DOCUMENT RESUME

ED 052 605 88 EM 009 059

AUTHOR Koetke, Walter
TITLE Computers in the Classroom: Teacher's Resource Manual for Algebra.
INSTITUTION Digital Equipment Corp., Maynard, Mass.
SPONS AGENCY Massachusetts State Dept. of Education, Boston.; Office of Education (DHEW), Washington, D.C.
PUB DATE 71
NOTE 134p.
AVAILABLE FROM Digital Equipment Corporation, Educational Marketing (5-2), 146 Main Street, Maynard, Massachusetts 01754 ($3.00)

EDRS PRICE EDRS Price MF-$0.65 HC Not Available from EDRS.
DESCRIPTORS *Algebra, *Computer Assisted Instruction, Computer Programs, *Mathematics Instruction, Mathematics Materials, Problem Sets, Programing, Remedial Mathematics, *Secondary School Mathematics, *Teaching Guides

ABSTRACT

Demonstration programs, possible assignments for students (with solutions), and remedial drill programs for students to use are presented to aid teachers using a computer or a computer terminal in the teaching of algebra. The text can be followed page by page or used as a well-indexed reference work, and specific suggestions are made on how and where to use the computer within the schools' present curriculum. Almost all topics discussed are completely self-contained. The order of major topics follows that used in the Modern Algebra series by Dolciani, Berman, and Wooton: arithmetic operations, variables, and sets; solving equations and inequalities in one variable; using the properties of equality, addition, and multiplication when solving equations; negative numbers; solving first degree equations and inequalities; operations with polynomials; and factoring. Sample programs are written using the Digital Equipment Corporation's FOCAL programing language, but they can be translated into any other interactive language that is suitable for student use. The manual is in loose-leaf form and provides a complete index. (Author/JY)

1-2

COMPUTERS IN THE CLASSROOM

TEACHER'S RESOURCE MANUAL FOR ALGEBRA

Written by:

Walter Koetke

Lexington High School

COMPUTERS IN THE CLASSROOM

TEACHER'S RESOURCE MANUAL FOR ALGEBRA

Written by:

Walter Koetke

Lexington High School

Produced jointly under the auspices of Digital Equipment Corporation and Project LOCAL (Title III, ESEA, 67-4533). Project LOCAL includes the Massachusetts public school systems of Westwood, Lexington, Natick, Needham and Wellesley and is directed by Robert N. Haven.

The work presented or reported herein was performed pursuant to a grant from the United States Office of Education, Department of Health, Education and Welfare. However, the opinions expressed herein do not necessarily reflect the position or policy of the United States Office of Education and no official endorsement by the United States Office of Education should be inferred.

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

CONTENTS

CONTENTS (Cont)

PREFACE

The purpose of this text is to suggest methods for using a computer or computer terminal in the teaching of Algebra I, II, and III to students with a wide range of mathematical ability. This edition constitutes about one-half of the complete book; it is being released in this form so as to be available for the 1968-69 academic year. The complete work will be available in 1969.

This text is intended to serve as a handbook for teachers rather than for students, as demonstration programs, possible assignments for students (with solutions), and remedial drill programs for students to use, are all included. Although the text can be followed page by page, it is primarily intended to be a well-indexed reference work as specific suggestions are made as to how and where to use the computer within the schools' present curriculum. Almost all topics discussed are completely self-contained, thus the suggestions made can be used without reference to programs or ideas presented in other sections of the text. The order of major topics is similar to that used in the Modern Algebra series by Dolciani, Berman, and Wooton[1].

The loose-leaf format and the detailed index of the text should complement the intended use of this material. When a particular topic is to be presented to the class, the teacher can use the index to locate various methods of presentation or possible class assignments. More often than not, the suggestions found in the text will be modified to better suit the specific needs of the teacher's planned presentation. These modifications can then be inserted into the text for easy reference in the future. Note that each of the sample programs included represents only one possible solution to a given problem. These samples were selected because of their exceptional clarity and/or because they are typical of many student solutions. Thus they rarely represent the best algorithm or most efficient program.

This text is not intended to teach either programming techniques or a programming language. Although all sample programs are written using the Digital Equipment Corporation's FOCAL language (4K version), they can be easily translated into any other interactive language that is suitable for student use. This text, accompanied by the FOCAL Programming Manual[2], might also be used to provide self-instruction in programming for teachers without previous experience using computers. Although the text does not contain an entire section devoted to techniques for teaching programming to students, Section VII-A contains several examples which are particularly useful in this endeavor.

When reading the examples in this text, one should keep in mind that almost all programs, including those that appear tutorial, are most beneficial when assigned as programs to be written by students. The complete program has been included as one possible solution to such an assignment and not as a "cook-book" program for teachers to type and students to use.

---

[1] Houghton Mifflin Company, Boston, 1965.
[2] Digital Equipment Corporation, 1968, Order No. DEC-08-AJAB-D

When used as intended, this text is not and cannot be complete, for new ideas and new programs will be added on a continuing basis. Additional material is already being prepared and will be made available as soon as possible.

Any suggestions concerning programs and/or topics that might be included, as well as reactions to the content and/or format of the text, should be addressed directly to the author. Every effort will be made to incorporate suggestions received in this manner into the forthcoming sections of the text.

Walter Koetke
Lexington High School
Lexington, Mass.

I. ARITHMETIC OPERATIONS, VARIABLES, AND SETS

A. Rules of Algebraic Precedence

A computer quite naturally reinforces the rules of algebraic precedence, since almost all computer languages require the user to write expressions on a single line according to these rules. An examination of these rules is frequently one of the first topics discussed when teaching programming.

Motivation for the need to establish rules of algebraic precedence can be quickly attained by having the students evaluate expressions written on a single line. For example, given expressions such as 2*6/3+3 and 16†1/2, there will be several solutions other than the correct answers of 7 and 8. By using the computer to evaluate several similar expressions, the students can be led to "discover" the precedence rules on their own.

One way of presenting this topic, which also reinforces the proper use of parentheses, is to present an expression such as 36/3*12 - 3*12/36, and then ask students to insert parentheses to obtain as many different values as possible. A typical student effort using only a single pair of parentheses might appear as:

```
*TYPE 36/3*12-3*12/36,!!
=+    0.0000

*TYPE (36/3)*12-3*12/36,!!
=+  144.0000

*TYPE 36/(3*12-3*12/36),!!
=+    1.0286

*TYPE 36/3*(12-3*12/36),!!
=+    1.0909

*TYPE 36/3*(12-3)*12/36,!!
=+    0.0031

*TYPE (36/3*12-3)*12/36,!!
=-    0.6667

*TYPE 36/3*12-3*(12/36),!!
=+    0.0000

*
```

NOTE

The last example would not be considered correct since the result is not different from all of the preceding answers.

Another type of problem often enjoyed by students is writing an expression without using parentheses. For example, write the expressions 84.6/(46.8*0.0056) and (16.8+13.052)↑2 without parentheses.

```
*TYPE ?84.6/(46.8*0.0056)?,!,?84.6/46.8/0.0056?,!!
84.6/(46.8*0.0056)=+ 322.8020
84.6/46.8/0.0056=+ 322.8020

*TYPE ?(16.8+13.052)↑2?,!,?16.8↑2+2*16.8*13.052+13.052↑2?,!!
(16.8+13.052)↑2=+ 891.1420
16.8↑2+2*16.8*13.052+13.052↑2=+ 891.1420

*
```

Even with this elementary use of the computer, better students might be motivated to study farther by having them express continued fractions on a single line. For example, several approximations to the continued fraction

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\ldots}}}}}$$

appear as:

```
*TYPE 1+1/1,!!
=+    2.0000

*TYPE 1+1/(1+1/1),!!
=+    1.5000

*TYPE 1+1/(1+1/(1+1/1)),!!
=+    1.6667

*TYPE 1+1/(1+1/(1+1/(1+1/1))),!!
=+    1.6000

*TYPE 1+1/(1+1/(1+1/(1+1/(1+1/1)))),!!
=+    1.6250

*TYPE 1+1/(1+1/(1+1/(1+1/(1+1/(1+1/1))))),!!
=+    1.6154
```

```
*TYPE 1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/1)))))),!!
=+    1.6190

*TYPE 1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/1)))))),!!
=+    1.6176

*TYPE 1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/1))))))),!!
=+    1.6182

*
```

The same topic can also be presented using variable names if students are familiar with this concept. This is also a good way to help introduce variables to those unfamiliar with them. A student solution to a problem such as evaluating

$$\frac{A+B}{A+C} \quad , \quad \frac{1}{C}+\frac{B}{A*C} \quad , \text{ and } \quad \frac{.01(A^2+C^2)}{B}-C$$

when A = 3, B = 5, and C = -4 might appear as:

```
*SET A=3; SET B=5; SET C=-4
*TYPE (A+B)/(A*C),!!
=-    0.6667

*TYPE 1/C+B/(A*C),!!
=-    0.6667

*TYPE .01*(A↑2+C↑2)/B-C,!!
=+    4.0500

*
```

One side benefit of using the computer to evaluate student work in this fashion is that the teacher can easily check the validity of a solution regardless of the form of the solution. This allows each student complete freedom to solve each problem in his own way.

The computer itself can also be utilized to check the student solutions. Consider the very brief problem set:

Given that A=2, B=3, and C=4, determine X,Y, and Z where X =

$$X=\frac{AB^C-AB}{1-B} \quad , \quad Y=\frac{6A}{B^2} \quad , \text{ and } \quad Z=\frac{1}{\sqrt{\frac{C}{AB}}}$$

A student solution would appear much like that shown in the previous example. However, by using the following teacher-written program, the students' solutions can also be checked by the computer. (See Page 5 for further description of this program.)

```
01.01 C SOLUTION ANALYSIS
01.02 SET N=0; SET A=2; SET B=3; SET C=4
01.03 IF (X-(A*B↑C-A*B)/(1-B)) 1.05, 1.04, 1.05
01.04 SET P[1]=1
01.05 IF (Y-(6*A/B↑2)) 1.07, 1.06, 1.07
01.06 SET P[2]=1
01.07 IF (Z-1/FSQT(C/(A*B))) 1.09, 1.08, 1.09
01.08 SET P[3]=1
01.09 TYPE !!!
01.10 FOR I=1,3; IF (P[I]-1) 1.11; SET N=N+1
01.11 CONTINUE
01.12 TYPE "YOU WERE CORRECT ON ",%1.0,N,%8.04," OF THE 3 PROBLEMS",!!
01.13 IF (-N) 1.14; TYPE "I'D SUGGEST SOME EXTRA HELP.",!!; QUIT
01.14 IF (N-3) 1.15; TYPE "AN EXCELLENT JOB.",!!; QUIT
01.15 TYPE "THE VARIABLES YOU MISSED WERE:  "
01.16 IF (1-P[1]) 1.17, 1.17; TYPE "X  "
01.17 IF (1-P[2]) 1.18, 1.18; TYPE "Y  "
01.18 IF (1-P[3]) 1.19, 1.19; TYPE "Z"
01.19 TYPE !!
01.20 ERASE
```

Student work to be checked using this program can be prepared by punching an off-line tape using direct commands:

```
*C STUDENT IDENTIFICATION
*
*SET A=2; SET B=3; SET C=4
*SET X=(A*B↑C-A*B)/(1-B)
*SET Y=6*A/B↑2
*SET Z=1/FSQT(C/(A*B))
*TYPE !,X,!,Y,!,Z,!

=-   78.0000
=+    1.3333
=+    1.2247
*DO 1


YOU WERE CORRECT ON =+3 OF THE 3 PROBLEMS

AN EXCELLENT JOB.

*
```

This job can also be done by preparing a short program such as:

```
*2.01 C STUDENT IDENTIFICATION
*
*2.1 SET A=2; SET B=3; SET C=4
*2.2 SET X=(A*B↑C-A*B)/1-B
*2.3 SET Y=6*A/B↑2
*2.4 SET Z=1/FSQT((C/A)*B)
*2.5 TYPE !,X,!,Y,!,Z,!
*2.6 DO 1
*DO 2

=+ 153.0000
=+   1.3333
=+   0.4082


YOU WERE CORRECT ON =+1 OF THE 3 PROBLEMS

THE VARIABLES YOU MISSED WERE:  X  Z

*
```

In both cases, the student work requires only the addition of the single extra command DO 1 to begin the checking program. Note that each student program begins with a comment line to identify the student. This technique has proven very useful, for it immediately identifies the program being run as well as forming an indelible label on the punched tape itself. Additional use could be made of the checking program by adding the capability of tabulating the number of students who missed each item and typing this information for the teacher after all programs have been checked.

Note that the solution analysis program can be easily modified to a teacher's specific needs. Within this program, the key steps are:

1.02 - A,B, C are defined because students will occasionally assign incorrect values to these variables.

N is used to count the total number of correct answers.

1.03 - 1.08 These steps, taken in pairs, compare the student defined variables X, Y, Z with the correct values. Note that this technique DOES NOT allow the teacher to assign a problem with a correct answer of zero, because this program assigns the value 0 to any undefined variable.

1.10 - 1.11 These steps count the number of correct responses

1.12 - 1.18 These steps are all used for typing the appropriate diagnostic with an appropriate

In order to change the assignment from that used in the example to another one involving different values of A,B,C and different expressions for X,Y,Z, only steps 1.02, 1.03, 1.05, and 1.07 need to be changed. To add more expressions to this assignment, steps to check the additional variables would be inserted between 1.08 and 1.09 and after 1.18.

B. The Signs of Equality and Inequality

In most classes the basic concepts of equality and inequality are already understood and little or no computer assistance is necessary. However, this topic does complement the simultaneous presentation of programming using an IF statement. A useful exercise for students is to write a program which will tell whether or not two given expressions are equal. As well as requiring the use of an IF statement, this program demonstrates a basic technique for writing a program which will accept a general expression for later evaluation. A good student-written program is:

```
01.10 C TEST FOR EQUALITY
01.20 TYPE "TO TEST FOR EQUALITY, SET YOUR FIRST EXPRESSION EQUAL",!
01.30 TYPE "TO  A  AND YOUR SECOND EXPRESSION EQUAL TO  B  , THEN",!
01.40 TYPE "TYPE DO 2.",!!

02.10 IF (A-B) 2.2, 2.3, 2.2
02.20 TYPE "YOUR EXPRESSIONS ARE NOT EQUAL.",!!; QUIT
02.30 TYPE "YOUR EXPRESSIONS ARE EQUAL.",!!
*
```

Note that part 1 is used only to type out directions and is not really a necessary part of the program. A sample run of this program is:

```
*DO 1
TO TEST FOR EQUALITY, SET YOUR FIRST EXPRESSION EQUAL
TO  A  AND YOUR SECOND EXPRESSION EQUAL TO  B  , THEN
TYPE DO 2.

*SET A=2↑5
*SET B=2*2*2*2*2
*DO 2
YOUR EXPRESSIONS ARE EQUAL.

*SET A=.001+8
*SET B=.008+1
*DO 2
YOUR EXPRESSIONS ARE NOT EQUAL.

*
```

The student exercise might be modified by requiring the program to distinguish between the three cases: equal to, greater than, or less than.

Even in the sample program, some round-off error will occur due to computer approximations (10*.2 and 10*(1/5) will be called not equal); thus better students might enjoy the challenge of trying to reduce this error. One possible solution is:

```
02.10 IF (FITR((A-B)*100)) 2.2, 2.3, 2.2
02.20 TYPE "YOUR EXPRESSIONS ARE NOT EQUAL.",!!; QUIT
02.30 TYPE "YOUR EXPRESSIONS ARE EQUAL.",!!
*
```

This use of the integer part function will eliminate all round-off error before the third decimal place, but will indicate equality for two expressions which differ after the second decimal place.

C. Introducing Exponents

When introducing exponential notation, the computer can be used with programs much like those already discussed in this chapter. By having students write exponential expressions on a single line (Section I-A) they receive practice with exponents as well as reinforcement of earlier work. Similarly, tests for equality (Section I-B) can be run with expressions like 5^{2^3} and $(5^2)^3$.

By limiting the set of numbers being considered, discussions centered around questions like "Is X^3 always greater than X^2?" are often beneficial. To supplement these discussions, one could use programs like those in Section I-B, or simply direct commands which can be used to examine several cases:

```
*FOR X=-5,5; TYPE X,"    ",X↑2,"    ",X↑3,!
=-   5    =+  25    =- 125
=-   4    =+  16    =-  64
=-   3    =+   9    =-  27
=-   2    =+   4    =-   8
=-   1    =+   1    =-   1
=+   0    =+   0    =+   0
=+   1    =+   1    =+   1
=+   2    =+   4    =+   8
=+   3    =+   9    =+  27
=+   4    =+  16    =+  64
=+   5    =+  25    =+ 125
*
```

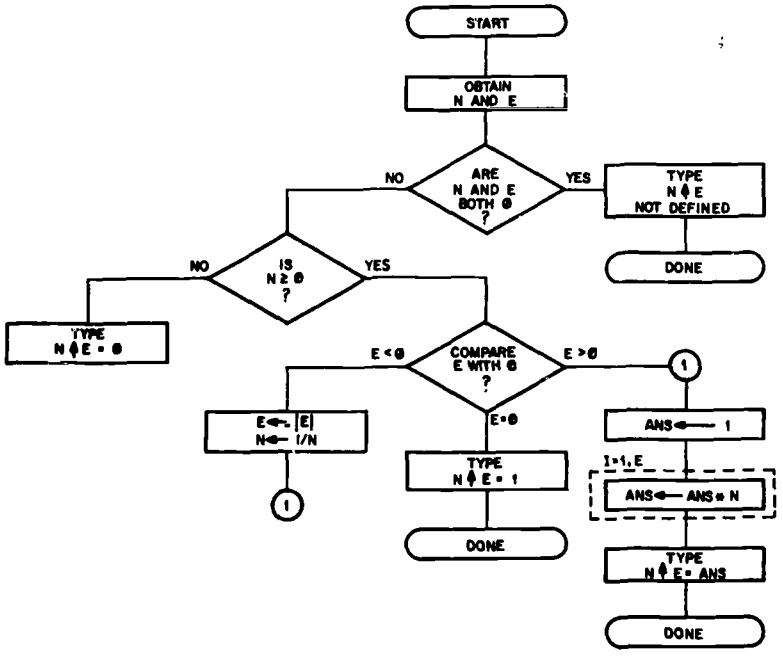
Students often enjoy finding the algebraic expressions used to generate a given table. For example, one might ask what expressions were used to generate:

-3	+8	+12
-2	+3	+6
-1	+0	+2
+0	-1	+0
+1	+0	+2
+2	+3	+6
+3	+8	+12

The correct expressions are X, X^2-1, and X^2-X.

Students are capable of finding surprisingly complex expressions, even though they are approaching this exercise on an intuitive level. Using this type of exercise also introduces a systematic method of search which has wide application in other areas.

The writing of a program which uses neither functions nor the operator ↑ to compute N^E (where N = any number and E = any integer) is a good assignment for better students in Algebra I and all students in Algebra II who are convinced they have already mastered the use of exponents. To write such a program, the student must completely understand the definition and all of the special cases that can occur. One solution to this problem is:



A program which follows this flow chart is:

```
01.01 C PROGRAM TO COMPUTE N↑E, WHERE E IS ANY INTEGER.
01.02 ASK ?N?,?   E?,"   N↑E "
01.03 IF (FABS(N)+FABS(E))  1.05,1.04,1.05
01.04 TYPE " IS NOT DEFINED."!!; QUIT
01.05 IF (N)  1.07,1.06,1.07
01.06 TYPE %,0,!!;  QUIT
01.07 IF (E)  1.09, 1.08, 1.10
01.08 TYPE %,1,!!;  QUIT
01.09 SET N=1/N; SET E=-E
01.10 SET ANS=1; FOR I=1,E; SET ANS=ANS*N
01.11 TYPE %, ANS,!!
*
```

Several runs of this program are:

```
*GO
N:-2   E:5    N↑E =-0.320000E+02

*GO
N:0    E:0  N↑E    IS NOT DEFINED.

*GO
N:9    E:-3    N↑E =+0.137174E-02

*GO
N:23    E:23    N↑E =+0.208805E+32

*
```

Note that the program shown does not contain provisions for rejecting fractional values of E. Classroom experience with this assignment has indicated that this is a challenging problem for the suggested group of students. The most common student error is that of omitting the exception 0 ↑ 0.

D. Plotting Equalities and Inequalities on the Number Line

This program is an excellent demonstration device as well as a challenging assignment for better students studying first year algebra. Any equality or inequality can be plotted on the number line over the interval -8 through +8. This choice of limits is quite arbitrary, and the program can easily be modified to accommodate any other limits desired. The basic program appears as:

```
01.10 TYPE !"USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:"!!" "
01.20 FOR N=-8,.25,8; DO 2
01.30 TYPE !"-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5"
01.40 TYPE "   6   7   8"!!!; QUIT

02.90 TYPE "."; CONTINUE

03.90 TYPE "X"; CONTINUE
*


*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ..................................................................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8


*
```

Only the number line was typed out because no other conditions were specified. The specific equality or inequality to be plotted must be described using any of the step numbers 2.01 through 2.89 and the IF command. This command should transfer to step 2.9 for a point that is NOT to be plotted and to step 3.9 for a point that is to be plotted. Suppose we wish to plot all $N \leq 0$. The instruction needed is

```
2.1  IF (N) 3.9,  3.9,  2.9
```

point IS included if ≤ 0.

point NOT included if > 0.

A sample run is:

```
*2.1 IF (N) 3.9, 3.9, 2.9
*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX................................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8


*
```

One can also indicate the inclusion or exclusion of a terminating point on the graph. The previous example plotted $N \leq 0$, while this example plots $N < 0$. Note that 0 is not included in this case.

```
*2.1 IF (N) 3.9, 2.9, 2.9
*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.................................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8


*
```

Expressions involving absolute value can also be plotted. For example, $|N| = 3$ appears as:

```
*2.1 IF (FABS(N)-3) 2.9, 3.9, 2.9
*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ....................X.........................X....................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8


*
```

The program can be used to plot inequalities according to student instructions, or to rapidly give inequalities which students must identify. By preparing a tape containing instructions for several different conditions, a wide variety of examples could be presented using only a small amount of class time.

