Gifted Students' Individual Differences in Computer-Based C Programming Course

Tammy Rosenthal* and Patrick Suppes**

* Education Program for Gifted Youth, Stanford University, and Computer Science Department, Holon Academic Institute of Technology, Israel.

** Education Program for Gifted Youth, Stanford University.

1. Introduction

The Education Program for Gifted Youth (EPGY) at Stanford University is dedicated to developing and offering multimedia computer-based distance-learning courses. The EPGY courses use multimedia technology to present lectures consisting of synchronized sound and graphics. The courses have been designed to be used in those settings where a regular class cannot be offered, either because of an insufficient number of students to take the course or the absence of a qualified instructor to teach the course. In this way it differs from traditional applications of computers in education, most of which are intended to be used primarily as supplements and in conjunction with a human teacher. For early examples of complete courses (Suppes 1980) EPGY courses are designed so that gifted students can take the courses at their homes without the need to leave their regular activities or go out-of-school hours to a special classroom for this purpose. The software presents demanding exercises and quizzes that require students to display a mastery of the subject before advancing to the next lessons. Instructors help students and parents by telephone and electronic mail. They also provide extra virtual classrooms over the Internet to clarify and discuss relevant topics.

The origins of this work can be traced to the development of the first computer science courses for distance learning at EPGY by the authors in 1998. Earlier efforts at Stanford are described in Suppes (1981). After a few months of running the course for the pilot student groups, it became evident that the performance of students in the course varied greatly. The study of the nature of these individual differences is the focus of this study.

1.1. Background and Preliminary Considerations

Getting Started – The Difficulties We Faced

Since computer-based courses in computer science (CS), and in C as the first language, especially for distant-learning gifted youth was a new topic in a new subject matter, we had to consider several crucial issues prior to the detailed design and curriculum considerations, in order to establish a pedagogical scheme that would work in this setup. The difficulties we faced resulted from the following factors:

- The need to find a good substitute for the traditional computer lab found in schools. This also often held for the human instructor.
- A lack of prior models for such constellations of CS courses to learn from and compare with. We could find courses with partially similar features: A distant learning course in Pascal as the first language for adult undergraduates, a Web based C course for adults which consisted of plain texts and little or no multimedia components (see Web References: Open University of Israel, Stanford Online).

- We could not take full advantage of the automated evaluation software of EPGY as done in mathematics or physics (Ravaglia, Sommer, Sanders, Oas 1999), since C programs, the most significant factor in the students' workload, could only be evaluated manually by the tutor at that time. This increased the tutor's workload significantly in comparison to courses that have automated evaluation.
- Teaching C as the first language is an independent issue. The question which programming paradigm should be the first to be taught in CS is controversial and has been discussed often in the literature (Harel 1991). In many CS faculties of various universities (such as Stanford (Web References: Stanford CS)) the imperative paradigm was chosen to be the first one. Others, such as MIT (Springer, Friedman 1989) chose to start with the functional paradigm with SCHEME as the first language. However even for the universities that chose to start with the imperative paradigm, for many years Pascal was the preferred choice because of its characteristics (Cooper 1993). Only within the last couple of years did several universities (including Stanford) start with C. The reasons to switch to C were mainly practical; C is very popular and provides background for C++ and object-oriented programming. The specific characteristics of C are such that it usually takes more time for a beginner to grasp the fundamentals of programming in general, as well as to master the essential syntax of C to enable writing the first programs. For example, in standard C the basic operation of reading values is done by the "scanf" function that includes the use of pointers. Pointer variables store addresses of other variables. This concept of address is considered hard to grasp, especially at the beginning stage where so many concepts are new. Therefore, pointers traditionally are being taught in CS introductory courses towards the end of the curriculum (Web References: Stanford CS, HU CS). However, since reading values from the user is such a common operation needed even for early programs, it was important to find a way to bypass this complex use of pointers and to enable students to carry out simple operations in a straightforward manner.

Preliminary Considerations

In developing the initial computer-based C course at EPGY we had to define directions and working methods in advance. Our preliminary design strategy was aimed to address the problems on the one hand and to provide a suitable framework for very young students on the other hand. We needed to develop a scheme that would compensate for the lack of hands-on instruction while taking advantage of the special features of the technology-based instruction as being a major factor for increasing the diversity and richness of learning (Suppes 1996).

Perhaps our most important goal was to establish a user-friendly, encouraging and pleasant framework for the young remote students who were being introduced for the first time to programming challenges. We wanted to emphasize the joy and satisfaction of programming over the technical details involved, namely not to "kill" the motivation and delight but rather cultivate it. This is an important concern for any responsible curriculum designer. However, in traditional

programming courses the physical presence of the teacher in the class lab can compensate, since he is able to give immediate attention and personal help to any student who is stuck with debugging his first programs. Therefore establishing a scheme and concept for teaching the first computer-based programming course for young children in this setup was a challenging task. We came out with several preliminary design decisions.

- In accordance with EPGY practices, we decided to take advantage of the individualized instruction that computers can offer. For example, computers can give immediate attention to individual responses. Computers can also correct responses and convey information about their character, especially when the students' answers are incorrect. Computers can adapt the pace of instruction in delicate and subtle ways to the individual student pace of learning. (Suppes 1996). EPGY's system uses these features and also stresses the virtue of the student's actively participating as opposed to passively listening. Our goal here was to follow and implement EPGY teaching methodology of short lectures (average length of 2-15 minutes each), many exercises and activity for students, and automated feedback to the student as much as possible, along with a supportive human tutor when needed.
- We decided to include many *gradual* programming assignments. Many academic CS courses include 6-8 larger assignments (Web References: Stanford CS, HU CS). We included 29 programming assignments, starting from very basic and trivial ones and gradually going to more complex ones. Giving gradual assignments to students is based on former experience of the first author with teaching programming courses to young children. Giving many gradual assignments is also important for the students, since they are able to create real, albeit simple programs by themselves very early in the course.
- We made it a point to start as soon as possible with writing programs and to defer theoretical topics. This idea is based also on former experience with teaching programming to children. It seems more efficient to let young students write programs and learn about CS from their own experience; Then, at a much later stage, to go back to the theoretical background which now seems more familiar and makes more sense. In particular, in C11A the theoretical basics were deferred to chapter 7.
- We selected a suitable compiler* to work with - simple to operate, reasonably low priced and with a good reputation for users' support services.

---

* A compiler is a special program that checks whether syntax errors were made in the code and that translates the high-level code into machine-language executable code.

- We decided to include interactive sessions, such as explanations as to how to use the compiler, sample run programs included within the lectures, embedded questions as part of the computer-based lectures with automated feedback to students. The goal here was to substitute the traditional computer lab hands-on environment.
- We decided to include extended libraries, in order to overcome some difficulties of teaching C as the first language. These extended libraries include useful operations required for programs, such as functions that deal with reading inputs from the user and other facilities that are not part of the standard C language. Thus, the usage of such operations became simple for the students. With these libraries in place, the students do not need to know how each of these functions is implemented. They only need to know it as a "black box"; include the library name at the top of their program, understand what it does in a high-level manner and how to operate it, and then call the desired function within their program.
- We had to choose a textbook for the course. The textbook we chose was "The Art and Science of C" by Eric Roberts (Roberts, 1995). It teaches C as the first programming language and includes extended libraries to make it easier for the beginning programmer.
- We decided to include many theoretical exercises with automated scoring and feedback to the students to help them learn new concepts and practice the details of the syntax, before they actually began to write the programming assignment itself.

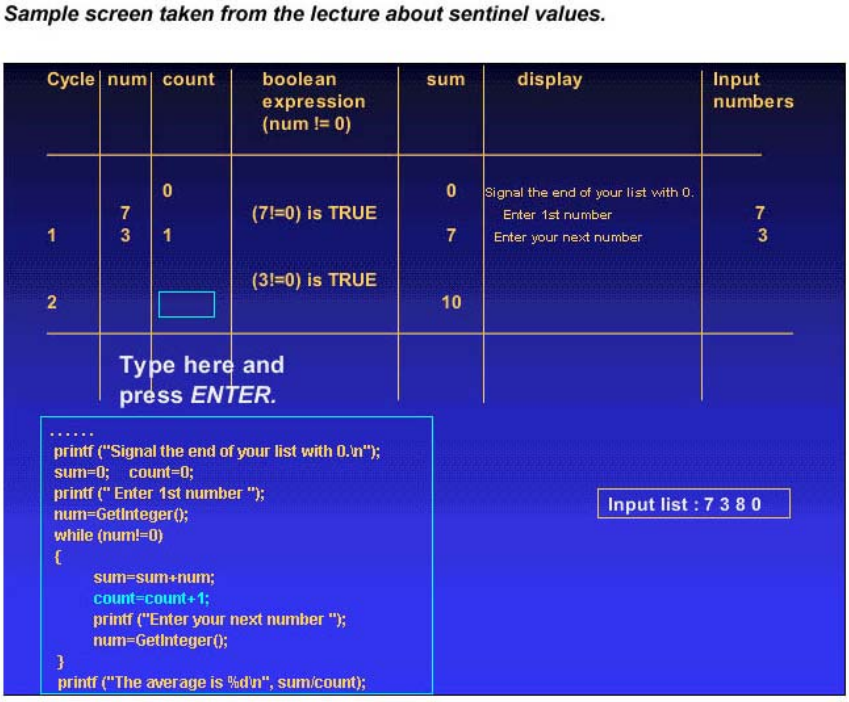
1.2. Student Selection

A total of sixty students took part in this study.
Forty students took the first five sessions of the EPGY introductory C course (C11A) and completed the course successfully during 1998-1999. These forty students consisted of twenty-six individual remote gifted students and fourteen from a private school. The twenty-six individual students took the course in their own homes and contacted their instructor via e-mail. The fourteen students from the private pilot school were in the 6^{th} and 7^{th} grades and had an instructor in the classroom lab while taking the course. The analysis of individual differences in student programming performance was obtained from applying evaluation-methods to student-programs. It was done with respect to this group.
The second pilot group consisted of twenty students who took the course in 2000-2001. These twenty students provided feedback on the evaluation methods.

1.3. Course Setup

The course contains 55 short lectures of 5-10 minutes each, 250 quizzes given by the software at the end of the lectures with automated immediate feedback to the student, and 29 programming assignments that the students are required to write, run and then send to their tutor by e-mail as attached files. Each multimedia lecture typically consists of 40-80 frames with synchronized sound (audio) that runs as a whole movie. Figure 1 presents screenshots from the lectures.
Each student receives 3 CDs: one system CD which contains the EPGY software, and two lecture CDs, as well as a package of written materials. The student needs first to install the system CD. Then the software instructs her how to progress. For example, the software tells the student which lecture CD to insert. It also specifies the title of the next lecture. The software actually plays the lecture itself and once the lecture ends it presents automated exercises and quizzes to the student on the lecture topics. For each answer the student enters, the software displays an automated message indicating if the answer was correct along with a detailed explanation about the correct answer and error type if one occurred. Figure 2 presents samples of exercises regarding syntax taught in the lecture. All the answers entered by students on the quizzes are stored by the software in a special file.

FIGURE 1
SCREEN SHOTS FROM LECTURES



Sample screen taken from the introductory lecture about "How do loops operate?"

