CMU-CS-85-175

Implementing a Mastery Examination in Computer Science

Mark J. Stehlik

Philip L. Miller

Computer Science Department
Carnegie-Mellon University

November 8, 1985

Summary

In this paper we describe the rationale behind, the content of, and the administration of the Mastery Examination as it is used in the Introductory Programming courses at Carnegie-Mellon University. Special attention is given to the actual delivery of the Mastery Examination.

The authors thank the Computer Science Department of Carnegie-Mellon University for its unfailing support. A special thanks goes to the faculty, staff, and thousands of students who have been a part of the Introductory Programming courses during our time at CMU.

Table of Contents

1. Background

1.1 Philosophy

Early in 1980, it was decided that the Introductory Programming courses at Carnegie-Mellon University should include a competency based examination called the **Mastery Exam**. The reason for this decision, an expected method of delivery, exam content, and the method for evaluation were set out fully in [2]. To summarize the reason for decision, it was hoped that the Mastery Exam would address a general problem of students successfully passing programming courses at CMU without also learning the rudiments of programming methodology. A review of the success of the Mastery Exam format is given in [1]. Briefly, it was found that use of the Mastery Exam appears to be closely linked with a dramatic improvement in student performance in these same courses.

The faculty and staff of the Introductory Programming courses at CMU subscribe to the thesis that the best way to measure a student's programming proficiency is by letting the student write a program with all the tools (editors, compilers, debuggers) normally attendant to the task. The program can then be graded by a number of different metrics (functionality, programming style, user-friendliness) either singly or in some combination. The main drawback of written exams in a programming course is that they typically require a student to design and write both semantically *and* syntactically correct modules (or programs!) in *one* pass without *any* feedback (at least until the graded exams are returned). This seems to be unrealistic at best and counterproductive at worst.

Programming in any language is a skill that is only developed over time. In the Introductory Programming courses at Carnegie-Mellon, we encourage this development through weekly labs (small programs) and a number of larger programming assignments scheduled throughout the term. The Mastery Exam seems a fitting end to this process in that it measures the ability of a student to write a program to solve a given problem within a certain period of time.

Despite the face validity of the Mastery Exam as a measure of programming ability, it is not 100% of a student's final grade. When it was used as such, students demonstrated a tendency to ignore most of the course assignments, since the Mastery Exam was all that mattered. This worked to their detriment since completing the other course assignments (labs and programs) is vital to developing programming skill. It is virtually impossible to write a Mastery-level program if a student has not attempted any other programs during the semester. As a result, the Mastery Exam is now viewed as a major part of the final grade, but course-work performance is factored into the final grade determination to provide additional incentive for its completion (see Appendix I - Final Grade Determination).

1.2 Short Description

The Mastery Exam is designed as a five hour task. A pool of possible questions is made public to the students at the beginning of the semester. On a given exam day, students report to the terminal room and are logged in to the computer with a specific mastery user-id to a specific mastery directory. Initially, there are no files on the directory. One of the questions from the pool is drawn at random for each student and when all students have been given their exams, the exam is started. Five hours later (at most), the students are asked to turn in their work, logout and leave.

1.3 Retakes

The Mastery Exam causes much more fear and anxiety in our student body than is rationally justifiable. In a number of cases, students balk during the taking of an exam and end up with a program that barely runs (or doesn't even compile). To deal with the legitimate cases of exam-induced nervousness causing poor performance on the Mastery Exam, we allow for the possibility of a student having to retake the exam subject to certain criteria. Typically, if a student ends up with a grade on the Mastery Exam that is two letter grades lower (or worse) than their coursework grade, that student is entitled to retake the exam (see Appendix II - Mastery Retake Policy).

