Incremental Steps for C3

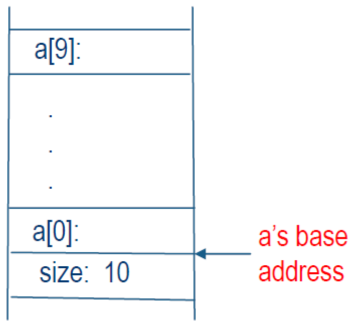
- **Subtask 1**: Get familiar with TM Simulator architecture and its assembly code and test the TM Simulator package.
- **Subtask 2**: Refactor the syntax trees and the visitor interface.

FunctionDec: funaddr (To be added)

VarDec: super-class for SimpleDec and ArrayDec
offset: location relative to fp (To be added)
nestLevel: gp or the current fp (To be added)

Implementation Detials

- Since an array is a static linear structure with a fixed size, we can store the "size" value right below the base address of the array
- When passing an array as an argument, the corresponding parameter will be given the base address of the array in the caller's stack frame, and right below the base address, we can access the "size" value



- Every time we access an indexed variable, we should always check the index value: if it is less than 0, we have a runtime error of "out of range below", and if it is greater or equal to "size", we have a runtime error of "out of arrange above"
- Since C- language only has integer values, we can show very large negative values such as "-1000000" for "out of range below" and "-2000000" for "out of range above" errors.

For a FunctionDec, add "int funaddr" to record the start address of the corresponding function, which is needed for a function call.

For a VarDec (either SimpleDec or ArrayDec), we need to add "int nestLevel" and "int offset". The former is either 0 for "global" scope or "1" for "local" scope, and the latter is the offset within the related stackframe for memory access.

If "nestLevel = 0" and "offset = -3", we will go the global frame pointed by "gp" and its 4th location to read/write data. If "nestLevel = 1" and "offset=-2", we will go the current stackframe pointed by "fp" and access its third location (right after "ofp"and "return addr").

For a Var (either SimpleVar or IndexVar) and a CallExp, we need to add a link to its related definitions: SimpleDec, ArrayDec, and FunctionDec. That's where we can find the memory location or the function address.

- Note that in the above illustration, we need to handle the "visit" for SimpleVar differently depending on whether we are computing the left-hand side of AssignExp or not. This is distinguished by "isAddr" parameter in the "visit(Absyn tree, int offset, boolean isAddr)".
- The value for "isAddr" is false for most cases except when calling "visit(tree.lhs, offset, true)" of AssignExp, since this is when we need to compute and save the address of a variable into a memory location.
- For the case of IndexVar, we naturally compute the address of an indexed variable, and that value can be saved directly into a memory location when used in the left-hand side of AssignExp.
- As a general principle, we use the given location to save the result of an OpExp, and the next two locations for its left and right children. In addition, register "0" is used heavily for the result, which needs to be saved to a memory location as soon as possible.