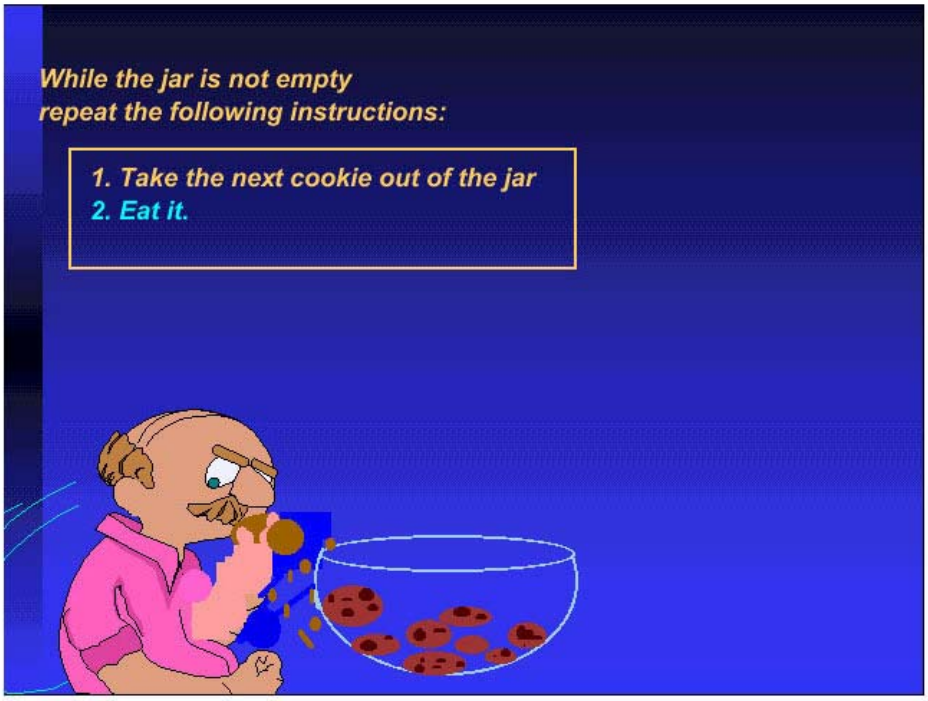


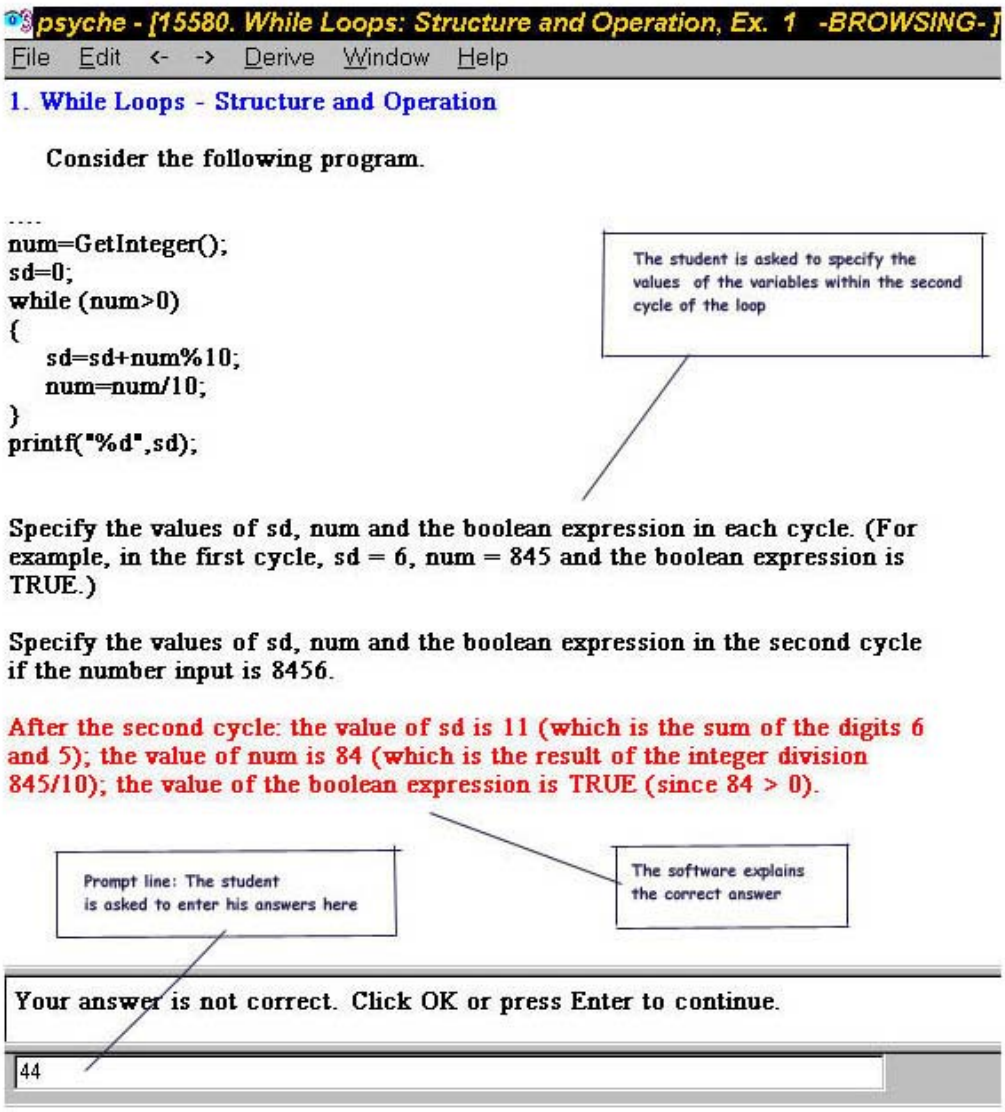
FIGURE 2
SAMPLES OF EXERCISES REGARDING SYNTAX

Below is a program which is intended to compute and display the factorials in the range of 1 to 10. The program's output should be . . .

The program contains some errors. Locate and correct them.

```
#include <stdio.h>
#define upperLimit 10
main ()
{
    double product=1,i;
        printf ("%s \t %d\n", "number", "factorials");
        for (i=1;i<=limit;i--)
                {product*=product;
                printf ("%d\t%d\n", i, product);}
}
```

This line should be: int product=1,i;



1.4. The Structure and Curriculum of the Course

The main objectives of C11A are to provide students with a general introduction to computer science and programming within the context of learning the C programming language, to cover the basic introduction to computer science and to fundamental programming concepts, and to enable students to write programs in C. C11A covers the following topics: Data types, program structure, statements and expressions, I/O operations, control statements, computer architecture and algorithms. The students are required to write approximately 30 programming exercises during the course, in addition to theoretical exercises and a test. Typical programming assignments during this course are: computing the solutions of quadratic equations, finding prime or perfect numbers in a given range, and processing words in a passage of text. There are no prerequisites (see Web References: C11A for students).

Once a week, the software reminds the student to send her "report" (done by choosing the "report" option from the software menu). The report contains all the answers to the quizzes. Once the student sends the report it is received and processed by EPGY and then transferred to the appropriate tutor for further follow up. In fact, students get two types of feedback on their answers – an automated online feedback from the software and a manual feedback from the tutor (by e-mail). The software registers the location students have quit and thus next time they wish to work it continues from this current location. For programming assignments, the software refers the student to the specifications in the assignment sheet, included within the course materials package students receive. The students work separately on their programs and send them as attached files for manual evaluation by the tutors. Table 1 presents the course curriculum and the suggested schedule for the course.

TABLE 1
CURRICULUM AND SCHEDULE OF THE COURSE

1. Introduction — Week 1

lectures	course numbers
1.1. Objectives and overview of the course.	1510
1.2.1. What is C ? The development of the computer languages.	1511
1.2.2. The development of C . The compilation and linking process.	1512

2. The compiler — Week 2

lectures	course number	Programming Assignments
2.1. Getting started: Installing the compiler and creating a new folder.	1520	
2.2. Running the "Hello World" program.	1521	
2.3. Running the "Hello You" program.	1522	prnt20.c

3. Fundamentals of C — Week 3, 4

Lectures	course number	Programming Assignments
3.1.The structure of C programs - 3.1.1. The general structure.	1530	Week 3: first.c, hello1.c, birthday.c,
3.1.2. The statements in main.	1531	
3.2. Data types: integer, Boolean.	1532	
3.3. Variables - Names, declarations.	1533	
3.4. The assignment operator.	1534	
3.5.Expressions - Arithmetic, Boolean - 3.5.1.Arithmetic expressions. 3.5.1.1. Operands and operators.	1535	Week 4: addMul.c + *extended library* volume.c, average4.c
3.5.1.2. Constructing complex expressions with the operands.	1536	
3.5.1.3. The expressions in the addMul.c program.	1537	
3.5.2. Boolean expressions.	1538	
3.6. Simple input and output.	1539	
3.7. Summary.	1540	

5. Basic control statements. — Weeks 5,6,7,8

Lectures	Course num.	Programming Assignments
5.1. The if statement - 5.1.1. Basic if statements.	1550	Week 5: max.c, positive.c, min3.c,
5.1.2. if/else statements.	1551	
5.1.3. Multiline if statements.	1552	
5.2. The ?: operator	1553	
5.3. Cascading if statements - 5.3.1. Structure and operation.	1554	Week 6: summer.c, binary.c, week.c,
5.3.2. The if/else blocking rule.	1555	
5.4. The switch statement.	1556	
5.5. The while statement. - 5.5.0. How do loops operate?	1557	Week 7: mulByAdd.c, minMax.c, power.c, circles.c,
5.5.1. While loops - structure and operation.	1558	
5.5.2. Abbreviations.	1559	
5.5.3. Using sentinel values.	1560	
5.5.3.1. A different approach - Infinite loops.	1561	Week 8: square50.c, star8.c, prime.c, reverse.c
5.5.4. Drawing circles.	1562	
5.6. The for statement - 5.6.1. For loops - structure and operation.	1563	
5.6.2. Nested for loops.	1564	
5.7. Evaluation of Boolean expressions in C.	1565	
5.8. Summary - Part 1	1566	
5.8. Summary - Part 2	1567	

7. The computer behind the scenes. Week 9

Lectures	Course numbers
7.1. Hardware - 7.1.1. Components.	1570
7.1.2. Memory.	1571
7.1.3. CPU	1572
7.1.4. I/O devices.	1573
7.1.5. Secondary storage devices.	1574
7.1.6. A historical point of view.	1575
7.2. Software.	1576
7.4. Computer languages.	1577
7.4. Summary - the general picture.	1578

8. Data types, constants, algorithms and output displays. Weeks 10, 11, 12

Lectures	course	Programming Assignments
8.1. Constants.	1580	Week 10: factorial.c, celsFahr.c, Week 11: quad.c, GCD1.c, euclid.c, Week 12: volumes.c, longStr.c, games.c
8.2. The floating-point data type.	1581	
8.3. Problem solving - algorithms and flow charts - 8.3.1. The quadratic equation.	1582	
8.3.2.1 GCD – Brute Force.	1583	
8.3.2.2. Euclid's Algorithm.	1584	
8.3.3. Flow charts.	1585	
8.4. The string data type - 8.4.1. Definition.	1586	
8.4.2. Simple operations with strings.	1587	
8.5. Controlling the output display.	1588	
8.6. Summary	1589	

----------Exam----------

2. Students' Performance

Within the C11A course the students are required to submit reports (which contain their answers to the theoretical exercises), send their C programs and take an exam. In this section, we analyze students' individual differences in their performance in the reports, and in their programming exercises. We also look at the time students spent on completing their programming assignments.

2.1. Students' Performance in Reports

The theoretical exercises include 70 exercises consisting of 254 questions about the lecture topics. The goal of these exercises is to provide students with practice to grasp new concepts, syntax and semantics taught in the lecture, as well as to prepare them for the corresponding programming assignments.

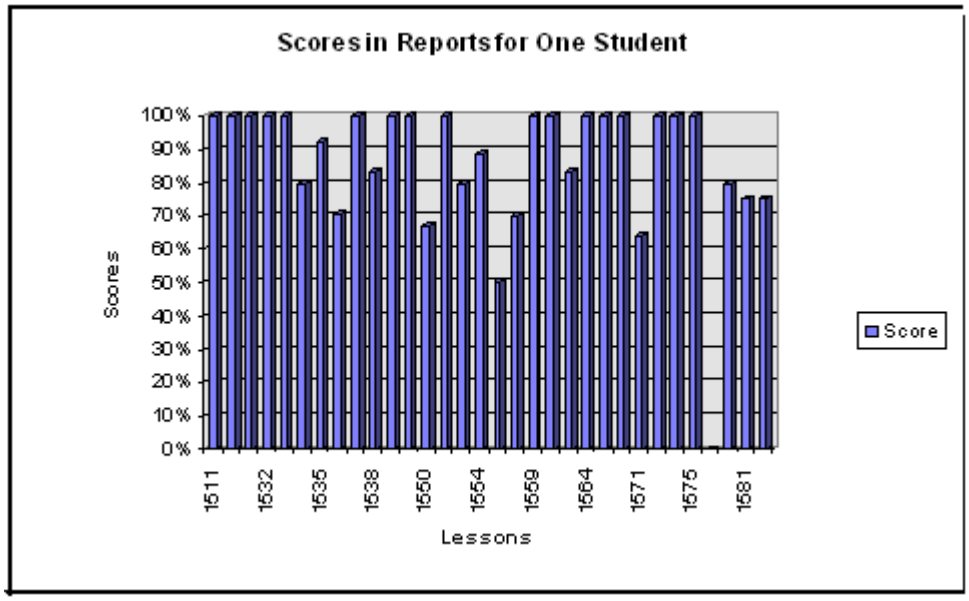
Each week the students are required to e-mail a report that includes all their work. This report is received by the server, processed, and updated in the student's record in the data base. The processed reports are then transferred to the tutor for further follow up. These processed reports are the data used for the following analysis. Figure 3 presents the average scores of one student for the whole set of the theoretical exercises of C11A.