2. Exam Questions

2.1 Development Criteria

A representative Mastery Exam (see Appendix III) requires a student to perform a number of specific tasks. There are many fields from which to draw potential Mastery Exam questions. Ones that have been used successfully at Carnegie-Mellon include a line-based editor, a payroll manager, an academic advisor (keeps track of students and courses they've passed), a simple flight reservation system, and a mailing-list generator. In order to be consistent, each of the exam questions (regardless of venue) is designed to fit the following schema:

- Data Structure - the problem must lend itself to solution with an array of records or a singly linked list. Either data structure is viewed as appropriate; there is no incentive to choose one or the other. It is up to the individual student to decide which one they would like to use. Approximately 95% of the students in the course write their exams using an array-based solution.

- Functionality - each exam question is written to require the individual student to write similar routines regardless of the specific question that is drawn. The levels of functionality required are as follows:

    - Minimum - in order to achieve a passing grade on the Mastery Exam, a student must be able to write a program that performs both simple file I/O and sequential search.

        - File I/O - In practice, this means that a student should be able to read data from an external file into a suitable data structure (i.e. an array of records or a linked list). The student should also be able to write the information back to a file from the program.
        - Search - the student must be able to write routines that prompt the user for a key and then search the structure for the record associated with that key. The information in the record is then displayed to the terminal.

    - Additional functions - a choice is offered among the following functions. The more that are correctly implemented by a student, the higher the grade.

        - Modifying records - the student is expected to be able to process and/or modify records in the data structure. This can include adding a field or changing a field in any record in the data structure.

        - Modifying the structure - the student is expected to be able to modify the data structure itself. This requires the insertion and deletion of entire records within the data structure. Typically, insertion and deletion are viewed as separate modules (see Appendix III).

        - Sorting the structure - the student is expected to be able to perform a sort of some type on the data structure.

2.2 Practical Issues

2.2.1 Distribution of Questions

It has been the policy in the Introductory Programming course at Carnegie-Mellon to make public the complete set of questions that will be used at the end of the semester. Students are given a packet at the beginning of the course that includes all the labs and programming assignments for that semester as well as the actual set of Mastery Exam questions that will be used at the end of the semester. There are no surprises.

This may come as quite a shock since the contents of an exam is traditionally a well-guarded secret. The Mastery Exam, though, is not a traditional exam. As the Mastery Exam is supposed to measure programming ability, we find no inconsistency in publishing the actual questions in advance. The theory behind this is that if, in advance, a student wants to write programs that solve all (or a good percentage of) the Mastery Exams, that student will both be well-prepared for the exam *and* will have learned how to program.

We have recently decided to add some measure of variability into the actual Mastery Exam that a student must write on exam day. Students are now expected to be able to read data files that differ somewhat from the sample format specification as well as add functionality that was not specified before exam day. This was done to be certain that students were not able to memorize solutions without coming to grips with issues of programming methods. The net effect of this is to dramatically increase the pool of possible examinations. Rather than face a possible seven or ten exams, the student is faced with possibilities in the hundreds (all differing slightly)!

Students in the course are encouraged to write programs for some of the exams in advance of finals week. The last programming assignment for the course is one of the actual exam questions! It is usually the case that the students who do well are the same students who have written a number of mastery programs prior to taking the actual exam.

2.2.2 File Formats

A description of a representative format for each of the input files for the Mastery Exams is made available to the students. It also serves to guide them as they develop the specifications for the individual records to be used in the data structure. The format serves only as a guide, however, and the file that the student is given on exam day may differ somewhat from that which is used in the sample illustration. Again, this is to thwart memorization of routines.

2.2.3 Algorithm Help and Exemptions

In a number of cases, the average student may not have been exposed to the algorithms necessary to solve the problem, even though they have the programming ability to implement the algorithms. For these exams, help files are included so that students can still attempt these exams. In the current Mastery Exam set, there are two exams that fit this mold (and help files are given for both): Matrix Math (which involves multiplying an arbitrary matrix by a scalar, and being able to add, subtract and multiply two matrices) and Gaussian Elimination (which involves performing Gaussian elimination on an arbitrary matrix). A sample help file (the one for the Matrix Math mastery) is given in Appendix IV.

