Ian Myers

CPSC 326

05/09/23

Video link: https://youtu.be/RG5ad9VLLYs

My final project consisted of the creation of a delete operator. This delete operator is able to deallocate memory from struct instances and arrays. This allows the memory to be reused. Anyone coding in mypl can create an array or struct, then, when done with it, can delete it and reuse the memory. This keeps the memory clean. One example is if a coder created an array in a for loop then, before leaving the for loop the coder can delete the array and deallocate the memory. The implementation for this required changing every step of mypl. First, I needed to add the delete_struct and delete_array to the lexer as special words so they would be picked up as tokens. Once they were put in as tokens I moved to the parser. I created two different statements, one for each new keyword. In those statements I ate the token then accepted the expression. The expression followed the normal path without any changes. Once the parser was done I created statements in the semantic checker. These statements made sure that if the delete_struct word was called, that there was a struct being given, and if the delete_array word was called, that there was an array being given. Once the semantic checker was complete I changed the vm. In the vm, I created two delete opcodes and functions: one for struct and one for array. The functions were almost the same as another. The functions involved getting the object id from the stack and calling erase(oid) on the struct/array heap. The heap function, erase, removes all values in the bucket of the given index. Once this was done I added two statements to the code generator, where it accepted the expression and pushed the proper delete opcode onto the stack. The original goal of creating a delete operator that deallocated memory and removed arrays/structs. Along with the original goal, I added checks in the get field and index making sure an error occurred when looking to get a struct/array that does not exist. I was unable, however, to make it so that because of the deletion, the code can run faster. If I had

additional time I would make it possible to use just the word delete on either a struct or array rather than having two different words. The way mypl is written makes it difficult to know whether to delete from the struct or array heap at the time of putting the opcode onto the stack. I also would have wanted to find a way to make delete cause the program to run faster. For my testing I created a new delete_tests gtest file. In this I made tests for the lexer and token all the way to the code generator. In the lexer tests, I made sure my keywords worked. In the ASTparser tests I made sure the code was following the right statement and caused no errors. In the semantic checker tests I made sure it would cause an error if a non struct or non array was asked to be deleted. In the vm tests I made sure the struct or array, once deleted, were inaccessible and checked that the size of the heap was reduced by one when the delete operator was called. In the code generator tests, I checked that a normal piece of code running the delete operator would cause no errors. Finally, I wrote programs that showed the OS running out of memory and crashing when not using the delete operator, then running fine with the operator in place. To run the code properly follow these steps. Git clone. Once cloned cmake . and that show set up all tests and code. From there all gtests can be run. To see the crashing example files run first reduce your memory capacity with these commands:

Ulimit -v 8192

Ulimit -m 8192

The memory will reset once the bash terminal is closed. From there comment and uncomment the delete operator line and watch it crash or not crash.