FIGURE 3
SCORES OF ONE STUDENT IN THE ENTIRE SET OF REPORTS

1511	1512	1530	1532	1533	1534	1535	1536	1537	1538	1539	1555
100%	100%	100%	100%	100%	80%	92%	71%	100%	83%	100%	50%

1540	1550	1552	1553	1554	1558	1559	1561	1563	1564	1565
100%	67%	100%	80%	89%	70%	100%	100%	83%	100%	100%

1570	1571	1572	1573	1575	1576	1580	1581	1582
100%	64%	100%	100%	100%	0%	80%	75%	75%



Each of the scores here represents the score of the student for all the exercises of the lesson. For example, in lesson- number 15580 the student received a score of 70%. This is the score for three exercises consisting of twenty-two questions that are included in lesson 15580. This score indicates that the student has answered correctly on 70% of the questions in lesson 15580. Altogether, this set represents 254 questions. Table 2(a) and Table 2(b) present the average scores of the students' reports within the two pilot groups (home and school students).

TABLE 2(a)
SCORES IN REPORTS FOR THE HOME STUDENTS

student's num	Average Scores in Reports
1	88
2	87
3	91
4	88
5	72
6	84
7	87
8	94
9	73
10	94
11	94
12	91
13	87

TABLE 2 (b)
SCORES IN REPORTS FOR THE SCHOOL STUDENTS

student's num	Average Scores in Reports
1	69
2	70
3	73
4	94
5	89
6	71
7	72
8	73
9	84
10	83
11	90
12	84
13	82

14	87
15	86
16	86
17	84
18	100
19	91
20	91
21	91
22	91
23	94
24	94
25	94
26	94

Average	88.96
min	72
max	100
median	91

14	75

Average	79.21
min	69
max	94
median	78.5

In Tables 2(a) and 2(b) the average score for each student is computed for the entire set of theoretical exercises of C11A (254). From these figures we can see that the total average score of the school's group is lower than the score of the home students (79.21% as opposed to 88.96%). Also, the results of the minimum, median and maximum average scores are higher for the home students. There are several plausible explanations for this difference in the performance between the two groups:

- Most of the home students had EPGY math background, but all the school's students had only the school's math curriculum when starting the course.
- As part of the prerequisites for taking EPGY math courses, most of the home students had passed test scores indicating their higher ability. In order to be admitted to the EPGY math program they had to score within the top fifteen percent of mathematical ability as based on commonly accepted standardized test scores, including the SAT, PSAT, ACT, AP exams, or other achievement tests commonly utilized.
- The home students were not limited as to the time they spent on the reports since they did it at their home computers. The school's computer time available to students was limited.

The scores in general are relatively high (average of 79% and higher for both groups). The questions were not tricky. They were aimed to help in absorbing the details needed to assist students in writing the corresponding programming assignments.

2.2. Students' Performance in Programming Assignments

In order to analyze students' performance in programming, three automated methods were developed:

- A criteria set for programs' quality-evaluation. The set consists of ten criteria, four of which are used to evaluate core skills: *operations, functions, len* and *mem*. *Operations* counts the total number of operations within the program as executed for a given input. *Functions* counts the number of calls to functions as executed for a given input. *Mem* measures the

amount of memory storage in bytes required to store the variables of the program. *Len* measures the length in lines of the program excluding comments or blank lines.

- Weighted scores for each programming assignment.
- Performance scores computed for each student. These scores express the students' programming performance in the entire set of the programs that were submitted.

The analysis relate to four main aspects:

1. Results obtained from applying the criteria to students' programs.
2. Correlation measures between the more significant criteria.
3. Scores computed for programs.
4. Scores as computed for students' performance in the entire set of assignments examined.

For each of these aspects significant individual differences in students' performance were obtained as well as differences between the two pilot groups (home and school students). The main results are presented below.

The full analysis is given in (Rosenthal, 2004).

The results we present relate to 320 programs for 8 types of assignments that were submitted by all the students in the pilot (home and school students).

Applying the Criteria

The results obtained from applying the criteria to students' programs demonstrated significant differences in students' performances. For some of the assignments, the criterion values differed between students by an order of magnitude. For example, in an assignment named *mulByAdd* the *operations* count range was 12-20032. In the *prime100* assignment the *mem* range was 2-218 bytes. In the *perfect* assignment the *functions* range was 22-1764 calls. Table 3 presents the *operations* count for three students over eight assignments.

TABLE 3
OPERATIONS COUNT FOR THREE STUDENTS OVER EIGHT TYPES OF ASSIGNMENTS

Students' numbers	Factorial	Euclid	MulByAdd	Min3	Prime50-100	GCD	Reverse	Quad	Average operations
1	33	18	17	16	450	122	11	17	**85.5**
12	121	14	7980	18	462	142	26	22	**1098**
school 3	421	39	20012	29	4053	175	49	27	**3518.57**

The range between these students varies greatly in breakdown of the assignments as well as in their average count (for eight assignments). For example, for student 1 the *operations* range was: [11-450] whereas, for school student 3 the range was much larger: [29-20012]. This large diversity relates to the variety of algorithm choices, implementation methods and details involved in programming tasks. It indicates the extent of the complexity involved in program evaluation. For example, students who have implemented the *prime100* problem by using the *sieve of Eratosthenes* algorithm had an *operations* count of 300-500 operations; whereas a more straightforward implementation with nested loops that run on the whole range of numbers yielded 5000-10,000

operations. The most significant differences measured between students were with regard to the criteria: *operations, functions* and *len*.

For example, the average count of operations for all the pilot students ranged from 44 to 3519. Table 4 presents the average count for these three criteria for the two groups (home and school students).

TABLE 4
AVERAGE COUNT OF OPERATIONS, FUNCTIONS AND LENGTH FOR 320 PROGRAMS

	Average operations	*Average functions*	*Average length*
26 Individual students	1413.56	18	20.68
14 School students	2875.32	27.23	32.10

Table 4 shows that on the average the home students performed better (for these three criteria).

Correlation Tests between the Criteria

In correlation tests for the more significant criteria (*operations, functions, mem* and *length*), we found a significant positive correlation (0.6) between *operations* and *length* in some of the more basic assignments. Namely, students who tended to write shorter programs used on the average fewer operations and vice versa. For the more advanced assignments no significant correlation occurred. Significant differences occurred in the correlation results between the two pilots (home and school students), as well as in the ranges of the criteria results. The correlation results for the school were more significant than for the home students. Also the criterion range was smaller for the school. Namely, the school's students' programs were more similar to each other and contained fewer differences. Examples of the differences between the ranges of the criteria for the two pilot groups are given in Table 5.

TABLE 5
THE CRITERIA RANGES

Criteria	**Assignment**	**Home students' range**	**School students' range**
operations	*reverse*	11 - 63 operations	32 - 63 operations
functions	*Euclid*	3 - 14 calls	13 - 17 calls
mem	*factorial*	6 - 66 bytes	6 - 20 bytes

We found that the diversity in program performance among the gifted-youth sample (home students) was larger than the diversity among the school-students sample.

Programming Scores

We found significant individual differences between programs' scores depending on the specific problem domain of the assignment. For some types of assignments the differences between students' scores were very drastic while for others milder. For example, the scores for the *mulByAdd*

assignment ranged from a minimum of 1007 to a maximum of 801953. In the GCD assignment the range was much smaller: [4195 – 8442]. We also found that students' performance level was not equal through the entire set of assignments. One student may have received a better score for one program and worse scores in others. But in Table 7 we can see that student 1 received the best relative score for the entire set of assignments, and yet in each of the corresponding assignments, except for one, his score is also best. (Note that scores with ** are better when they are lower.) We also identified differences between the ranges of the scores for the two groups (home and school students). For the school the programs were more alike and therefore contained fewer differences both in the criteria and scores' results. For example, the range of the scores as computed for the school in the *reverse* assignment was [min=2422, max=3932] whereas the range of the home students in *reverse* was larger [min=757, max=3650]. We found that the school students on the average received lower (and worse) scores than the home students.

TABLE 6
AVERAGE RELATIVE SCORES RESULTS FOR THE HOME ND SCHOOL STUDENTS

Average relative scores

	school	Home
min	19.46	33.44
average	42.94	91.99
max	76.87	134.99

To illustrate the individual differences in students' performance in their programs, Table 7 presents results measured for three students in the pilot. The student who got the best score for his performance (student 1), from among all the students in the pilot: the student who got an average score for his performance (student 16); and the student who got the worst score (school 3).

TABLE 7
A COMPARISON OF PERFORMANCE IN PROGRAMMING ASSIGNMENTS AMONG THREE STUDENTS

	relative	**average**	**average**	**average**	**average**		**Absolute scores for the assignments****							
Students	**scores***	**operations ****	**functions ****	**length ****	**memory ****		**min3**	**mulByAdd**	**prm50-100**	**reverse**	**factorial**	**GCD**	**Euclid**	**Quad**
(Best) **student1**	134	85.5	7.7	16	8.5		1047	1297	18541	757	1740	5306	1167	1497
(Average) **student16**	68	2609.2	23.7	31.6	15.2		1857	800907	15241	2476	11705	7637	2106	2391
(Worst) **School3**	19	3518.5	27.8	31.8	15.5		19516	801078	163831	3348	1997	7827	2511	2127
Average results for 40 students ***	74.8	1925.1	21.2	24.6	12.9		6206	491148	68961	2452	6361	6796	1794	2003

* In this computation scale, higher scores are better. The results are computed for 8 types of assignments and reflect the performance of students in their programs in terms of the difference to the average score of the entire group (of 40 students).

**In this computation scale, lower scores are better. The scores are based on the criteria results.

*** The average values here were computed for the entire group (40 students)

In comparing the performance of these three students, we see that student 1 who got the best relative total score (of 134) used on the average in his 8 programs fewer operations (=85.5), fewer function calls (=7.7), less memory bytes (=8.5), and wrote shorter code (of average length of 16 lines). The scores he received for most of his programs were better than the scores of student 16 and of the school student 3. The performance of student 1 (with an average count of operations of 85.5) is extremely good relative to the average count of operations of the entire group for all the 8 assignments (= 1925.1.) In the comparison between the performance results of student 16 and school 3, in most aspects, student 16 had performed better.

Correlation between Students' Age, Math background and Program scores

In correlation tests conducted between age of students and average scores in the entire set of programs examined, Table 8 presents the results were obtained.

TABLE 8
CORRELATION BETWEEN STUDENTS' AGE AND SCORES IN PROGRAMS

The pilot as a whole (40 students)	The home students (26 gifted students)	The school students (14 students from the 6th and 7th grades)
0.13	0.006	-0.53

We received a negative correlation (-0.53) for the school students. Namely, younger students got higher scores in programs. The instructor of the course in the school indicated that the younger class (6th graders) were more motivated, worked also at home on some of the programs and thus on the average performed better. This may explain this result of better performance scores in programs for the younger students in the school. For the entire pilot group (of 40 students) we got a significant positive correlation (0.7) between having a former EPGY math background and having higher average scores in programs. As a whole, the differences we got between the criterion measures as well as between the scores were often at least an order of magnitude (such as the range of *operations* in the *mulByadd* assignment: 12 - 20032). The reasons for this large diversity relate to the complexity involved in programming: diversity of implementation methods, choice of algorithms, enormous amount of details, and differences in characteristics of specific problem domains. For all the aspects examined (criteria, correlation between the criteria, programs' scores, students' performance on time scale, and performance-scores for students), the patterns we found are summarized below:

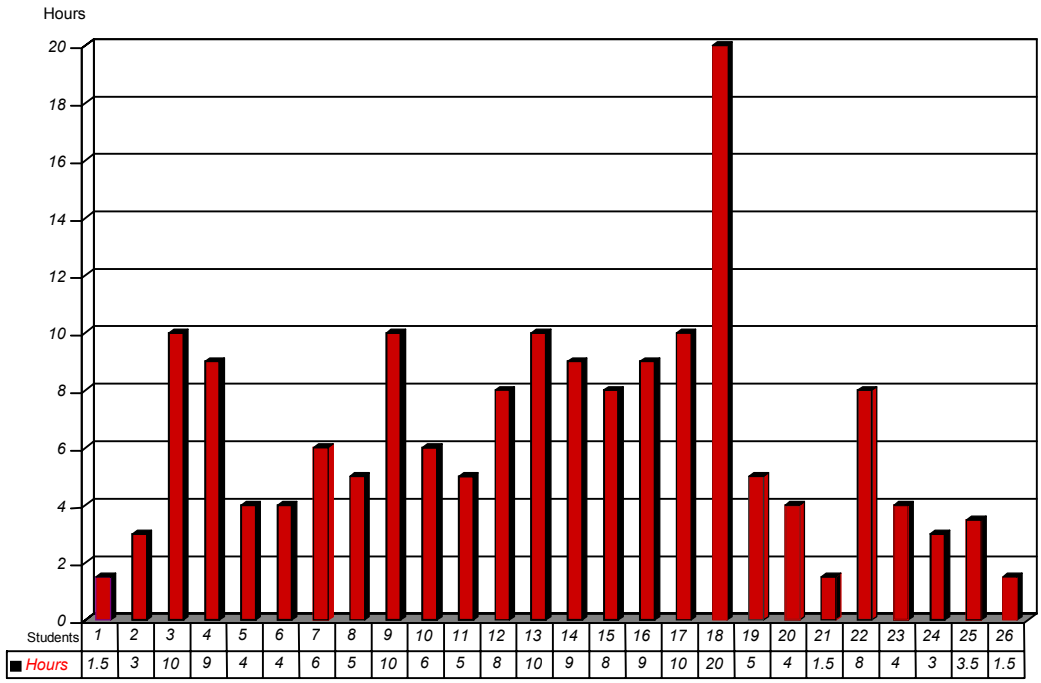
- We found correlation among students between the criteria of length and operations in some of the more basic assignments.
- We found that for students who got better total scores (for the entire set of assignments examined), their performance on time scale was smoother.
- We found that on the average the gifted students' population had performed better in programming than the school students.
- We found larger diversity in programs' performance among the gifted students than among the school students.