In addition to the help files, we typically allow students to exempt a number of the questions from the set from which their exam will be drawn. In general, students were allowed to exempt three exams from consideration out of the set of ten exams available. Most chose to exempt the matrix-oriented exams, with the remaining exemptions spread out over all other choices.

2.2.4 Executable Versions

In addition to making the exam questions public, executable versions of each of the exams are made available on-line at the beginning of the semester. This allows a student to examine the functionality of a working program and provides them with a reasonable model on which to base their own work. We believe that the source code which is used to generate the executable versions should be kept protected and inaccessible to the students.

3. Scheduling

3.1 Resources

There are, on average, 600 students per semester enrolled in the three versions of Introductory Programming offered at Carnegie-Mellon. The three courses cover essentially the same material; one is designed for Science and Engineering students, one for Humanities and Social Science students, and the last for Architecture students. For purposes of the Mastery Exams, however, all three groups will be considered the same.

We have two VAX 11/780[1] computers (running UNIX™) available for use by students. There are 20 terminals connected to each machine, so the terminal room can accommodate a maximum of 40 people. For the Mastery Exam 'sessions', we choose to use only 30 terminals (15 per machine). This is done both to spread people out (to minimize looking at someone else's work) and to keep the load level manageable throughout the exam. As a result, any one Mastery Exam session can accommodate a maximum of 30 students.

3.2 Early Exams

The idea behind offering the Mastery Exam early in the semester is twofold:

- to allow students with prior programming experience the opportunity to 'place out' of the course by taking the exam
- to allow eligible students from the semester before the opportunity to retake the exam

3.2.1 Scheduling

In the Introductory Programming course at Carnegie-Mellon, we offer early Mastery Exam sessions the last two weekends (2 each weekend - one Saturday and one Sunday) of the first month of the semester. As any one session can accommodate at most 30 students, we can schedule as many as 120 students for the early Mastery Exam. In a typical semester, anywhere from 60 to 90 students will sign up for the early sessions (about 30% of these are people attempting to place out of the course).

---

[1] VAX is a trademark of Digital Equipment Corporation

3.2.2 Registration

Any student wishing to register for an early Mastery Exam session must obtain a permission slip from the instructor in charge of the student's section of the course. If the student is attempting a retake, the permission must be obtained from the instructor that the student had in the prior semester. This permission may be obtained if the student met the previously mentioned criteria for a retake (mastery grade 2 letter-grades below coursework grade).

If the student is trying to place out, some evidence of programming ability must be furnished to the instructor before permission will be granted to take the early exam. This is insisted upon to allow only those students with a reasonable chance of actually passing the exam the opportunity to take it.

3.3 Final Exams

It would seem that since there are approximately 600 students in the course and any one Mastery Exam session can accommodate 30 people that we would only have to schedule 20 sessions. In practice, however, this doesn't work because most of the sessions are not filled to capacity. As a matter of fact, the only sessions that are completely filled are the last 4 or 5 that are offered. This is mainly due to conflicts with other final exams, but there are other factors (such as procrastination). As a result of some experimentation, it has been shown that optimal number of slots to allow for is 1.5 times the number of students registered for the course. For 600 students, then, this would demand 900 slots, or 30 sessions at 30 students per session.

3.3.1 Scheduling

Carnegie-Mellon has a final exam period of one week. The Mastery Exam sessions typically start the week before finals begin and continue through to the end of finals week. There are two sessions scheduled every day for those two weeks, giving us 28 sessions. In addition, there is a single session run the Saturday and Sunday before the two-week period begins. This normally gives us enough slots to accommodate everyone.

3.3.2 Registration

For the Mastery Exams offered during finals, there is no permission required. Every student registered for the course is allowed to sign up for one slot on any day that has an opening. A book of the sign-up sheets is kept in the terminal room and students register for an exam by signing their name, user-id and section on the sheet for the day that they desire. Students are allowed to sign up for only one session. If they have already signed up and desire to change to another day, they must come in and have a tutor remove their name from the original day and add it to the new one, assuming that there is an opening available.

