

In this assignment you will implement garbage collection for the MiniJava compiler. The project includes the two programming projects specified at the end of Chapter 13 in the Appel book. This assignment can be done in groups of one, two, or three. Larger groups will be expected to show more experimental results in terms of how their garbage collection is working.

1 Motivation

Garbage collection is becoming quite common in modern day programming languages. By implementing a simple garbage collector, you will learn how garbage collection works and gain insight into how garbage collection can affect application performance.

2 The Assignment

Your assignment is to extend the MiniJava compiler so that it generates record descriptors and pointer maps, and to implement a *mark-sweep* garbage collector in C.

In the MiniJava compiler, you should add a `GarbageCollection` class to the `Mips` package. It should include the following functionality:

- **Label `funcCall(MethodInformation callerInfo)`** internally creates a pointer map for the given call and returns a label for the return address right after the function call. The `GarbageCollection` class should store all generated pointer maps until queried for all such data at once.
- **Tree.Exp `heapAllocCall(ClassInformation classInfo)`** should return the appropriate IR tree for the call to `halloc_gc()`. Internally it should also create a descriptor string for the given class. The `GarbageCollection` class should store all generated descriptors until queried for all such data at once. Keep in mind that a heap allocation call is like other function calls in that it should have a return label associated with it and a pointer map. NOTE: The parameters to `halloc_gc()` should be listed in reverse so that the parameters are pushed in the order expected by functions generated by the C- compiler.
- **Tree.Exp `heapAllocCall(Tree.Exp length)`** should return the appropriate IR tree for the call to `halloc_gc` for an array where the length is specified with the given `Tree.Exp`. Internally it should also create a descriptor string for the given array. You can assume that all arrays in MiniJava only contain integers.

- **String mips_data()** returns a string containing all of the descriptors and pointer maps that have been generated.
- **String gc_implement()** should read in a file that contains all of the MIPS code for implementing garbage collection and return the contents of that file as a string.

The following interface should be implemented in C and compiled to MIPS using the Wisconsin C-compiler (<http://www.cs.wisc.edu/~lenz/compiler.html>).

- **init_gc(struct ptrmap * last, int heapsize)** will traverse all of the pointer maps generated by the compiler and put them into a global data structure that can access each pointer map based on the return address it is associated with. It will also set the global variable that indicates the allocatable heap size.
- **halloc_gc(int * return_address, struct descriptor class_descr)** will replace the current heap allocation routine. It will call garbage collection if necessary.

It should be possible to indicate the size of the heap in bytes available for dynamic memory allocation. For example,

```
% java -jar MiniJavaCompiler.jar --heap=64 file.java
```

With the suggested design, it will be difficult to turn garbage collection on and off. You should make a branch in subversion for the garbage collecting version of the MiniJava compiler.

NOTE: For this project, your compiler should not print all debug information. Instead the compiler should just print information about dynamic memory allocations and garbage collecting. For example,

```
Allocating an instance of class Foo at line XX.
Failed attempt to allocate an instance of class Bar at line YY.
***** Starting garbage collection
    Marked ## bytes.
    Put ## bytes on the freelist.
Allocating an instance of class Bar at line YY.
...
```

Your group will need to create a test input programs and run those test input programs using varying amounts of available storage.

- The program should gracefully fail when an allocation is requested and there is no more allocatable space left.

- The `alloc_gc()` function should call garbage collection, if it is not possible to allocate the requested space.
- You should have test cases that create a lot of garbage and those that do not.

Feel free to share such test cases with other groups in the class, but each group must submit their own test cases.

Graph the number of times garbage collection occurs versus heap size for some test cases where garbage collection occurs at least once for the largest heap size: one test case if there is one person in your group, two if there are two people, etc. Explain the results.

3 Getting Started

1. You can use your current version of the MiniJava compiler or start from the version that was sent out for project 3. To make things simpler, the MiniJava compiler should allocate local variables to the stack and just use a spill all register allocation.
2. The Wisconsin C- compiler has been compiled on the CS linux machines. Copy the jar file from the following location:

```
/s/bach/b/class/cs553/GCTesting/cmm.jar
```

NOTE: The `cmm` compiler generates code that expects the parameters to a function to be pushed onto the stack in order. This is opposite of the ordering used within the MiniJava compiler. When generating the calls to `init_gc()` and `halloc_gc()` make sure the push the parameters onto the stack in order.

