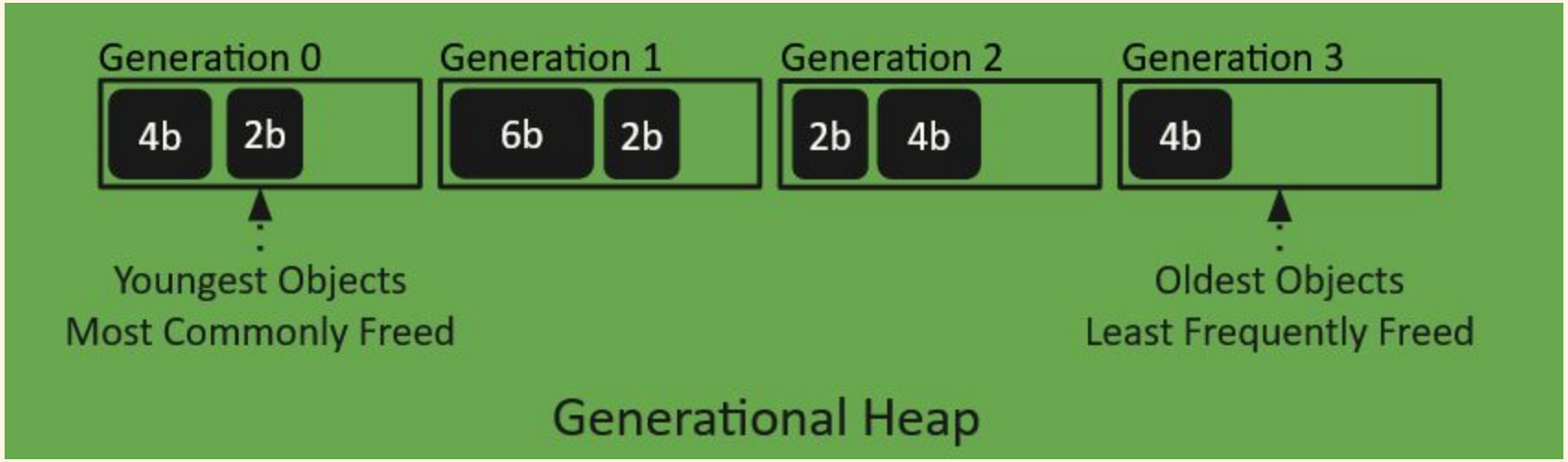
Heap Heap Hooray

Milestone 6

Finished Generational GC

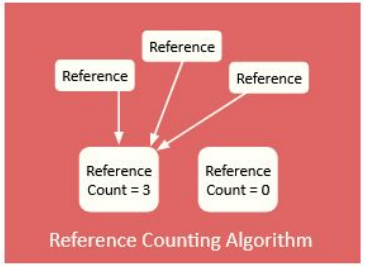


NORTHROP GRUMMAN
ENGINEERING & SCIENCE
STUDENT DESIGN SHOWCASE
FLORIDA TECH

Heap Heap Hooray: Reinventing the Garbage Collector

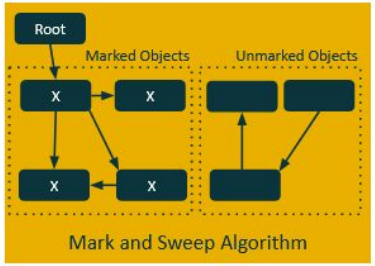
Trevor Schiff, Tyler Gutowski

Faculty Advisor(s): Ryan Stansifer, Dept. of Electrical Engineering and Computer Science, Florida Institute of Technology