4. Exam Administration

4.1 Materials

The following materials are required during a Mastery Exam session:

- copies of the exam questions
- copies of any help files
- several copies of Pascal textbooks (for student reference during the exam)
- scratch paper and pencils

4.2 Mastery Accounts

Rather than let the students login to their own accounts (which would make it extremely hard to restrict access to their files), we have established special Mastery Exam accounts on each computer. As we have chosen 15 as the maximum number of students per machine during an exam, the accounts are numbered from mex1 through mex15 (hereafter referred to as *mex**). These accounts are for use only during Mastery Exams and are all 'owned' by another special account, the *proctor* account. The proctor account serves as a central repository for all mastery software as well as a convenient place from which to monitor the mex* accounts.

Having the mex* directories owned by another directory prevents a student running as one of the mex* from changing the protection of the files that are there (so that someone else could copy them). It also makes a number of other preventive measures easier to implement. As UNIX™ is, like most operating systems, not designed to provide exam-type security, a number of precautions must be taken to ensure that there is no 'sharing of information' between mex* accounts during a Mastery Exam.

4.3 Mastery Software

4.3.1 Initializing the System

One of the thorniest security problems faced by someone implementing a Mastery Exam system is how to prevent access to other parts of the system (files and accounts) during the exam. The reason this is necessary should be clear: these accounts contain some files which, in all likelihood, are solutions to various of the Mastery Exams. There are two possible ways to solve this problem:

- remove all offending files and accounts

- leave the files and accounts alone, but prevent outside access to them during the exam

The means that we have ultimately chosen (having tried both) is the former. As the two computers are used solely by students and staff connected with the Introductory Programming course, we have total control over what files and directories are available. All student files are on a separate file structure, which is dismounted before an exam session is scheduled to begin. All files and utilities necessary for the exam (the editor, compiler, any data files required by the various Mastery Exams) are also on a separate structure, which is mounted before an exam session is scheduled to begin.

It appears that this is the easiest way to ensure that students taking an exam will have access to those things which are necessary and nothing more. On some systems, such as the one at Carnegie-Mellon, certain system servers (such as the mail server, file-transfer programs and inter-machine network servers) must be disabled before an exam environment can be thought of as secure. Having done this, though, most security problems will have been solved.

4.3.2 Individual Account Software

Once the exam environment is secure, the issue of student logins must be addressed. The password to the mex* directories should not be divulged to students taking the exam. The person proctoring the exam should be apprised of the password and should be responsible for logging students in to the accounts that they will be using during the exam. This person should also keep track of the account that the student has been assigned (we make this notation in a space on the sign-up sheet). This is used as backup verification of the student/mex assignment.

Students should present some form of ID in order to be logged in to take the exam. As students are logged in to the mex* accounts (trying to distribute the students as evenly as possible between the two machines), a program is run which removes all leftover files (presumably from a prior exam) and ensures that the only files that are left are any necessary system files (if there are such on your system). After this program has been run, the contents of the directory should be displayed. If the proctor decides that nothing is amiss, he should allow a student to use the terminal. Alternatively, this could all be done from the proctor directory before starting the exam.

Once a student has been assigned a terminal, a program (called *HELLO* on our system) should run which greets the student and acquires any necessary information from the student (name, user-id, course section, etc.). This program should then prompt the student for the exams that he wishes to exclude and then randomly choose one of the remaining exams. Once this is done, an entry should be created in a file in the proctor directory with the following information:

- student name
- student userid
- class and section
- semester and year
- mastery directory (one of the mex*)
- the exam that was randomly drawn

Every student who is taking an exam should have an entry in this file which should be located in the proctor directory. At the end of the exam, the file will be placed in a separate location and a new file will be created for the next exam.

As we don't want the people who were logged in first to have an unfair advantage, the last function of the *HELLO* program should be to 'freeze' the terminal until a flag is raised on the proctor directory. The easiest way to do this is to have *HELLO* look for a specific file on the proctor directory. It should check every 10 seconds or so for this file and when it finally sees it, should terminate, allowing the student to begin programming.

