A Tool for Basic Programmer Training

by Jay Hocott

The purpose of this article is to introduce teachers, instructors, or activity leaders who are tasked with instilling sequential disciplined thinking in young people to a programming language particularly suited to this goal. The name of the language is M, formerly called MUMPS. It is often considered by some to be a niche language used primarily in the medical community. In reality, it is more than a language, rather it is a superior database engine. It resides in a niche not because of any limitations of the language itself, but because it lacks exposure to the computing community as a whole. It is frequently disparaged and dismissed as somewhat of a dying or dead-end language by those who have not kept up with its development and by those who fail to understand its power and applicability. It is an ANSI standard language whose strengths are character string manipulation and data storage and retrieval. We "MUMPSTERS," or perhaps I should say "M-sters," have a tendency to laud its strengths to each other, preaching to the choir, as it were, enhancing the niche image. Hopefully this article will help spread the word while aiding you in your educational effort.

In the incubation period between the commitment to write this article and actually beginning work on it, I ran across a thought that to me had power. The thought itself is not new, the way it was expressed is what impressed me. I share it with you hoping it will be as germane and powerful for you as it was to me:

"First, the author's protagonist was quoting a Park Ranger on grizzly bears. The Ranger described the animal as being nine feet tall, weighing a thousand pounds or more, ill-tempered, and could run faster than a human for short distances. He cautioned his listeners not to approach the bear. Then he offered this advice if chased by a grizzly. 'Your best chance,' he said, 'was to climb a tree.' If there were no trees close enough, then drop to the ground and play dead. The bear would, hopefully, just sniff around and perhaps turn you over and soon lose interest. However, this was entirely up to the grizzly and each bear was different.

With a knowing grin the Ranger went on to say that most of those listening would remember this for a while, some might even make notes. The chances were, however, that unless they really appreciated the danger, they wouldn't remember this advice if they were charged by a grizzly bear. They'd panic and run for their lives even though they'd just been told that the grizzly was considerably faster that they were. But, he said with a grin, if there was any one of his listeners who had actually been chased by a bear, that listener would remember the advice and drop to the ground as if shot, instantly.

Then the protagonist drew this conclusion. 'Specific advice is something I give only when asked. Like teaching, advice is wasted until the listener already has a vital reason for wanting to know it. When that's true, then suddenly that knowledge makes immediate sense to them. But until that moment of recognition, all they've gotten from me is a bag full of words. They may listen to those words. They may even write them down; but until it connects with something important to them, what they hear is just words.' "[1]

The obverse side of this coin is the thrust of this article. Not only do we have to make our curriculum germane in the manner of the above illustration, we need to purify the task of introducing computer programming concepts of as many arbitrary definitions as possible. A classic example would be COBOL's environment division, and more currently, the definition of file organization techniques; sequential, indexed sequential, random, etc. The beginning student, especially the young student without much life experience, has no hooks on which to hang this information. He/she is not yet being chased by this particular bear. So the task changes from turning them on to not turning them off.

The concept of a variable is the first turn-off usually encountered. Some students never absorb this concept. Others, who seem to grasp the concept, fail when variable types are introduced. That makes as much sense to them as me hitting you with the statement that pink widgets can only go into green blivits. The survivors then are hit with type and dimension statements with rigid and arbitrary rules. Far too many times his/her eyes glaze over, and that student becomes convinced that he/she cannot do this thing. This belief, either that he/she can do or cannot do a thing, almost certainly determines the outcome of the instruction. The M language avoids every complication to the concept of a variable by its "untyped" variable structure. The concept of a variable is then simplified to be the name of a place that may store data, without conditions imposed by variable types.

Many computer concepts are accepted rather than understood. An interpretive language aids in achieving that acceptance. A variable, or other concept, can be immediately and repeatedly

demonstrated until the acceptance or understanding comes. While M is obviously not the only interpretive language, it has several useful attributes. In direct mode, a line of code may be typed as part of a routine, stored in a buffer, then saved and executed later, or the same code snippet may be typed for immediate execution. A single keystroke defines the mode, there is no need to change windows, use function keys, exit an editor, or take any attention-diverting action. Using this alternating mode capability, the teacher can demonstrate, and the student can verify and validate, the effect of each statement or segment on the variables or process being investigated. This immediate identification of correct or incorrect assumptions facilitates rapid concept formation and avoids false or ambiguous concept absorption that must later be located and corrected. This technique allows the limits of an operation or command to be studied, increasing understanding. M's dual modes that allow for either immediate or delayed statement execution, have a high value in getting a point across the communications gap between teacher and student.

A challenge, issued as an opening remark to beginning students: "Name the five things a computer can do" produces silence. After a while, frequently a long while, someone will usually "break the ice" and suggest that a computer can add. Then other arithmetic operations follow. The disappointment is palpable when told "That's one, a computer can do arithmetic." Decision-making usually is the next thing identified. Sometimes, input/output is recognized. Data storage and movement usually must be coaxed from the class. The ability to stop and handle an interruption must be presented to them. Again, immediate mode best demonstrates data input, output and storage. Arithmetic and arithmetic expressions are also easy to demonstrate. M operates in strict left-to-right order that removes the need for introducing the complication of a hierarchy of operations and provides a good platform to show the effect of parenthetical expressions. M has a special variable that contains the result of a logical (If) expression. Thus, the more difficult concept of logic can be made visual by examining M's truth variable. The especially difficult concept of a truth value can be demonstrated until acceptance comes. The ability to see the yes/no value of the truth variable enables the student to recognize the pattern of defining "truth."

