as GCC, GDB, etc.
  - All GC testing occurs within this container

System Design



Evaluation

- Ensure GC correctness/reliability
  - Before evaluating performance by algorithm, the GC must first work correctly
- Compare GC performance by source code algorithms
  - Gather metrics using MiniJava source code containing different algorithms
- Determine optimal GC configuration for source code
  - Based on algorithms exhibited in said source code
  - In an abstract sense, should be applicable outside MiniJava/mjc

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

Milestone 4

1. Implement “copying” GC method
   - Involves managing two heaps
   - GC cycles copy data between heaps to defragment live allocations, and prepares unreachable allocations to be freed
2. Write and execute tests for copying GC
   - MiniJava test cases to test GC correctness
3. Allow GC configuration when compiling MiniJava programs
   - Add MJC compiler flags and other configurations
   - Requires considerable further system design
4. Run tests and gather metrics across the three GC methods
   - Ref-count, Mark-sweep, Copying
   - Collect more detailed data, such as memory/execution overhead

Milestone 5

1. Implement “generational” GC method
   - Implementation somewhat built on top of copying GC
   - Popularized in environments such as the Java Virtual Machine
2. Write and execute tests for generational GC
   - MiniJava test cases to test GC correctness
3. Conduct evaluation and analyze results
4. Create poster and e-book page for Senior Design Showcase

Milestone 6

1. Write test suites
2. Make debugging less verbose
3. Last debugging before showcase
4. Test/demo entire system
5. Conduct last evaluation and analyze results
6. Create user & developer manual(s)
7. Create demo video