After the exam has been chosen, a copy of the actual exam question that was selected by the program should be given to each of the students. This can be done by one proctor, while another is logging in the remaining students. You are ready to start the exam when all students have been logged in and given a copy of the exam question that has been drawn for them.

4.3.3 Central Proctor Software

The proctor directory should be where all information about a current exam is stored. There should be a facility for raising the flag that 'unfreezes' the terminals as mentioned above. There should also be a file that contains all the information about the students currently taking the exam. Since we want this to be a repository, there must be a provision for students to append files to a subdirectory that is owned by the proctor directory. This subdirectory will contain the files that students turn in at the end of the exam (the Pascal source code and any additional files, for example, a photo file). It must be decided how to archive the files from each exam (how long they must be kept on-line and where they should be kept).

4.3.4 System Crash

If the system should crash during the exam (and it will!), some provision must be made to minimize the impact that this has upon the students' work. We inform the students that any down time will be added to the duration of the exam so that they will be given the full 5 hours on the machine. Students ARE allowed to work on their programs during the time the system is down (of course, they won't have the use of the computer to do so).

When the system comes back up, students are logged back in to the directories that they were logged into at the start of the exam. We have also provided for a means to indicate that a *restart* is in progress, so that the program (*HELLO*) which initializes the exam directories does not remove all the files!! This is probably the most easily overlooked part of the Mastery Exam software. It is inevitable that your system will crash during a Mastery Exam. Plan for it in advance and it won't be nearly as traumatic.

Students will undoubtedly lose some work (depending on the auto-save frequency of your system's editor), but any time lost in this manner can be added to the time they have for the exam.

4.4 During the Exam

4.4.1 Student Privileges

Since there should be no possibility of students accessing any files that they are not supposed to access, the mex* accounts should have normal operating privileges. The one exception to this is that all files created on an mex* directory should be protected so that only the individual mex account can read them. Since the mex* accounts cannot change the protection on files, there should be no problem with students accessing each other's code.

Students are allowed to copy data files from a central area (the proctor directory would be ideal for this) and they should be allowed free rein in using the editor and compiler. Students should also be allowed to print files at will to aid in debugging their programs. Experience has shown that students print programs at random intervals so there is never a large number of students at the printer at any one time. If this becomes an issue, a proctor could stand by the printer to ensure that no information is passed between students.

Students should be allowed access to a Pascal textbook during the exam. Rather than have them bring in their own, a supply should be available during the exam.

Out of concern for security, we restrict students to one trip to the bathroom during the course of the exam. It may sound draconian, but has been proven to be necessary.

4.4.2 Duties of Proctors

It should be the case that only students who are taking the Mastery Exam should be allowed into the terminal room. The proctor should make sure that the room remains QUIET and that the exam runs smoothly. Be aware that people do try to cheat on the exam and keep an eye on the students.

Students will inevitably ask questions of the proctors. Proctors should be aware of this and should be cognizant of the fact that students should write their own programs. Proctors may help students with any difficulties encountered in using the editor, and, if necessary, can explain system commands. They should help them get files they can't find, make sure they are using the proper data files and, if they have trouble, make sure that the copy in their directory is not damaged.

The proctor should also keep tabs on the time of the exam and inform the students at every hour how much time is left in the exam. When you get down to the last hour, telling the students at fifteen minute intervals is a good idea.

4.4.3 Documentation

We have prepared a document that describes, in detail, the procedures involved in starting and running a Mastery Exam session on our system. A copy of this documentation is kept with the proctor materials so that it can be referenced during an exam. This is to assist the proctor in keeping the exam running smoothly.

As we have become familiar with known problems and their fixes, we have added them to this document. This will has helped tide the proctors over until the problems were permanently fixed (which is always later than you want them to be fixed).

4.5 Ending the Exam

At the end of the time for the exam, students should be informed that they must finish making their last edit and that they should wrap things up. One nice facility of the computers that we use in the course at Carnegie-Mellon is a program called *photo* which enables a student to record, verbatim, an interactive session at the terminal in a separate file.