In short, all the "bricks and mortar" necessary to build a computer program can be easily demonstrated. It seems, however, that I recall a story about being chased by a bear. Even if the student knows the length, width, depth, texture, color of a brick and has similar knowledge of mortar, he/she has never seen anything made from these materials. Some scenario must be chosen for computer application. A real danger exists in that the application will be so trivial [as] to be irrelevant. Any significant application has to involve data storage and retrieval. This is frequently accomplished by data stored as part of the program, again trivializing the example. How much more realistic and useful to concept formation would [be] retrieving data from a permanent repository? It is seldom done due to having to introduce arbitrary file organization and techniques or simply presenting data handling code as a given. Using M, the commands to handle data in program variables are used identically to handle data on disk. Data can be retrieved from disk and placed in a program variable with a Set command. Data can be stored on disk from a program variable by changing the order of the operands with the same Set command. This ability to fetch and store data from disk enables the course content to be made meaningful to the student. He/she can see the checkbook, or the address book, or another entity that is real to him/her, that can be built with all this "bricks and mortar" stuff. The ability to use meaningful data in a realistic application without a plethora of arbitrary rules that must be accepted without understanding, empowers the student to take control of the computer. The young student responds to this power. The freedom to create from his/her own imagination can establish the foundation to understand the need of organization techniques and structure rather than to present them as rules that exist just because somebody said so.

Computer programming can be divided into two broad groups, scientific or engineering applications and business or commercial applications. If you plan to launch your own satellite or write arcade games, M is not for you. However, if you want to know about your customers, patients, inventory or products, the advantages of M become readily apparent. Customers and patients have names and addresses, products have descriptions, specifications and locations. These data are examples of character strings. In fact, the ability to manipulate character strings all but determines categorization of a program into the broad groups above. M is ***THE*** string manipulation language. It was designed from the ground up to be a string manipulation language. It has every string manipulation function you can imagine, all syntactically simple and logically easy to understand.

The M language has a free-form syntax. This syntax supports program structure but does not enforce it. The lack of structure enforcement removes a significant level of complexity in forming concepts about elementary computer functionality. Structural rules and discipline can be introduced as the student gains confidence and capability. Using this procedure, structure is seen as good technique and as a productivity tool instead of some arbitrary set of rules that interfere with coding flexibility. The discipline required to write structured code can be introduced at the time the student is capable of accepting it. Just like the good advice about grizzly bears, it can be timed appropriately.

The concept of data storage is virtually a "blackbox" in other database implementations. Data stored by M is visible to the student. The student may control the way in which data are stored. Different storage schemes may be evaluated. The student can navigate through the open array data storage provided by M. He/she can evaluate the effect of storage schemes on retrieval effectiveness and can design his/her own data structure suited to his/her process. The visibility of stored data allows the student to appreciate the value of raw data in diagnosing problems and debugging programs. The student gains insight into the relationship of data to programmatic processes.

OK! So M is easy to teach, aids in concept formation, supports realistic applications, easily stores and retrieves data from disk, and manipulates strings. Assuming that you are convinced that M is the best language for training programming to programmers, so what! Few, if any, teachers would select or create a curriculum based solely on how easy it was to teach or to learn for that matter. Selecting M as a beginning language will do all the above. However, the reasons for using M are the best reasons for teaching M. M is a superior database engine. It is faster and more efficient than any other database. As mentioned earlier, it is an ANSI standard language, continually evolving to the state of the art. M supports visual interfaces and open database queries. In short, M has the necessary attributes to be the future standard for data storage and retrieval. The primary reason for teaching M is its applicability to "real world" situations, an important bonus is that it is well-suited as a primary training facilitator.

How will teaching M benefit your student? Admittedly, M is not as widely used as many other computer languages. There is a demand for M programmers now and the demand is growing. The demand is also worldwide. Even if your students never work in M, many advantages still accrue. The first computer language learned is the most difficult. Basic concepts unique to computer programming must be formed.

These concepts are independent of the language used to develop a program. Similarly, basic techniques must be developed. Insights into computer behavior must be acquired. A beginning student has much to absorb. It makes good sense to ease that assimilation in every way possible. M does exactly that.

For additional information on M, contact the M Technology Association, Micronetics Design Corporation, InterSystems Corporation, or Greystone Technology Corporation. Both Micronetics and InterSystems offer no-cost implementations of M that run on an IBM compatible. The Micronetics product is called MSM-Student and can be downloaded from their Web site. The InterSystems product is called DTM Student. Both of these products are full implementations of the language. They are restricted somewhat but the restrictions are based on size. No restrictions are placed on the functionality of the language.

Finally, I offer my personal assurance of your satisfaction with M. I have taught computer programming since 1960 to both young and not-so-young students. I conducted classes in COBOL, FORTRAN, RPG, Basic dialects, macro languages, and several assemblers in addition to M. M has been, by far, the most easily absorbed and has produced the greatest success ratio. On request, I will make available a syllabus for a beginning course in M starting with a "checkbook" example and extending into an elementary cost distribution and accounts payable application. A very elementary data dictionary is developed and used for file maintenance.

M Technology Association
<http:\\members.aol.com/mtal994/mta.htm>

Micronetics Design Corp <http:\\www.micronetics.com>

InterSystems Corp. <http:\\www.intersys.com>

Author: jayhocott~mail.snider.net

M

REFERENCES

1. *Other* by Gordon R. Dickson

Jay Hocott was the Training Coordinator for the Little Rock VA Medical Center in charge of training 4000 employees in the use of the DHCP system and is now retired. He also supported major portions of VA's DHCP program from 1984 through 1995 including the VA Kernel and FileManager. He has been in computer programming and education since 1960.

1997 MTA Annual Conference
May 18-22, 1997
Boston