The large diversity we found for the gifted students' sample in programming skills is supported by similar findings on gifted-students' performance in other computer-based courses (Stillinger, Suppes, 1999), (Tock, Suppes, 2002).

2.3. Time Spent by Students on Programming Assignments

Figure 4 presents the average time spent on programming assignments per student, as based on students' own estimations from their evaluation forms. The figures represent the 26 individual (home) students of the pilot group. We can see significant differences between the average-time spent by students on programs (per chapter). It ranged from a minimum of 1.30 hours (for students 21, 1 and 26) to a maximum of 20 hours (for student 18).

FIGURE 4
AVERAGE TIME SPENT BY STUDENTS ON PROGRAMS



3. Calendar Time

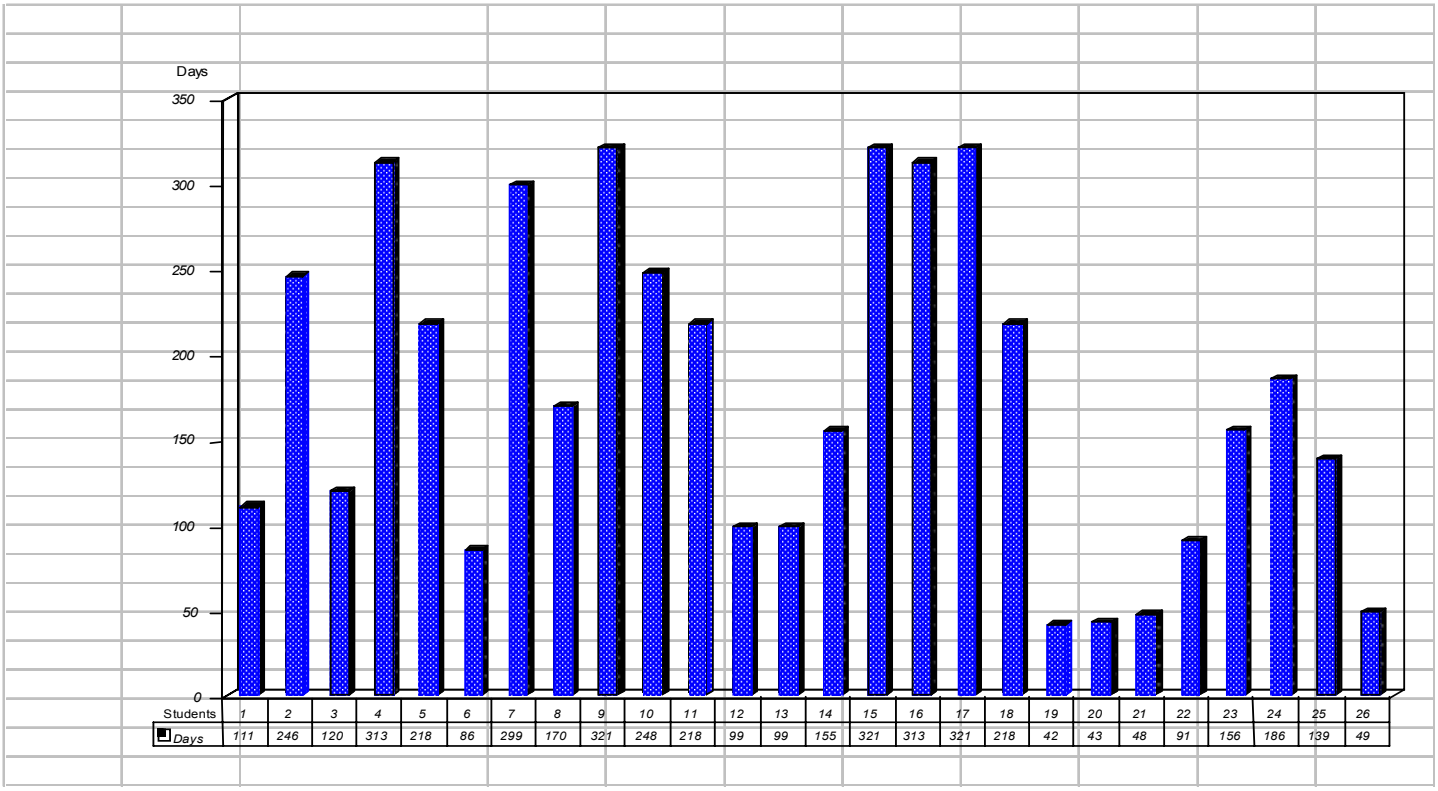
In previous studies it has been shown that *time* plays an important role in characterizing students' individual differences in learning (Suppes, Zanotti 1996). The *calendar time* is the number of days it took each student to complete the course as a whole. Table 9 presents the *calendar time* for the home students in a breakdown of the average, minimum and maximum time. The time is measured in days from the date the students started until they were done.

TABLE 9
CALENDAR TIME FOR HOME STUDENTS

Average calendar time	*Minimum calendar time*	*Maximum calendar time*
178 days	*42 days*	*321 days*

For the school students the time frame was strict and equal for all. The school students all started the course in mid October and completed it on June 20th 98. It took them 211 days to complete the course. Figure 5 presents the total number of days it took each of the home students to complete the course.

FIGURE 5
CALENDAR TIME FOR HOME STUDENTS



In Figure 5 we can see that student 19 has completed the course in the shortest time (=42 days) while student 9 took the longest time (=321 days). The calendar time varied from a month and a half to almost one full year. In computing the calendar time we took into account factors such as leaves, and sick days that we knew about. However, there is still room for errors. Moreover, since the students learned the course in different versions we assumed that differences in calendar times might occur also as a result of this fact; that is, that a student who took the course in its first version would face more difficulties and therefore take longer in completing it rather than students who took the course in its later versions.

Allowing for possible errors, it is interesting to note that the findings suggested the opposite trend. For example, for student 1 who took the 1st version of the course it took 111 days to complete it and he got a high score in his programs (135); Whereas for student 13 who took the 5th version it took 99 days to complete (12 days less) and his score in the programs was worse (58). These samples as well as the results obtained from correlation measures between the parameters of "*completion time*" and "*performance in programs*" - indicate that no correlation was found between *completion time* and *performance in programs*.

4. Students' Difficulties

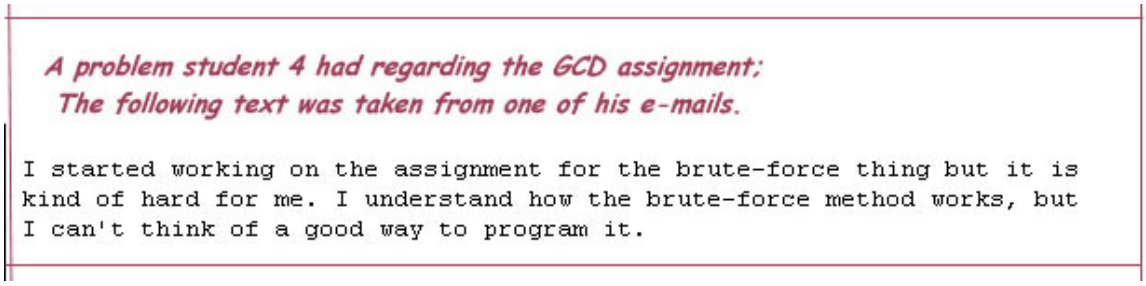
The analysis presented in this section is performed based on: Evaluation feedback forms completed by students by the end of the course, students' e-mails throughout the course, students' comments in the reports, and telephone conversations with students and parents. The main difficulties students faced were related to: translating problems into algorithmic steps; debugging programs; technical difficulties with installations required; and compilation of the programs.

4.1. Translating Problems into Algorithmic Steps

It turns out that the translation process from a problem stated in regular text to an algorithmic solution is problematic. The following is typical of introductory programming courses. The details, syntax and semantics may be very clear. Yet, when the student needs to sit by himself and translate the problem into algorithmic steps (for a computer program) it is much more difficult. Several CS references such as (Harel, 1992) refer to this issue. Also, in some introductory programming courses additional curriculum is added at the beginning of the course to introduce and deal with algorithms and algorithmic approaches. For example, in the Stanford CS introductory course (CS106a) a similar approach is taken; before starting with C, the students read an introductory chapter on "Karel the Robot" to introduce students to algorithmic thinking (Pattis, 1995).

Based on students' feedback, the lecture topics and their presentation were clear. Also, there were almost no questions asked by students regarding the lectures' content. Yet, the task of understanding fully the specifications given for the programming assignments and translating the problems into computer programs were less straightforward. An example is given in Figure 7.

FIGURE 7
A SAMPLE FROM STUDENTS' QUESTIONS ABOUT TRANSLATING A PROBLEM INTO AN ALGORITHM



4.2. Problems Students Faced with Debugging their Programs

Most of the difficulties students faced were in debugging* their programs. Typical bugs might occur as a result of programs' failure to treat extreme cases or wrong inputs properly.

For example, computing expressions such as (x/y) without checking first whether y is equal to zero might crash the program in cases for which y is equal to 0. Also, trying to access array cells in locations for which the index is out of the array range will result in unpredictable behavior. Other typical bugs occur when a loop runs infinitely since the boundary conditions were not set correctly. Typical bugs in C also might result from incorrect use of the operators = and = =. The = operator is used for assignment, while the operator = = is a Boolean operator (that returns FALSE or TRUE). The = = operator checks whether two values are equal. Mixing between these two operators is a common mistake that creates bugs.

Within C11A students faced debugging problems for which they asked help from their tutor. Typically, some students had many problems and asked many questions while others almost none. Figure 5 presents the number of debugging questions the home students asked. For the school's students debugging questions were dealt with immediately. Students the tutor in class for help. Therefore, the number of such calls for the school was not measured.

As we can see in Table 10, there are significant differences with regard to the number of debugging questions asked by students. It ranges between 0 to 31 questions. Also, the complexity level of questions can be very different.

---

* Debugging is the process of finding and correcting bugs; Bugs are mistakes in the logic of the programs. Bugs occur after the program is clean from syntax errors. Then one may discover that the program comes up with incorrect answers or fails to produce answers at all. This means that there are bugs – errors in the logic of the program.

Syntax errors are different, and result from breaking the syntactic rules of the language. Each language has its own vocabulary and a set of grammar rules. If we have violated the syntactic rules, the compiler displays an error message and then we have to go back to our program and correct it.

4.3. Technical Difficulties and Installation Problems

Students faced technical difficulties, in particular during installations. Most problems occurred in the first versions of the course. Within C11A there were several installations students were required to perform: Installing the EPGY software, installing the compiler and including the extended libraries in C projects. In the first sessions we also included updated versions of the EPGY system on the Web to correct bugs and errors. Students were notified and were asked to download these updates from the Web. Altogether, for most of the students the installations were smooth (according to their feedback forms), especially the installation of the compiler, since this topic was covered in detail in an interactive manner within the lectures of the course. The lectures included step-by-step instructions with interactive parts for the students to do it by themselves. Some students had difficulties in including the extended libraries or with managing their files in general. A few students asked for help to operate the EPGY software, in particular they needed help in the procedure of sending reports. For the school students, the difficulties resulted also from the massive use of the lab's computers by other classes.

