Homework 1 Answers

1. For strategy A, the average cost of a lookup within the same scope is $\frac{1+2+...+10}{10} = 5.5$ and the average cost of a single lookup in an immediately enclosing scope is $\frac{11+12+...+20}{10} = 15.5$ (since we'll have to go through the list for the current scope first). Overall, 100 lookups will cost $90 \cdot 5.5 + 10 \cdot 15.5 = 650$ on average. In the worst case, the element will be the last one probed each time, so the cost is $90 \cdot 10 + 10 \cdot 20 = 1100$.

   For strategy B, assuming perfect hashing, the average and worst case cost is the same and equal to 100.

   For strategy C, assuming perfect hashing, the cost of an access in the current scope is 1 while the cost of an access in the outer scope is 2 (1 to look in the current table and fail plus 1 to look in the second table). In total we have a cost of $90 \cdot 1 + 10 \cdot 2 = 110$.

   It appears that the second scheme is best. However, if we consider maintainance costs, we may choose to go for strategy A under certain circumstances: Strategy A requires only as much space as necessary and adding entries to the symbol table is always constant, while a hash table may need rehashing if the number of variables is much larger than anticipated.

2. This method is called worst-fit. In terms of speed, the answer depends on how the free list is maintained. If it requires a linear search, then worst-fit is comparable to best-fit since both need to traverse the whole free list to find a suitable block, while first-fit will stop as soon as it finds a suitable one. Alternate implementations (e.g. sorted list) will make worst-fit faster.

   In terms of fragmentation, we would expect our method to be better than best-fit. The latter results in many small and useless fragments. Worst-fit, by selecting the largest block, increases the chance that the remaining chunk will be large enough to be useful but also reduces the number of large blocks and may fail faster when large requests come in. First fit might still suffer from the same disadvantages, but since this would depend on the memory requests and not the way the method works, we expect it to give better results on average.

   In practice, worst-fit is not used much.

3. Using parameters would not be very flexible because then we'd have to specify their values every time we invoke a new shell. Furthermore, using dynamic variables makes it easier to temporarily modify the environment for a specific execution.

4. I am shamelessly copying James's answer to this question:

   Global variables are initialized outside of program execution scope. (This must be true: they need to be available even at the program's entry point.) However, function calls exist in the stack space and follow the scoping rules of program execution. Calling functions within an "outer-outer" scope prior to the entry point of a program could in effect implement a confusing "multiple entry-point" scheme where functions initializing different global variables would all behave like programs prior to the main entry point. While I see no technical reason that this could not be done, it does seem like it would be a confusing and counter-intuitive idea.

5. STATIC LINKS: Every time a procedure is invoked, a new link to its immediately enclosing procedure's frame is stored. Then, in order to reach non-local variables, we need as many loads as the difference between the current scope and the variable's scope. When the procedure terminates, no additional loads or stores are needed.

   DISPLAY: Every time a procedure is invoked, we need one load for the Display's old value (for that scope), one store for that value (so it can be restored afterwards) and one store in the Display of the new value. Then, in order to reach non-local variables, we need one load only to reach the correct scope. When the procedure terminates, we need to restore the Display by loading the old value and storing it in the Display.

Even though the Display requires more expensive book-keeping, it provides faster access to non-locals. So if there is a large number of non-local accesses at deep scopes, then the Display will be cheaper.

6. No. If part of the expression causes side effects and the optimizer does not evaluate them, then the program will not behave in the way it would had the "optimization" not taken place.