REPORT

Shrinivas Khiste 19CS30043
Rupinder Goyal 19CS10050

A. What is the structure of your internal page table? Why?

Our page table is made of two parts:

a. An array indexed by the local address that stores the physical address of the variable. This is the classic page Table.

b. Then we also have stored the list of Variables that have been declared till now again indexed by their local address. This list contains the details of Variables including their name, type, size, local address, array length (1 if it is not an array and the array size if it is) and a flag that marks the variables to be cleaned by the garbage collector. This is like the symbol table.

The first part is necessary to access the memory locations of the Variables and modify and read their values as user will give the logical address of the variables

The second part is necessary to know any information about the variable as and when necessary. For example the type is needed for type checking, then the size is when we are freeing and allocating space to the variable and so on.

We have indexed them by the local address so that we have an O(1) access to all information regarding the variable.

—---

B. What additional data structures functions are used in your library? Describe all with justifications.

Additional Data Structures:

1. Stack: We have implemented a stack to maintain the variables that are currently in scope. Variables are pushed into this stack when they get in scope and are popped from this stack when their scope ends. The Stack stores only pointers to the Variables to avoid extra memory usage

2. Physical Address to Local Address Mapping: We have implemented a mapping between the physical address and local address too. This is used when we are running compaction as we have the physical address and need to update it and save it on the page Table and variable list at the local address index.

3. Variable Class: This is made to store all the details of the variable that is its name, type, size, local Address, array length (1 if it is not an array and the array size if it is) and a flag that marks the variables to be cleaned by garbage collector.

4. Medium Int: We have implemented a class for medium int that defines a variable of 3 bytes size and we have defined an appropriate constructor and a function to convert it into an integer

Additional Functions:

1. Functions in Variable Class:
    a. isValid() to check if the Variable is valid given the type and name (limit the size)
    b. getSizeFromType() to get the size occupied by the Variable in terms of number of blocks of 4 bytes

2. We have implemented standard functions of stack

3. typeCheck(): To check the type of a given Variable given its local Address by matching it with the second argument

4. addToVar() and multToVar(): to add and multiply a value to an integer variable given its local address

5. getValueVar(): To get the value stored at a given local Address. This is defined for all the variable types

6. getValueArr(): To get the value stored at a particular index of an array given the local address. This is defined for all variable types

5. addToArr() and multToArr(): To add and multiply a value to an integer at a given index of an array given its local address

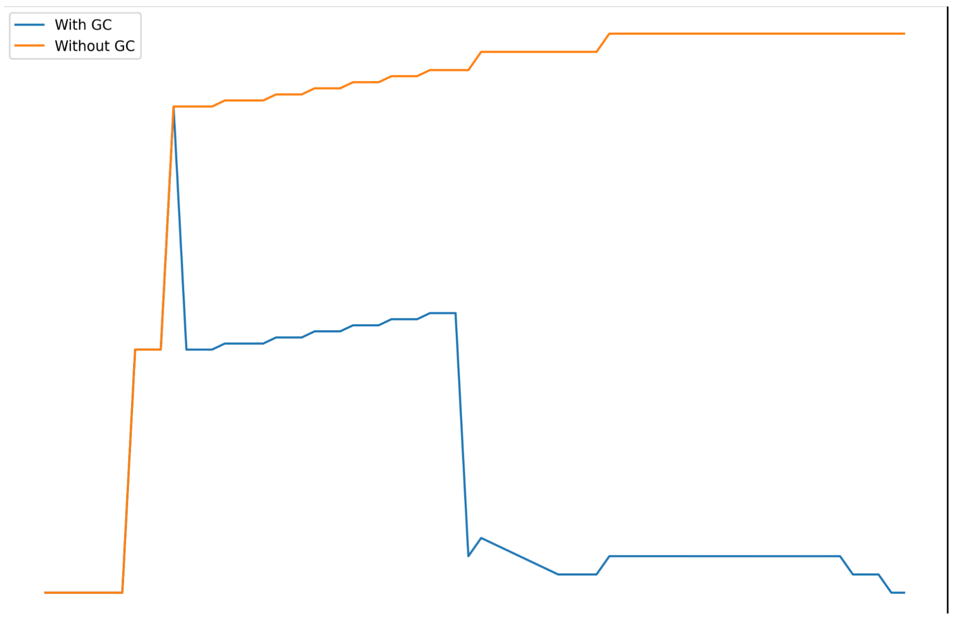
6. functionStart(): Run at the start of any function to push a NULL pointer in the variable stack to mark the limit till which we pop the stack