4.4. Compilation Problems

Occasionally, students asked for help in finding their syntax errors. However, this improved quickly as the students accumulated more experience in mastering the language and in locating syntax errors. The main set of problems the school students faced focused on different issues:

- Technical problems with the lab's computers. Some difficulties resulted from the massive use of the lab's computers by other classes.
- Motivation problems, since this was a compulsory class with no choice. This may have affected the lower rates of completion (14 out of 20) among the school as opposed to the home students.
- Limited computer time to work on reports as well as on programming assignments. The school had worked on the EPGY program only in class sessions in the lab.
- Lack of appropriate math background was more crucial within the school since all the students had only the school's math curriculum as their background. For the younger class some concepts were less familiar and therefore, they had more difficulties in solving problems such as, quad, GCD or Euclid. This is supported by the results obtained from the correlation tests conducted. We found significant positive correlation (0.7) for the entire pilot group between having a former EPGY math background and between program-scores.

TABLE 10
NUMBER OF DEBUGGING QUESTIONS ASKED BY HOME STUDENTS

Individuals	Debug. Questions
1	0
2	5
3	0
4	11
5	7
6	3
7	31 *
8	1
9	3
10	2
11	1
12	2
13	4

Individuals	Debug. Questions
14	1
15	5
16	8
17	6
18	1
19	1
20	0
21	0
22	0
23	1
24	3
25	4
26	2

average	3.9
minimum	0
maximum	31

* For student 7, several questions were asked for one type of assignment, plus several questions from the mother too. Altogether the debugging questions of student 7 related to 15 types of assignments within C11A.

5. Correlation Measures

In order to examine whether there is a connection between different parameters of students in learning C, we performed correlation tests. The results are presented in Table 11.

Gender: programs

We did not find any significant connection between students' gender and students' scores in their programs. However, since the correlation results were negative (-0.29 within the entire pilot group), it seems that there is a slight tendency indicating that the boys, on the average, got better scores in their programs than the girls. We should note, though, that the home students group consisted of 22 boys and 4 girls. On the average the home students got much higher scores for their performance in programs than the school students. This difference in performance may have affected this negative correlation result as well.

Age: programs

Within the school group we got a significant negative correlation result (-0.53). According to the school's tutor the younger class of the 6th graders was more motivated and wrote better programs than the older class. Some of the younger students took some of the problems to write drafts for programs at home as well. This may explain why the younger class received higher scores in their programs than the older class.

Math: programs

For this pair we got a significant correlation result (0.7) for the entire pilot group. Having EPGY math background was strongly correlated to getting better scores in programs. Strong math

background is helpful for programming. In particular since some programs were aimed to solve mathematical problems such as computing quadratic equations, or finding the GCD according to Euclid's algorithm. However, we should note that most of the home students (21/26) had EPGY math background. All the other home students as well as the school students had the school curriculum math as their background.

Completion: programs

We did not get any significant correlation for this pair. However, it is still interesting to note that for the entire pilot as well as for the home students group a negative correlation (-0.3) was measured. This suggests that students who managed to complete the course in a shorter time received higher scores in their programs. This finding makes sense in programming tasks. Having fewer breaks between assignments and lectures is an advantage. Students absorb the new concepts and while they are still fresh they apply them. They tend to forget less of the details of the semantics and syntax if they work more sequentially. Moreover, students who were more motivated and enthusiastic and who enjoyed more the programming challenges were eager to progress faster. For example, student 21 got the second best programming score from among the entire sample. He completed the course in 48 days which is a very short period relative to the range of the group [42 - 321 days].

Reports: programs

For this pair, a significant positive correlation result was measured for the entire pilot (0.52) as well as for the school (0.61). Namely, higher scores in programs were correlated to higher scores in the reports. This makes sense. The reports consist of questions that are aimed to give practice of new concepts, syntax and semantics presented within the corresponding lecture. The programs require students to implement the same concepts. If students master well the new concepts it helps in the implementation.

Gender: reports

We did not get a significant correlation for this pair. The negative figure of (-0.4) which was measured for the entire pilot, suggests that the male students tended to perform better in their reports than the female students. Since the home students consisted of mostly boys (22/26), and since they had higher scores than the ones of the school, this tendency may be related.

Completion: reports

We got a significant positive correlation (0.56) for the entire group between the completion time and the scores in the reports. Students who took more time to complete the course had scored better in their reports. On the other hand, for the home students there was no significant result. On the contrary the result we got was negative (-0.36). Namely, home students who completed the course in a shorter time tended to get better scores in their reports. Therefore, since we have contradictory trends we can not indicate a definite pattern here.

Version: reports

We got a significant positive correlation for the home students (0.58) and less significant but relatively high result (0.49) for the entire pilot. This correlation between the version number and students' scores in the reports makes sense. The reports are part of the EPGY software. The first versions contained more errors and bugs that were fixed in the next versions.

TABLE 11
CORRELATION BETWEEN PARAMETERS

Parameters	*For 40 students*	*For the 26 individual stud*	*For 14 school's students*
Gender; programs	-0.29	Not relevant (only 4 girls)	-0.01
Age; programs	0.13		**-0.54**
Math; Programs			
Completion; programs	-0.33	-0.30	Not relevant (all have completed in the same time)
Version; programs	0.19	-0.01	Not relevant (all have taken the same version)
Reports; programs	0.53	0.06	**0.61**
Gender; reports	-0.40	Not relevant (only 4 girls)	0.29
Math; reports	0.39	-0.13	Not relevant (all had the same background)
Completion; reports	0.57	-0.37	Not relevant (all have completed in the same time)
Version; reports	0.49	**0.58**	Not relevant (all have taken the same version)
Age; completion	-0.10	-0.05	Not relevant (all have completed in the same time)
Age; reports	0.11	-0.02	-0.11
Debug; Programs		-0.29	Was not measured for the school
Debug; age		0.03	Was not measured for the school
Completion; Time spent on programs, per chapter		0.38	Was not measured for the school
Programs; Time spent on programs per chapter		-0.1	Was not measured for the school

Key tables

version of the course	started
1	July
2	October
3	November
4	January
5	April

parameter	**values**
Gender	female=1 ; male=0;
Reports	Average score in the entire set of exercises.
Debug	Number of debugging questions students have asked. See Table 10 for full results.
Age	Given in years and decimal digits representing the months at the time the course started.
Math	Background of school curricula only; 1- Background of EPGY math courses prior to taking the course
completion	In days. The full results are presented in Graph 2.
programs	Students' scores for the entire set of assignments (from Figure 30).
Time spent on programs	Average time students spent on programs as based on feedback forms. Time is given in hours. The decimal digits represent the minutes. For example, 5.30 stand for 5 hours and a half.

6. Measures of Students' Individual Ability and Achievement

In order to determine how much variation and dimensionality in performance are present among the pilot group, we conducted qualitative tests for ability and achievement. We have included the most significant measures of performance as analyzed within the study.

6.1. List of Concepts used in the Analysis

The following criteria were used for this analysis:
oper - Average number of operations each student used in the entire set of programs examined.
len - Average line length of students' programs for the entire set of programs examined.
calls - Average number of calls to functions as executed for a certain input.
mem - Average number of bytes required to store all the variables declared within the program.
comple - The calendar time that measured the number of days it took each student to complete the course as a whole.
rep - Average score each student got for the automated exercises of the course.
deb - The number of debugging questions students sent by e-mail to their instructor within the course.
timProg - The average time students spent on programs per chapter.
Age - Age of student in years, calculated at the time students began the course. The age was taken optionally. Namely one set of tests was done without including the age as a measure and a second test was done with including age as a measure. The goal was to view the effect of this factor on other performance measures.
For some measures a better performance is indicated by lower values. In order to run these tests we used the negation for the computation. For example, *oper, len, calls ,mem, comple, deb, timProg* and *age* were all negated before conducting the tests. Also, in computing the partial orderings (explained in the next paragraph), any missing values were replaced by very poor scores. This was necessary as transitivity of the partial ordering is violated if values are missing. The full table for the measures of student performance is presented in Table 12 which contains the values before the negation. Missing values in this table are marked as NA.

6.2. Students' Measures of Performance

The qualitative tests were done by constructing a partial ordering of student performance according to subsets of performance scales. The partial ordering used is *domination.* Student A dominates student B with respect to a set of measures, if all his scores are equal to or better than the ones of student B, in all measures, and better on at least one. A *dominance chain* is defined as an ordered list of students where each one dominates the next one in the list on the set of selected performance measures. *Undominated students* stand at the top of the dominance chains. Namely, an undominated

set of students consists of students who are not dominated by any other student with respect to the set of selected performance measures. We characterize the partial orderings by computing the sizes of the undominated sets as well as the length of the dominance chains of students.

For each nonempty subset of the set of all the performance measures, we computed the sizes of the undominated sets of students with respect to the partial ordering, as well as the lengths of the dominance chains of students. If we only consider one performance measure, then the longest dominance chain will have the same length as the size of the sample, assuming that the values are strictly ordered and that none of the measure values are missing in the sample. As we add more measures, however, the lengths of the longest chain will decrease. If the longest chain becomes very small relative to the sample size, then there is little hierarchy evident in the partial ordering, so we may conclude that these measures are more independent in nature. If, given some or all of these measures, only a small group of students are undominated with respect to the partial ordering then we may consider the various performance measures to be highly dependent in character, possibly leading to the conclusion that a single performance measure might suitably capture student ability. On the other hand, if the set of undominated students is large, then it may indicate that the measures are independent of each other or are only weakly correlated, and to characterize student ability requires knowing a set of measure values (Cope, Suppes 2000).

The results of the partial orderings are summarized in Tables 13-15. Several breakdowns are used:

- The pilot as a whole (40 students: gifted and school students) as opposed to the 26 gifted students.

  The goal here was to examine the differences between the two pilot groups in dimensionality of students' performance.
- Results with eight or nine measures - with and without including age as a performance measure.

  The goal here was to better view the effect of age on other performance measures.
- For each count of measures included, we computed the maximum sizes of undominated sets with minimal sizes of dominance chains. The goal here is to find the degrees of dependence present among the various measures of performance.
- For each count of measures included, we computed the minimum sizes of undominated sets with the corresponding length of the dominance chains. The goal here was to evaluate which measures of performance appear more correlated.

TABLE 12: STUDENTS' MEASURES OF PERFORMANCE FOR THE QUALITATIVE TESTS

Students	oper	len	calls	mem	age	comple	rep	deb	timProg
1	85.5	16	7.75	8.5	11	111	88	0	1.3
2	2644.5	24.125	34.63	12.75	15.08	246	87	5	3
3	55.87	17.37	7.25	9.25	9.02	120	91	0	10
4	2618.75	19.62	28.38	11.25	17	313	88	11	9
5	108	19.12	14.63	9	13.02	218	72	7	4
6	568.12	17.25	8.875	13.5	14	86	84	3	4
7	816	21.75	22.38	10.5	13.01	299	87	31	6
8	3047.88	16.62	8.375	12	12.08	170	94	1	5
9	113.25	21.75	9.38	11.5	14.03	321	73	3	10
10	138.5	18.37	39	9.625	14.05	248	94	2	6
11	103.12	19.5	25.75	15.625	14.05	218	94	1	5
12	1098	16.62	8.38	18.375	11.06	99	91	2	8
13	3085.25	18.25	27.375	12.875	13.03	99	87	4	10
14	2647.13	26.12	27.38	13.75	13.02	155	87	1	9
15	2603.38	22.87	14.625	11.25	13.06	321	86	5	8
16	2609.25	31.62	23.75	15.25	9.11	313	86	8	9
17	2620.75	23	25.75	15	14.05	321	84	6	10
18	93.37	16.5	11.16	10.5	11.01	218	100	1	20
19	90.5	25.25	10.63	12.5	14.11	42	91	1	5
20	81.62	21.5	10.25	12	17.03	43	91	0	4
21	44.25	12	7.5	7.5	15.11	48	91	0	1.3
22	3089.75	20.12	38.63	11.5	10	91	91	0	8
23	3064.38	19	14.25	13.25	17	156	94	1	4
24	2620.63	36.25	13.75	13.5	9.03	186	94	3	3
25	2613.25	18	16.75	13.25	16.03	139	94	4	3.3
26	91.75	19.12	11.5	11.5	16.05	49	94	2	1.3
school 1	3005.14	30	42.5	13.75	12.09	211	69	NA	NA
school 2	2975.43	29.62	28.13	14.25	11.01	211	70	NA	NA
school 3	3518.57	31.87	27.86	15.5	13.03	211	73	NA	NA
school 4	121	31.37	28.38	15.25	11.09	211	94	NA	NA
school 5	2947.86	34.25	16.13	12.75	13	211	89	NA	NA
school 6	3519.43	31.5	37	13.5	13.06	211	71	NA	NA
school 7	3492.14	30.62	13.38	14	11.06	211	72	NA	NA
school 8	3519.43	31.5	37.25	14.5	12.04	211	73	NA	NA
school 9	640.71	31.37	15	14.5	11.06	211	84	NA	NA
school 10	2995.71	32	38.63	14.75	12.09	211	83	NA	NA
school 11	2985.71	31.5	16	14	12.03	211	90	NA	NA
school 12	3515.57	33.75	39.25	15.25	13	211	84	NA	NA
school 13	3499.86	34.87	16.13	15	11.07	211	82	NA	NA
school 14	3518	35.25	25.63	14	13.01	211	75	NA	NA
average	***1925.2***	***24.68***	***21.23***	***12.91***	***13.01***	***189.6***	***85.55***	***3.92***	***6.43***
Maximum	***3519.4***	***36.25***	***42.5***	***18.37***	***17.03***	***321***	***100***	***31***	***20***
minimum	***44.25***	***12***	***7.25***	***7.5***	***9.02***	***42***	***69***	***0***	***1.3***

Key

oper, len, mem, calls	Criteria for program evaluation
age	Given in years and decimal digits representing the months at the time the course started.
comple	In days.
deb	Number of debugging questions students have asked.
rep	Average score in the entire set of automated exercises.
timProg	Average time students spent on programs per chapter as based on feedback forms. Time is given in hours. The decimal digits represent the minutes. For example, 3.30 stand for 3 hours and a half.