We have the students finish editing their program and then have them run their programs in photo.

Once inside photo, the student demonstrates everything that the program is capable of doing by running the program in accordance with a test script that is handed out with the copy of the exam question at the beginning of the session. The test script consists of a sequence of commands (and the data to be used with each of them) designed to completely exercise and demonstrate a working program. Students are encouraged to follow the script as best they can and then exit photo.

Once the student is finished demonstrating the program's execution, there should be some on-line facility for handing in their work. This program should allow the student to specify the name of the file that contains the Pascal source code as well as the name of a photo file (if there is one) and should append them to a subdirectory in the proctor directory. All these files can then be printed to be graded and moved elsewhere to be archived.

I. Final Grade Determination

The final grade for the course is a combination of the grade for course work and the mastery grade. Below is a matrix detailing how the final grade is determined:

COURSEWORK GRADE	MASTERY GRADE A	MASTERY GRADE B	MASTERY GRADE C	MASTERY GRADE D	MASTERY GRADE R
A	A	A	B	C	R
B	A	B	B	C	R
C	A	B	C	C	R
D	B	C	C	D	R
R	C	C	D	R	R

When a student's performance on the mastery is significantly worse than his performance in class during the year (as measured by labs, programs and exams), a retake of the mastery exam may be permitted. The criteria for a retake are established in the table below (essentially, the mastery grade must be two grades lower (or worse) than the coursework grade). Under *no* circumstances can the final grade for the course based on a retake be higher than the coursework grade, as detailed in the following matrix:

COURSEWORK GRADE	MASTERY RETAKE GRADE A	MASTERY RETAKE GRADE B	MASTERY RETAKE GRADE C	MASTERY RETAKE GRADE D	MASTERY RETAKE GRADE R
A	A	A	B	C	R
B	B	B	B	C	R
C	C	C	C	C	R
D		NO RETAKE ALLOWED			
R		NO RETAKE ALLOWED			

II. Mastery Retake Policy

When your performance on the mastery is significantly worse than your performance in class during the year (as measured by labs, programs and exams), a retake of the mastery exam may be permitted. The criteria for a retake are established in the table below (essentially, your mastery grade must be two grades lower (or worse) than your coursework grade).

MASTERY GRADE	COURSEWORK[2]	RETAKE
R,D,C	A	Yes
R,D	B	Yes
R	C	Yes
R	D	NO
R	R	NO

If there is room (available slots), a student should see his recitation instructor to get permission to retake the exam and schedule the retake *during* the semester in which he is registered for the course. Otherwise, the student ***must*** take the retake during the period of time allotted for mastery retakes during the following semester. The retake period usually will be the last 2 weekends of the first month of the following semester. If you are retaking the exam in the retake period, you **must** see your instructor during the ***first*** week of the following semester in order to get permission to retake the exam.

If you meet the above criteria, you generally can expect to be allowed ***one*** mastery retake. Under ***no*** circumstances can you retake the exam more than once during the retake period.

Some notes concerning the mastery exam and your final grade for the course:

- You ***must*** take the mastery in the semester in which you are registered for the course. Failure to do so will normally result in a grade of R (failure) for the course.
- If your mastery grade is higher than your coursework grade, your final grade is determined according to the grading matrix found in the Course Syllabus. As an example, assume that you get an A on the mastery, but have a D in coursework; your final grade will be a B (NOT an A).
- A retake cannot improve your final grade for the course above the coursework grade as detailed in the matrix found in the Retake Policy section of the syllabus.

---

[2] *Coursework* in the above table refers to the average grade of a student's programs, exams, labs, and homework.

III. Representative Mastery Exam

ACADEMIC ADVISOR

You have just been hired as an academic advisor to the students of this university. You must write a program to assist you in advising the students as to what courses they must take to graduate. A file of student information, **courses**, exists. The file contains all the student names and the courses they have passed thus far. To graduate, a student must pass all of the following: Math, Biology, Physics, Computing, Chemistry, and English. Your program must accept the following commands:

Required for a Passing Grade:

Must read the file into an appropriate data structure.