The same program can also plot more complicated conditions such as $N \leq -4$ or $N > 2$. The needed instructions are:

```
                                                  plot if N>2
2.1  IF (N-2)  2.2,  2.2,,  3.9
                2nd condition must be checked if N≤2

2.2  IF (N+4)  3.9,  3.9,,  2.9
                plot if N≤-4        point rejected since N>-4 AND N≤2
```

A run of the program including these instructions, appears as:

```
*2.1 IF (N-2) 2.2, 2.2, 3.9
*2.2 IF (N+4) 3.9, 3.9, 2.9
*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 XXXXXXXXXXXXXXXXX.....................XXXXXXXXXXXXXXXXXXXXXXX
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

By working with several examples similar to this one, the students receive an introduction to logical AND/OR statements. The following graph represents the condition $2 < N \leq 4$ (i.e., a program using AND).

```
*2.1 IF (N-2) 2.9, 2.9, 2.2
*2.2 IF (N-4) 3.9, 3.9, 2.9
*DO 1

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ..........................................XXXXXXXX.............
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

When students are convinced that they understand this topic, the following two examples will check their confidence.

```
*2.1 IF (N-2) 2.2, 2.9, 2.9
*2.2 IF (N-4) 2.9, 3.1, 3.1
*DO 1
```

represents $2 > N \geq 4$

```
USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ..........................................................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8
```

```
*2.1 IF (N-2) 3.9, 2.2, 2.2
*2.2 IF (N+4) 2.9, 3.9, 3.9
*DO 1
```

represents $N < 2$ or $N \geq -4$

```
USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

E. Sets (Union, Intersection, and Subsets)

Determining an expression to represent the number of subsets of a given set containing N elements is a good individual or class project. Students might be led to discovering the expression 2^N by using the following program:

```
01.01 TYPE "HOW MANY ELEMENTS IN YOUR SET?  "; ASK N
01.02 TYPE "THAT SET HAS ", %19.0, 2↑N, " SUBSETS.",!!
*


*GO
HOW MANY ELEMENTS IN YOUR SET?  :23
THAT SET HAS =+            8388610 SUBSETS.

*GO
HOW MANY ELEMENTS IN YOUR SET?  :63
THAT SET HAS =+9223370000000000000 SUBSETS.

*
```

NOTE

2^{63} is the largest number acceptable in the indicated format.

Another program which can be used for demonstration when presenting the topic of subsets is the following which lists all subsets of a given set.

```
01.01 C PROGRAM WHICH WILL LIST ALL SUBSETS OF A GIVEN SET
01.02 TYPE "THIS PROGRAM WILL LIST ALL SUBSETS OF A GIVEN SET",!
01.03 TYPE "OF NUMBERS.  HOW MANY ELEMENTS ARE IN YOUR SET?  "
01.04 ASK N
01.05 TYPE "NOW TYPE EACH OF THE ELEMENTS.",!
01.06 FOR I=1,N; ASK E[I]
01.07 TYPE !,"THE SUBSETS ARE:",!,"NULL SET",!,%4.0
01.08 FOR I=1,N; SET P[I]=0
01.09 FOR C=1,2↑N-1; DO 2

02.01 FOR I=1,N; IF (1-P[I]) 2.02, 2.02; SET J=I; SET I=N
02.02 CONTINUE
02.03 FOR I=1,J; SET P[I]=FABS(P[I]-1)
02.04 FOR I=1,N; IF (P[I]-1) 2.05; TYPE E[I]," "
02.05 CONTINUE
02.06 TYPE !
```

A sample run of this program is:

```
*DO 1
THIS PROGRAM WILL LIST ALL SUBSETS OF A GIVEN SET
OF NUMBERS.  HOW MANY ELEMENTS ARE IN YOUR SET?  :3
NOW TYPE EACH OF THE ELEMENTS.
:1
:2
:3

THE SUBSETS ARE:
NULL SET
=+    1
=+    2
=+    1   =+    2
=+    3
=+    1   =+    3
=+    2   =+    3
=+    1   =+    2   =+    3
*
```

The writing of this program is a difficult assignment for most students, but a class discussion of the algorithm used has several benefits. The program constructs an N-digit binary counter (N is the number of elements in the set), associates each element of the set with the digit position in the binary counter, and then types out each element of the set when its corresponding binary position has a value of 1. In the previous sample run the program set up the three-digit binary counter:

1	2	3	elements of set binary positions
x	x	x	

The typeout then followed the pattern:

BINARY COUNTER	TYPE OUT
0 0 0	NULL SET
1 0 0	1
0 1 0	2
1 1 0	1 2
0 0 1	3
1 0 1	1 3
0 1 1	2 3
1 1 1	1 2 3

Note that the positons of the binary counter must be considered rather then the base 10 representation of the binary number.

An unexpectedly challenging student exercise is the writing of a program which tells whether or not two given sets are equal. One student's solution to this problem is:

```
01.01 TYPE "THIS PROGRAM WILL TELL WHETHER OR NOT TWO SETS ARE EQUAL."
01.02 TYPE !"HOW MANY ELEMENTS ARE IN EACH OF YOUR SETS? "; ASK N
01.03 IF (N) 1.02, 1.02; IF (FITR(N)-N) 1.02, 1.04, 1.02
01.04 TYPE !"THE ELEMENTS OF YOUR 1ST SET ARE?"!; FOR I=1,N; ASK A[I]
01.05 TYPE !"THE ELEMENTS OF YOUR 2ND SET ARE?"!; FOR I=1,N; ASK B[I]
01.06 FOR Q=1,N; DO 2
01.07 TYPE !"THE SETS ARE EQUAL."!!; QUIT

02.01 FOR J=0; FOR I=Q,N; IF (A[Q]-B[I]) 2.03, 2.02, 2.03
02.02 SET S=B[Q]; SET B[Q]=B[I]; SET B[I]=S; SET J=1; SET I=N
02.03 CONTINUE
02.04 IF (-J) 2.05; TYPE !"THE SETS ARE NOT EQUAL."!!; QUIT
02.05 CONTINUE
```

Two runs of this program are:

```
*DO 1
THIS PROGRAM WILL TELL WHETHER OR NOT TWO SETS ARE EQUAL.
HOW MANY ELEMENTS ARE IN EACH OF YOUR SETS? :3

THE ELEMENTS OF YOUR 1ST SET ARE?
:2
:3
:1

THE ELEMENTS OF YOUR 2ND SET ARE?
:3
:1
:2

THE SETS ARE EQUAL.

*


*DO 1
THIS PROGRAM WILL TELL WHETHER OR NOT TWO SETS ARE EQUAL.
HOW MANY ELEMENTS ARE IN EACH OF YOUR SETS? :2

THE ELEMENTS OF YOUR 1ST SET ARE?
:5
:5

THE ELEMENTS OF YOUR 2ND SET ARE?
:5
:0

THE SETS ARE NOT EQUAL.

*
```

Note that step 1.03 of this program is used to reject improper values for N and is not a necessary step in the solution. When this problem is assigned to a class, many different, yet correct, algorithms usually result. If time permits, a class discussion of several different solutions is a good introduction to the concepts of searching and comparing two files.

The intersection and union of two sets can be demonstrated using the following program. This problem is a good programming exercise for students familiar with programming techniques, but is best used as a demonstration program when this topic is being introduced to more elementary students. Note that the program will correctly handle the case when one or both sets are empty, but will not work when one of the sets contains two or more identical elements. The program is:

```
01.01 ERASE
01.02 TYPE "PROGRAM TO FIND THE UNION AND INTERSECTION OF TWO SETS."
01.03 TYPE !!"HOW MANY ELEMENTS IN SET A? "; ASK N
01.04 IF (N) 1.06, 1.06, 1.05
01.05 TYPE "THESE ARE?"!; FOR I=1,N; ASK A[I]
01.06 TYPE !"HOW MANY ELEMENTS IN SET B? "; ASK M
01.07 IF (M) 1.09, 1.09, 1.08
01.08 TYPE "THESE ARE?"!; FOR I=1,M; ASK B[I]
01.09 IF (M+N) 4.02, 4.02; TYPE !"THE UNION CONTAINS:"!
01.10 SET I=1; IF (N) 1.11, 1.11; FOR I=1,N; SET U[I]=A[I]
01.11 IF (M) 1.12, 1.12; FOR J=1,M; SET MT=0; DO 2
01.12 FOR J=1,I-1; TYPE U[J],!
01.13 TYPE !"THE INTERSECTION CONTAINS:"!
01.14 SET T=0; FOR I=1,N; FOR J=1,M; DO 3
01.15 IF (-T) 1.16; TYPE "NO ELEMENTS"!!; QUIT
01.16 FOR I=1,T; TYPE IN[I],!
01.17 QUIT

02.02 FOR K=1,N; IF (B[J]-A[K]) 2.04, 2.03, 2.04
02.03 SET MT=-1; SET K=N
02.04 CONTINUE
02.05 IF (MT) 2.06; SET U[I]=B[J]; SET I=I+1
02.06 CONTINUE

03.02 IF (A[I]-B[J]) 3.04, 3.03, 3.04
03.03 SET T=T+1; SET IN[T]=A[I]; SET J=M
03.04 CONTINUE

04.02 TYPE !"UNION AND INTERSECTION ARE EMPTY."!!
*
```

Within this program, part 2 is used to determine the union of the sets ond port 3 determines the intersection of the sets. Steps 1.02 through 1.08 ore used to obtain the elements of the two sets to be compared. Three runs of this program are:

```
*DO 1
PROGRAM TO FIND THE UNION AND INTERSECTION OF TWO SETS.

HOW MANY ELEMENTS IN SET A? :0

HOW MANY ELEMENTS IN SET B? :0

UNION AND INTERSECTION ARE EMPTY.

*


*DO 1
PROGRAM TO FIND THE UNION AND INTERSECTION OF TWO SETS.

HOW MANY ELEMENTS IN SET A? :3
THESE ARE?
:5
:6
:2

HOW MANY ELEMENTS IN SET B? :4
THESE ARE?
:0
:4
:2
:5

THE UNION CONTAINS:
=+      5
=+      6
=+      2
=+      0
=+      4

THE INTERSECTION CONTAINS:
=+      5
=+      2
*


*DO 1
PROGRAM TO FIND THE UNION AND INTERSECTION OF TWO SETS.

HOW MANY ELEMENTS IN SET A? :0

HOW MANY ELEMENTS IN SET B? :3
THESE ARE?
:1
:2
:3

THE UNION CONTAINS:
=+      1
=+      2
=+      3

THE INTERSECTION CONTAINS:
NO ELEMENTS

*
```

Many students will enjoy writing programs which simply list the elements of a set whose definition is given. For example, assuming the Universal Set to be the positive integers 1 through 99, the following student program lists the set whose elements are divisible by 16 and end in 2 or 0.

```
01.10 IF (FITR(N/16)-N/16) 1.9, 1.2, 1.9; C DIVISIBLE BY 16?
01.20 IF (N-10*FITR(N/10)) 1.3, 1.8, 1.3; C IS LAST DIGIT 0?
01.30 IF (N-10*FITR(N/10)-2) 1.9, 1.8, 1.9; C IS LAST DIGIT 2?
01.80 TYPE N,!
01.90 CONTINUE
*
```

The Universal Set being considered is determined by this FOR command.

```
*FOR N=1,99; DO 1
=+   32
=+   80
*
```

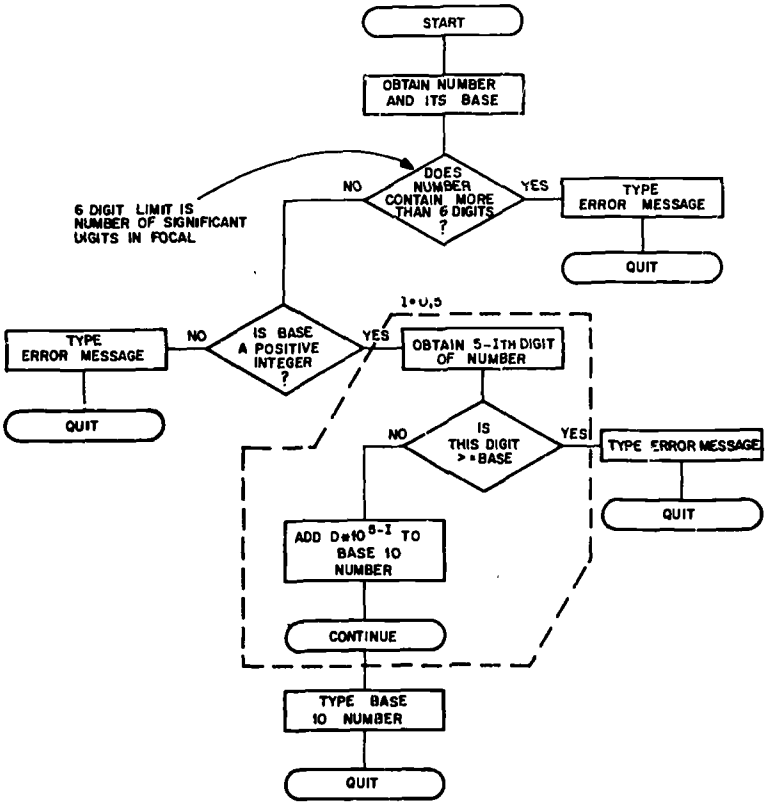
By writing programs such as this, students reinforce their knowledge of logical AND/OR statements. They also begin to appreciate the value of considering all cases when correct but unexpected elements appear in their result. This type of assignment is also easily individualized, as the set definition can be made as complicated as desired for each student. As a final example, consider the following program which lists the set of two-digit positive integers (a leading zero is assumed) in which one digit is divisible by the other OR vice versa. Note that the student must eliminate the possibility of trying to divide by zero. The program is:

```
01.10 SET T=FITR(N/10); SET U=N-10*T; C FIND TENS AND UNITS DIGITS
01.20 IF (U) 1.3, 1.3; IF (FITR(T/U)-T/U) 1.3, 1.8, 1.3
01.30 IF (T) 1.9, 1.9; IF (FITR(U/T)-U/T) 1.9, 1.8, 1.9
01.80 TYPE N,!
01.90 CONTINUE
*

*FOR N=1,99; DO 1
=+    1
=+    2
=+    3
=+    4
=+    5
=+    6
=+    7
=+    8
=+    9
=+   10
=+   11
=+   12
=+   13
=+   14
=+   15
=+   16
=+   17
```

```
=+   18
=+   19
=+   20
=+   21
=+   22
=+   24
=+   26
=+   28
=+   30
=+   31
=+   33
=+   36
=+   39
=+   40
=+   41
=+   42
=+   44
=+   48
=+   50
=+   51
=+   55
=+   60
=+   61
=+   62
=+   63
=+   66
=+   70
=+   71
=+   77
=+   80
=+   81
=+   82
=+   84
=+   88
=+   90
=+   91
=+   93
=+   99
*
```

F. Numbers in Other Bases (Supplementary)

A program to convert integers from base N to base 10 is useful for both demonstrations and student assignments. Equally useful, but decidedly more challenging, is a program to convert any number from base N to base 10. The following flow chart represents an algorithm for the first of these problems.



The program appears as:

```
01.01 TYPE "THIS PROGRAM CONVERTS INTEGERS (UP TO 6 DIGITS) FROM"!;ERASE
01.02 TYPE "BASE N TO BASE 10."!; ASK ?NUMBER?, ?BASE?
01.03 IF (FABS(NU)-999999) 1.04,1.04;TYPE !"NO MORE THAN 6 DIGITS"!;QUIT
01.04 IF (BA) 2.03, 2.03; IF (FITR(BA)-BA) 2.03, 1.05, 2.03
01.05 SET A=0; SET S=FSGN(NU); SET NU=FABS(NU); FOR I=0,5; DO 2
01.06 TYPE "NUMBER IN BASE 10 IS ", %8.0, S*A,!!!; QUIT

02.01 SET D=FITR(NU/10↑(5-I)); SET NU=NU-D*10↑(5-I)
02.02 IF (D-BA) 2.04, 2.03, 2.03
02.03 TYPE !"YOU'VE USED AN IMPROPER BASE."!; QUIT
02.04 SET A=A+D*BA↑(5-I)
*
```

Note that steps 1.03, 1.04, 2.02, and 2.03 are all used to eliminate incorrect values of number and base. When using this problem as an assignment for students, the program is much simpler if these checks or input numbers are not required. Several runs of this program are:

```
*DO 1
THIS PROGRAM CONVERTS INTEGERS (UP TO 6 DIGITS) FROM
BASE N TO BASE 10.
NUMBER:1101
BASE:2
NUMBER IN BASE 10 IS =+       13


*DO 1
THIS PROGRAM CONVERTS INTEGERS (UP TO 6 DIGITS) FROM
BASE N TO BASE 10.
NUMBER:4321
BASE:7
NUMBER IN BASE 10 IS =+     1534


*DO 1
THIS PROGRAM CONVERTS INTEGERS (UP TO 6 DIGITS) FROM
BASE N TO BASE 10.
NUMBER:1234
BASE:3

YOU'VE USED AN IMPROPER BASE.
*
```

A program to convert an integer from base 10 to base N is quite similar to the previous example in both usefulness and form. The amount of class time required for the two problems can be reduced by discussing one program in class, then assigning the other as homework. A student written program for converting integers from base 10 to base N is:

```
01.01 TYPE "THIS PROGRAM CONVERTS AN INTEGER FROM BASE 10 "; ERASE
01.02 TYPE "TO BASE N."!; ASK ?NUMBER?, ?BASE?,!
01.03 SET SN=FSGN(NU); SET NU=FABS(NU)
01.04 IF (BA-2) 1.05; IF (10-BA) 1.06; IF (FITR(BA)-BA) 1.05,1.07,1.05
01.05 TYPE "THAT IS NOT A VALID BASE."!!; QUIT
01.06 TYPE "PROGRAM CAN ONLY HANDLE BASES LESS THAN 11"!!;QUIT
01.07 IF (NU-BA↑6) 1.08; TYPE "NUMBER TOO LARGE FOR PROGRAM"!!; QUIT
01.08 FOR I=0,5; DO 2
01.09 TYPE %8.0,"THE NUMBER IN BASE", BA, "  IS", T*SN,!!; QUIT

02.01 SET X=FITR(NU/BA↑(5-I)); SET NU=NU-X*BA↑(5-I); SET T=T+X*10↑(5-I)
*
```

Note that steps 1.04 through 1.07 are all used to eliminate incorrect values of number and base. Since these steps are not a part of the actual base-changing algorithm, their inclusion might be an optional or extra-credit part of an assignment. Several runs of this program are:

```
*DO 1
THIS PROGRAM CONVERTS AN INTEGER FROM BASE 10 TO BASE N.
NUMBER:123 BASE:2
NUMBER TOO LARGE FOR PROGRAM

*DO 1
THIS PROGRAM CONVERTS AN INTEGER FROM BASE 10 TO BASE N.
NUMBER:4321 BASE:8
THE NUMBER IN BASE=+        8  IS=+    10341

*DO 1
THIS PROGRAM CONVERTS AN INTEGER FROM BASE 10 TO BASE N.
NUMBER:12 BASE:-3
THAT IS NOT A VALID BASE.

*DO 1
THIS PROGRAM CONVERTS AN INTEGER FROM BASE 10 TO BASE N.
NUMBER:-234 BASE:5
THE NUMBER IN BASE=+        5  IS=-     1414

*
```

II. SOLVING (BY SUBSTITUTION) EQUATIONS AND INEQUALITIES IN ONE VARIABLE

A. Solving Equations

Students are often introduced to the concept of solving equations on an intuitive basis. The programs in this section are designed to complement this approach as well as to assist students in developing their ability to guess approximate roots. All of the programs are suitable for demonstration or student assignments. Since there is little programming skill involved in most of the examples shown, the problems are good exercises for students who are still becoming familiar with the computer.

The first program allows the user to check the solution to any equation. Note that the left and right sides of the equation are entered independently - the students do not have to be able to put all terms on one side of the equation in order to write or use the program. The program appears as:

```
01.01 C PROGRAM TO CHECK THE SOLUTION TO AN EQUATION BY SUBSTITUTING
01.02 C ANY GIVEN VALUE INTO THE EQUATION. TO USE THE PROGRAM, USE
01.03 C PART 2 TO SET L = TO LEFT SIDE OF EQUATION AND R = RIGHT SIDE
01.04 C OF EQUATION.  YOU MUST USE  X  AS THE VARIABLE IN PART 2.
01.05 ASK "YOUR SOLUTION IS X=", X; DO 2
01.06 IF (L-R) 1.08, 1.07, 1.08
01.07 TYPE "THAT VALUE IS CORRECT!!"!!; QUIT
01.08 TYPE "THAT VALUE IS NOT CORRECT!!"!!; GOTO 1.05
*
```

Two runs of the program, first with $-3x = 9$, then $4x - 5 = 2x+7$, are

```
*2.1 SET L=-3*X
*2.2 SET R=9
*DO 1
YOUR SOLUTION IS X=:-3
THAT VALUE IS CORRECT!!

*2.1 SET L=4*X-5
*2.2 SET R=2*X+7
*DO 1
YOUR SOLUTION IS X=:-6
THAT VALUE IS NOT CORRECT!!

YOUR SOLUTION IS X=:6
THAT VALUE IS CORRECT!!

*
```

Although both examples had integral solutions, the program will work for any solution expressible in six significant digits.

The next program is simply an extension of the last one. Instead of just labeling the user's value of x as correct or incorrect, the program types out the value of each side of the equation when an in-

correct solution is given. Since this program is almost identical to the previous one (steps 1.09 and 1.10 have been added and step 1.08 changed), a good teaching sequence is to use the first program for demonstration and the following as a student assignment.

```
01.01 C PROGRAM TO CHECK THE SOLUTION TO AN EQUATION BY SUBSTITUTING
01.02 C ANY GIVEN VALUE INTO THE EQUATION. TO USE THE PROGRAM, USE
01.03 C PART 2 TO SET L = TO LEFT SIDE OF EQUATION AND R = RIGHT SIDE
01.04 C OF EQUATION.  YOU MUST USE  X  AS THE VARIABLE IN PART 2.
01.05 ASK "YOUR SOLUTION IS X=", X; DO 2
01.06 IF (L-R) 1.08, 1.07, 1.08
01.07 TYPE "THAT VALUE IS CORRECT!!"!!; QUIT
01.08 TYPE "THAT VALUE IS NOT CORRECT!!"!"USING THAT VALUE OF X, "
01.09 TYPE "YOU HAVE:"!"LEFT SIDE OF EQUATION  ",%8.04,L
01.10 TYPE !"RIGHT SIDE OF EQUATION ",R,!!; GOTO 1.05
*
```

Runs of this program using the equations $-3x = 9$ and $4x-5 = 2x+7$ are:

```
*2.1 SET L=-3*X
*2.2 SET R=9
*DO 1
YOUR SOLUTION IS X=:-3
THAT VALUE IS CORRECT!!