NA - Missing values. The parameters, *timPrg* and *deb* and *comple* were not measured for the school.

Tables 13(a), 14(a) and 15(a) present the results obtained for the entire pilot group of 40 students consisting of gifted students and school students. Tables 13(b), 14(b) and 15(b) present the results obtained only for the gifted students (26 home students). Tables 13(a) and 13(b) present the partial orderings results without including age as a measure. In Tables 14(a) and 14(b) age is included as one of the measures. Tables 13-14 present results obtained for each count of measures for the maximum undominated set sizes and the minimum sizes of dominance chains. Table 15 presents the opposite direction. Namely, it focuses on the minimum sizes of undominated sets with the corresponding sizes of the dominance chains.

TABLE 13(a)
DIVERSITY IN STUDENTS' PERFORMANCE FOR THE ENTIRE PILOT GROUP (40 STUDENTS)
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGARDING AGE)

Number of Measures	Maximum undominated set sizes		Measures Included	Minimum sizes of dominance chains		Measures Included
2	11	27%	comple, rep	5	12%	deb, timPrg
3	15	37%	comple, rep, deb	3	7%	comple, rep, deb
4	16	40%	mem, comple, rep, deb	3	7%	oper, comple, rep, deb
5	16	40%	oper, mem, comple, rep, deb	3	7%	oper, len, mem, comple, deb
6	16	40%	oper, len, mem, comple, rep, deb	3	7%	oper, len, calls, mem, comple, deb
7	16	40%	oper, len, calls, mem, comple, rep, deb	3	7%	oper, len, calls, mem, comple, rep, deb
8	16	40%	oper, len, calls, mem, comple, rep, deb, timPrg	3	7%	oper, len, calls, mem, comple, rep, deb, timPrg

TABLE 13 (b)
DIVERSITY IN STUDENTS' PERFORMANCE FOR THE GIFTED STUDENTS (26 STUDENTS)
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGRADING AGE)

Number of Measures	Maximum undominated set sizes		Measures Included	Minimum sizes of dominance chains		Measures Included
2	10	38%	compl, rep	5	19%	oper, rep
3	14	53%	comple, rep, deb	3	11%	comple, rep, deb
4	15	57%	mem, comple, rep, deb	3	11%	oper, calls, rep, deb
5	15	57%	oper, mem, comple, rep, deb	2	7%	oper, calls, comple, rep, deb
6	15	57%	oper, len, mem, comple, rep, deb	2	7%	oper, len, calls, comple, rep, deb
7	15	57%	oper, len, calls, mem, comple, rep, deb	2	7%	oper, len, calls, mem, comple, rep, deb
8	15	57%	oper, len, calls, mem, comple, rep, deb, timProg	2	7%	oper, len, calls, mem, comple, rep, deb, timPrg

Key

oper	Average number of operations each student used in the entire set of programs examined.
len	Average code length of students programs in the entire set of programs examined.
calls	Average number of calls to functions as executed for certain input.
mem	Average number of bytes required to store all the variables and structures declared in students' programs.
comple	The calendar time. That is, the number of days it took each student to complete the course as a whole.

rep	Average score each student got for the automated exercises of the course.
deb	The number of debugging questions students sent by e-mail to their instructor within the course.
timProg	The average time students spent on programs per chapter.

TABLE 14 (a)
DIVERSITY IN STUDENTS' PERFORMNCE FOR THE ENTIRE PILOT GROUP (40 STUDENTS)
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (INCLUDING AGE)

Number of Measures	Maximum undominated set sizes		Measures Included	Minimum sizes of dominance chains		Measures Included
2	11	27%	age, rep	5	12%	deb, timPrg
3	16	40%	comp, age, rep	3	7%	age, rep, deb
4	18	45%	oper, comp, age, rep	3	7%	len, mem, comp, age
5	19	47%	oper, mem, com, age, rep	3	7%	oper, len, mem, comp, age
6	19	47%	oper, len, mem, comp, age, rep	3	7%	oper, len, calls, mem, comp, age
7	19	47%	oper, len, calls, mem, comp, age, rep	3	7%	oper, len, calls, mem, comp, age, rep
8	19	47%	oper, len, calls, mem, comp, age, rep, deb,	3	7%	oper, len, calls, mem, comp, age, rep, deb
9	19	47%	oper, len, calls, mem, comp, age, rep, deb, timPrg	3	7%	oper, len, calls, mem, comp, age, rep, timPrg

TABLE 14 (b)
DIVERSITY IN STUDENTS' PERFORMANCE FOR THE HOME STUDENTS (26 STUDENTS)
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (INCLUDING AGE)

Number of Measures	Maximum undominated set sizes		Measures Included	Minimum sizes of dominance chains		Measures Included
2	10	38%	comple, rep	5	19%	oper, rep
3	15	57%	age, comple, rep	3	11%	comple, rep, deb
4	17	65%	oper, age, comple, rep	3	11%	len, mem, age, comple
5	18	69%	oper, mem, age, comple, rep	2	7%	oper, calls, comple, rep, deb
6	18	69%	oper, len, mem, age, comple, rep	2	7%	oper, len, calls, comple, rep, deb
7	18	69%	oper, len, calls, mem, age, comple, rep	2	7%	oper, len, calls, mem, comple, rep, deb
8	18	69%	oper, len, calls, mem, age, comple, rep, deb	2	7%	oper, len, calls, mem, age, comple, rep, deb
9	18	69%	oper, len, calls, mem, age, comple, rep, deb, timPrg	2	7%	oper, len, calls, mem, age, comple, rep, deb, timPrg

Key

oper	Average number of operations each student used in the entire set of programs examined.
len	Average code length of students programs in the entire set of programs examined.
calls	Average number of calls to functions as executed for certain input.
mem	Average number of bytes required to store all the variables and structures declared in students' programs.
comple	The calendar time. That is, the number of days it took each student to complete the course as a whole.
rep	Average score each student got for the automated exercises of the course.
deb	The number of debugging questions students sent by e-mail to their instructor within the course.
timProg	The average time students spent on programs per chapter.
age	Age of student in years, calculated at the time students began the course.

TABLE 15 (a)
MINIMUM UNDOMINATED SETS FOR THE ENTIRE PILOT GROUP
MEASURES INCLUDED: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGARDING AGE)

Number of Measures	Minimum Size of undominated Set for 40 studs		Measures Included	Size of Dominance chain for 40 studs	
2	1	2.5%	oper, len	14	35%
3	1	2.5%	oper, len, mem	10	25%
4	3	7%	oper, len, mem, comp	5	12%
5	4	10%	oper, len, calls, mem, timPrg	5	12%
6	6	15%	oper, len, calls, mem, comp, deb	3	7%
7	7	17%	oper, len, calls, mem, comp, deb, timPrg	3	7%
8	16	40%	oper, len, calls, mem, comple, rep, deb, timPrg	3	7%

TABLE 15 (b)
MINIMUM UNDOMINATED SETS FOR THE HOME STUDENTS FOR THE SAME MEASURES:
OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGARDING AGE)

Number of Measures	Minimum Size of undominated Set for 26 studs		Measures Included	Size of Dominance chain for 26 students	
2	1	3.8%	oper, len	9	34%
3	1	3.8%	oper, len, mem	7	26%
4	3	11%	oper, len, mem, comp	5	19%
5	4	15%	oper, len, calls, mem, timPrg	4	15%
6	6	23%	oper, len, calls, mem, comp, deb	3	11%
7	7	26%	oper, len, calls, mem, comp, deb, timPrg	3	11%
8	15	57%	oper, len, calls, mem, comple, rep, deb, timPrg	2	7%

Key

oper	Average number of operations each student used in the entire set of programs examined.
len	Average code length of students programs in the entire set of programs examined.
calls	Average number of calls to functions as executed for certain input.
mem	Average number of bytes required to store all the variables and structures declared in students' programs.
comple	The calendar time. That is, the number of days it took each student to complete the course as a whole.
rep	Average score each student got for the automated exercises of the course.
deb	The number of debugging questions students sent by e-mail to their instructor within the course.
timProg	The average time students spent on programs per chapter.

6.2.1. Diversity of Students' Measures of Performance

Table 13(a) presents results for the maximum undominated set sizes and minimum sizes of the dominance chains for every size of subset of the eight performance measures considered. For example, the combination of three parameters yielding the largest undominated set when age was not included was: Completion time, average scores in the reports and number of debugging questions asked by students within the course. In the same line in the table, we can see that the combination of three parameters yielding the minimum size of the dominance chain was the same one: completion time, reports and number of debugging questions. Thus, the same combination of three measures gave relatively a large number of undominated students (37% of the sample size) and also a relatively small size of dominance chain (7% of the sample size). This may indicate that these specific measures are relatively independent of each other. We can see in Table 13(a) that the undominated set sizes grow from 27% to 40% of the total student sample size (where age is not included among the performance measures). Thus, a very large percentage of students are not dominated in eight measures. Also, the length of the dominance chains gets very small, as more measures are included, to the point where the dominance chain size is 3. In addition, there are increases in the undominated set sizes without dramatic decreases in the length of the maximal dominance chain up to the inclusion of about four parameters in both Tables 13 and 14. This may indicate that the number of parameters relevant to characterizing performance is about four (from among this set of measures). However, since the undominated sets are already large relative to the sample size (37% in Table 13(a) and 57% in Table 13(b)) at the point where 3 measures are included, it is difficult to conclude the exact effect of adding the fourth measure. Furthermore, we have included here average measures for criteria for each student and for each criterion. However, as analyzed before hand, the most significant differences between students' performance in programs are dependent to a great extent on the specific problem domain. Therefore, we will also look into the variation in program-performance in breakdown of different assignments.

In Table 13(b), we present the results for the same measures as in Table 13(a), but only for the gifted students' sample. We can see that the sizes of the undominated sets are larger than for the pilot as a whole. Namely, for the gifted students sample the undominated set grows from 38% to 57% from the sample size, as opposed to 27%-40%for the entire pilot in Table 13(a). Each student in the undominated set stands at the top of one list of measures included. The undominated set cannot be ordered. Thus, a larger undominated set for the gifted students sample means that it is more difficult to order the gifted students by means of a hierarchy of performance measures. There are more "excellent" students at the top of the list according to some measures that cannot be ordered in groups. It also suggests that the diversity in performance among the gifted students is larger than the diversity of the pilot group as a whole. Table 16 presents the ranges for the selected measures for the gifted and school students. For most of the measures, the range for the gifted students' sample is larger.

TABLE 16
THE RANGES OF MEASURES FOR THE GIFTED AND SCHOOL STUDENTS

Measure	**26 Home students**	**14 School students**
oper - Average operations count	44 - 3089 operations	121 - 3519 operations
len - Average code length	12 - 36 lines	29 - 35 lines
calls - Average calls to functions	7 - 39 calls	13 - 42 calls
mem - Average memory	7 - 17 bytes	12 - 15 bytes
comple - Completion time in days	42 - 321 days	211 days (for all)
rep - Average score in reports	72% - 100%	69% - 94%
deb - Number of debugging questions asked	0 - 31 questions	NA (not measured)
timPrg - Average time spent by students on programs, per chapter	1.5 - 20 hours	NA (not measured)
Age - Age of student in years, calculated at the time students began the course.	9.02 - 17.03 years	11.01 - 13.06 years

However, there are three measures included in the tests and this table that either had the same value for all the school students (i.e. comple) or that was not measured for the school (i.e. deb and timPrg). Thus, we have computed an additional set of tests including only six measures with full values for all. The results were similar with regard to the difference between the pilots in diversity in performance. For the gifted students sample the undominated sets grew from 23% to 50% from the total sample size. For the entire pilot the undominated set sizes grew from 15% to 32%. Here too we got more diversity in performance for the gifted students' sample. Furthermore, previous analysis and results we got with regard to the diversity in programming performance of the groups support this finding. For example, the diversity both in applying the criteria to students' programs and in programs-scores was larger for the gifted-student sample than for the school students'. Examples are given in table 17.