7. EndScope(): Run at the end of any function. This pops all the variables from the stack till a NULL pointer is found. The variables are also marked to be freed

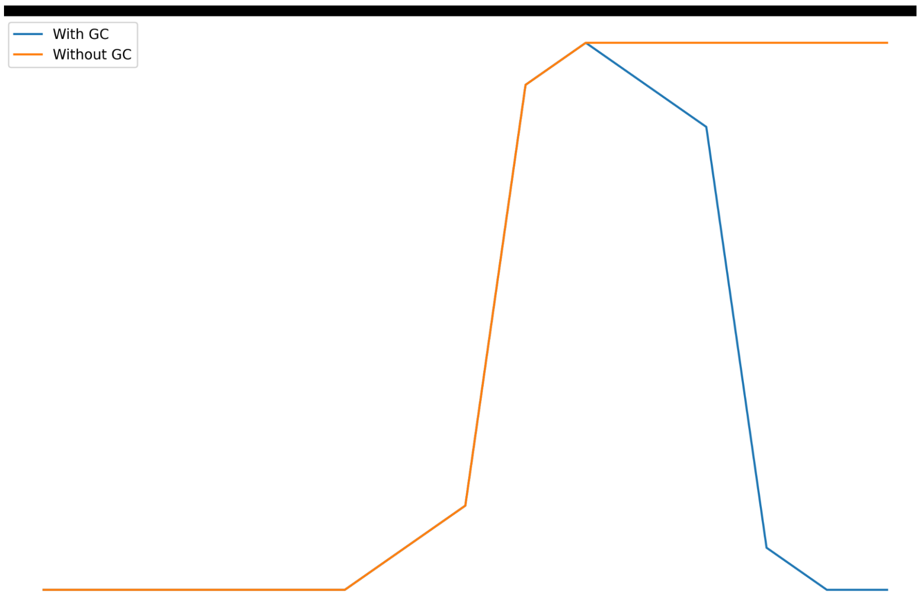
C. What is the impact of mark and sweep garbage collection for demo1 and demo2. Report the memory footprint with and without Garbage collection. Report the time of Garbage collection to run.

The following graphs explain the impact.

DEMO 1:



DEMO 2:



The garbage collection took negligible time to run. It took 0.49 seconds with GC and 0.47 seconds without GC for the first demo and 0.00807 seconds without GC and 0.00968 and with GC.

D. What is your logic for running compact in Garbage collection, why?

We have maintained a boolean array that stores whether a given block in the memory is free or not. When we run compaction using Two Pointer Algorithm. The first pointer points to the first empty location in the array. The second pointer points to the first non-empty location to the right of the first pointer. Now the data at the location pointed by the second pointer is copied to the location pointed by the first pointer. Then the first pointer is shifted to the right and the second pointer is shifted to the next non-empty location. The page table and variable structure are updated accordingly. In the case of an array, the entire array is moved at a time.

E. Did you use locks in your library? Why or why not?

Yes, we have used locks in our code. We have used it to make sure that while compacting the memory, we don't create a variable/array or assign any value to the variable/array element. If a variable is being moved to a new location, we can either update the table first and then move the variable value or move the variable value and update the table later. In both cases, assigning the variable a new value might assign the value to the wrong location or wrong time. Also if we are moving the free spaces then a wrong free space may be allocated to the new array or variable.