4 Your Report

The report is an essential part of your completed assignment. Use it to describe your mark and sweep algorithms, how you implemented those algorithms, your assumptions, difficulties, insights, and results. Organize and present your document as if it were the only basis for your assignment's grade. The format of your writeup is up to you, but it should minimally include the following:

- Introduce the main goals of the project and in a couple of sentences summarize what you have accomplished.
- Briefly describe the mark and sweep algorithms you chose to implement. Motivate your selection.

- Present and explain graphs that study the number of times garbage collection is called based on the heap size.
- How did you test your garbage collection implementation?
- What problems did you encounter while implementing mark and sweep garbage collection? If you knew someone who was just about to start work on this assignment, what advice would you give them?
- What, if any, outside sources did you use (e.g., articles, books, other students)? This is particularly important. It's OK to look at books and articles and speak with your professor and fellow students (although sharing code and working together is strictly forbidden), but as with any scientific document, you should always cite your references and collaborations. You can either cite collaborations in footnotes or in a separate Acknowledgment section.
- Extra Credit (10 points): Provide a detailed, incremental schedule for the implementation. Indicate what belonged to each piece you implemented and how you tested that piece. For full points, you will need to break the project up into at least 5 pieces.

There's no exact number of pages you should write, but if you've got between four and six then you're in the right ballpark.

5 Hints for doing the assignment

You should start this assignment early!! You only have three weeks to complete this assignment, and one of those weeks is Thanksgiving break. I recommend spending a couple hours a day on the project starting now. Start writing your report as you are planning the implementation. A well-written report with some missing implementation guarantees a much higher grade than a poorly written report with all the implementation.

I can look at your code and help point you in the right direction, but the amount of help I can give may be inversely proportional to the amount of time until the due date. By no means should you spend several hours trying to figure out a weird bug; consult me for help. When e-mailing me about the project, send a copy of the relevant section of your code (not as an attachment; send it as text pasted into your message). Give a good description of the problem including information about the stack in a debugger. Also, indicate possible solutions you are considering.

5.1 Extensions Needed in MiniJava Compiler

Notice that both the `init_gc()` and `halloc_gc()` functions take a label as a parameter. Specifically, `init_gc()` will want the label for the last pointer map for its first parameter. `halloc_gc()` requires the label for the class or array descriptor of what is being allocated.

- Extend `structures/ClassInformation` with a method that generates an ordered list of field names. These field names can then be used by `GarbageCollection.heapAllocCall()` to query the `ClassInformation` for field information and create a descriptor.
- In `Mips/Codegen.java`, `munchExpNAME()` needs implemented so that MIPS code which loads the address of the label is generated.

```
lda $t0, Label
```

Also, `munchExp()` should be modified to call `munchExpNAME()` when appropriate.

- Modify the `outANewExp` and `outANewArrayExp` in `ast_walkers/Translate.java` so that they use `GarbageCollection.heapAllocCall()`.
- Modify the `Tree.ExpCALL` data structure so that it has a field for a return address label.
- Modify the `outACallExp()` method in `ast_walkser/Translate.java` so that it calls `GarbageCollection.funcCall()`.
- Modify `munchExpCall` in `Codegen.java` so that the return address label is generated right after the call.
- Have `Mips/MipsFrame.programTail()` call `GarbageCollection.mips_data()` and `gc.implement()`, and append the string to the returned program tail.
- Modify `Mips.procEntryExit3()` so that it generates a call to `init_gc()` at the top of main.

5.2 Working with the Wisconsin C- Compiler

- Read their documentation. There isn't much, but it will help your understanding.
- For some reason their compiler allocates 16 bytes per field in a class, but accesses fields at 4 byte boundaries. This should not cause any problems, since the pointer map structure instances will be generated by the MiniJava compiler and read by functions generated by the `cmm` compiler.
- The `cmm` compiler will only compile a program that has a main. Therefore, you will need to compile your program with an empty main and then remove that main from the `.s` file that is read in with the MiniJava compiler.
- The C- language does not have handy data structures like maps. I recommend using lists. The lookup will be slow, but you should be able to get it to work.

6 What to turn in

Turn in a hard copy of the report and email a copy in pdf format to mstrout@cs.colostate.edu.

Create a jar file that includes all of the bytecode files, the source files, your test cases, and your benchmarks. The jar file should also include a README that gives specific command-lines for running the jar file on your provided test cases.

7 Due date

This assignment is due Friday November 27th, at **2:10pm** and is worth 10% of your final grade. Late assignments will be penalized 10% per day.