TABLE 17
CRITERION RANGE FOR GIFTED STUDENTS AND SCHOOL STUDENTS

Criteria	**Assignment**	**Home students range**	**School range**
operations	*reverse*	11 - 63 operations	32 - 63 operations
functions	*prm50-100*	10 - 65 calls	32 - 56 calls
mem	*factorial*	6 - 66 bytes	6 - 20 bytes

We can see in these examples that the differences in the maximum and minimum values computed for these groups are significant. For example, for the home students, the range for *mem* in the *factorial* assignment was 60 (66-6) but for the school it was 14 (20-6).

TABLE 18
AVERAGE RELATIVE SCORES

	School	Home students
min	19.46	33.44
average	42.94	91.99
max	76.87	134.99

In Table 18, the range for the school students in the relative scores (for the entire set of programs) is 57 (76-19). For the home students the range is larger -- 101 (134-33).

The differences between these two groups in the degrees of diversity in performance may be related to characteristics of gifted-students as supported in previous studies (Cope, Suppes 2000). This last study analyzes gifted students' individual differences in the EPGY's physics computer-based courses. Thus, although the measures of performance selected for physics are naturally different than the ones we chose here, still the diversity and dimensionality in students' performance was established. However, the difference may be also related to the different conditions these two groups had while taking the course. The gifted students studied alone at home. The school students were all working on their programs in one class lab, and they could have consulted with each other.

Tables 14(a) and 14(b) present the same measures with including age as an additional measure. We have considered age as a performance measure separate from the other parameters to better view the effect of this factor on the other performance measures. If older students had a significant advantage over younger students, we should see a large increase in the undominated set sizes. For the entire pilot (Table 14(a)) we can see an increase in the sizes of the undominated sets. Thus, without including age the undominated sets grew from 27% to 40% of the sample size. Including age the undominated sets grew from 27% to 47% of the sample size.

In Table 14(b) for the gifted students only with out including age the size of the undominated sets grew from 38% to 57%. With including age the size grew from 38% to 69%. Thus, the conclusion is similar to the one drawn in previous study for the computer-based physics courses (Cope, Suppes, 2000). That is, we can see an increase in the sizes of the undominated sets, as we are adding another degree of freedom into the sets of possible orderings; but none of these increases are very dramatic. However, because the undominated sets are very large already, it is difficult to conclude from this that older students possess any significant advantage over younger students.

Further more, in correlation tests between age and performance in programs (programs' scores), we found a negative correlation (-0.53) for the school students and no significant correlation for the gifted students. Namely, younger students in the school had better performance scores in programs. Also, we did not get any significant correlation between age and other performance measures included here (i.e. *deb, comple, rep*, etc.,). This finding as well as previous results may stress the conclusion, that age does not significantly affect other measures of students' performance.

6.3. Analysis of Undominated Sets of Measures

In Table 15 we show the minimum sizes of the undominated sets with the corresponding lengths of dominance chains with combinations of 2-8 measures. Table 15(a) presents the results for 40 students for every size of subset of the eight performance measures. Table 15(b) takes the same combinations of measures and presents the sizes of the undominated sets and dominance chains for the 26 gifted students. The goal here was to look closely at which measures were more correlated or connected when two or more measures were considered. For example, the combination of two parameters yielding the smallest undominated set was: operations (*oper*), and length (*len*). We can

see in Table 15(a) that for this combination of two measures the size of the undominated set is 1 (2.5% from the sample size) with length of 14 (35% from the sample size) for the corresponding dominance chain. For the same combination of two measures (*oper* and *len*) we can see in Table 15(b) that the size of the undominated set is 1 (3.8%) with dominance chain length of 9 students (34% for the gifted-student sample). Since the undomintaed set is small, along with a relatively large dominance chain, these results indicate that these two measures of *len* and *oper* have some dependency. This finding is in fact supported by previous findings in correlation tests between the criteria according to specific assignments (see "correlation between the criteria"): For the entire pilot of 40 students in 4 out of 8 assignments we found a significant positive correlation between *oper* and *len* (0.7 on the average). For the gifted students, we found a positive significant correlation (0.6 on the average) between *len* and *oper* in 3 out of 10 assignments. However, there was no significant correlation between these two measures in the more advanced assignments. Thus, the conclusion earlier was that in the more basic assignments students who tended to write shorter programs have used less operations in their programs and vice versa. Here, we computed the average length for each student for the entire set of assignments to get a measure of performance for each individual student. So, we cannot see here the nuances and differences as dependent on the problem domain of the assignment. But, the same pattern was followed – length of programs and count of operations were connected. On the other hand, some correlation results we found previously, (see in "correlation tests") were not supported by these results. For example, we found a positive correlation (0.61) between scores in reports (*rep*) and programs scores only for the school students. This finding was not indicated here. Namely, including the measure *rep* in both Tables (12a and 12b) did not yield smaller undominated sets. Moreover, this was the eighth measure to be included in both tables. In particular, for the pair of operations (which is the most significant factor constructing the score) and reports scores we did not get a significant small undominated set with a large dominance chain. Also, the correlation found earlier between completion time and reports (0.56) for the entire pilot group was not supported by these new findings since *comple* was included as the third measure and *rep* as the eighth measure. We can see more dramatic increases in the undominated set sizes, as more measures are included. The percentages of the sizes of the undominated sets of the total sample size are summarized below: For the entire pilot the undominated set grows from 2.5% to 40% and for the gifted students it grows from 3.8% to 57%. Namely, we can see more diversity in the gifted- student sample than for the entire pilot.

TABLE 19
SIZE OF UNDOMINATED SETS

Number of measures	For the whole pilot - 40 students: Size of minimum undominated sets from the sample	For the gifted students - 26 students: Size of minimum undominated sets from the sample
3	2.5%	3.8%
4	7%	11%
5	10%	15%
6	15%	23%
7	17%	26%
8	40%	57%

We can see in both Tables (15a and 15b) that by adding the third measure there were no dramatic changes in the sizes of the undominated sets as well as in the lengths of the dominance chains. Namely, it suggests that *mem* was correlated to *oper* and *len*. In particular for the pair [*oper* and *mem*], it indicates that students who wrote programs using fewer operations used less memory to store their variables and vice versa. However, this last finding is problematic, since it contradicts previous findings in the context of specific assignments. Thus, we found earlier that the criteria of *oper* and *mem* were not correlated in the context of specific problem domains. On the contrary, there was a tradeoff between these two criteria—as more operations were executed less memory was used and vice versa. This was analyzed in prior research by several persons and was also supported by the correlation measures obtained for these two criteria in the breakdown of the assignments. The explanation for this may be related to the fact that the measures selected for these tests computed ***average*** counts for the criteria for a set of assignments and for each student. This average count, however, did not reflect the diversity between the criteria in the context of specific problem domains of the assignments. In order to capture the complexity we need to look also into students' performance measures in the context of specific assignments. Tables 20(a) and 20(b) present results for the gifted youth sample with regard to the *prime100* assignment. Namely, the four measures for program evaluation (*oper, len, mem, calls*) were computed for this assignment. Table 20(a) presents the maximum undominated set sizes and the minimum sizes of the dominance chains for every size of subset of the eight performance measures considered. Table 20(b) presents the opposite direction; that is, the minimum undomintaed set sizes with the corresponding lengths of the dominance chains.

TABLE 20 (a)
DIVERSITY IN STUDENTS' PERFORMANCE FOR THE PRIME100 ASSIGNEMNT, FOR 26 STUDENTS.
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGARDING AGE)

Number of Measures	Maximum undominated set sizes		Measures Included	Minimum sizes of dominance chains		Measures Included
2	10	38%	compl, rep	5	19%	calls, mem
3	14	53%	mem, comple, rep	3	11%	mem, comple, rep
4	16	61%	mem, comple, rep, deb	3	11%	oper, mem, comple, rep
5	17	65%	oper, mem, comple, rep, deb	2	7%	calls, mem, comple, rep, timPrg
6	18	69%	oper, len, mem, comple, rep, deb	2	7%	oper, calls, mem, comple, rep, timPrg
7	18	69%	oper, len, calls, mem, comple, rep, deb	2	7%	oper, len, calls, mem, comple, rep, timPrg
8	18	69%	oper, len, calls, mem, comple, rep, deb, timProg	2	7%	oper, len, calls, mem, comple, rep, deb, timPrg

TABLE 20 (b)
MINIMUM UNDOMINATED SETS FOR THE PRIME100 ASSIGNMENT, FOR 26 STUDENTS
MEASURES: OPER, LEN, CALLS, MEM, COMPLE, REP, DEB, TIMPROG (DISREGARDING AGE)

Number of Measures	Minimum Size of undominated Set for 26 studs		Measures Included	Size of Dominance chain for 26 students	
2	3	11%	oper, len	7	26%
3	4	15%	oper, len, mem	6	23%
4	6	23%	oper, len, calls, deb	4	15%
5	7	26%	oper, len, calls, mem, deb	4	15%
6	8	30%	oper, len, calls, mem, comp, deb	3	11%
7	15	57%	oper, len, calls, mem, rep, deb, timPrg	3	11%
8	16	61%	oper, len, calls, mem, comple, rep, deb, timPrg	2	7%

Key

oper	Number of operations each student used in the prime100 assignment.
len	Code length of the prime100 assignment.
calls	Number of calls to functions as executed.
mem	Number of bytes required to store all the variables and structures declared by students in prime100.
comple	The calendar time. That is, the number of days it took each student to complete the course as a whole.
rep	Average score each student got for the automated exercises of the course.
deb	The number of debugging questions students sent by e-mail to their instructor within the course.
timProg	The average time students spent on programs per chapter.

In Table 20(a) we can see that the undominated sets grows from 38% to 69% of the total sample size, as more measures were included. In Table 13(b) that presented the average counts for the programming criteria we got a smaller growth (from 38% to 57% of the total sample size). Thus, according to this last table, about six measures were relevant to characterize students' performance and not four as indicated beforehand. We can also see a dramatic difference between Tables 20(b) and 15(b) with regard to the minimum undominated set sizes and their corresponding dominance chains' lengths. For example, if we look more closely at the results we got for the pair, [*oper, len*] and then for the three measures, [*oper, len, mem*], we can see that when the average measures were considered, a higher correlation was indicated than when measures were taken for the *prime100* assignment.

TABLE 21
AVERAGE MEASURES (FROM TABLE 15 (b)

Number of measures	Minimum size of undominated set for 26 studs		Average measures	Size of Dominance chains	
2	1	3.8%	oper, len	9	34%
3	1	3.8%	oper, len, mem	7	26%

MEASURES IN PRIME100 (FROM TABLE 20 (b))

Number of measures	Minimum size of undominated set for 26 studs		Measures Included for the prime100 assignment	Size of Dominance chains	
2	3	11%	oper, len	7	26%
3	4	15%	oper, len, mem	6	23%

For the average counts, the undominated set did not grow when *mem* was included. It remained the same, as the third measure was included (3.8% of the total sample size). In the prime100 results, when *mem* was included the undominated set grew from 11% to 15% from the total sample size. Namely, *mem* in *prime100* is not correlated to *len* and *oper*. Also, the dominance chains lengths are larger for the average counts than for the *prime100* count.

As a whole, when average measures were used, the undominated sets grew from 3.8% to 57% from the total sample size. In *prime100* the undominated sets grew more, from 11% to 61% with more dramatic differences, as more measures were included. This indicates that the measures are more correlated when average counts are considered than for the specific assignment *prime100*.

Summary notes

We summarize the main findings.