*2.1 SET L=4*X-5
*2.2 SET R=2*X+7
*DO 1
YOUR SOLUTION IS X=:-6
THAT VALUE IS NOT CORRECT!!
USING THAT VALUE OF X, YOU HAVE:
LEFT SIDE OF EQUATION  =-  29.0000
RIGHT SIDE OF EQUATION =-   5.0000

YOUR SOLUTION IS X=:6
THAT VALUE IS CORRECT!!

*
```

Student interest will quickly turn to writing a program which will solve an equation rather then merely confirm the user's solution. This can be done at an introductory stage of equation solving by writing a program which will search a given interval for a possible solution. Doing this provides an excellent introduction to simple searching techniques and, more important, helps students develop the ability to find an approximate interval in which a solution lies.

The following program accepts equations in the same form as the previous two examples, asks for the interval (from A to B) to be searched, and then types all integral solutions in that interval.

The program assumes that A < B and that A and B are both integers, but these restrictions can easily be removed. The restriction that only integral solutions can be found is dictated by the incrementing of x in step 1.09, thus this restriction can also be easily removed. The program is:

```
01.01 C PROGRAM TO SEARCH FOR THE INTEGRAL SOLUTION TO AN EQUATION
01.02 C OVER A GIVEN INTERVAL. TO USE THE PROGRAM, USE PART 2 TO
01.03 C SET L = LEFT SIDE OF EQUATION AND R = RIGHT SIDE OF EQUATION.
01.04 C YOU MUST USE  X  AS THE VARIABLE IN PART 2.  THE PROGRAM WILL
01.05 C ASK FOR THE INTERVAL YOU WISH TO SEARCH.
01.06 ASK "SEARCH INTERVAL FROM ",A," TO ",B,!; SET X=A
01.07 DO 2; IF (L-R) 1.09, 1.08, 1.09
01.08 TYPE "A SOLUTION IS ", %6.0, X,!
01.09 SET X=X+1; IF (X-B) 1.07, 1.07; QUIT
*
```

Two runs of the program are:

```
*2.1 SET L=4*X-5
*2.2 SET R=2*X+7
*DO 1
SEARCH INTERVAL FROM :0  TO :10
A SOLUTION IS =+      6
*
```

```
*2.1 SET L=X*(X-1)
*2.2 SET R=2
*DO 1
SEARCH INTERVAL FROM :5  TO :15
*DO 1
SEARCH INTERVAL FROM :-10  TO :5
A SOLUTION IS =-      1
A SOLUTION IS =+      2
*
```

← note that no solution was found in this interval.

The writing of this program, or the rewriting of the program to eliminate the restrictions on A, B, and the nature of the solution, are both good student exercises. Better students might be assigned the task of writing a program which would type out the value of x which comes closest to making both sides of the equation equal when the exact solution is not found.

B. Solving Inequalities

Introductory programs for solving inequalities by substitution are quite similar to the three previous examples used for solving equations. The following program allows the user to confirm a solution to any inequality. The input form is exactly as in the second equality program, with the added

necessity of entering the appropriate inequality sign (GR for >, GE for >=, LE for <=, or LS for <). Note that FOCAL automatically assigns the values GR = 72, GE = 7, LE = 12, and LS = 123 when these characters are typed in response to an ASK command. The program is:

```
01.01 C PROGRAM TO CHECK A SOLUTION TO AN INEQUALITY BY SUBSTITUTING
01.02 C ANY GIVEN VALUE INTO THE INEQUALITY. TO USE THE PROGRAM, USE
01.03 C PART 2 TO SET L = LEFT SIDE OF INEQUALITY AND R = RIGHT SIDE
01.04 C OF INEQUALITY.  X MUST BE USED AS THE VARIABLE IN PART 2.
01.05 C WHEN PROGRAM ASKS FOR "INEQ", TYPE GR, GE, LE, OR LS
01.06 C TO INDICATE THE SENSE OF YOUR INEQUALITY.
01.07 ASK ?INEQ?
01.08 ASK "A SOLUTION IS X=",X; DO 2
01.09 IF (7-IN) 1.10; IF (L-R) 1.14, 1.13, 1.13
01.10 IF (12-IN) 1.11; IF (L-R) 1.13, 1.13, 1.14
01.11 IF (72-IN) 1.12; IF (L-R) 1.14, 1.14, 1.13
01.12 IF (L-R) 1.13, 1.14, 1.14
01.13 TYPE "THAT VALUE IS CORRECT!!"!!; QUIT
01.14 TYPE "THAT VALUE IS NOT CORRECT."!"USING THAT VALUE OF X, "
01.15 TYPE "YOU HAVE:"!"LEFT SIDE OF INEQUALITY  ",%8.04,L
01.16 TYPE !"RIGHT SIDE OF INEQUALITY ",R,!!; GOTO 1.08
*
```

Two runs of this program are:

```
*2.1 SET L=2*X+3
*2.2 SET R=X-1
*DO 1                    } Input: 2x + 3<x - 1
INEQ:LS
A SOLUTION IS X=:4
THAT VALUE IS NOT CORRECT.
USING THAT VALUE OF X, YOU HAVE:
LEFT SIDE OF INEQUALITY  =+  11.0000
RIGHT SIDE OF INEQUALITY =+   3.0000

A SOLUTION IS X=:-4
THAT VALUE IS NOT CORRECT.
USING THAT VALUE OF X, YOU HAVE:
LEFT SIDE OF INEQUALITY  =-   5.0000
RIGHT SIDE OF INEQUALITY =-   5.0000

A SOLUTION IS X=:-5
THAT VALUE IS CORRECT!!

*DO 1
INEQ:GE          <-- Note that the inequality is
A SOLUTION IS X=:0   changed to: 2x + 3≥x - 1
THAT VALUE IS CORRECT!!
*
```

A final example considers the inequality $|x| < 3$:

```
*2.1 SET L=FABS(X)
*2.2 SET R=3
*DO 1
INEQ:LS
A SOLUTION IS X=:-4
THAT VALUE IS NOT CORRECT.
USING THAT VALUE OF X, YOU HAVE:
LEFT SIDE OF INEQUALITY  =+    4.0000
RIGHT SIDE OF INEQUALITY =+    3.0000

A SOLUTION IS X=:2
THAT VALUE IS CORRECT!!

*
```

A similar program that searches a given interval for integral solutions to an inequality is:

```
01.01 C PROGRAM TO SEARCH FOR THE INTEGRAL SOLUTIONS TO AN INEQUALITY
01.02 C OVER A GIVEN INTERVAL.  TO USE THE PROGRAM, USE PART 2 TO
01.03 C SET L = LEFT SIDE OF INEQUALITY AND R = RIGHT SIDE OF
01.04 C INEQUALITY.  YOU MUST USE  X  AS THE VARIABLE IN PART 2.
01.05 C THE PROGRAM WILL ASK FOR THE INTERVAL YOU WISH TO SEARCH.
01.06 C WHEN PROGRAM ASKS FOR "INEQ", TYPE GR, GE, LE, OR LS TO
01.07 C INDICATE THE SENSE OF YOUR INEQUALITY.
01.08 ASK ?INEQ?,!"SEARCH INTERVAL FROM ",A, " TO ",B,!; SET X=A
01.09 TYPE "INTEGRAL SOLUTIONS IN THAT INTERVAL ARE:"!
01.10 DO 2; IF (7-IN) 1.11; IF (L-R) 1.15, 1.14, 1.14
01.11 IF (12-IN) 1.12; IF (L-R) 1.14, 1.14, 1.15
01.12 IF (72-IN) 1.13; IF (L-R) 1.15, 1.15, 1.14
01.13 IF (L-R) 1.14, 1.15, 1.15
01.14 TYPE %6.0, X, !
01.15 SET X=X+1; IF (X-B) 1.10, 1.10; QUIT
*
```

Three runs of this program are:

```
*2.1 SET L=2*X+3
*2.2 SET R=X-1
*DO 1
INEQ:LE
SEARCH INTERVAL FROM :0  TO :10
INTEGRAL SOLUTIONS IN THAT INTERVAL ARE:
*
```

$2x+3 \leq x-1$

← No solutions found in this interval.

```
*DO 1
INEQ:LE
SEARCH INTERVAL FROM :-10  TO :0
INTEGRAL SOLUTIONS IN THAT INTERVAL ARE:
=-     10
=-      9
=-      8
=-      7
=-      6
=-      5
=-      4
*
```

```
*2.1 SET L=FABS(X)
*2.2 SET R=3
*DO 1
INEQ:LE
```

Inequality changed to $|x| \leq 3$

```
SEARCH INTERVAL FROM :-15  TO :15
INTEGRAL SOLUTIONS IN THAT INTERVAL ARE:
=-      3
=-      2
=-      1
=+      0
=+      1
=+      2
=+      3
*
```

and finally to $|x| > 3$

```
*DO 1
INEQ:GR
SEARCH INTERVAL FROM :-8  TO :8
INTEGRAL SOLUTIONS IN THAT INTERVAL ARE:
=-      8
=-      7
=-      6
=-      5
=-      4
=+      4
=+      5
=+      6
=+      7
=+      8
*
```

Note that in the preceding two programs no provisions were made to prevent the input of incorrect data. If the interval given does not begin with an integer or the interval begins with a larger number than it ends with, the programs will appear to run, but they will not perform the expected search.

C. Words To Algebraic Expressions

Introducing students to the techniques of converting a series of words to a mathematical equation or expression is probably best done without utilizing the computer. The computer will have previously helped clarify the meaning of a variable, but the actual process of translating a phrase like: "Express the cost of 12 apples and 6 oranges if 3 apples cost x cents and the oranges are five times as expensive as apples" into the expression COST = 4x + 5(2x) is well introduced without involving the computer. However, the computer can be used to reinforce the learning of this concept in either a drill or a "testing" situation. For example, consider the following problems:

1. At the shallow end of a swimming pool, there are 5 steps (each i inches high) leading out of the water. Express the depth of the water in feet.
2. This year Bill earned $400 doing odd jobs. This was an increase of q dollars over his earnings of last year. Express his earnings for last year.
3. A picture frame is 6 inches longer than twice its width (w). Express the length of the frame in inches.
4. Express a number which is 17 less than 3 times a given number N.
5. Tomatoes cost 28¢ per pound more than potatoes. If potatoes cost p cents per pound, express the cost of 5 pounds of potatoes and 2 pounds of tomatoes.
6. Carol is 3 years older than twice Alice's age. If Alice is x years old, express Carol's age.

By having students prepare punched tapes containing their answers in the form

```
2.1 SET A1 = 5*I/12
2.2 SET A2 = 400-q
      :
      :
2.6 SET A6 = 2*X+3
DO 1
```

the following program can be used to check their work

```
01.10 SET I=60; SET Q=190; SET W=30; SET N=8
01.11 SET P=.15; SET X=11.5
01.20 DO 2
01.30 SET S1=25; SET S2=210; SET S3=36; SET S4=7
01.31 SET S5=1.61; SET S6=26
01.40 TYPE "YOU WERE CORRECT ON PROBLEMS:"
01.41 IF (-FABS(A1-S1))  1.42; TYPE " 1"
01.42 IF (-FABS(A2-S2))  1.43; TYPE " 2"
01.43 IF (-FABS(A3-S3))  1.44; TYPE " 3"
01.44 IF (-FABS(A4-S4))  1.45; TYPE " 4"
01.45 IF (-FABS(A5-S5))  1.46; TYPE " 5"
01.46 IF (-FABS(A6-S6))  1.60; TYPE " 6"
01.60 TYPE !!; ERASE
01.70 ERASE 2
01.80 QUIT
*
```

Two different student responses are:

```
*2.1 SET A1=5*I/12
*2.2 SET A2=400-Q
*2.3 SET A3=W+6
*2.4 SET A4=3*N-17
*2.5 SET A5=2*(.28+P)+5*P
*2.6 SET A6=2*X+3
*DO 1
YOU WERE CORRECT ON PROBLEMS: 1 2 3 4 5 6

*
```

and

```
*2.1 SET A1=5*I*12
*2.2 SET A2=400-Q
*2.3 SET A3=W+6
*2.4 SET A4=3*N+17
*2.5 SET A5=2*(28+P)+5*P
*2.6 SET A6=2*X+3
*DO 1
YOU WERE CORRECT ON PROBLEMS: 2 3 6

*
```

NOTE

This program is similar in purpose to that shown on page 4.

The program verifies the student answers by substituting a value (assigned in steps 1.10 and 1.11) into the student's expressions and comparing their results (determined by DOing 2) with the correct results (assigned in steps 1.30 and 1.31). Thus to utilize this program with a different set of six problems, one need only change steps 1.10, 1.11, 1.30, and 1.31. If the number of problems is to be changed, the sequence of steps 1.41 through 1.46 can be increased or decreased as necessary. The only restriction placed on problems to be checked with this program is that no problem should be used for which zero is the correct answ , for if a student fails to do a problem the answer variable is assumed to be zero.

The preceding program is also useful as a drill exercise. This application, however, uses much more computer time since it is most beneficial if the student does all work on-line rather than first preparing punched tapes off line, because while on-line he can immediately modify an incorrect problem. Another program specifically for drill work could be written which would check one problem at a time, and in doing so tell whether the student's expression would yield a result greater than, less than, or equal to the desired result.

D. Solving (by substitution) Equations and Inequalities in Two Variables (Supplementary)

The following program searches a given interval for solutions to an inequality in two variables. The operation of this program is identical to that of the second example given in section B of this chapter; only the input and output forms differ. In this example the program ASKs for both an x and y interval, and the typeout is in the form of ordered pairs. The program is:

```
01.01 C PROGRAM TO SEARCH FOR THE INTEGRAL SOLUTIONS TO AN INEQUALITY
01.02 C (WITH TWO VARIABLES) OVER A GIVEN INTERVAL.  TO USE THE PROGRAM,
01.03 C USE STEPS 2.1 AND 2.2 TO SET L = LEFT SIDE OF INEQUALITY AND
01.04 C R = RIGHT SIDE OF INEQUALITY.  YOU MUST USE  X  AND  Y  AS THE
01.05 C VARIABLES IN PART 2.  THE PROGRAM WILL ASK FOR THE INTERVAL YOU
01.06 C WISH TO SEARCH.  WHEN PROGRAM ASKS FOR "INEQ", TYPE GR, GE,
01.07 C LE, OR LS TO INDICATE THE SENSE OF YOUR INEQUALITY.
01.08 ASK ?INEQ?,!"X  INTERVAL IS FROM ",XS, " TO ",XF,!
01.09 ASK "Y  INTERVAL IS FROM ",YS, " TO ",YF,!
01.10 TYPE "INTEGRAL SOLUTIONS  (X,Y)  IN THAT INTERVAL ARE:"!
01.11 FOR X=XS,XF; FOR Y=YS,YF; DO 2
01.12 QUIT

02.30 IF (7-IN) 2.31; IF (L-R) 2.35, 2.34, 2.34
02.31 IF (12-IN) 2.32; IF (L-R) 2.34, 2.34, 2.35
02.32 IF (72-IN) 2.33; IF (L-R) 2.35, 2.35, 2.34
02.33 IF (L-R) 2.34, 2.35, 2.35
02.34 TYPE %3.0, "(",  X, ",", Y, " )"!
02.35 CONTINUE
*
```

A run of this program using the inequality $x(x+1) \geq y+6$ is:

```
*2.1 SET L=X*(X+1)
*2.2 SET R=Y+6
*DO 1
INEQ:GE
X  INTERVAL IS FROM :-4  TO :4
Y  INTERVAL IS FROM :-2  TO :2
INTEGRAL SOLUTIONS  (X,Y)  IN THAT INTERVAL ARE:
(=-   4,=-   2 )
(=-   4,=-   1 )
(=-   4,=+   0 )
(=-   4,=+   1 )
(=-   4,=+   2 )
(=-   3,=-   2 )
(=-   3,=-   1 )
(=-   3,=+   0 )
(=+   2,=-   2 )
(=+   2,=-   1 )
(=+   2,=+   0 )
(=+   3,=-   2 )
(=+   3,=-   1 )
```

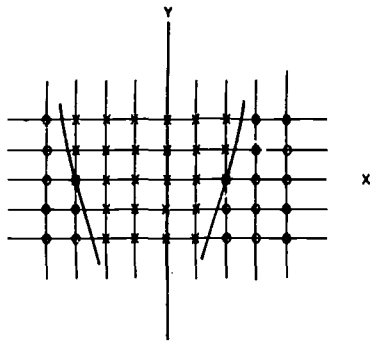
The magnitudes of the limits on the two intervals do not have to be the same as those used in this example.

```
(=+   3,=+   0 )
(=+   3,=+   1 )
(=+   3,=+   2 )
(=+   4,=-   2 )
(=+   4,=-   1 )
(=+   4,=+   0 )
(=+   4,=+   1 )
(=+   4,=+   2 )
*
```

Another run of this program, using the inequality $x(x+1) \leq y+6$, is:

```
*DO 1
INEQ:LE
X  INTERVAL IS FROM :-4  TO :4
Y  INTERVAL IS FROM :-2  TO :2
INTEGRAL SOLUTIONS  (X,Y)  IN THAT INTERVAL ARE:
(=-   3,=+   0 )
(=-   3,=+   1 )
(=-   3,=+   2 )
(=-   2,=-   2 )
(=-   2,=-   1 )
(=-   2,=+   0 )
(=-   2,=+   1 )
(=-   2,=+   2 )
(=-   1,=-   2 )
(=-   1,=-   1 )
(=-   1,=+   0 )
(=-   1,=+   1 )
(=-   1,=+   2 )
(=+   0,=-   2 )
(=+   0,=-   1 )
(=+   0,=+   0 )
(=+   0,=+   1 )
(=+   0,=+   2 )
(=+   1,=-   2 )
(=+   1,=-   1 )
(=+   1,=+   0 )
(=+   1,=+   1 )
(=+   1,=+   2 )
(=+   2,=+   0 )
(=+   2,=+   1 )
(=+   2,=+   2 )
*
```

If students are asked to plot the points that are typed out, they are apt to make several discoveries. In the following graph, the points from the first run ($x\ (x+1) \geq y+6$) are plotted using o's and the points from the second run ($x\ (x+1) \leq y+6$) are plotted using x's.



Students can approximate the location of the points which satisfy $x(x+1) = y+6$ by sketching a curve between the x's and o's. This can be used as a different technique for introducing the graphing of functions. Graphs which represent equations like $y-1 = x + 3$ and $y = x^3$ can be introduced very early in algebra to help students develop an intuitive feeling for the behavior of these expressions before a more detailed study of them is made later in the course. Students can also discover approximate or even exact (as in this example) zeroes of functions by noting where the curve intersects the x-axis. The exact roots will appear in both of the typeouts if $\geq$ and $\leq$ are used.

E. Number Guessing (Supplementary)

The technique of binary search has several mathematical and programming applications. This idea can be introduced to students with a wide range of mathematical background by using the following demonstration program:

```
01.01 TYPE "I HAVE CHOSEN AN INTEGER 0 THROUGH 100.  TRY TO GUESS"!
01.02 TYPE "MY NUMBER IN AS FEW TRIES AS POSSIBLE."!!, %4.0
01.03 SET COUNT=0; SET N=FITR(FABS(FRAN()*100))
01.04 SET COUNT=COUNT+1; ASK "YOUR GUESS IS ",G
01.05 IF (N-G) 1.06,1.07; TYPE "  TOO LOW"!; GOTO 1.04
01.06 TYPE "  TOO HIGH"!; GOTO 1.04
01.07 TYPE "  CORRECT IN  ",COUNT," GUESSES."!
01.08 IF (3-COUNT) 1.09; TYPE "YOU WERE LUCKY."!!; QUIT
01.09 IF (7-COUNT) 1.10; TYPE "GOOD JOB."!!; QUIT
01.10 TYPE "BUT YOU SHOULDN'T NEED MORE THAN 7 GUESSES."!!
*
```

Three runs of this program are:

```
*DO 1
I HAVE CHOSEN AN INTEGER 0 THROUGH 100.  TRY TO GUESS
MY NUMBER IN AS FEW TRIES AS POSSIBLE.

YOUR GUESS IS :78    CORRECT IN  =+   1  GUESSES.
YOU WERE LUCKY.

*


*DO 1
I HAVE CHOSEN AN INTEGER 0 THROUGH 100.  TRY TO GUESS
MY NUMBER IN AS FEW TRIES AS POSSIBLE.

YOUR GUESS IS :50    TOO LOW
YOUR GUESS IS :75    TOO HIGH
YOUR GUESS IS :62    TOO HIGH
YOUR GUESS IS :56    CORRECT IN  =+   4 GUESSES.
GOOD JOB.

*


*DO 1
I HAVE CHOSEN AN INTEGER 0 THROUGH 100.  TRY TO GUESS
MY NUMBER IN AS FEW TRIES AS POSSIBLE.

