An Interpreter for a Sublanguage of C

Project Information

Ciaran Gruber – A0269276M

Project Repository available at: https://github.com/CiaranGruber/cs4215-project

Scope

The primary scope for the project was to create a sublanguage that would cover a number of use cases within C. In order to do this, the following requirements were set out to be included:

- Specification to be developed and adhered as closely as possible to the C17 ballot available at: https://web.archive.org/web/20181230041359if_/http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf
- Type checking of variables by ensuring incompatible types are not assigned to each other and corrupting data.
- Correct type casting as expected when performing operations with variables such as assignment or returning from functions.
- The use of any primary types consisting of: Char; Short; Int; Long, Functions, Pointers of any type
    - These variables should be able to have memory.
- Ability to use minimal type qualifiers such as const to prevent editing of variables correctly.
- Memory allocation on the Heap such that values can be allocated and deallocated with complete deallocation of memory excluding fragmentation of empty pointers.
- All “permanent” memory should be stored in a single place in the memory in order to ensure JavaScript is not required to perform garbage collection on said objects.
    - This excludes working memory which is used for performing the constructs within the language or used for viewing the memory.
- Ensuring memory which is in use by the evaluator as meta-data can not be changed by the programmer to ensure program correctness and reduce bugs.
- A basic visualisation of the heap used to show the physical bytes occupied as well as the protection status of each byte.
- Complete documentation within the program with descriptions of the memory format shown with each relevant class.

Progression of Objectives

Unfortunately, the main evaluator was not complete for the project, however the main components of the language were developed such that the evaluation can be simulated using code. These examples can be seen in the Jest test cases. In full, the objectives met (programmatically) included:

- Parsing files/code in C using Antlr4 into the CompilationUnitContext class.
- Correct type casting for all assignments and returning of functions.
- Ability to run built-in functions specified using Function types.
- Pointer representations of every potential type including checks to ensure that floats (as according to spec) cannot be cast.

- Entering of call-frames and block-frames with call-frames automatically returning the top-most value from the stash (if available) onto the stash of the previous frame.
- Correct manipulation of pointer references and dereferences within the stash.
- Calling of functions using pointers or functions retrieved from the stash.
    - These are retrieved using the FunctionManager JavaScript object which maps a given 'key' to the function to be run.
- Correct allocation and deallocation of the C Stack and the Heap
- Small optimisations made to the deallocation of the Heap as well as a way to manually defragment consecutive empty spaces in the heap (this does not perform any form of garbage collection such as stop-and-copy, etc, and is not automatic)
- Partial progress of the evaluator has been made such that the evaluator can process instructions, however the full set of instructions has not been included.

Incomplete sections include:

- Completion of the main evaluator bridging the gap between the parsed output and the visible running of the program
- Function Handling for functions defined by the user of the program.

Tests

The tests available are in the form of Jest tests. These can be run using 'npm run test,' and cover most use cases of how manipulation of the C memory can occur. The primary tests of interest include:

- Within 'Environment.test.ts'
    - Storing and retrieving variables
        - This tests the ability to store and retrieve variables in the environment from the same scope.
    - Retrieving variables from parent scope
        - This tests retrieving values from a parent scope such as the built-in environment after a variable is not present in the current environment.
- Within 'CValue.test.ts'
    - Referencing and dereferencing a value
        - Tests the referencing and dereferencing of values by showing the ability to create pointers and retrieve the value it points to
    - Setting the value of a constant
        - Tests to ensure the program throws an error when attempting to set the value of a constant
    - Calling a function
        - This tests the ability to call a function that has been defined within a program and allocated to a specific variable name
    - Casting to a smaller value
        - Shows the effects of casting values to smaller spaces and what happens during an overflow (such as Int->Short)
    - Casting to a larger value
        - This tests the effects of casting values to a larger value (such as Short->Int)
- Within 'CMemory.test.ts'
    - Allocating a value on the heap

        - Tests the ability to allocate values within the heap and retrieve the middle data.
    - Freeing values in the heap with basic merge
        - This shows the ability for the heap to completely recover after allocating and deallocating variables in a specific order.
    - Freeing values in the heap with complex merge
        - This is used to show the ability for the memory to merge heap values such that a fragmented heap makes the best use of its internal empty spaces.
    - Defragmenting an empty but fragmented heap
        - This tests the ability for the memory to perform a manual defragmentation if necessary.

Additional tests show the properties of the RestrictedHeap instance which the CMemory is based upon showing its ability to protect values from edits except where it is specifically told to overwrite any protected attributes.

Build Instructions

The project repository can be found at: https://github.com/CiaranGruber/cs4215-project

In here, the build instructions have been described in full as well as the relevant programs.