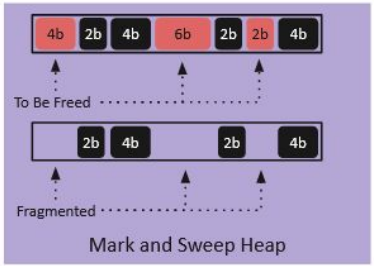
Motivation/Goal

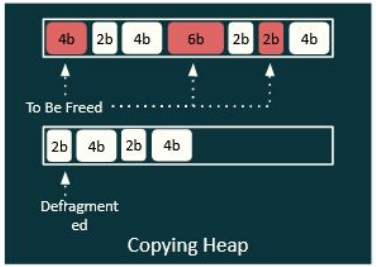
- Demystify garbage collection (GC)
- Integrate with MiniJava compiler from the *Compiler Theory* course, extended with C runtime
- Demonstrate how each newly implemented GC algorithm fixes the previously implemented algorithm's downfall.
- Provide easy to use/extend environment for other developers designing brand new GC algorithms



Implementation

- Runtime
  - Written in C, compiled with GCC for SPARC.
  - Linked with MiniJava programs.
- Heap
  - C standard library (malloc/free)
  - Chunk heap, used for the copying and generational algorithms.
- Algorithms
  - Reference Counting, Mark-and-sweep, Copying and Generational





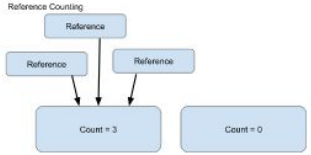
Features

- Several garbage collection algorithms.
- De-abstracts garbage collectors.
- Implemented parameterization for:
  - Garbage Collection type
  - Heap type
  - Heap size
- Verbose memory usage and debugging

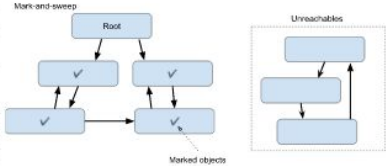
Project Name **Heap Heap Hooray**
Team Lead: Trevor Schiff
Team Member(s): Trevor Schiff, Tyler Gutowski
Faculty Advisor(s): Dr. Ryan Stansifer, Dept. Of Electrical Engineering and Computer Science

Heap Heap Hooray is an advanced, yet easy to understand memory management system for the MiniJava compiler, encompassing reference counting, mark-sweep, copying, and generational garbage collection techniques. The system aims to optimize memory usage, minimize fragmentation, enhance performance, and be fully customizable with parameters.

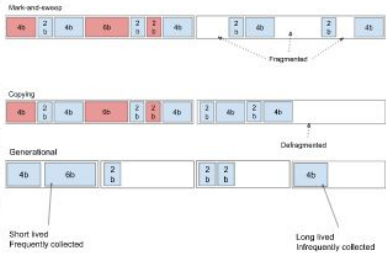
Reference counting is a simple garbage collection method that relies on objects maintaining an atomic count of existing references to themselves. The reference count is updated when an object alias is created or leaves scope, and the object is safe to free when this count reaches zero.



Mark-and-sweep is a garbage collection algorithm that finds and marks reachable objects via the program stack. Any items that have not been marked are unreachable (garbage), and can be freed. The algorithm sweeps through the heap, freeing anything that hasn't been marked. The issue with this algorithm is that the memory becomes fragmented, which can detract from performance.



Copying is a garbage collection algorithm that acts similarly to mark-and-sweep. Copying solves the fragmentation issue by using two heaps. During the sweep phase, everything that is marked is moved to the opposite heap, defragmenting the layout of these objects in the process. Lastly, unmarked objects are freed, emptying the first heap.



Generational is a garbage collection algorithm built on the copying algorithm, but with more heaps. Objects are categorized into different "generations" based on their age. The garbage collection efforts primarily target the younger generations, where short-lived objects are more prevalent, reducing the frequency and duration of garbage collection pauses.