YOUR GUESS IS :50    TOO HIGH
YOUR GUESS IS :25    TOO LOW
YOUR GUESS IS :37    TOO LOW
YOUR GUESS IS :44    TOO HIGH
YOUR GUESS IS :41    TOO HIGH
YOUR GUESS IS :42    TOO HIGH
YOUR GUESS IS :40    TOO HIGH
YOUR GUESS IS :39    TOO HIGH
YOUR GUESS IS :38    CORRECT IN  =+   9 GUESSES.
BUT YOU SHOULDN'T NEED MORE THAN 7 GUESSES.

*
```

Note that the program does not reveal the technique of binary search, but merely alludes to the fact that no more than 7 guesses are necessary. Classroom use of this program has resulted in almost all students "discovering" the idea of binary search after only a few runs of the program.

An excellent student assignment is the writing of a program which reverses the roles of the user and computer in the previous example. The computer should be programmed so that it will guess a number the user has determined. After each guess, the user inputs LOW, HIGH or CORRECT, and the program continues until the correct number is found. One such program is:

```
01.01 TYPE "YOU THINK OF AN INTEGER 1 THROUGH 100, AND I WILL GUESS"!
01.02 TYPE "YOUR NUMBER.  AFTER EACH GUESS TYPE HIGH, LOW, OR CORRECT."!
01.03 SET LOW=0; SET HIGH=101; SET C=0
01.04 SET GUESS=FITR((HIGH+LOW)/2)
01.05 TYPE %5.0,"I GUESS ",GUESS," THIS VALUE IS "; ASK ANS,!; SET C=C+1
01.06 IF (1357-ANS) !.07; SET LOW=GUESS; GOTO 1.04
01.07 IF (8978-ANS) 1.08; SET HIGH=GUESS; GOTO 1.04
01.08 TYPE !, C, "  GUESSES ISN'T SO BAD."!!
*
```

(Note that FOCAL automatically assigns LOW = 1357, HIGH = 8978, and CORRECT > 8978 when these responses are given to the ASK command.)

A run of this program is:

```
*DO 1
YOU THINK OF AN INTEGER 1 THROUGH 100, AND I WILL GUESS
YOUR NUMBER.  AFTER EACH GUESS TYPE HIGH, LOW, OR CORRECT.

I GUESS =+    50 THIS VALUE IS :LOW
I GUESS =+    75 THIS VALUE IS :LOW
I GUESS =+    88 THIS VALUE IS :HIGH
I GUESS =+    81 THIS VALUE IS :LOW
I GUESS =+    84 THIS VALUE IS :LOW
I GUESS =+    86 THIS VALUE IS :HIGH
I GUESS =+    85 THIS VALUE IS :CORRECT

=+     7  GUESSES ISN'T SO BAD.

*
```

This program will guess any number chosen within 7 guesses, providing that all user clues (LOW, HIGH, and CORRECT) are consistent. There are no checks built into the program which prevent the user from supplying contradictory information. Writing the few steps necessary to do this should be a part of the assignment for better students.

III. USING THE PROPERTIES OF =, +, AND * WHEN SOLVING EQUATIONS

A. Postulated Properties: Closure, Commutative, Associative, Distributive, Inverse, and Identity

The reflexive, symmetric, and transitive properties of equality and inequality are excluded from this section since little assistance is needed when introducing these postulates. One might, however, make use of problems like: "Using the integers 1, 2, 4, and 8, and the operations +, -, *, /, and ↑, express 64 in as many different ways as possible." Five, but not all possible, solutions are:

```
*TYPE 1*2*4*8,!
=+ 64
*TYPE 1*8↑(4-2),!
=+ 64
*TYPE 8↑((4-2)/1),!
=+ 64
*TYPE 8*2↑(4-1),!
=+ 64
*TYPE 4↑(8/2-1),!
=+ 64
*
```

By doing problems such as this, students gain experience with arithmetic operations, since most of their work is in the creation and mental evaluation of many different expressions before using the computer to confirm their results.

The commutative, associative, and distributive properties can be demonstrated in a variety of ways. One effective method is to introduce the three postulates for + and * in a conventional way, and then run the following demonstration program:

```
01.10 ASK ?A?,?B?,?C?,!
01.20 TYPE %8.04,! "COMMUTATIVE PROPERTY:"!!
01.30 TYPE ?A*B?,!,?B*A?,!!
01.40 TYPE ?A-B?,!,?B-A?,!!!
01.50 TYPE "ASSOCIATIVE PROPERTY:"!!
01.60 TYPE ?(A*B)*C?,!,?A*(B*C)?,!!
01.70 TYPE ?(A-B)-C?,!,?A-(B-C)?,!!!
01.80 TYPE "DISTRIBUTIVE PROPERTY:"!!
01.90 TYPE ?A*(B-C)?,!,?A*B-A*C?,!!
01.91 TYPE ?A-(B*C)?,!,?(A-B)*(A-C)?,!!
*


*DO 1
A:3 B:4 C:5
```

```
COMMUTATIVE PROPERTY:

A*B=+   12.0000
B*A=+   12.0000

A-B=-    1.0000
B-A=+    1.0000


ASSOCIATIVE PROPERTY:

(A*B)*C=+   60.0000
A*(B*C)=+   60.0000

(A-B)-C=-    6.0000
A-(B-C)=+    4.0000


DISTRIBUTIVE PROPERTY:

A*(B-C)=-    3.0000
A*B-A*C=-    3.0000

A-(B*C)=-   17.0000
(A-B)*(A-C)=+    2.0000

*
```

This program checks the validity of the three properties with the operations * and -. Many students seem to understand better the importance of these properties when they also see several cases for which the properties are not valid.

By following this demonstration with an assignment requiring that a similar program be written to check the validity of these properties with the operations of + and /, the student must do little more than write the necessary expressions. This writing, however, is enough to build the student's confidence in his understanding of these properties. Better students might be interested in exploring the changes in the validity of these properties if absolute value is used on all expressions or if a finite set with different operations (such as binary Boolean algebra) is used.

The introduction of additive and multiplicative inverses may be supplemented by assigning the apparently easy task of writing a program which will ASK for a number and then typeout the two inverses. Such a program is:

```
01.10 ASK ?A?,!
01.20 TYPE "ADDITIVE INVERSE IS:       ", -A,!
01.30 TYPE "MULTIPLICATIVE INVERSE IS: "; IF (A) 1.4, 1.5, 1.4
01.40 TYPE 1/A,!!; QUIT
01.50 TYPE "NOT DEFINED"!!
*


*DO 1
A:16
ADDITIVE INVERSE IS:       =-  16.0000
MULTIPLICATIVE INVERSE IS: =+   0.0625
```

```
*DO 1
A:0
ADDITIVE INVERSE IS:          =+    0.0000
MULTIPLICATIVE INVERSE IS: NOT DEFINED

*DO 1
A:5
ADDITIVE INVERSE IS:          =-    5.0000
MULTIPLICATIVE INVERSE IS:    =+    0.2000

*
```

Experience with this program has shown that many students will NOT write a correct solution the first time because they do not completely understand the two definitions. After the students have written this program, the teacher can quickly identify and then clarify the specific misunderstandings of individual students.

The idea of closure is an easy one, and use of the computer should not be necessary to reinforce a usual presentation. If, however, a little extra time is available, the concept of closure can be used to introduce other ideas to students new to programming. One such possibility is to assign the problem of writing a program which will determine whether the property of closure holds for each of the operations +, -, *, and / on some infinite set - say all positive multiples of 6. The initial reaction of many students has been that this is an easy assignment, but they soon discover that the computer CAN NOT PROVE even elementary concepts involving infinite sets. When this discovery is made, the teacher may alter the assignment to check only a finite set - say all positive multiples of 6 less than or equal to 150. The following program is a student written solution to this problem.

```
01.01 FOR N=6,6,150; DO 2
01.02 QUIT

02.01 TYPE !"NOW CHECKING ", %3.0, N, " FOR: "!
02.02 TYPE "ADDITION"!; FOR M=6,6,150; SET ANS=N+M; DO 3
02.03 TYPE "SUBTRACTION"!; FOR M=6,6,150; SET ANS=N-M; DO 3
02.04 TYPE "MULTIPLICATION"!; FOR M=6,6,150; SET ANS=N*M; DO 3
02.05 TYPE "DIVISION"!; FOR M=6,6,150; SET ANS=N/M; DO 3

03.01 IF (ANS-FITR(ANS)) 3.04, 3.02, 3.04
03.02 IF (150-ANS) 3.04; IF (ANS) 3.04, 3.04, 3.03
03.03 IF (ANS/6-FITR(ANS/6)) 3.04, 3.05, 3.04
03.04 TYPE " NOT CLOSED USING ", N, " AND ", M,!; SET M=151
03.05 CONTINUE
*
```

Note that this program assumes very little. Even the commutative law is not assumed, for both "A operation B" and "B operation A" are checked. Note that this student even verified (step 3.01) that the result of each operation is an integer. The program checks "N operation M" by fixing N, then substituting elements of the set for M until closure is verified or a single exception is found. This is done for all N's, regardless of the outcome of checking previous N's. The output appears as:

```
*GO

NOW CHECKING =+  6 FOR:
ADDITION
 NOT CLOSED USING =+  6 AND =+150
SUBTRACTION
 NOT CLOSED USING =+  6 AND =+  6
MULTIPLICATION
 NOT CLOSED USING =+  6 AND =+ 30
DIVISION
 NOT CLOSED USING =+  6 AND =+  6

NOW CHECKING =+ 12 FOR:
ADDITION
 NOT CLOSED USING =+ 12 AND =+144
SUBTRACTION
 NOT CLOSED USING =+ 12 AND =+ 12
MULTIPLICATION
 NOT CLOSED USING =+ 12 AND =+ 18
DIVISION
 NOT CLOSED USING =+ 12 AND =+  6
            .
            .
            .
NOW CHECKING =+150 FOR:
ADDITION
 NOT CLOSED USING =+150 AND =+  6
SUBTRACTION
 NOT CLOSED USING =+150 AND =+150
MULTIPLICATION
 NOT CLOSED USING =+150 AND =+  6
DIVISION
 NOT CLOSED USING =+150 AND =+  6
*
```

By checking all operations in this way, the student has demonstrated an understanding of the important programming concept of using multiple FOR statements.

B. Modular Arithmetic (Supplementary)

The introduction of modular arithmetic can be beneficial in several ways at this stage of student study. In addition to motivating independent study and reinforcing computational skills, modular arithmetic provides a simple yet useful mathematical system in which the postulated properties of the previous section appear in a different light. (A good reference, written for students, about

modular arithmetic is Secret Codes, Remainder Arithmetic, and Matrices by Lyman C. Peck; N.C.T.M. 1965.) The five programs shown in this section are all intended as topics for short assignments, as the students will benefit most by actually writing these programs themselves.

The first program allows the user to input a positive integer and a mod m, and then converts the given integer to mod m. The program is:

```
01.10 C CONVERSION OF A POSITIVE INTEGER TO AN INTEGER MOD M.
01.20 ASK "THE INTEGER ",I," IN MOD ",M," IS "
01.30 SET A=I-FITR(I/M)*M; TYPE %5.0,A,!!
*


*GO
THE INTEGER :1526  IN MOD :27  IS =+   14

*GO
THE INTEGER :123  IN MOD :123  IS =+    0

*GO
THE INTEGER :8  IN MOD :7  IS =+    1

*
```

By writing this program, students demonstrate an understanding of the meaning of a number in mod m as well as the important programming technique of retaining the remainder after a division.

The definition of congruent integers will be new to the majority of students. Their understanding of this definition can be tested by having them write a program similar to:

```
01.10 C IDENTIFICATION OF CONGRUENT POSITIVE INTEGERS IN MOD M.
01.20 ASK "THE INTEGERS ",A," AND ",B," IN MOD ",M," ARE "
01.30 SET N1=A-FITR(A/M)*M; SET N2=B-FITR(B/M)*M
01.40 IF (N1-N2) 1.5, 1.6, 1.5
01.50 TYPE "NOT CONGRUENT."!!; QUIT
01.60 TYPE "CONGRUENT."!!
*

*GO
THE INTEGERS :190  AND :394  IN MOD :17  ARE CONGRUENT.

*GO
THE INTEGERS :125  AND :104  IN MOD :11  ARE NOT CONGRUENT.

*
```

Another way of approaching the definition of congruent numbers is the following:

```
01.10 C PROGRAM TO LIST ALL MODS (2 THROUGH 100) IN WHICH TWO GIVEN
01.20 C POSITIVE INTEGERS ARE CONGRUENT.
01.30 ASK "THE INTEGERS ",A," AND ",B," ARE CONGRUENT IN MODS:"!
01.40 FOR M=2,100; DO 2
01.50 QUIT

02.10 SET N1=A-FITR(A/M)*M; SET N2=B-FITR(B/M)*M
02.20 IF (N1-N2) 2.4, 2.3, 2.4
02.30 TYPE %5.0,M,!
02.40 CONTINUE
*

*GO
THE INTEGERS :101  AND :37  ARE CONGRUENT IN MODS:
=+     2
=+     4
=+     8
=+    16
=+    32
=+    64
*

*GO
THE INTEGERS :1584  AND :1980  ARE CONGRUENT IN MODS:
=+     2
=+     3
=+     4
=+     6
=+     9
=+    11
=+    12
=+    18
=+    22
=+    33
=+    36
=+    44
=+    66
=+    99
*
```

A problem that appeals to many students is that of determining, WITHOUT using the computer, the two numbers less than 1000 that are congruent in the greatest number of mods. By working on this problem, students might "discover" several important properties of divisors and prime numbers.

Tables for the arithmetic operations in a given mod can be generated easily with a short program. The following program ASKs for a mod, then types out the multiplication tables for the given mod. (The entered mod must be less than or equal to 11 so that the table will fit on a page.)

```
01.10 ASK "MULTIPLICATION TABLES, MOD ",M
01.15 TYPE %2.0,!"     *  "; FOR C=0,M-1; TYPE C,"  "
01.16 TYPE !; FOR I=1,6*M+8; TYPE "*"
01.17 TYPE !"     *  "!
01.20 FOR R=0,M-1; DO 2
01.30 QUIT

02.10 TYPE R," *  "; FOR C=0,M-1; TYPE C*R-FITR((C*R)/M)*M,"  "
02.20 TYPE !,"     * ",!
*
```

```
*GO
MULTIPLICATION TABLES, MOD :5

     *  =+ 0  =+ 1  =+ 2  =+ 3  =+ 4
*************************************
     *
=+ 0 *  =+ 0  =+ 0  =+ 0  =+ 0  =+ 0
     *
=+ 1 *  =+ 0  =+ 1  =+ 2  =+ 3  =+ 4
     *
=+ 2 *  =+ 0  =+ 2  =+ 4  =+ 1  =+ 3
     *
=+ 3 *  =+ 0  =+ 3  =+ 1  =+ 4  =+ 2
     *
=+ 4 *  =+ 0  =+ 4  =+ 3  =+ 2  =+ 1
     *
*
```

By changing the typed statement in step 1.10, and replacing the operation * with + in step 2.10, the same program will generate addition tables. After making these changes, a run appears as:

```
*GO
ADDITION TABLES, MOD :5
     *  =+ 0  =+ 1  =+ 2  =+ 3  =+ 4
*************************************
     *
=+ 0 *  =+ 0  =+ 1  =+ 2  =+ 3  =+ 4
     *
=+ 1 *  =+ 1  =+ 2  =+ 3  =+ 4  =+ 0
     *
=+ 2 *  =+ 2  =+ 3  =+ 4  =+ 0  =+ 1
     *
=+ 3 *  =+ 3  =+ 4  =+ 0  =+ 1  =+ 2
     *
=+ 4 *  =+ 4  =+ 0  =+ 1  =+ 2  =+ 3
     *
*
```

The existence of reciprocals in modular arithmetic should be contrasted with that of reciprocals in base 10 arithmetic. Such a contrast helps students to appreciate the magnitude of the definition of reciprocals, for only in some mods do all elements except 0 have a reciprocal, in all mods the reciprocals are integers, and in most instances the value of the reciprocal of a given number is not immediately apparent.

A program for computing the reciprocals of integers in a given mod is:

```
01.10 C COMPUTATION OF RECIPROCALS OF INTEGERS IN MOD M.
01.20 ASK "MOD ",M,!"   NUMBER  RECIPROCAL"!
01.30 FOR N=0,M-1; FOR R=0,M-1; DO 2
01.40 QUIT

02.10 SET A=N*R-FITR((N*R)/M)*M; IF (A-1) 2.3, 2.2, 2.3
02.20 TYPE %5.0,N,"  ",R,!
02.30 CONTINUE
*
```

Two runs of this program are:

```
*GO
MOD :23
   NUMBER  RECIPROCAL
=+     1  =+     1
=+     2  =+    12
=+     3  =+     8
=+     4  =+     6
=+     5  =+    14
=+     6  =+     4
=+     7  =+    10
=+     8  =+     3
=+     9  =+    18
=+    10  =+     7
=+    11  =+    21
=+    12  =+     2
=+    13  =+    16
=+    14  =+     5
=+    15  =+    20
=+    16  =+    13
=+    17  =+    19
=+    18  =+     9
=+    19  =+    17
=+    20  =+    15
=+    21  =+    11
=+    22  =+    22
*
```

```
*GO
MOD :14
   NUMBER  RECIPROCAL
=+     1  =+     1
=+     3  =+     5
=+     5  =+     3
=+     9  =+    11
=+    11  =+     9
=+    13  =+    13
*
```

C. Developing and Using Basic Theorems for Solving Simple Equations

A presentation of the basic theorems used for solving simple equations (*i.e.*, if a = b, then a + c = b + c; if c = -(a + b), then c = -a + (-b)) provides an excellent opportunity to demonstrate that the computer will not do everything. The following program examines the "proposed" theorem: if a = b, then a*c = b*c by checking the validity of the conclusion with over 160,000 pairs of values for c and a = b.

```
01.10 FOR A=-100,.5,100;TYPE %4.01,A,!;SET B=A;FOR C=-100,.5,100;DO 2

02.10 IF (A*C-B*C) 2.2,2.3,2.2
02.20 TYPE "NOT TRUE FOR ",?A?,?  B?,?  C?,!; QUIT
02.30 CONTINUE
*
```

The output of this program appears as:

```
*DO 1
=-100.0
=- 99.5
=- 99.0
=- 98.5
   .
   .
   .
=-  0.5
=+  0.0
=+  0.5
=+  1.0
   .
   .
   .
=+ 98.5
=+ 99.0
=+ 99.5
=+100.0
*
```

The running of this program is best done overnight as the complete procedure requires a little over 14 hours on the PDP-8/S. Each time A and B change value, this value is typed simply to demonstrate that the program is running properly.

What, however, has been proved by running this program? Only that the theorem is valid for the 160,000+ pairs of values that were used. To further emphasize this point, a similar program which checks the "proposed" theorem $b-\frac{1-4a}{4a-1}=1+b$, using the same 160,000+ pairs of values is:

```
01.10 FOR A=-100,.5,100; TYPE %4.01,A,!; FOR B=-100,.5,100; DO 2

02.10 IF (B-(1-4*A)/(4*A-1)-(1+B)) 2.2,2.3,2.3
02.20 TYPE "NOT TRUE FOR ",?A?,?  B?,!; QUIT
02.30 CONTINUE
*
```

The output of this program is identical to that of the previous program - the "theorem" is valid for all of the values checked. In this case, however, the value A = 1/4, which was not checked, makes the theorem invalid as the term $\frac{1-4a}{4a-1}$ is undefined.

After students have proved WITHOUT further use of the computer, the basic theorems used for solving simple equations, and the focus changes to the solving of these equations, the computer can again be useful.

For students who need extra work on this topic, a drill program can be written which will supply them with an equation, accept their solution, and give an immediate diagnostic. The program might have 16 to 20 different equations written into it. A motivating factor might also be included which will prematurely stop presenting problems to a student if he responds with five or six consecutive correct answers. The program can also be written to maintain an error frequency record on each problem and/or student. This record can be useful to the teacher, both for giving individual help and for preparing lesson plans. This type of application is, however, quite boring for the average or better students.

Another approach to this same type of drill program is the following example, which supplies the student with equations of the form Ax ± B = C ± Dx (this form can easily be altered) and lets the student solve the problem a step at a time. Before examining the program, look at a sample run:

```
*DO 10.1; DO 1
                    must be ignored
PROBLEM IS:
=+  3.00*X MINUS =+  4.00  === =-  8.00 PLUS =+  7.00*X

OPERATION TO BE A,S,M, OR D? :A USING THE QUANTITY :4 TIMES (X OR 1):1
                                  student input

PROBLEM IS:
=+  3.00*X MINUS =+  0.00  === =-  4.00 PLUS =+  7.00*X

OPERATION TO BE A,S,M, OR D? :S USING THE QUANTITY :7 TIMES (X OR 1):X


PROBLEM IS:
=-  4.00*X MINUS =+  0.00  === =-  4.00 PLUS =+  0.00*X

OPERATION TO BE A,S,M, OR D? :D USING THE QUANTITY :-4 TIMES (X OR 1):1


PROBLEM IS:
=+  1.00*X MINUS =+  0.00  === =+  1.00 PLUS =+  0.00*X

YOU'VE SOLVED THE PROBLEM!
```

The command DO 10.1 determines which problem is to be done (10.1 means problem 1, 10.3 means problem 3, etc.). The command DO 1 then begins the student/computer interaction.

Following is a sample PART 10 containing the values for three different problems.