D --> must prompt for a student's last name and first name and display both the courses passed and the courses needed to graduate for the specified student. Must indicate if the student is not found on the rolls.

Q --> writes all updated information back to **courses** and terminates the execution of the program. All commands must be allowed to repeat until Q.

In addition, you must do all of the above plus:

- **2** of the following **5** items for a **C**
- **4** of the following **5** items for a **B**
- **all** of the following **5** items for an **A**

P --> must prompt for a student's last name and first name, and a newly passed course. The course will be added to the list of courses that the student has passed. If the student has already passed that course, the user should be so notified. Must inform the user if the student is not found.

T --> should display the following information for each course: course name, the total number of students who have passed the course, and the total number of students who have not passed the course.

A --> will prompt the user for the name of a new student as well as the courses that the student has passed and add that student to the list. Must inform the user if the student is already on the list.

R --> will prompt the user for the name of a student and remove that student from the list. Must inform the user if the student is not found.

Sort the names in alphabetical order when they are read in and maintain the students in alphabetical order throughout the program. Note that adding a new name will require to be inserted alphabetically (or the list to be re-sorted).

IV. Representative Help File

What Is A Matrix?

A matrix is a rectangular array of numbers arranged in rows and columns. This is a matrix:

```
1 2 3
4 5 6
7 8 9
```

Rows go from left to right. In the matrix above the first row is 1 2 3. Columns run from top to bottom. In the matrix above the first column is:

```
1
4
7
```

A matrix may have any number of rows and columns. This is a 2x3 matrix (which means that it has 2 rows and 3 columns):

```
1 1 2
6 9 4
```

This is a 1x3 matrix. It has 1 row and 3 columns:

```
1 0 1
```

This is a 3x1 matrix. It has 3 rows and 1 column:

```
1
2
3
```

Scalar Multiplication

Several arithmetic operations can be applied to matrices. The first of these to be explained here is scalar multiplication. Scalar multiplication is multiplying a matrix by a scalar (number). This operation is performed by multiplying each element of the matrix by the scalar.

Example:

```
                1 2 3          3  6  9
3       *       4 5 6    =    12 15 18
                7 8 9          21 24 27

scalar          matrix         result
```

Matrix Addition

Two matrices are added together by adding the corresponding elements of each matrix. Consider the following matrices:

```
(1)   a[1,1]  a[1,2]        (2)   b[1,1]  b[1,2]
      a[2,1]  a[2,2]              b[2,1]  b[2,2]
```

{ [ ] denote a subscript.}

When we add these two matrices, we get the following resultant matrix:

```
a[1,1] + b[1,1]   a[1,2] + b[1,2]

a[2,1] + b[2,1]   a[2,2] + b[2,2]
```

Here is an example using numbers:

```
1 2 3          1 2 1          2 4 4
4 5 6    +     3 4 3    =     7 9 9
```

Notice that two matrices may be added together only if the number of rows in the first matrix equals the number of rows in the second and the number of columns in the first matrix equals the number of columns in the second.

Matrix Subtraction

One matrix may be subtracted from another in the same way that one matrix is added to another.

```
(a[1,1] a[1,2]) - (b[1,1] b[1,2]) = (a[1,1]-b[1,1]  a[1,2]-b[1,2])
(a[2,1] a[2,2])   (b[2,1] b[2,2])   (a[2,1]-b[2,1]  a[2,2]-b[2,2])
```

Matrix subtraction may also be thought of as a two step process. The first step is scalar multiplication. The matrix to be subtracted is multiplied by -1. The second step is addition of the first matrix and the new matrix resulting from scalar multiplication.

```
                  a[1,1]  a[1,2]  -  b[1,1]  b[1,2]
                  a[2,1]  a[2,2]     b[2,1]  b[2,2]

(1)      -1   *  b[1,1]  b[1,2]  =  -b[1,1] -b[1,2]
                 b[2,1]  b[2,2]     -b[2,1] -b[2,2]

(2)              a[1,1]  a[1,2]  +  -b[1,1] -b[1,2]
                 a[2,1]  a[2,2]     -b[2,1] -b[2,2]

                 a[1,1]-b[1,1]   a[1,2]-b[1,2]
           =     a[2,1]-b[2,1]   a[2,2]-b[2,2]
```

Matrix Multiplication

To multiply matrices, think of them in terms of rows and columns. To find the result of multiplying matrices A and B, A * B, the rows of A are multiplied by the columns of B.