Table of Contents

```
void refcount_ref_decr(GC* gc, Object* obj);
   Decrements an object's reference count. If the counter reaches zero, the
   object is freed.
```

marksweep_gc.c/.h

This file contains the mark-sweep garbage collector implementation. If you would like to use these functions, please use the following public API in marksweep.h:

```
GC* marksweep_create(void);
   Create a mark-sweep garbage collector.

void marksweep_destroy(GC* gc);
   Destroy this garbage collector.

void marksweep_collect(GC* gc);
   Perform mark-and-sweep garbage collection cycle.

void marksweep_stack_push(GC* gc, void* frame, u32 size);
   Pushes a new stack frame to traverse later using garbage collection.

void marksweep_pop_stack(GC* gc);
   Pops the current stack frame.
```

copying_gc.c/.h

This file contains the copying garbage collector implementation. If you would like to use these functions, please use the following public API in copying.h:

```
GC* copying_create(void);
   Create a copying garbage collector.

void copying_destroy(GC* gc);
   Destroy this garbage collector.

void copying_collect(GC* gc);
   Perform a copying garbage collection cycle.

void copying_stack_push(GC* gc, void* frame, u32 size);
   Push a new active stack frame to traverse later during garbage collection.
```

Jabberwocky

Install Poetry from python-poetry.org.

Install Jabberwocky from https://github.com/Kippiii/jabberwocky-container-manager.

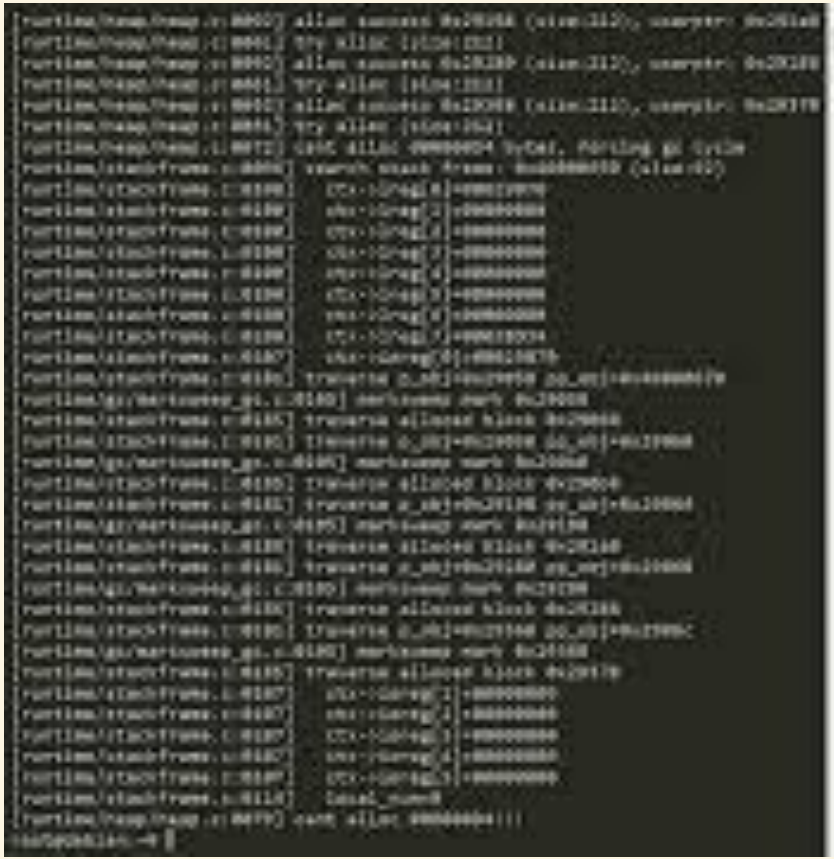
Navigate to the installed Jabberwocky folder.

```
poetry install
poetry shell

python build.py
```

MJC Usage

```
Usage: {compiler-jar} [option(s)] input-file
Options:
--help              Display this information again.
--verbose           Log verbose compiler information to "/verbose.txt".
--gc=<type>         Set <type> as the program's garbage collection method.
                    (None|RefCount|MarkSweep|Copying|Generational)
--heap=<type>       Set <type> as the program's heap type.
                    (Stl|Chunk|Buddy)
```



Executing a fragmentation test using the mark-and-sweep GC.

Although enough memory exists overall, it is not contiguous. Fragmentation results in the program terminating.