```
10.10 S A=3;S 01=3;S B=4;S C=-8;S 02=0;S D=7
10.20 S A=4;S 01=3;S B=-8;S C=3;S 02=0;S D=2
10.30 S A=5.9;S 01=0;S B=3;S C=-4.2;S 02=0;S D=20.3
*
```

The variables used represent the equation: A*X 01 B = C 02 D*X. When assigning the operations 01 and 02, MINUS is indicated with a 3 and PLUS with a 0.

Thus Problem 1 (10.1) is: 3x MINUS 4 = -8 PLUS 7

Problem 2 (10.2) is: 4x MINUS -8 = 3 PLUS 2

Problem 3 (10.3) is: 5.9x PLUS 3 = -4.2 PLUS 20.3

The teacher can change the number of problems, as well as alter the values in any or all problems, simply by modifying the steps in part 10. Part 1 of the program can remain unchanged - it will handle all equations of the given form which have a unique solution. Part 1 is:

```
01.10 T %5.02,!"PROBLEM IS:"!,A,"*X"
01.11 IF (01-3) 1.12;T " MINUS ",B;GOTO 1.13
01.12 T " PLUS ",B
01.13 T "  === ",C;IF (02-3) 1.14;T " MINUS ",D,"*X"!!; GOTO 1.20
01.14 T " PLUS ",D,"*X"!!
01.20 IF (A-1) 1.22,1.21,1.22
01.21 IF (FABS(B)+FABS(D)) 1.22,1.24,1.22
01.22 IF (D-1) 1.30,1.23,1.30
01.23 IF (FABS(A)+FABS(C)) 1.30,1.24,1.30
01.24 T "YOU'VE SOLVED THE PROBLEM!"!!!;QUIT
01.30 A "OPERATION TO BE A,S,M, OR D? ",O,"USING THE QUANTITY ",Q
01.31 A "TIMES (X OR 1)",T,!!;IF (T-1) 1.40,1.32,1.40
01.32 IF (O-13) 1.33;S A=A*Q;S B=B*Q;S C=C*Q;S D=D*Q;GOTO 1.10
01.33 IF (O-4) 1.34;S A=A/Q;S B=B/Q;S C=C/Q;S D=D/Q;GOTO 1.10
01.34 IF (O-3) 1.36;S C=C-Q;IF (01-3) 1.35;S B=B+Q;GOTO 1.10
01.35 S B=B-Q;GOTO 1.10
01.36 S C=C+Q;IF (01-3) 1.37;S B=B-Q;GOTO 1.10
01.37 S B=B+Q;GOTO 1.10
01.40 IF (O-4) 1.42;T !"THAT OPERATION IS LEGAL, BUT DOING IT WILL"
01.41 T " ONLY COMPLICATE THE PROBLEM."!"TRY ANOTHER."!!;GOTO 1.30
01.42 IF (O-3) 1.44;S A=A-Q;IF (02-3) 1.43;S D=D+Q;GOTO 1.10
01.43 S D=D-Q;GOTO 1.10
01.44 S A=A+Q;IF (02-3) 1.45;S D=D-Q;GOTO 1.10
01.45 S D=D+Q;GOTO 1.10
```

In this program, steps 1.10 through 1.14 typeout the problem, steps 1.20 through 1.24 check to see if the problem has been solved, steps 1.30 through 1.37 allow the next theorem to be applied in solving the equation and execute the theorems involving only a constant term, and steps 1.40 through 1.45 execute the theorems involving an x term. A second sample run of the program is:

```
*DO 10.2; DO 1
PROBLEM IS:
=+  4.00*X MINUS =-  8.00  === =+  3.00 PLUS =+  2.00*X

OPERATION TO BE A,S,M, OR D? :S USING THE QUANTITY :4 TIMES (X OR 1):X


PROBLEM IS:
=+  0.00*X MINUS =-  8.00  === =+  3.00 PLUS =-  2.00*X

OPERATION TO BE A,S,M, OR D? :S USING THE QUANTITY :3 TIMES (X OR 1):1


PROBLEM IS:
=+  0.00*X MINUS =-  5.00  === =+  0.00 PLUS =-  2.00*X

OPERATION TO BE A,S,M, OR D? :D USING THE QUANTITY :-2 TIMES (X OR 1):X


THAT OPERATION IS LEGAL, BUT DOING IT WILL ONLY COMPLICATE THE PROBLEM.
TRY ANOTHER.

OPERATION TO BE A,S,M, OR D? :D USING THE QUANTITY :-2 TIMES (X OR 1):1

PROBLEM IS:
=+  0.00*X MINUS =+  2.50  === =+  0.00 PLUS =+  1.00*X

YOU'VE SOLVED THE PROBLEM!
```

This program will allow the student to use, as often as desired, any of the basic theorems:

if $a = b$, then $a + c = b + c$
if $a = b$, then $a - c = b - c$
if $a = b$, then $a * c = b * c$
if $a = b$, and $c \neq 0$, then $a/c = b/c$

(The only exception is that division by an x term is not allowed.) By seeing the computer execute the specific theorem requested using the numbers he has indicated, students are less likely to incorrectly believe they are applying one theorem when they are actually using another. After the teacher develops programs such as this for several of the equation forms often found difficult by many students, the computer can become a valuable teaching assistant, and the typed copy produced by the students' work can provide an excellent source of specific information about the difficulties of individual students.

A challenging problem for better students is the task of writing a program which will accept the two sides of any first degree equation and then solve the equation. In accomplishing this task, the student must not only apply the basic theorems being studied, but also must create a solution algorithm that is unlike those he has previously used. This is indeed a good exercise in "problem solving." The equation to be solved should be entered as in the examples of Section II-A (SET L = left side of equation and SET R = right side of equation in part 2 of the program). Regardless of the number of terms entered on each side of the equation, the equation can be considered to be composed of: $L = Ax + B$ and $R = Cx + D$. One possible algorithm for obtaining the solution $x = \frac{D-B}{A-C}$ is

(1) let $x = 0$, then $LS(0) = B$ and $RS(0) = D$

(2) let $x = 1$, then $LS(1) = A+B$ and $RS(1) = C+D$

(3) now A-C can be determined using:

$$SX = RS(1) - RS(0) = C$$

and

$$CX = LS(1) - LS(0) - SX = A-C$$

Thus the solution is

$$x = \frac{RS(0) - LS(0)}{CX} \qquad (\text{if } CX \neq 0)$$

The algorithm described is student written, as is this program which follows it:

```
01.10 FOR X=0,1; DO 2; SET LS[X]=L; SET RS[X]=R
01.20 SET SX=RS[1]-RS[0]; SET CX=LS[1]-LS[0]-SX
01.30 TYPE "THE SOLUTION IS X", %6.03, (RS[0]-LS[0])/CX,!!; QUIT
*
```

Several runs of this program are:

```
*2.01 SET L=X; SET R=9                          solving x = 9
*GO
THE SOLUTION IS X=+  9.000

*2.01 SET L=-4; SET R=X                         solving -4=x
*GO
THE SOLUTION IS X=-  4.000

*2.01 SET L=X+5-3*X; SET R=4*X-7                solving x+5 - 3x=4x - 7
*GO
THE SOLUTION IS X=+  2.000

*2.01 SET L=3*X/5-7; SET R=(2/3)*X+5            solving 3x/5 - 7=2/3x+5
*GO
THE SOLUTION IS X=-180.000

*2.01 SET L=X+4; SET R=X+4                      solving x+4=x+4
*GO
THE SOLUTION IS X?02.80
*
```

Note that this program did not properly describe the solution in the last example. The student did not anticipate any possibility other than a unique solution. Although this may not be considered a serious oversight at this early stage of learning to solve equations, the discovery naturally leads the student to further inquiry. A program which will handle all types of first degree equations is:

```
01.10 FOR X=0,1; DO 2; SET LS[X]=L; SET RS[X]=R
01.20 SET SX=RS[1]-RS[0]; SET CX=LS[1]-LS[0]-SX
01.25 IF (CX) 1.3,1.4,1.3
01.30 TYPE "THE SOLUTION IS X", %6.03, (RS[0]-LS[0])/CX,!!; QUIT
01.40 IF (RS[0]-LS[0]) 1.6,1.5,1.6
01.50 TYPE "ALL VALUES OF X SATISFY THE EQUATION"!!; QUIT
01.60 TYPE "NO VALUES OF X SATISFY THE EQUATION"!; QUIT
```

Several runs of this program are:

```
*2.01 SET L=X+4; SET R=X+4
*GO
ALL VALUES OF X SATISFY THE EQUATION

*2.01 SET L=X+3*X+5; SET R=6+2*X-4*X+7
*GO
THE SOLUTION IS X=+  1.333

*2.01 SET L=5+X; SET R=X-7
*GO
NO VALUES OF X SATISFY THE EQUATION
*
```

The same program can be extended (either as an assignment or for demonstration) to type out the steps used in solving the equation. If a student can complete this program, he is certainly demonstrating sufficient evidence that he is capable of solving any first degree equation he might encounter. Such a student written program is:

```
01.01 TYPE %6.03;SET B=0;FOR X=0,1;DO 2;SET LS[X]=L;SET RS[X]=R
01.02 SET SX=RS[1]-RS[0];SET CX=LS[1]-LS[0]-SX;TYPE "TO BOTH SIDES:",!
01.03 IF (SX) 1.04,1.05;TYPE "SUBTRACT ", SX,"*X",!;SET B=B-1;GOTO 1.05
01.04 TYPE "ADD ",-SX,"*X"!; SET B=B-1
01.05 IF (LS[0]) 1.06,1.07;TYPE "SUBTRACT ",LS[0],!;SET B=B-1;GOTO 1.07
01.06 TYPE "ADD ",-LS[0],!;SET B=B-1
01.07 IF (CX-1) 1.08,1.11,1.08
01.08 IF (FABS(CX)-1) 1.09,1.09;TYPE "DIVIDE BY ",CX,!;GOTO 1.10
01.09 TYPE "MULTIPLY BY ",1/CX,!
01.10 SET B=B-1
01.11 IF (B) 1.12;TYPE "DO NOTHING",!
01.12 TYPE "THE SOLUTION IS X",(RS[0]-LS[0])/CX,!!
```

(Note - this program has not provided for the case of no solution or multiple solutions.)

Several runs of this program are:

```
*2.1 SET L=X; SET R=7                                    solving  x=7
*GO
TO BOTH SIDES:
DO NOTHING
THE SOLUTION IS X=+  7.000

*2.1 SET L=X+3*X+3-2*X+1; SET R=5-2*X+1                  solving
*GO                                                      x+3x+3-2x+1=5-2x+1
TO BOTH SIDES:
ADD =+ 2.000*X
SUBTRACT =+ 4.000
DIVIDE BY =+ 4.000
THE SOLUTION IS X=+  0.500

*2.1 SET L=4*X+X-10; SET R=-2*X+2                        solving
*GO                                                      4x+x-10=-2x+2
TO BOTH SIDES:
ADD =+ 2.000*X
ADD =+10.000
DIVIDE BY =+ 7.000
THE SOLUTION IS X=+  1.714

*2.1 SET L=(1/4)*X+(1/8); SET R=(5/4)*X-(7/8)            solving
*GO                                                      1/4x+1/8=5/4x-7/8
TO BOTH SIDES:
SUBTRACT =+ 1.250*X
SUBTRACT =+ 0.125
THE SOLUTION IS X=+  1.000

*
```

If this program is used for demonstration, the teacher should ensure that students realize that the steps given for the solution illustrate only one of many ways to solve the equation.

D. Very Elementary Statistics - Sum, Mean, and Standard Deviation (Supplementary)

The two programs shown in this section are suggested as topics for student assignments. Although not related to the other topics in this chapter, the mathematics involved can be handled easily by students just beginning algebra, and the writing of the programs might stimulate some independent study.

The first example is a program that simulates the response of an adding machine. The program repeatedly ASKs for a number until the user responds with the word TOTAL. When this occurs, the sum of the input numbers, the total number of input numbers, and the average of these numbers is typed out. The program and a sample run are:

```
01.10 TYPE "ADDING MACHINE SIMULATOR"!!; ERASE
01.20 ASK NUMBER,!; IF (NUMBER-55422) 1.3,1.4,1.3
01.30 SET SUM=SUM+NUMBER; SET ENTRIES=ENTRIES+1; GOTO 1.2
01.40 SET AVERAGE=SUM/ENTRIES; TYPE %6.02,!!,?SUM    ?,!,ENTRIES?,!
01.50 TYPE ?AVERAGE?,!!
*


*GO
ADDING MACHINE SIMULATOR

:3
:9
:-6
:0
:4
:-2
:5.5
:-2.25
:11.75
:4
:TOTAL


SUM     =+  27.00
ENTRIES=+  10.00
AVERAGE=+   2.70

*
```

Classroom experience with this assignment has shown that often several students expand upon the idea by writing programs to simulate more elaborate calculators. Many of the programs allow the user to input ADD, SUB, MUL, DIV, TOTAL, and SUBTOTAL, as well as the arguments for these operations. Such a program may not give immediate reinforcement to the mathematics being studied in class, but in writing the program, students develop programming techniques that are useful throughout the year. If limited class time does not permit this program to be a class assignment, it might be offered as an " extra credit " problem or as a small-group project which will later be explained to the entire class by the students in the group.

A second problem is that of computing the mean and standard deviation of N numbers. This topic often is never presented in high school, yet the mathematics involved is suitable for inclusion in Algebra I. Although the subject can be avoided, is there really a good reason for not explaining the meaning of the notation $\Sigma(x_i - m)^2$? A student written program to accomplish this task is:

```
01.10 TYPE "COMPUTATION OF MEAN AND STANDARD DEVIATION "; ERASE
01.20 ASK "OF N TERMS."!"NUMBER OF TERMS IS ", N,!
01.30 TYPE "THE TERMS ARE";FOR I=1,N;ASK "  ",A[I];SET TOTAL=TOTAL+A[I]
01.40 SET MEAN=TOTAL/N; FOR I=1,N; SET SUM=SUM+(A[I]-MEAN)+2
01.50 SET ST=FSQT(SUM/N); TYPE %7.03,!!,?MEAN?,!,?STANDARD-DEVIATION?,!!
*
```

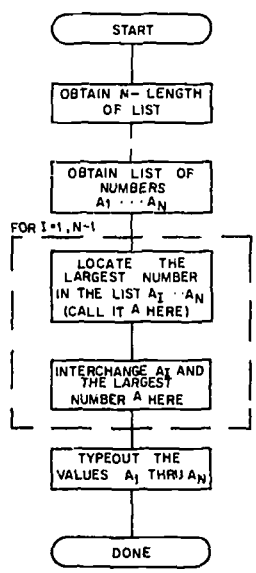
```
*GO
COMPUTATION OF MEAN AND STANDARD DEVIATION OF N TERMS.
NUMBER OF TERMS IS :9
THE TERMS ARE  :1   :3   :2   :5   :3   :3   :4   :2

MEAN=+   3.000
STANDARD-DEVIATION=+   1.155

*
```

This assignment has been successfully presented to a class by simply stating the problem and suggesting that several books in the library might be useful in finding the definition of standard deviation. If this assignment did no more than demonstrate that there are mathematics books other then textbooks in a library, the time required would be well spent.

E. Ordering (Supplementary)

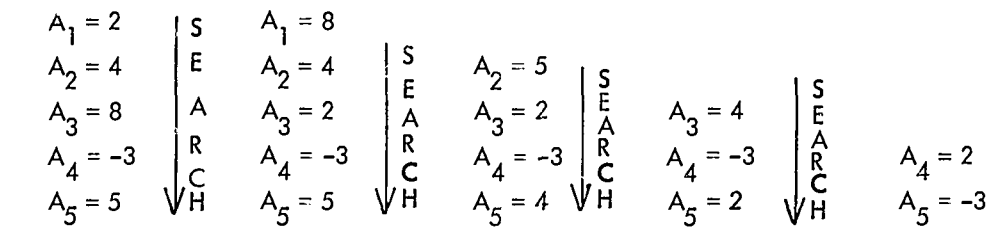
The concept of ordering a list of N numbers is an intuitive one for most students, but the writing of a program to accomplish this task is not quite as obvious. With average classes, this problem should be prefaced with a program to find the maximum and/or minimum values in a list of N numbers.

There are many algorithms for ordering numbers, and guiding the students' "discovery" of some of these algorithms provides a significant contribution to their abilities in general problem solving. Consider first the problem of arranging N numbers in descending order when nothing is known about the numbers. A valid algorithm for doing this is shown in the following flowchart.



A step-by-step example of this technique is:

Let N = 5, then



original list of numbers The resulting list A_1 through A_5 is arranged in descending order.

A program which uses this algorithm is:

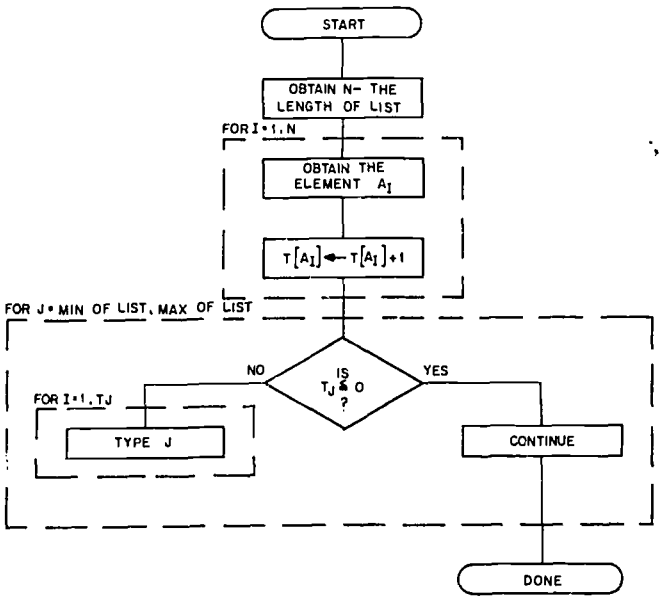
```
01.10 TYPE "PROGRAM TO ARRANGE N NUMBERS IN DESCENDING ORDER."!; ERASE
01.20 ASK !"HOW MANY NUMBERS? ",N,!; FOR I=1,N; ASK A[I],"  "
01.30 FOR J=1,N-1; SET HERE=J; SET LARGE=A[J]; DO 2
01.40 TYPE !!"THE NUMBERS IN DESCENDING ORDER ARE:"!, %6.02
01.50 FOR K=1,N; TYPE A[K],!
01.60 QUIT

02.10 FOR I=J+1,N; IF (A[I]-LARGE) 2.2,2.2; SET HERE=I; SET LARGE=A[I]
02.20 CONTINUE
02.30 SET A[HERE]=A[J]; SET A[J]=LARGE
*
```

A run of this program is:

```
*GO
PROGRAM TO ARRANGE N NUMBERS IN DESCENDING ORDER.
HOW MANY NUMBERS? :8
:3.4    :-17.5    :0    :12.34    :-5.1    :0    :18    :-3.4

THE NUMBERS IN DESCENDING ORDER ARE:
=+  18.00
=+  12.34
=+   3.40
=+   0.00
=+   0.00
=-   3.40
=-   5.10
=-  17.50
*
```

Note that to use this program to arrange numbers in ascending order, one need only change the argument in the IF command of step 2.10 from A [I] - LARGE to LARGE-A[I].

Student exploration of various methods for ordering numbers will invariably lead to discussions of the speed and efficiency of each technique. When this occurs, the following algorithm for ordering N INTEGERS when the minimum and maximum values are known will have particular appeal, since each number is examined only once rather than several times as in most methods. A flowchart for this algorithm is:



The following program is written according to this flowchart:

```
01.10 TYPE %3.0,"PROGRAM TO ARRANGE N INTEGERS (+- 25) IN"; ERASE
01.20 ASK " ASCENDING ORDER"!!"HOW MANY NUMBERS? ",N,!
01.30 TYPE "THESE ARE"; FOR I=1,N; ASK "  ",A[I]; SET T[A[I]]=T[A[I]]+1
01.40 FOR J=-25,25; IF (T[J]) 1.5,1.5; FOR I=1,T[J]; TYPE !,J
01.50 CONTINUE
01.60 TYPE !!
*
```

A run of this program appears as:

```
*GO
PROGRAM TO ARRANGE N INTEGERS (+- 25) IN ASCENDING ORDER

HOW MANY NUMBER? :30
THESE ARE  :-16  :21  :0  :23  :25  :-17  :-2  :6  :10  :1
  :6  :0    :15  :23  :5  :-9  :12  :-23  :-4  :0  :17  :6
  :0  :1    :21  :19  :-15  :-24  :-16  :0
=- 24
=- 23
=- 17
=- 16
=- 16
=- 15
=-  9
=-  4
=-  2
=+  0
=+  0
=+  0
=+  0
=+  0
=+  1
=+  1
=+  5
=+  6
=+  6
=+  6
=+ 10
=+ 12
=+ 15
=+ 17
=+ 19
=+ 21
=+ 21
=+ 23
=+ 23
=+ 25

*
```

Note: Although the method used is completely general, limited memory availability may necessitate restricting integer ordering to the interval ±25 as in the example.

IV. NEGATIVE NUMBERS

A. Plotting Equalities and Inequalities on the Number Line

The program shown on page 10 will plot expressions involving negative numbers on the number line, as demonstrated in some of the sample runs. Use of this program might provide additional interest when negative numbers are "introduced" in Algebra I.

Students may be asked to determine which of the programs they have already written and/or run (union and intersection of sets, ordering, exponents, etc.) will continue to work if negative numbers are used. The value of such an assignment is, of course, directly proportional to the nature of the programs previously written by the students. Use of such an assignment will also help the teacher to identify problems that students might yet be having with earlier topics. For example, if a student has written a program to type the Nth power of an input number and this program does not work for negative numbers, the student probably does not completely understand the definition of exponents.

B. Addition and Subtraction on the Number Line (Using Vector Representation)

A demonstration program to assist a presentation of vector addition is the following: (The program can be easily altered to perform vector subtraction).