- We found significant differences in the diversity in performance between the two samples. For the entire pilot, not surprisingly the undomintaed set grew from 27% to 40% of the total sample size, as more measures were included. For the gifted students' sample, the growth was larger: 38% to 57%. Among the gifted-student sample the diversity in performance was much larger than among the school sample.
- We found differences between the diversity in student performance depending on whether average programming criteria (for several assignments) were selected, or whether measures in the context of a specific assignment were selected. For the average counts of student performance in their programs, the undomitaed sets sizes grew, as more measures were included, up to four measures. This indicates that when average counts were considered, about four measures were relevant to characterizing student performance. However, for the criteria-counts in the context of the specific assignment named *prime100*, the undominated sets grew more for six measures. This indicated that students' diversity in performance in the C course depended also on the specific problem domains of the assignments.
- We found that age was not a major factor affecting other performance measures. Namely, older students did not have a significant advantage over younger students.
- As expected, the sizes of the undominated sets became larger as more measures were included. However, the quantitative details are less obvious. For the entire pilot the sizes grew up to 40% of the total sample size and for the gifted-student sample it grew up to 57% when eight measures were included. Thus, a large percentage of students were undominated with eight measures. This may indicate that the dimensionality in students' performance was large. It suggests that we should not select only one or two measures by which to rank students' performance.

TABLE 22
MAIN FINDINGS

Number of measures	Minimum sizes of undominated sets	Minimum sizes of undominated sets	Minimum sizes of undominated sets
	40 students	26 gifted students	26 gifted students
	Average program measures	Average program measures	Program measures in *prime100*
2	1 (2.5%)	1 (3.8%)	3 (11%)
3	1	1	4
4	3	3	6
5	4	4	7
6	6	6	8
7	7	7	15
8	16 (40%)	15 (57%)	16 (61%)

Number of measures	Maximum sizes of undominated sets	Maximum sizes of undominated sets	Maximum sizes of undominated sets
	40 students	26 gifted students	26 gifted students
	Average program measures	Average program measures	Program Measures in *prime100*
2	11 (27%)	10 (38%)	10 (38%)
3	15	14	14
4	16	15	16
5	16	15	17
6	16	15	18
7	16	15	18
8	16 (40%)	15 (57%)	18 (69%)

7. References

Bagno F. , Franci F., and Sterbini A. Webtech: Web Tools for Teachers and Students. *32nd ASEE/IEEE Frontiers in Education Conference,* November 6 - 9, 2002, Boston, MA

Bently J.L., Writing Efficient Programs, Prentice-Hall, Englewood Cliffs, NJ, 1982.

Bieman J.M., and Kang B., Measuring Design-Level Cohesion, IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp 111- 124, 1998.

Briand L. C., Daly J. W., and Wust J. K., A Unified Framework for Coupling Measurement in Object-Oriented Systems, IEEE Transactions on Software Engineering, Vol. 25, No. 1, pp 91- 121, 1999.

Briand L. C., Morasca S., and Basilli V. R., Defining and Validating Measures for Object-Based High-Level Design, IEEE Transactions on Software Engineering, Vol. 25, No. 5, pp 722- 743, 1999.

Clarke E.M., Grumberg O., and Peled D. A., Model Checking, The MIT Press, 2000

Coad, P., and Yurdon, E. Object-Oriented Analysis, Second Edition, Prentice Hall, 1991.

Cooper, D. Oh! Pascal. Third edition. *W.W Norton & Company, New York and London.* 1993.

Cope, E., Suppes, P. Gifted Students' Individual Differences in Distance-Learning Computer-Based Physics, 2000. In preparation.

Cope, E., Suppes, P. Gifted Students' Individual Differences in Computer-Based Calculus and Linear Algebra, Education Program for Gifted Youth Technical Report, Stanford, CA. 1999

Gilnert, E. Introduction to Computer Science Using Pascal. *Prentice-Hall International*. Translated and processed for distance learning purposes by The Open University of Israel. 1983. (In Hebrew).

Harel D. , Algorithmics - The Spirit of Computing Second Edition, Addison-Wesley, England, 1992.

Kerninghan, B. W., and Ritchie D. M. The C Programming Language. Second Edition. *Prentice Hall*, Englewood Cliffs, N.J. 1988. ISBN: 0-13-110370-9

Lakhotia, A. Rule-Based Approach to Computing Module Cohesion," *Proc. 15th Int'l Conf. Software Eng*., pp 35-44, 1993.

Larsen, I., Morkosian, L.Z., and Suppes, P. Performance models of Undergraduate Students on Computer-Assisted Instruction in Elementary Logic. *Instructional Science,* 1978, **7**, 15-35.

Lorton, P., and P. Cole. Computer-Assisted Instruction in Computer Programming: Simper, LOGO, and Basic, 1968-1970. *University-Level Computer-Assisted Instruction at Stanford: 1968-1980.* 1981. 841-876.

Macken, E., Suppes, P., and Zanotti, M. Considerations in Evaluating Individualized Instruction, *Journal of Research and Development in Education,* 1980, **14**, 79-83.

Malone, T.W., Macken, E., and Suppes, P. Toward Optimal Allocation of Instructional Resources: Dividing Computer-Assisted Instruction Time among Students. *Instructional Science,* 1979, **8,** 107-120.

Malone, T.W., Suppes, P., Macken, E., Zanotti, M., and Kanerva, L. Projecting Student Trajectories in a Computer-Assisted Instruction Curriculum. *Journal of Educational Psychology,* 1979, **71**, 74-84.

Manna Z. , Verification of Programs, *Mathematical Theory of Computation*, McGraw-Hill, pp 161 - 222, 1974.

Manna, Z., Browne, A., Sipma, H.B., and Uribe, T,E. Visual Abstractions for Temporal Verification. *In AMAST'98, vol. 1548 of LNCS*, 1998, pp. 28-41, Spinger-Verlag.

McAlpin D.A., O'Docherty B.A., O'Donoghue P.G., and Teague K.G. Flexible Automatic Assessment of Clarity, Complexity and Style of Students' Modula2 Programs. County Antrim, Northern Ireland. Available at: http://www.ulster.ac.uk/cticomp/mcalp.html

McCabe, T.J. and Bulter, C.W. Design Complexity Measurement and Testing. Comms ACM, Volume 32, No 12, Decemeber 1989, pp 1415-1425.

Pattis, R.E. Karel the Robot, A Gentle Introduction to the Art of Programming. *John Wiley & Sons, Inc.*1995. ISBN: 0-471-59725-2.

Ravaglia, R., Alper, T., Rozenfeld, M., and Suppes, P. Successful Pedagogical applications of Symbolic Computation. In *Computer-human Interaction in Symbolic Computation,* Ed. N. Kajler, New- York, NY: Spinger-Verlag, 61-87, 1998.

Ravaglia, R., Sommer, R., Sanders M., Oas, G. and DeLeone, C. Computer-based Mathematics and
Physics for Gifted Remote Students. In *Proceedings of the International Conference on Mathematics/Science Education & Technology,* ed. D. Thomas. Association for the Advancement of Computing in Education, Charlottesville, VA, 1999, pp.~405-410.

Ravaglia, R., Suppes, P. , Stillinger, C., and Alper, T.M. Computer-based Mathematics and Physics for Gifted Students. *Gifted Child Quarterly.* 1995, **39,** 7-13.

Rees, M.J. (1982), "Automatic Assessment Aids for Pascal Programs", ACM Sigplan Notices, Volume 17, No 10, October 1982, pp 33-42.

Roberts, E. S. The Art and Science of C. *Addison-Wesley Publishing Company*. 1995. ISBN: 0-201-54322-2.

Roberts, E. S. Programming Abstractions in C, A second Course in Computer Science. *Addison-Wesley Publishing Company.* 1998. ISBN: 0-201-54541-1.

Roberts, E. S., Lilly, J., and Rollins, B. Using Undergraduates as Teaching Assistant in Introductory Programming Courses: On Update on the Stanford experience. Department of Computer Science Stanford University. *SIGCSE 195 3/95 Nashville*, 1995, TN USA.

Rosenthal, T., Professional Updating Through Computer Training. Master's Thesis. The Hebrew University, Jerusalem, Israel (in Hebrew), 1995.

Rosenthal, T., Automated Evaluation Methods with Attention to Individual Differences- A Study of A Computer-Based Course in C. PhD. Dissertation. The Hebrew University, Jerusalem, Israel, 2004.

Rosenthal T., Suppes P., and Ben-Zvi N. Automated Evaluation Methods with Attention to Individual Differences- A Study of A Computer-Based Course in C. Proceedings of the *32nd ASEE/IEEE Frontiers in Education Conference,* November 6 - 9, 2002, Boston, MA

Saikkonen, R., Malmi, L., and Korhonen, A. Fully Automatic assessment of Programming Exercises. *6th Annual SIGCSE/SIGUE conference on Innovation and Technology in Computer*

Science Education ITiCSE, June 25 – 27, 2001, Canterbury, England. Available at: http://www.cs.hut.fi/Research/Scheme-robo/

Springer, G., Friedman, D. P. Scheme and The Art of Programming. *The MIT Press Cambridge.* 1989.

Stevens W., Myers G. and Constantine L.L., Structured Design, IBM Sys. J. , 13: 115-139, 1974.

Steward D. V., Software Engineering with Systems Analysis and Design, CA State University, Sacramento, Brooks/Cole Publishing Company, Monterey CA, 1987.

Stillinger, C., and Suppes, P. Gifted Student' Individual Differences in Computer-Based Algebra and Pre Calculus Courses. 1999. Published on the Web at: http:// www-epgy.stanford.edu.

Suppes, P., Editor. *University-level Computer-assisted Instruction at Stanford: 1968-1980.* Stanford University, Institute for Mathematical Studies in the Social Sciences, Stanford, California, 1981.

Suppes, P. On the Effectiveness of Educational Research. In D.B.P. Kallen, G.B. Kosse. Wagenaar, J.J.J. Kloprogge, and M. Vorbeck (eds.), Windsor, Berks., England:1982.

Suppes, P., and Zanotti, M. Mastery Learning of Elementary Mathematics: Theory and Data. 1996. To appear in: P.Suppes & Zanotti, Fundamentals of Probability with Applications, Selected Papers: 1974-1975: New York: *Cambridge University Press*.

Suppes, P., and Sheehan, J. CAI Course in Axiomatic Set Theory, *University-level Computer-assisted Instruction at Stanford: 1968-1980.* Stanford University, Institute for Mathematical Studies in the Social Sciences, Stanford, California, pp 3 - 80, 1981.

Suppes, P. Current Trends in Computer-Assisted Instruction. In M.C. Yovits (Ed.), *Advances in Computers* (Vol. 18). New York: Academic Press, 1979.

Suppes, P., Fletcher, J.D. , and Zanotti, M. Models of Individual Trajectories in Computer-Assisted Instruction for Deaf Student. *Journal of Educational Psychology,* 1976, **68**, 117-127

Suppes, P. Education and Technology at Stanford in the Twenty-first Century. In K. Arrow, R. Cottle, B.C. Eaves & I. Olkin (Eds.), Education in research University, 1996, *Stanford University Press,* 143-158.

Suppes, P., Macken, E., and Zanotti, M. The Role of Global Psychological Models in Instructional Technology. In R. Glaser (Ed). *Advances in Instructional Psychology* (Vol. 1) Hillsdale, N.J.: Erlbaum. 1978. 229-259.

Tenenbaum, M., and Augenstein, M. Data Structures Using Pascal. Published by: *Prentice-Hall International*, 1981, Englewood Cliffs, New Jersey.

Tock K., and Suppes, P. The High Dimensionality of Students' Individual Differences in Performance in EPGY's K6 Computer-Based Mathematics Curriculum. June 2002. Available at: http://epgy.stanford.edu/research/index.html?trajectories.html

Valenti S., Cucchiarelli A., and Panti M. Computer Based Assessment Systems Evaluation via the ISO9126 Quality Model. Journal of Information Technology Education Volume 1 No. 3, 2002

Woodward, M. Sifficulties Using Cohesion and Coupling as Quality Indicators, *Software Quality J.*, Vol. 2, no. 2, pp109-127, June 1993.

Yourdon E. and Constantine L. L, Structured Design, Prentice Hall, Englewood Cliffs, NJ, 1979.

Zin A.M, and Foxley E. Automatic Assessment System, *Nottingham University*, UK, 2002. Available at: http://www.cs.nott.ac.uk/CourseMarker/more_info/html/ASQA.HTM

Web References

EPGY: http://www-epgy.stanford.edu

C11A for students: http://epgy.stanford.edu/courses/cs/C11A/

Stanford Online: http://stanford-online.stanford.edu/

Stanford CS (Introductory courses): http://www.stanford.edu/class/cs106a/

OUI, The Open University of Israel: http://www.openu.ac.il/

HU CS, The Hebrew University CS: http://www.cs.huji.ac.il/

OU CS, United Kingdom: http://computing.open.ac.uk/

HU, The Hebrew University, Snunit -

The Center for the Advance of Web Based Learning: http://www.snunit.k12.il/English/

C++ TEST, by Parasoft: http://www.parasoft.com/products/ctest/index.htm