Example:

```
2 4     3 1
3 8  *  4 2
```

The result of this multiplication will be a 2x2 matrix because the first matrix has two rows and the second has two columns. To find the number in the first row and first column of the result, the first row in the first matrix is multiplied by the first column in the second matrix. The answer is:

```
2 4   *  3   =  (2 * 3) + (4 * 4) = 6 + 16 = 22
         4
```

The result in the first row and second column of the multiplication is the first row in the first matrix multiplied by the second column in the second matrix:

```
2 4   *  1   =  (2 * 1) + (4 * 2) = 2 + 8 = 10
         2
```

The pattern is continued and the answer is found to be:

```
(2 * 3) + (4 * 4)     (2 * 1) + (4 * 2)        22  10
(3 * 3) + (8 * 4)     (3 * 1) + (8 * 2)     =  41  19
```

Any individual element of the result of the multiplication of two matrices can be determined without finding the entire resultant matrix. For example, suppose the following matrices are to be multiplied and we wish to know the element in the second row and third column of the result.

```
1 2 3                 10 11 12 13
4 5 6        *        14 15 16 17
7 8 9                 18 19 20 21
```

To find the element in the second row and third column of the result, the second row of the first matrix is multiplied by the third column of the second matrix:

```
4 5 6        *        12
                      16
                      20

= (4 * 12) + (5 * 16) + (6 * 20)
= 48 + 80 + 120
= 248
```

Now we know the element in the second row and third column of the matrix resulting from the multiplication:

```
x x x x
x x 248 x
x x x x
```

Multiplication of two matrices is only possible when the number of columns in the first matrix equals the number of rows in the second matrix. Matrix multiplication is not commutative. For instance, if A is a 3x4 matrix and B is is a 4x5 matrix, it is possible to multiply A * B because A has 4 columns and B has 4 rows. However, it is not possible to multiply B * A because B has 5 columns and A has 3 rows. To make it clear why this multiplication can not be performed, let's try to do it.

```
1  2  3  4                    13 14 15 16 17
5  6  7  8                    18 19 20 21 22
9 10 11 12                    23 24 25 26 27
                              28 29 30 31 32

  A                             B
(3x4)                         (4x5)

                B * A

13 14 15 16 17                 1  2  3  4
18 19 20 21 22          *      5  6  7  8
23 24 25 26 27                 9 10 11 12
28 29 30 31 32
```

In the first row and first column of the result matrix the answer would be (13 * 1) + (14 * 5) + (15 * 9) . . . We can not continue because there are too few rows in A. It should now be obvious why the number of columns in the first matrix must equal the number of rows in the second.

The size of the matrix resulting from the multiplication of two matrices is determined by the number of rows in the first matrix and the number of columns in the second matrix. For example, if a 6x5 matrix is multiplied by a 5x3 matrix, the result will be a 6x3 matrix.

References

[1] Jacobo Carrasquel, Dennis R. Goldenson and Philip L. Miller.
Competency Testing in Introductory Computer Science: The Mastery Examination at Carnegie-Mellon University.
In Harriet G. Taylor (editor), *The Papers of the Sixteenth SIGCSE Technical Symposium on Computer Science Education*. Association for Computing Machinery Special Interest Group on Computer Science Education, ACM, March, 1985.
Abstract only.

[2] Phil Miller and Mary Shaw.
Proposal to Reorganize the Introductory Computer Science Courses.
February, 1980.
Internal memorandum.