```
01.01 C PROGRAM TO ADD TWO NON-ZERO VECTORS WHOSE INDIVIDUAL
01.02 C AND COMBINED MAGNITUDES HAVE INTEGRAL VALUES <= 10.
01.03 ASK "VECTORS TO BE ADDED ARE ",A," AND ",B,!!
01.04 SET S=0; SET E=A; IF (-A) 1.05; SET S=A; SET E=0
01.05 SET V=A; DO 2
01.06 SET S=A; SET E=A+B; IF (-B) 1.07; SET S=A+B; SET E=A
01.07 SET V=B; DO 2
01.08 TYPE "  "; FOR I=-10,10; TYPE ".  "
01.09 TYPE !"-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6"
01.10 TYPE "  7  8  9 10"!
01.11 SET S=0; SET E=A+B; IF (-(A+B)) 1.12, 1.13; SET S=A+B; SET E=0
01.12 SET V=A+B; DO 2
01.13 TYPE "RESULT IS ",%3.0, A+B,!!; QUIT

02.10 TYPE "  "; IF (S+10) 2.2,2.2; FOR I=-10,S-1; TYPE "  "
02.20 IF (V) 2.3; TYPE "---"; GOTO 2.4
02.30 TYPE "<--"
02.40 IF (E-S-1) 2.5,2.5; FOR I=S+1,E-1; TYPE "---"
02.50 IF (V) 2.6; TYPE ">"!; GOTO 2.7
02.60 TYPE "-"!
02.70 CONTINUE
*
```

Several runs of this program are:

```
                                        Input
                                  ↓         ↓
*GO
VECTORS TO BE ADDED ARE :2  AND :-5

                                  ------>
                          <---------------
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
                          <---------
RESULT IS =-  3

*GO
VECTORS TO BE ADDED ARE :2  AND :5

                                  ------>
                                        ---------------->
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
                                  ---------------------->
RESULT IS =+  7

*GO
VECTORS TO BE ADDED ARE :-2  AND :-5

                            <------
                  <----------------
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
                  <----------------------
RESULT IS =-  7

*GO
VECTORS TO BE ADDED ARE :-2  AND :5

                            <------
                            ---------------->
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
                                  ---------->
RESULT IS =+  3

*
```

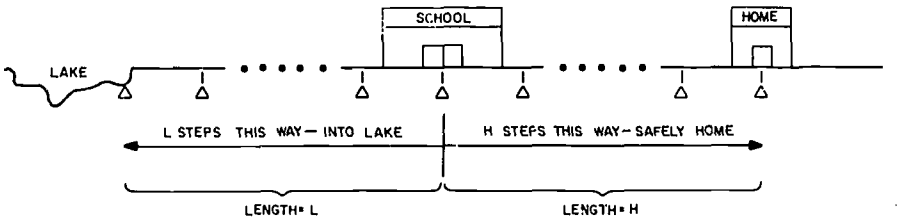
Note that the program types out the vector sum in both vector and numerical form. The description states that only non-zero vectors whose individual and combined magnitudes have integral values ≤ 10 may be used. There are, however, no provisions in the program for rejecting illegal values. Actually, the first vector and the combined magnitudes must have integral values ≤ 10, but the second vector is not restricted to these values. For example:

```
*GO
VECTORS TO BE ADDED ARE :-4  AND :14

                        <------------
                        ------------------------------------------>
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
                                 ------------------------------>

RESULT IS =+  10

*
```

This program can be modified to provide remedial help for students having trouble with this topic. Such a modification would operate by using the random number function to generate the two vectors rather than ASKing the user to input them. The two vectors would then be typed using the present output form, but the user would be ASKed for the vector sum rather than just seeing it typed. If he responded correctly, an appropriate message would be typed. If he responded incorrectly, the correct vector sum would be typed using the format of the sample program.

An assignment that helps reinforce the topic of vector addition and subtraction, and that has a great deal of appeal for many students, is the writing of a program that simulates the following problem:

> Consider the pseudo-random walker who, upon leaving school in a somewhat confused state, must attempt to make his way home. When he leaves the school he faces home and is confronted with the following situation:



> At every point on his journey, the walker is as likely to step backward as he is to step forward. Your program should ASK for L, H, and T (the total number of trips from school to a final destination), then simulate each of the walker's journeys. The output should contain the total number of steps taken and the ultimate destination for each trip.

A student written solution to this problem is:

```
01.10 ASK "RANDOM WALKER HOME IN ",H," STEPS AND INTO LAKE IN ",L
01.20 ASK " STEPS."!"HOW MANY TRIPS SHALL HE MAKE? ",T,!!
01.30 FOR I=1,T; SET COUNT=0; SET PLACE=0; DO 2
01.40 QUIT

02.10 TYPE "TRIP NUMBER ",%4.0,I,!"STEPS ARE: "
02.20 SET STEP=FRAN(); SET COUNT=COUNT+1
02.3  IF (STEP) 2.4,2.4; SET PLACE=PLACE+1; TYPE "H"; GOTO 2.5
02.40 SET PLACE=PLACE-1; TYPE "L"
02.50 IF (H-PLACE) 2.6, 2.8, 2.6
02.60 IF (L+PLACE) 2.2, 2.7, 2.2
02.70 TYPE !"DUNKED IN ",COUNT, " STEPS."!!; RETURN
02.80 TYPE !"SAFELY HOME IN ",COUNT, " STEPS."!!; RETURN
*
```

A run of this program appears as:

```
*GO
RANDOM WALKER HOME IN :5  STEPS AND INTO LAKE IN :5  STEPS.
HOW MANY TRIPS SHALL HE MAKE? :4

TRIP NUMBER =+   1
STEPS ARE: LHLHHHHLHHLLLLLLHLLLHHHHHHLHHH
SAFELY HOME IN =+  29 STEPS.

TRIP NUMBER =+   2
STEPS ARE: HHLLHHLLHHLLHLHHHHLHH
SAFELY HOME IN =+  21 STEPS.

TRIP NUMBER =+   3
STEPS ARE: LHLHLHHLLHHLLLLHLLHHHHLHLLLHHLLLHLL
DUNKED IN =+  35 STEPS.

TRIP NUMBER =+   4
STEPS ARE:  HLHHLLHHHHH
SAFELY HOME IN =+   11 STEPS.

*
```

Note that this program types out each step as well as all the information requested in the problem.

Students working on this problem are often discovered asking questions about probability. They are quite naturally interested in the "average" number of steps expected after several trips, the chances of exactly L or H steps in a single trip, etc. These questions become even more interesting if the numbers L and H are not alike. For example:

```
*GO
RANDOM WALKER HOME IN :6  STEPS AND INTO LAKE IN :4  STEPS.
HOW MANY TRIPS SHALL HE MAKE? :3

TRIP NUMBER =+   1
STEPS ARE: LHLHLHHHLHHHHLLLHHLLHHLLLHHHHLHLLLLHLHHLLHLHLLLHLHHHHLHHH
SAFELY HOME IN =+  56 STEPS.

TRIP NUMBER =+   2
STEPS ARE: LHLLHLHHLLLL
DUNKED IN =+   12 STEPS.

TRIP NUMBER =+   3
STEPS ARE: LHLHHLLLHHLLHLLHHLLL
DUNKED IN =+  20 STEPS.

*
```

Trip number 1 of this example suggests a problem which this student's program did not anticipate - if more than 60 steps are taken, the 61st and all remaining steps will be typed in the same position. This problem can be eliminated by adding the step: 2.25 IF (COUNT-61) 2.3; TYPE !

This program is also an excellent vehicle for introducing the important concept of simulation. Students should be encouraged to determine empirical probabilities after simulation of 100 or more trips of the random walker. This will help in demonstrating the real value of using the computer to simulate real, more complex problems.

NOTE

The use of the random function in this example - <= 0 means step backward and >0 means step forward - will produce satisfactory results if 100 or more runs are used.

C. Absolute Value, Operations with Negative Numbers, and More Theorems for Solving Equations

Classroom use has shown that the very short assignment of writing a program which will ASK for a number and then type out the absolute value of that number will benefit most students being introduced to absolute value. Such a program, with two sample runs, is:

```
01.10 ASK "THE ABSOLUTE VALUE OF ",X," IS "
01.20 IF (X) 1.3; TYPE %6.02,X,!!; QUIT
01.30 TYPE %6.02,-X,!!
*
```

```
*GO
THE ABSOLUTE VALUE OF :5.23  IS =+   5.23

*GO
THE ABSOLUTE VALUE OF :-7.1  IS =+   7.10

*
```

Although the definition of absolute value seems quite easy, not all students will correctly write this program. An incorrect program probably reflects a misunderstanding of the real meaning of a variable. Thus, students writing invalid programs are most likely in need of individual help.

Programs to assist in the presentation of techniques for addition and subtraction using negative numbers were shown in the previous section of this chapter. To assist in the presentation of techniques for multiplication and division using negative numbers, one might use the following demonstration program:

```
01.10 ASK ?A?,?  B?,!
01.20 TYPE %6.02,?A*B     ?,!,?(-A)*B    ?,!,?A*(-B)    ?,!,?(-A)*(-B)?,!
*

*GO
A:3   B:4
A*B       =+   12.00
(-A)*B    =-   12.00
A*(-B)    =-   12.00
(-A)*(-B)=+    12.00
*


*GO
A:-2   B:6
A*B       =-   12.00
(-A)*B    =+   12.00
A*(-B)    =+   12.00
(-A)*(-B)=-    12.00
*
```

The results of this simple program will not be obvious to all beginning students. An understanding that -A*B is not always a negative number requires a confident grasp of the real meaning of a variable. Thus, this program provides another means of discovering any misunderstandings on the part of the individual student. An assignment that might provide a student with some surprise is that of writing a program which ASKs for two numbers, and then ASKs for the four products which were typed in the previous example. If the user responds with a correct product, the next product is ASKed. If the user responds with an incorrect product, this product is ASKed again. An example of such a program and two sample runs is:

```
01.10 ASK ?A?,?  B?,!
01.20 ASK !"A*B=       ",ANS; IF (ANS-(A*B)) 1.2,1.3,1.2
01.30 ASK !"(-A)*B=    ",ANS; IF (ANS-((-A)*B)) 1.3,1.4,1.3
01.40 ASK !"A*(-B)=    ",ANS; IF (ANS-(A*(-B))) 1.4,1.5,1.4
01.50 ASK !"(-A)*(-B)=",ANS; IF (ANS-((-A)*(-B))) 1.5,1.6,1.5
01.60 TYPE !!!; QUIT
*


*GO
A:3    B:-4

A*B=        :-12
(-A)*B=     :12
A*(-B)=     :12
(-A)*(-B)=-12


*GO
A:-2    B:-5

A*B=        :-10
A*B=        :10
(-A)*B=     :-10
A*(-B)=     :10
A*(-B)=     :-10
(-A)*(-B)=:10

*
```

The surprise that may occur is that a student, using his own program, will discover that this program is correcting some of his own mathematics. This technique provides an effective means of provoking individual thought by students who have some uneasiness concerning the use of variables.

When proving and using elementary theorems involving negative numbers and absolute value to solve simple equations, many of the programs shown in Section III can be utilized with no change. For example, consider the programs for: verifying the commutative, associative, and distributive properties (page 37); finding additive and multiplicative inverse (page 38); verifying the property of closure (page 39); checking the validity of proposed theormes (page 45); the step-by-step solution of simple equations (page 46); and solving simple equations (page 49).

These programs can all be used or assigned in exactly the manner already presented. However, rather than use these programs a second time, students will gain more if the assignments appear somewhat different. This can probably be done easily, as class time usually does not permit the use of all the different programs already suggested by the time the topics in this section are reached. The teacher should not overlook the possibility of now assigning students the task of writing a program that was earlier used for demonstration. The computer is most useful as a teaching aid when students write their own programs, and thus, repeating a particular topic in this manner is quite beneficial.

When using negative numbers and absolute value, one begins to encounter several theorems which are true only with restricted values for the variables. For example, consider:

(1) $a+b = -(|a|+|b|)$; valid if $a<0$ and $b<0$

(2) $a+b = |a|-|b|$; valid if $a>0$, $b<0$, and $|a| \geq |b|$

(3) $a+b = |a|+|b|$; valid if $a>0$ and $b>0$

Such theorems can be simply stated for the class as propositions, the assignment being to determine the special conditions under which they are valid. If this is done, be sure to include at least one proposition such as $|a| + |b| < |a + b|$ which is never true. If students have not previously had an opportunity for attacking a problem involving the computer in their own way (i.e., without a specific program suggested), this is a good time to start. Thus the complete assignment is: "Determine the conditions under which a given proposition is valid, using the computer to assist you in any way you choose." Classroom experience with this type of assignment has shown that the number of students who have previously written programs for the computer is inversely proportional to the number of students who will seek computer assistance in completing the assignment. There are several possible positive reasons for this relationship. First, the students may have indirectly learned how to approach this problem as a result of previous programming experience. Second, the students might recognize the few cases that need to be tried - previous programming experience does assist this recognition - and not bother with the computer for the simple arithmetic required. Finally, better students will have already done problems similar to this while working on earlier programs.

Slower students might enjoy the problem of writing expressions such as A-B-C using as many different arrangements of parentheses as possible without using more than one pair for the same purpose. (This assignment is not very challenging for better students.) Several possible solutions are:

```
*SET A=1; SET B=2; SET C=4
*TYPE %4.0, A-B-C,!
=-   5
*TYPE A+(-B)+(-C),!
=-   5
*TYPE A-(B+C),!
=-   5
*TYPE ((A+(-B))+(-C)),!
=-   5
*
```

A challenging assignment for average or better students is the writing of a program which will reduce a given expression to the form P=X+Q. The technique required is similar to that shown in the program on page 49 which would solve a first-degree equation. The program appears as:

```
01.10 SET X=0; DO 2; SET Q=EXP
01.20 SET X=1; DO 2; SET P=EXP-Q
01.30 TYPE "REDUCED EXPRESSION IS IN THE FORM  P*X + Q, WHERE:"!,%6.02
01.40 TYPE ?P ?,!,?Q ?,!!
*
```

The output from two runs of this program is:

```
*2.1 SET EXP=3+4*X-6+5*X
*GO
REDUCED EXPRESSION IS IN THE FORM  P*X + Q, WHERE:
P =+    9.00
Q =-    3.00

*2.1 SET EXP=7*X+12-3*X+4.5-3.25+4.5*X
*GO
REDUCED EXPRESSION IS IN THE FORM  P*X + Q, WHERE:
P =+    8.50
Q =+   13.25

*
```

An effective approach to this problem is to make the assignment with no suggestion as to how to solve the problem. This requires the student to find a general approach to the problem that is unlike the many problems he has discussed in class. An understanding of the method of solution also helps develop an intuitive feeling for the idea of a function. This same program could be modified easily for use in giving extra practice for those students having trouble evaluating expressions. Such a modification might ASK the student for his answer for a given value of x and then give an immediate appraisal of this answer.

D. Addition and Subtraction of Binary Numbers Using Complement Arithmetic (Supplementary)

Although treated in a different manner, the topic of negative numbers is not new to students of Algebra I. A short lesson on addition and subtraction using complement arithmetic is, however, new material for almost all students and has proven to be an excellent motivational device. Since many computers actually execute these operations using complement arithmetic, the example of binary numbers is quite natural.

The following program is suitable for use as a demonstration, but the main reason for including it in this text is to provide an example of several programming techniques. In its entirety it would be too difficult to assign to students in Algebra I, but isolated portions of it provide excellent exercises. The program is:

```
01.01 E
01.10 A "THE BINARY NUMBERS ARE ",A," AND ",B,!
01.11 A "THE OPERATION IS PLUS OR MINUS?  ",OP,!!
01.20 F I=1,6;D 2
01.21 T %2.0,"THE PROBLEM IS:"!!"           ";F I=1,6;T A[7-I]
01.22 IF (2000-OP) 1.23;T !"PLUS       ";GOTO 1.24
01.23 T !"MINUS
01.24 F I=1,6;T B[7-I]
01.30 IF (2000-OP) 1.31,1.4,1.4
01.31 F I=1,6;S B[7-I]=FABS(B[7-I]-1)
01.32 T !!"USING THE COMPLEMENT OF B, THIS PROBLEM BECOMES:"
01.33 T !!"           ";F I=1,6;T A[7-I]
01.34 T !"PLUS       ";F I=1,6;T B[7-I]
01.40 F I=1,7;D 3
01.45 T !;F I=1,35;T "-"
01.46 T !"EQUALS ";F I=1,7;T S[8-I]
01.50 IF (2000-OP) 1.51;T !!!;Q
01.51 F I=1,6;S A[I]=0;S B[I]=S[I];S S[I]=0;S C[I]=0
01.52 S A[1]=S[7];F I=1,6;D 3
01.53 T !!"USING THE END AROUND CARRY, THE ANSWER"!!
01.54 T "EQUALS     ";F I=1,6;T S[7-I]
01.55 T !!!;Q

02.10 S A[7-I]=FITR(A/10↑(6-I));S A=A-A[7-I]*10↑(6-I)
02.20 S B[7-I]=FITR(B/10↑(6-I));S B=B-B[7-I]*10↑(6-I)

03.10 S S[I]=A[I]+B[I]+C[I]
03.20 IF (S[I]-2) 3.3,3.25,3.3
03.25 S S[I]=0;S C[I+1]=1
03.30 IF (S[I]-3) 3.4,3.35,3.4
03.35 S S[I]=1;S C[I+1]=1
03.40 C
*
```

NOTE

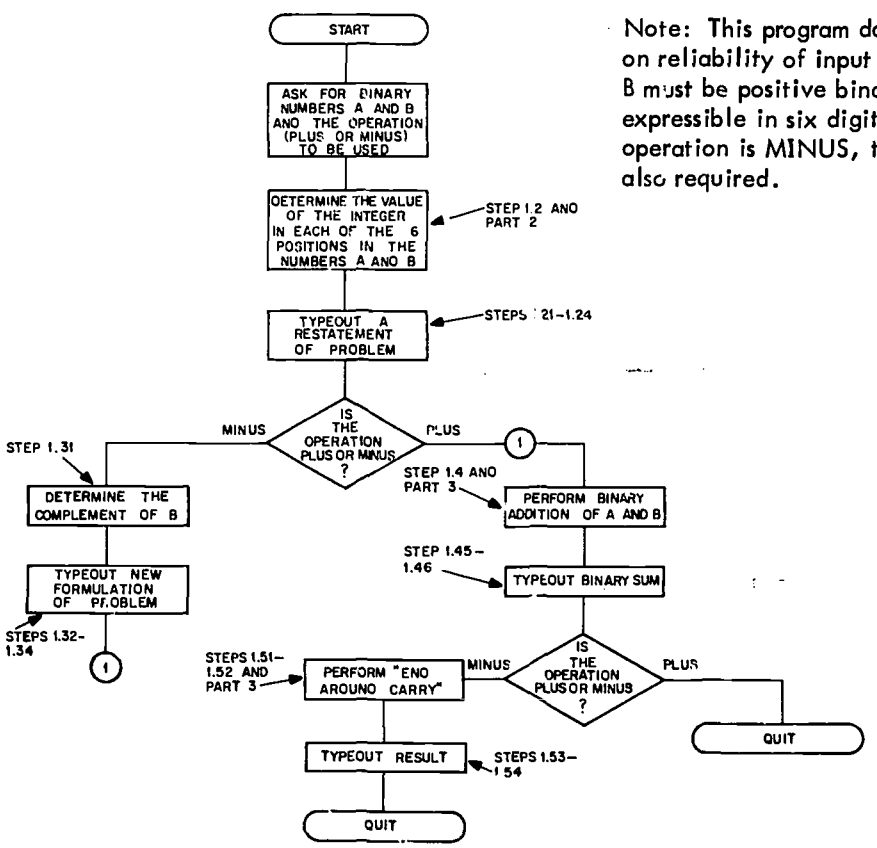
Abbreviated commands are necessary when using the 4K version of FOCAL.

The program will add and subtract two six-digit binary numbers. Complement arithmetic is used for subtraction, and the major steps required to attain the solution are typed out. The output from two runs of this program is:

```
*GO
THE BINARY NUMBERS ARE :101101  AND :110101
THE OPERATION IS PLUS OR MINUS?  :PLUS

THE PROBLEM IS:

           =+ 1=+ 0=+ 1=+ 1=+ 0=+ 1
PLUS       =+ 1=+ 1=+ 0=+ 1=+ 0=+ 1
-----------------------------------
EQUALS =+ 1=+ 1=+ 0=+ 0=+ 0=+ 1=+ 0
```

```
*GO
THE BINARY NUMBERS ARE :111010  AND :100111
THE OPERATION IS PLUS OR MINUS?  :MINUS

THE PROBLEM IS:
           =+ 1=+ 1=+ 1=+ 0=+ 1=+ 0
MINUS      =+ 1=+ 0=+ 0=+ 1=+ 1=+ 1

USING THE COMPLEMENT OF B, THIS PROBLEM BECOMES:

           =+ 1=+ 1=+ 1=+ 0=+ 1=+ 0
PLUS       =+ 0=+ 1=+ 1=+ 0=+ 0=+ 0
-----------------------------------
EQUALS =+ 1=+ 0=+ 1=+ 0=+ 0=+ 1=+ 0

USING THE END AROUND CARRY, THE ANSWER

EQUALS     =+ 0=+ 1=+ 0=+ 0=+ 1=+ 1


*
```

A very basic flowchart for this program is:



Note: This program does not check on reliability of input data. A and B must be positive binary numbers expressible in six digits. If the operation is MINUS, then A>B is also required.

Excluding the various typeout formats, there are four distinct sections of this program that might be used as separate assignments. These are:

1. Determining the integral value of each position in the numbers A and B. When FOCAL ASKs for A, the six-digit number which is entered is really accepted as a base 10 number. Thus, to perform binary operations on it, each digit must be isolated. (This problem occurs several times in many types of problems.) This program uses subscripts as:

   $A = \overline{A6}\ \ \overline{A5}\ \ \overline{A4}\ \ \overline{A3}\ \ \overline{A2}\ \ \overline{A1}$,

   and similarly for B. The necessity of facing this problem can be avoided by altering the program so that each digit of A and B is ASKed for separately.

2. Performing the binary addition of two numbers. The students must here realize that FOCAL is computing in base 10, thus 1 + 1 = 2 will have to be converted to 1 + 1 = 0 with a "carry" of 1. Similarly 1 + 1 + 1 (carried) = 3 will have to be converted to 1 + 1 + 1 (carried) = 1 with a "carry" of 1.

3. Performing the "end-around carry" this is mainly a problem of determining how to set up the initial conditions. This program resets the numbers A and B to represent the "end-around carry", then uses the same part of the program that was written for binary addition of A and B.

4. Determining the complement of a binary number. Students should be encouraged to use the absolute value function in their solution rather than the obvious sequence

   IF (A[I]) α, α; SET A [I] = 0; GOTO β
   α SET A [I] = 1
   β ...

   Although this sequence is perfectly valid, it becomes quite unwieldly if the 9's complement is being sought, while the method shown in the program (step 1.31) works equally well if a 9 replaces the 1 in the argument of the absolute value function.

Once again, this program in its entirety is a very difficult, time consuming assignment for students in the early stages of Algebra I, but any one of the four suggested subprograms is quite reasonable and beneficial. Interesting results can often be attained by assigning each of these subprograms to different groups within the class, then combining the better programs from various groups into one larger program similar to that given in the example.

The topic, using complements to perform subtraction, is also of interest to more advanced students. A complete proof of why the method works is a worthwhile task for students of Algebra II. In courses of Algebra I through Algebra III, classroom experience with this topic has been very rewarding when the techniques for adding and subtracting, using complement arithmetic, were presented only for binary numbers. In all cases, the majority of students sought and discovered a way of performing similar operations in base 10 (i.e., using the 9's complement) and many extended this to other bases as well. Many students of Algebra II were also able to discover methods for handling the case where A <= B and cases where A and/or B were negative numbers.

V. SOLVING FIRST-DEGREE EQUATIONS AND INEQUALITIES

A. Solving Inequalities

A program, which solves simple equations and then types each step of the solution is shown on page 50. Having students write a similar program which solves inequalities and types each step of the solution is a very good exercise. For better students, the problem can simply be assigned as stated. Average students will have to tackle the main problems of such an assignment if the program on page 50 is given to them and they are asked to modify the program so that it works for inequalities. One such modification is:

```
01.01 S GR=1;S LE=-1;T %6.03;S B=0;FOR X=0,1;DO 2;S LS[X]=L;S RS[X]=R
01.02 SET SX=RS[1]-RS[0];SET CX=LS[1]-LS[0]-SX;TYPE "TO BOTH SIDES:",!
01.03 IF (SX) 1.04,1.05;TYPE "SUBTRACT ",SX,"*X",!;SET B=B-1;GOTO 1.05
01.04 TYPE "ADD ",-SX,"*X"!; SET B=B-1
01.05 IF (LS[0]) 1.06,1.07;TYPE "SUBTRACT ",LS[0],!;SET B=B-1;GOTO 1.07
01.06 TYPE "ADD ",-LS[0],!;SET B=B-1
01.07 IF (CX-1) 1.08,1.11,1.08
01.08 IF (FABS(CX)-1) 1.09,1.09;TYPE "DIVIDE BY ",CX,!;GOTO 1.10
01.09 TYPE "MULTIPLY BY ",1/CX,!
01.10 SET B=-5
01.11 IF (B) 1.12;TYPE "DO NOTHING",!
01.12 TYPE "THE SOLUTION IS X  "; IF (B+4) 1.13,1.13,1.14
01.13 IF (-CX) 1.14; SET INEQ=-INEQ
01.14 IF (INEQ) 1.15; TYPE ">  "; GOTO 1.16
01.15 TYPE "<  "
01.16 TYPE (RS[0]-LS[0])/CX,!!
```

The operation of this program is identical to thar of the program on page 50. The output of three runs of the program is:

```
*2.1 SET L=X+5; SET INEQ=LE; SET R=3*X-3
*GO
TO BOTH SIDES:
SUBTRACT =+  3.000*X
SUBTRACT =+  5.000
DIVIDE BY =-  2.000
THE SOLUTION IS X  > =+  4.000

*2.1 SET L=4*X-3; SET INEQ=LE; SET R=4+2*X
*GO
TO BOTH SIDES:
SUBTRACT =+  2.000*X
ADD =+  3.000
DIVIDE BY =+  2.000
THE SOLUTION IS X  < =+  3.500
```

solving

x+5<3x-3

solving

4x-3<4+2x

```
*2.1 SET L=2*X+3; SET INEQ=GR; SET R=3*X-2
*GO
TO BOTH SIDES:
SUBTRACT =+   3.000*X
SUBTRACT =+   3.000
MULTIPLY BY =-   1.000
THE SOLUTION IS X   < =+   5.000

*
```

solving

$2x+3>3x-2$

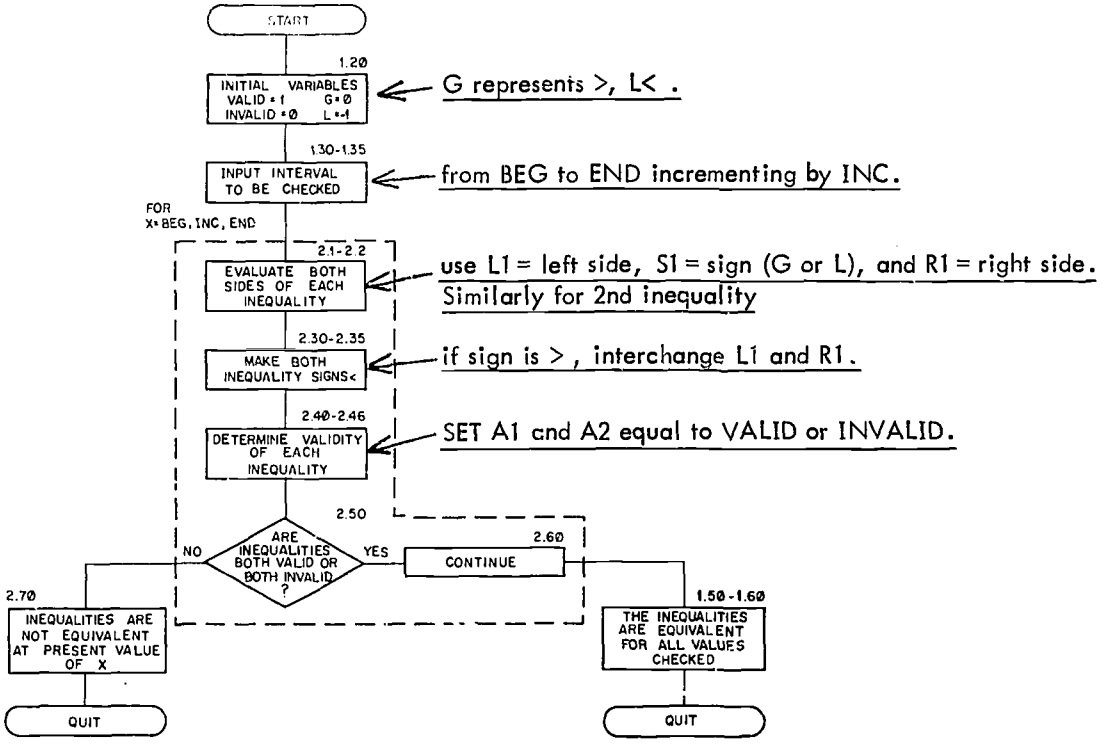
Certainly the students who can write a similar program fully understand the techniques used in solving simple inequalities.

Note that when modifying the program on page 50, very few changes are necessary. LE = -1 and GR = 1 are SET in the first step, SET B = -5 replaces step 1.10, and steps 1.11 through 1.16 are all used for typing the solution. This emphasizes the point that the steps used in solving inequalities are identical to those used in solving equations, with the exception of reversing the sense of the inequality sign when multiplying or dividing by a negative number.

In addition to being a good program for use as a student assignment, this program is also useful for demonstrations. It might also be modified in several different ways to provide extra practice for students having trouble with this topic. An easily made modification is to have the program type the steps of the solution as it already does, but then ASK rather than type the final result. This would supply students with the correct steps for solution but still require them to execute these steps. The program could then confirm or correct the student's solution. Yet another useful variation would be a program that ASKed for each of the steps of the solution, confirming or denying each step as the student responds.

Another problem that can be effectively used as an assignment is the writing of a program to identify equivalent inequalities. One approach to the solution of the problem is to utilize a technique similar to that of the previous program to actually solve both inequalities and then compare the results. (The very short program on page 49, used for solving equations, is an excellent start on this program.) A second approach is to verify the equivalence of the two inequalities over an ASKed interval without actually solving the inequalities.

No sample program has been included using the first approach, as this would be quite similar to previous programs. A flow chart for a program using the second approach appears as:



This program appears as:

```
01.10 C PROGRAM TO IDENTIFY EQUIVALENT INEQUALITIES
01.20 SET VALID=1; SET NOTVALID=0; SET G=0; SET L=-1
01.30 ASK "INTERVAL TO CHECK IS FROM ",BEG," TO ",END
01.35 ASK " INCREMENTING BY ",INC,!
01.40 FOR X=BEG,INC,END; DO 2
01.50 TYPE "THE INEQUALITIES ARE EQUIVALENT FOR ALL OF THE VALUES"
01.60 TYPE !"CHECKED IN THE INTERVAL."!!; QUIT

02.30 IF (S1) 2.35; SET T=L1; SET L1=R1; SET R1=T
02.35 IF (S2) 2.4; SET T=L2; SET L2=R2; SET R2=T
02.40 SET A1=VALID; SET A2=VALID
02.43 IF (L1-R1) 2.46; SET A1=NOTVALID
02.46 IF (L2-R2) 2.5; SET A2=NOTVALID
02.50 IF (A1-A2) 2.7,2.6,2.7
02.60 RETURN
02.70 TYPE "INEQUALITIES NOT EQUIVALENT WHEN X",%6.02,X,!!; QUIT
*
```

Following is the output of two runs of this program. In the first run, the inequalities checked are $x + 5 < 3x - 3$ and $2x + 9 > 21 - x$, while in the second run they are $2x + 9 > 14 - x$ and $x + 5 < 2x - 3$.

```
*2.1 SET L1=X+5; SET S1=L; SET R1=3*X-3
*2.2 SET L2=2*X+9; SET S2=G; SET R2=21-X
*GO
INTERVAL TO CHECK IS FROM :-5  TO :20 INCREMENTING BY :.25
THE INEQUALITIES ARE EQUIVALENT FOR ALL OF THE VALUES
CHECKED IN THE INTERVAL.

*2.1 SET L1=2*X+9; SET S1=G; SET R1=14-X
*2.2 SET L2=X+5; SET S2=L; SET R2=2*X-3
*GO
INTERVAL TO CHECK IS FROM :-10  TO :15  INCREMENTING BY :.5
INEQUALITIES NOT EQUIVALENT WHEN X=+   2.00

*
```

A common student error when using the approach exemplified by this program is to label a pair of inequalities as not equivalent because both are not valid for a given number. This error reflects a basic misunderstanding of the meaning of equivalence, which is not always uncovered if only the first approach to the solution is considered.

Pairs of inequalities can be plotted on the number line by using a program similar to that shown on page 10. Such a program appears as:

```
01.10 TYPE !"USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:"!!" "
01.20 FOR X=-8,.25,8; DO 2
01.30 TYPE #," "
01.40 FOR X=-8,.25,8; DO 3
01.50 TYPE :"-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5"
01.60 TYPE "   6   7   8"!!!; QUIT

02.90 TYPE "."; CONTINUE

03.90 TYPE " "; CONTINUE

04.90 TYPE "\"; CONTINUE  }
                          }  <----\ is used to plot one inequality
05.90 TYPE "/"; CONTINUE  }       and / is used for the other.
*
```

The output of a run of the program is:

```
*GO

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

.................................................................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

This is correct, since no conditions of inequality were given.

Inequalities are entered in much the same way as shown on page 10. Basically, parts 2 and 3 are used to define the inequalities to be plotted. For a given value of X, if the first inequality is valid, 4.9 is executed, if it is invalid 2.9 is done. Similarly, for a given value of X, if the second inequality is valid, the program branches to 5.9, and if it is invalid, a branch is made to 3.9.

In the following run of this program, $5x - 1 \leq 9$ and $x + 3 > 2$ are entered and then plotted.

```
02.10 IF (5*X-1-9) 4.9, 4.9, 2.9

03.10 IF (X+3-2) 3.9, 3.9, 5.9
*GO

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS

 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\XXXXXXXXXXXXX/////////////////////////
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

The output of three more runs of this program is:

```
02.10 IF (FABS(X)-1) 4.9, 4.9, 2.9 <----- |x| <= 1

03.10 IF (FABS(X)-3) 3.9, 5.9, 5.9 <----- |x| >= 3
*GO

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ////////////////////////.......\\\\\\\\\\.......////////////////////////
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

```
02.10 IF (X-4) 4.9, 4.9, 2.9 <----- x - 4 <= 0

03.10 IF (-3-X) 3.2, 3.2, 3.9 <----- -3 <= x < 2
03.20 IF (X-2) 5.9, 3.9, 3.9
*GO

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 \\\\\\\\\\\\\\\\\\\\\\\\XXXXXXXXXXXXXXXXXXX\\\\\\\\.................
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3  4   5   6   7   8

*
```

```
02.10 IF (-3-X) 2.2, 2.2, 2.9
02.20 IF (X-2) 4.9, 2.9, 2.9 <------ -3 <= x < 2

03.10 IF (5-X) 3.9, 3.9, 3.2 <------ 0 < x < 5
03.20 IF (X) 3.9, 3.9, 5.9
*GO

USING YOUR CONDITIONS, THE NUMBER LINE APPEARS AS:

 ....................\\\\ \\\\ \\\\ \XXXXXXX/////////////.............
-8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8

*
```

If this program is not given as an assignment, it can be used for demonstration. By presenting inequalities and having the class write the steps needed to enter them, individual misunderstandings might be eliminated. Note that in doing this, students are also getting practice using logical AND/OR operations.

B. Solving Equations

The solving of literal equations can be reinforced by giving students an equation such as

$$2A + B = \frac{3C}{D+4}$$

and asking them to write a program which will ASK for all four of the variables. The user should then be able to enter numerical values for three of the variables and the word FIND for the fourth. The program should then type the value of the unknown variable. Such a program would appear as:

```
01.10 C FORMULA EVALUATION
01.20 ASK ?A ?,? B ?,? C ?,? D ?,!
01.30 IF (A-7044) 1.4,1.31,1.4
01.31 SET A=((3*C)/(D+4)-B)/2; TYPE %8.03,?A?,!!; QUIT
01.40 IF (B-7044) 1.5,1.41,1.5
01.41 SET B=(3*C)/(D+4)-2*A; TYPE %8.03,?B?,!!; QUIT
01.50 IF (C-7044) 1.6,1.51,1.6
01.51 SET C=((2*A+B)*(D+4))/3; TYPE %8.03,?C?,!!; QUIT
01.60 SET D=(3*C)/(2*A+B)-4; TYPE %8.03,?D?,!!; QUIT
*
```

Although the writing of this program involves little more than the conventional paper and pencil task of solving the equation for each of the variables, classroom experience with the assignment has repeatedly given more accurate results when the program is also expected. The output of several runs of this program is:

```
*GO
A :2 B :4 C :8 D :FIND
D=-   1.000

*GO
A :2 B :4 C :FIND D :-1
C=+   8.000

*GO
A :2 B :FIND C :8 D :-1
B=+   4.000

*GO
A :FIND B :4 C :8 D :-1
A=+    2.000

*
```

If the programs suggested for solving equations, shown on pages 46 through 51, have not all been used, they are also applicable at this point. The writing of a program similar to that on page 72, to tell whether or not two equations have identical solution sets, is also a worthwhile exercise.

Classroom experience with better than average students has demonstrated that the assignment "Write a program which will allow the user to enter an equation (first degree) and then type out the solution, " will yield very beneficial results. Students will create, evaluate, and reject many different schemes for attacking this problem, and in so doing, will learn some significant mathematical ideas. The majority of students will, however, need a little more guidance when writing such a program. In Section II, programs were considered which solved an equation with an integral solution within an ASKed domain. Many students will want to continue this same approach and write a program which will not require a domain to be entered. (The programs on pages 46 through 51 already do this using a VERY different approach.)

Following is a program typical of those which students write when attempting this problem:

```
01.10 SET X=0; DO 2
01.20 SET X=X+1; DO 2
01.30 SET X=-X; DO 2
01.40 SET X=-X; GOTO 1.2

02.20 IF (L-R) 2.3,2.4,2.3
02.30 RETURN
02.40 TYPE "SOLUTION IS X",%6.02,X,!!; QUIT
*
```

The output from two runs of this program is:

```
*2.1 SET L=2*X+3; SET R=3*X-2          solves 2x + 3 = 3x -2
*GO
SOLUTION IS X=+   5.00

*2.1 SET L=3*X+220+2*X; SET R=X-192      solves 3x + 220+2x = X-192
*GO
SOLUTION IS X=- 103.00

*
```

The program does exactly as intended - it gives an integral solution to a first -degree equation without inputting a domain to search. There are, however, several serious disadvantages to this program which leave many students dissatisfied. First, the program is of no use at all if the solution is not integral, and second, if a root is suspected to be near, for example, -100, there is no way to avoid trying approximately 200 incorrect values before the correct solution is found. Thus there remain several problems to be overcome before the majority of students are satisfied. Since the students seem to benefit more when allowed to "discover" as much as possible, the following suggestions for modifications of this program should be made as brief as possible if given to the students.

1. Alter the program so that the starting point of the search is ASKed rather than always starting at zero. This will completely eliminate the second objection mentioned above.
2. Alter the program so that the value used to increment X is ASKed rather than always being 1. This will allow the program to identify some non-integral solutions.
3. With modifications 1 and 2, the program will only solve an equation with a non-integral solution if the input value for the increment happens to allow X to exactly equal the solution. To help eliminate this problem, suggest that the program be altered to accept a value of X as the solution if the two sides of the equation differ by no more than an ASKed value of "tolerance." This will permit the solution to be found to any degree of accuracy desired by simply re-running the program with different input values.

A program that incorporates all three of these suggestions is:

```
01.10 ASK ?START ?,?  INCREMENT ?,?  TOLERANCE ?,!
01.20 SET ADD=0; SET X=START; DO 2
01.30 SET ADD=ADD+INCREMENT; SET X=START+ADD; DO 2
01.40 SET X=START-ADD; DO 2
01.50 GOTO 1.3

02.20 IF (FABS(L-R)-TOLERANCE) 2.4,2.4,2.3
02.30 RETURN
02.40 TYPE "APPROXIMATE SOLUTION IS X",%6.03,X,!!; QUIT
*
```

A complete solution to the equation 6x - 5.7 = 8 + 2x can be found using this program as follows:

```
*2.1 SET L=6*X-5.7; SET R=8+2*X
*GO
START :0   INCREMENT :1   TOLERANCE :2
APPROXIMATE SOLUTION IS X=+  3.000

*GO
START :3   INCREMENT :.5   TOLERANCE :1
APPROXIMATE SOLUTION IS X=+  3.500

*GO
START :3.5   INCREMENT :.25   TOLERANCE :.5
APPROXIMATE SOLUTION IS X=+  3.500

*TYPE L,R,!
=+ 15.300=+ 15.000
*

*GO
START :3.5   INCREMENT :.1   TOLERANCE :.25
APPROXIMATE SOLUTION IS X=+  3.400

*GO
START :3.4   INCREMENT :.05  TOLERANCE :.1
APPROXIMATE SOLUTION IS X=+  3.450

*GO
START :3.45   INCREMENT :.01   TOLERANCE :.05
APPROXIMATE SOLUTION IS X=+  3.430

*GO
START :3.43   INCREMENT :.005   TOLERANCE :.02
APPROXIMATE SOLUTION IS X=+  3.425

*GO
START :3.425   INCREMENT :.0001   TOLERANCE :.001
APPROXIMATE SOLUTION IS X=+  3.425

*TYPE L,R,!
=+ 14.850=+ 14.850
*
```

Note: The result of the first run is used as START in the second run. Also, the increment and tolerance are reduced on each succeeding run.

Since the solution did not change, the two sides of the equation were examined to see if an exact solution had been found.

Evidently the exact solution has been found.

Certainly this type of program begins to teach the important ideas of iteration and approximation. At this point the better students might be encouraged to continue work on this program so that it alters the increment itself until an acceptable solution is found. However, for the majority of students, the introduction of this technique should be postponed until graphs have been discussed.

C. Word Problems

An effective technique for stimulating interest as well as for teaching methods of solving word problems is to have students write a program which will create and present a problem, ASK the user to solve the problem, and then verify the user's response. For example, consider the problem:

> A car traveling CS (have computer generate an integer 40 through 65) miles per hour can make a certain trip in D (have computer generate an integer 5 through 20) hours less than a train traveling at TS (have computer generate an integer 20 through 39) miles per hour. How long does the trip take by car?

The student's program should type out this problem with numbers generated by the random function appropriately substituted, then ASK for the user's solution to the problem, and then verify this solution. Note that to verify the user's solution, the student's program must also correctly solve the problem. A program which does all of this is:

```
01.10 C DISTANCE, RATE, TIME PROBLEM
01.20 TYPE "A CAR TRAVELING "; SET CS=FITR(FABS(FRAN())*25)+40
01.21 TYPE %3.0,CS," MILES PER HOUR CAN MAKE A CERTAIN TRIP IN"!
01.22 SET D=FITR(FABS(FRAN())*15)+5;TYPE D," HOURS LESS THAN A"
01.23 TYPE " TRAIN TRAVELING AT ";SET TS=FITR(FABS(FRAN())*19)+20
01.24 TYPE TS," MILES PER HOUR."!"HOW LONG DOES THE TRIP TAKE BY CAR?"!
01.25 ASK ?ANSWER ?;SET VALUE=D*TS/(CS-TS)
01.26 IF (FABS(ANSWER-VALUE)-.009999) 1.27,1.27,1.28
01.27 TYPE !"CORRECT"!!; QUIT
01.28 TYPE !"NOT SO. THE CORRECT VALUE IS ",%6.02,VALUE," HOURS."!!
*
```

The output from two runs of this program is:

```
*GO
A CAR TRAVELING =+ 58 MILES PER HOUR CAN MAKE A CERTAIN TRIP IN
=+ 10 HOURS LESS THAN A TRAIN TRAVELING AT =+ 21 MILES PER HOUR.
HOW LONG DOES THE TRIP TAKE BY CAR?
ANSWER :4.38
NOT SO.  THE CORRECT VALUE IS =+  5.68 HOURS.
```

```
*GO
A CAR TRAVELING =+ 55 MILES PER HOUR CAN MAKE A CERTAIN TRIP IN
=+  9 HOURS LESS THAN A TRAIN TRAVELING AT =+ 20 MILES PER HOUR.
HOW LONG DOES THE TRIP TAKE BY CAR?
ANSWER :5.14
CORRECT

*
```

If students have not been exposed to the random number function, these numbers might be ASKed rather than be generated by the computer. Using random numbers is, however, a topic that appeals to the majority of students, and these should be used unless time does not permit their introduction.

An effective use of this type of problem is to assign one problem from a list of several different problems to each student or to small groups of students. After the programs are written, students are directed to check programs written by others. In this way, each student not only fully understands the type of problem assigned to him, but he also gets practice in solving all of the other types as well. Note that the student must solve the problem in general in order to write his program. This should give him more valuable experience than solving two or three similar problems with specific numeric values.

Consider the following mixture problem as a second example:

> How many ounces of gold worth GV (generate a number 48 through 52) dollars per ounce must be packed with S (generate a number 10 through 40) ounces of silver worth SV (generate a number 2 through 4) dollars per ounce to produce a package worth TV (generate a number 5 through 35) dollars per ounce?

A program that presents this problem is:

```
01.10 C MIXTURE PROBLEM
01.20 TYPE "HOW MANY OUNCES OF GOLD WORTH ",%5.02
01.21 SET GV=FABS(FRAN())*3+48; TYPE GV," DOLLARS PER OUNCE"!
01.22 TYPE "MUST BE PACKED WITH ";SET S=FABS(FRAN())*10+30
01.23 TYPE S," OUNCES OF SILVER WORTH ";SET SV=FABS(FRAN())*2+2
01.24 TYPE SV,!"DOLLARS PER OUNCE TO PRODUCE A PACKAGE WORTH"
01.25 SET TV=FABS(FRAN())*30+5;TYPE TV," DOLLARS PER OUNCE?"!
01.26 ASK ?ANSWER ?;SET VALUE=S*(TV-SV)/(GV-TV)
01.27 IF (FABS(ANSWER-VALUE)-.009999) 1.28,1.28,1.29
01.28 TYPE !"CORRECT"!!; QUIT
01.29 TYPE !"NOT SO. THE CORRECT VALUE IS ",VALUE," OUNCES."!!; QUIT
*
```

The output from two runs of this program is:

```
*GO
HOW MANY OUNCES OF GOLD WORTH =+ 50.26 DOLLARS PER OUNCE
MUST BE PACKED WITH =+ 38.00 OUNCES OF SILVER WORTH =+  2.60
DOLLARS PER OUNCE TO PRODUCE A PACKAGE WORTH=+  8.16 DOLLARS PER OUNCE?
ANSWER :50.18
NOT SO. THE CORRECT VALUE IS =+  5.02 OUNCES.

*
```

```
*GO
HOW MANY OUNCES OF GOLD WORTH =+ 49.96 DOLLARS PER OUNCE
MUST BE PACKED WITH =+ 32.50 OUNCES OF SILVER WORTH =+  3.76
DOLLARS PER OUNCE TO PRODUCE A PACKAGE WORTH=+ 12.62 DOLLARS PER OUNCE?
ANSWER :7.71
CORRECT

*
```

Note that in both of the above programs the comparison between the computed result and the user's inputted result is not a simple IF (ANSWER-VALUE) statement. Instead, the input answer is accepted as correct if it is the same through two decimal places. With this exception, these are very straightforward programs. As a result, the major part of a student's time will be spent on the main purpose of the assignment, i.e., solving the problem in general.

A third, more difficult example is to solve in general the problem:

> If one of two (type complementary or supplementary) angles measures X (generate a number 1 through 88 if complementary, 1 through 178 if supplementary) degrees (type more or less) than the other, what is the measure of the smaller angle?

A program that presents this problem is:

```
01.10 C COMPLEMNTARY AND SUPPLEMENTARY ANGLE PROBLEM
01.20 TYPE "IF ONE OF TWO "; IF (FRAN()) 1.21,1.21,1.22
01.21 T "SUPPLEMENTARY";S X=FITR(FABS(FRAN())*177)+1;S SIG=1;GOTO 1.23
01.22 T "COMPLEMENTARY";S X=FITR(FABS(FRAN())*87)+1;S SIG=-1
01.23 TYPE " ANGLES MEASURES ",%4.0,X," DEGREES "!
01.24 IF (FRAN()) 1.25,1.25; TYPE "MORE"; GOTO 1.26
01.25 TYPE "LESS"
01.26 TYPE " THAN THE OTHER, WHAT IS THE MEASURE OF THE SMALLER ANGLE?"!
01.27 ASK ?ANSWER ?
01.28 IF (SIG) 1.29; SET SMALLER=90-X/2; GOTO 1.30
01.29 SET SMALLER=45-X/2
01.30 IF (ANSWER-SMALLER) 1.32,1.31,1.32
01.31 TYPE !"CORRECT!"!!; QUIT
01.32 TYPE !"NOT SO.  THE SMALLER ANGLE CONTAINS ",%5.01,SMALLER
01.33 TYPE " DEGREES."!!; QUIT
*
```

The output from two runs of this program is:

```
*GO
IF ONE OF TWO SUPPLEMENTARY ANGLES MEASURES =+  46 DEGREES
MORE THAN THE OTHER, WHAT IS THE MEASURE OF THE SMALLER ANGLE?
ANSWER :77
NOT SO.  THE SMALLER ANGLE CONTAINS =+  67.0 DEGREES.

*GO
IF ONE OF TWO COMPLEMENTARY ANGLES MEASURES =+  73 DEGREES
LESS THAN THE OTHER, WHAT IS THE MEASURE OF THE SMALLER ANGLE?
ANSWER :8.5
CORRECT!

*
```

This problem requires the student's program to remember which condition, complementary or supplementary, was used, and then to adjust the angular measure accordingly. When writing this program, many students will be surprised when they discover that the conditions "more or less" do not affect the solution. In some cases students may actually treat the two cases separately and not make this discovery until the two cases are programmed. In either case, the Algebra I students who successfully write a program such as this either knew or have learned some techniques of both solving equations and general problem solving.

As a final, more challenging example for better students consider the problem:

> Find the smallest of N (type 2, 3, or 4) consecutive (type even or odd) integers whose sum is S (generate a number 16 through 100).

A program which does this is:

```
01.10 C CONSECUTIVE INTEGER PROBLEM
01.20 SET N=FITR(FABS(FRAN())*2.9)+2; TYPE "FIND THE SMALLEST OF "
01.21 TYPE %3.0,N," CONSECUTIVE "; SET EVEN=-1; SET ODD=0
01.22 IF (FRAN()) 1.23,1.23; SET CK=EVEN; TYPE "EVEN"; GOTO 1.24
01.23 SET CK=ODD; TYPE "ODD"
01.24 SET S=FITR(FABS(FRAN())*84)+16; TYPE " INTEGERS WHOSE SUM IS "
01.25 TYPE %4.0,S," ."!ANSWER IS (ENTER NONE IF NO SOLUTION) "
01.30 ASK A,!; SET V=S/N-(N-1)
01.31 IF (FITR(V)-V) 1.34,1.32,1.34
01.32 IF (CK) 1.33; IF (FITR(V/2)-V/2) 1.4,1.34,1.4
01.33 IF (FITR(V/2)-V/2) 1.34,1.4,1.34
01.34 SET V=1564; C THIS SETS V=NONE
01.40 IF (A-V) 1.42,1.41,1.42
01.41 TYPE "CORRECT"!!; QUIT
01.42 IF (V-1564) 1.43,1.44,1.43
01.43 TYPE "NOT SO.  THE SMALLEST INTEGER IS ",%3.0,V,!!; QUIT
01.44 TYPE "NOT SO.  THERE IS NO SOLUTION."!!; QUIT
*
```

The output of several runs of this program is:

```
*GO
FIND THE SMALLEST OF =+ 2 CONSECUTIVE ODD INTEGERS WHOSE SUM IS =+   16 .
ANSWER IS (ENTER  NONE  IF NO SOLUTION) :7
CORRECT

*GO
FIND THE SMALLEST OF =+ 4 CONSECUTIVE EVEN INTEGERS WHOSE SUM IS =+  31.
ANSWER IS (ENTER  NONE  IF NO SOLUTION) :6
NOT SO.  THERE IS NO SOLUTION.

*GO
FIND THE SMALLEST OF =+ 2 CONSECUTIVE EVEN INTEGERS WHOSE SUM IS =+  22.
ANSWER IS (ENTER NONE IF NO SOLUTION)  :12
NOT SO.  THE SMALLEST INTEGER IS =+  10
```

```
*GO
FIND THE SMALLEST OF =+ 3 CONSECUTIVE ODD INTEGERS WHOSE SUM IS =+  85.
ANSWER IS (ENTER  NONE  IF NO SOLUTION) :NONE
CORRECT

*GO
FIND THE SMALLEST OF =+ 2 CONSECUTIVE ODD INTEGERS WHOSE SUM IS =+  40 .
ANSWER IS (ENTER  NONE  IF NO SOLUTION) :17
NOT SO.  THE SMALLEST INTEGER IS =+  19

*
```

This problem adds the dimension of identifying values for which no solution exists. Once a numerical answer is computed, the program must verify that it is both integral and even or odd as earlier specified. For this as well as the three preceding examples, a student must solve the problem in the general case before writing his program. In finding this general solution, the student demonstrates that he can adequately solve the problem for all specific numerical cases. By also having students use programs written by others, a task they are usually eager to undertake, they receive practice with many different types of word problems.

D. Magic Squares (Supplementary)

Although basically a frivolous topic, the techniques for construction of magic squares appeal to many students. This somewhat self-motivating topic can be used to provide an excellent set of exercises in analytical thought and organization.

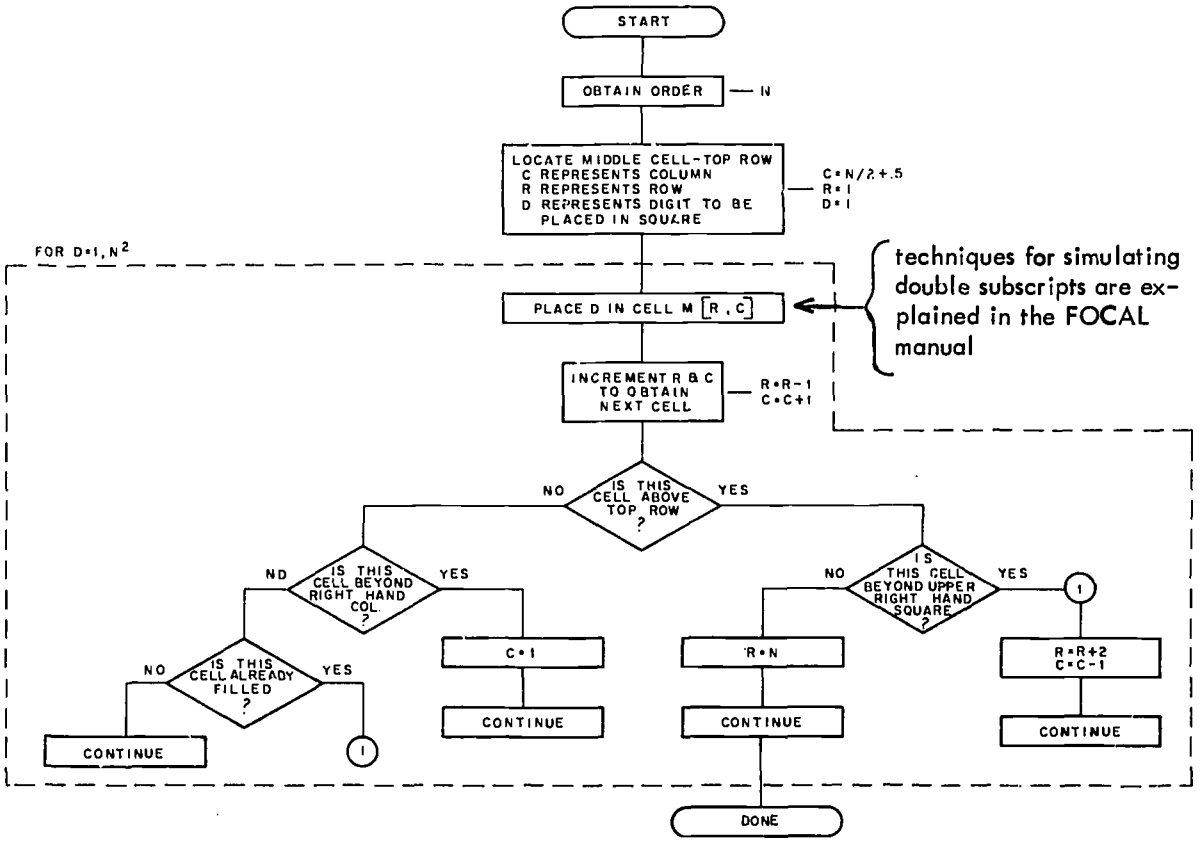
> Given a square matrix with n elements on a side (order n), a magic square is formed by arranging the integers 1 through n^2 so that the sum of the numbers in every row, column, and diagonol is equal to $1/2\ (n^3 + n)$.

Techniques for construction of magic squares are divided into three cases (1) for squares with an odd number of elements on a side; (2) for squares with x elements on a side where x is divisible by 2, but not 4; (3) for squares with x elements on a side where x is divisible by 4. A good exercise for students is to give them a rule for constructing magic squares of an odd order and then to ask them to write a program which uses this rule to construct squares of an ASKed odd order. The following such rule is taken from Mathematical Recreations and Essays. by W. W. Rouse Ball; Macmillan and Co.; 1940, page 195.

> "First, the number 1 is placed in the middle cell of the top row. The successive numbers are then placed in their natural order in a diagonal line which slopes upwards to the right, except that i) when the top row is reached, the next number is written in the bottom row as if it came immediately above the top row; ii) when the

right-hand column is reached, the next number is written in the left-hand column, as if it immediately succeeded the right-hand column; and iii) when a cell which has been filled up already or when the top right-hand square is reached, the path of the series drops to the row vertically below it and then continues to mount again."

The task of going from this explanation to the following flow chart is a good exercise in analytical thought.



A student-written flow chart such as this reflects a nicely organized approach to solving the problem. The following program is written using the algorithm shown in the flow chart.

```
01.10 ERASE
01.20 ASK "ORDER ",N,!; SET R=1; SET C=N/2+.5
01.30 FOR D=1,N+2; SET M[R+C*N]=D; DO 2
01.40 FOR R=1,N; TYPE %3.0,!; FOR C=1,N; TYPE M[R+C*N]
01.50 TYPE !!!; QUIT

02.10 SET R=R-1; SET C=C+1; IF (1-R) 2.4,2.4,2.2
02.20 IF (N-C) 2.3; SET R=N; GOTO 2.6
02.30 SET R=R+2; SET C=C-1; GOTO 2.6
02.40 IF (C-N) 2.5,2.5; SET C=1; GOTO 2.6
02.50 IF (M[R+C*N]) 2.6,2.6,2.3
02.60 CONTINUE
*
```

The output of two runs of this program is:

```
*GO
ORDER :3

=+  8=+  1=+  6
=+  3=+  5=+  7
=+  4=+  9=+  2


*GO
ORDER :9

=+  47=+ 58=+ 69=+ 80=+  1=+ 12=+ 23=+ 34=+ 45
=+  57=+ 68=+ 79=+  9=+ 11=+ 22=+ 33=+ 44=+ 46
=+  67=+ 78=+  8=+ 10=+ 21=+ 32=+ 43=+ 54=+ 56
=+  77=+  7=+ 18=+ 20=+ 31=+ 42=+ 53=+ 55=+ 66
=+   6=+ 17=+ 19=+ 30=+ 41=+ 52=+ 63=+ 65=+ 76
=+  16=+ 27=+ 29=+ 40=+ 51=+ 62=+ 64=+ 75=+  5
=+  26=+ 28=+ 39=+ 50=+ 61=+ 72=+ 74=+  4=+ 15
=+  36=+ 38=+ 49=+ 60=+ 71=+ 73=+  3=+ 14=+ 25
=+  37=+ 48=+ 59=+ 70=+ 81=+  2=+ 13=+ 24=+ 35


*
```

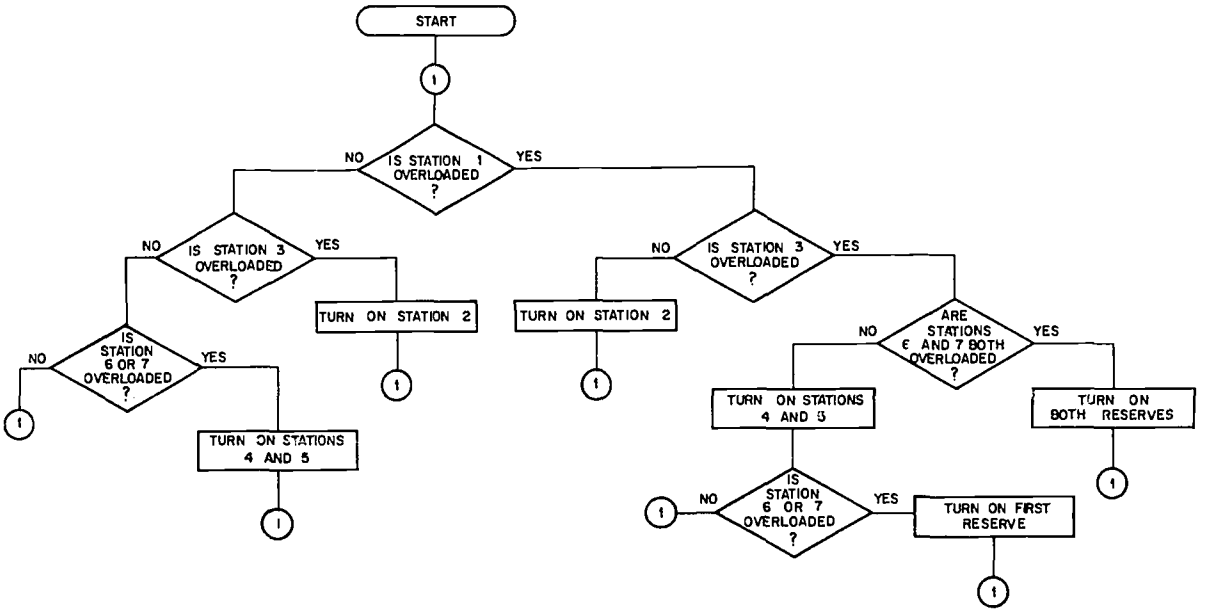
Using the 4K version of FOCAL, the program shown can only be used for squares of order 3, 5, 7, and 9 even though the algorithm is valid for any odd order square. Note that the program does not reject even numbers when order ASKed.

Classroom presentations of techniques for generating magic squares have indicated that the majority of students will benefit by writing a program similar to that shown in the example for odd order magic squares, but that the programming for the two types of even order squares frequently requires more time than is justified by the additional return. The even-order cases are better used as assignments for a course in computer science or for only the very best students in Algebra I and II.

A problem that lends itself well to teaching analytical thought and flow charting while also illustrating one type of real-time computer application is charting the logic of a method for controlling the power supply to a hypothetical city by manipulating seven regular and two reserve generation stations. The description of such a method is:

> If station 1 is overloaded, turn on station 2 unless station 3 is also overloaded. If so, turn on stations 4 and 5 unless stations 6 and 7 are both overloaded. If so, turn on both reserves. If not, turn on the first reserve if either 6 or 7 is overloaded.
>
> If station 1 is not overloaded and station 3 is, turn on station 2. If not, and station 6 or 7 is overloaded turn on stations 4 and 5. Otherwise do nothing.

One flow chart which represents this system is:



If time permits, the students might also write a program which ASKs the initial conditions of each station (on, off, or overloaded) and then performs the described checking sequence. Such an exercise is useful in developing programming skills as well as making the checking of student algorithms an easy process.

VI. OPERATING WITH POLYNOMIALS

A. Addition and Subtraction

The operations of addition and subtraction with polynomials are usually completely understood by most students without much difficulty, thus the presentation of this topic will require little or no computer assistance. The following program, however, might be used in several ways; as a demonstration program for the entire class; to provide drill work for students having difficulty; or, as a problem which better students are asked to program. This program uses the random function to generate problems involving polynomial addition and subtraction for the user to solve. The problems are generated in the following format:

(___+___+...) PLUS OR MINUS (___+___+...)

1 to 4 terms 1 to 4 terms

Each term is Cx^E

where C is an integer 1 through 50

E is an integer 0 through 4

The program is:

```
01.10 ERASE
01.20 T %3.0,"PROBLEM IS:"!;S SG=1;DO 2
01.30 IF (FRAN()) 1.31;T "PLUS ";GOTO 1.4
01.31 T "MINUS ";S SG=-1
01.40 DO 2
01.50 T !!"ENTER COEFFICIENT FOR EACH TERM";FOR E=0,4;DO 4
01.60 IF (OK) 1.7;T !"CORRECT"!!;QUIT
01.70 T !"THESE ARE NOT ALL CORRECT. WANT TO TRY AGAIN? (YES OR NO)"
01.80 A R;S OK=0;IF (R-155) 1.5,1.9,1.5
01.90 T !!;QUIT

02.10 T "(";S T=FITR(FABS(FRAN())*3)+2;FOR I=1,T;DO 3
02.20 T ") "

03.10 S C=FITR(FRAN()*49)+1;S E=0;T %3.0,C;IF (FRAN()) 3.2,3.4,3.4
03.20 T "*X";S E=FITR(FABS(FRAN())*3.9)+1;IF (1-E) 3.3,3.4,3.4
03.30 T "↑",%1.0,E,"  "
03.40 S TERM[E]=TERM[E]+C*SG;S P[E]=1

04.10 IF (P[E]) 4.2,4.7,4.2
04.20 IF (E) 4.4,4.3,4.4
04.30 A !"CONSTANT TERM IS ",A[E];GOTO 4.5
04.40 T !"COEF. OF X";IF (E-1) 4.45,4.45;T "↑",%1.0,E
04.45 A " TERM IS ",A[E]
04.50 IF (A[E]-TERM[E]) 4.6,4.7,4.7
04.60 S OK=-1
04.70 CONTINUE
*
```

The output of two runs of this program is:

```
*GO
PROBLEM IS:
(=- 45*X↑=+4  =- 26*X=-  1*X↑=+3  ) MINUS (=- 26*X=- 34*X=- 15)

ENTER COEFFICIENT FOR EACH TERM
CONSTANT TERM IS :15
COEF. OF X TERM IS :-52
COEF. OF X↑=+3 TERM IS :1
COEF. OF X↑=+4 TERM IS :-45
THESE ARE NOT ALL CORRECT. WANT TO TRY AGAIN? (YES OR NO):YES

ENTER COEFFICIENT FOR EACH TERM
CONSTANT TERM IS :15
COEF. OF X TERM IS :34
COEF. OF X↑=+3 TERM IS :1
COEF. OF X↑=+4 TERM IS :-45
CORRECT

*

*GO
PROBLEM IS:
(=+ 16*X↑=+3  =+  1) PLUS (=+  1=+ 17*X↑=+3  =+ 42=+ 17*X↑=+2  )

ENTER COEFFICIENT FOR EACH TERM
CONSTANT TERM IS :44
COEF. OF X↑=+3 TERM IS :51
CORRECT

*
```

When this program is used as an assignment for better students, they gain practice in anticipating all cases as well as in general problem solving since they must devise a scheme for presenting the problem, solving the problem and then checking the user's solution as well. Note that the program as shown may occasionally not have enough space to type an entire problem on a single line. This occurs so infrequently, however, that no provisions were made to prevent the resulting over typing in the rightmost print position.

B. Multiplication

The theorems used in raising a monomial to a power ($(x*y)^z=x^z.y^z$ and $(x^y)^z=x^{y*z}$ can be demonstrated and/or reinforced in several ways. A very brief program for either demonstration or use as a student assignment is the following:

```
01.10 ASK ?X ?,?  Y ?,?  Z ?,!
01.20 TYPE %,!?(X*Y)↑Z   ?,!?(X↑Z)*(Y↑Z) ?,!!
01.30 TYPE ?(X↑Y)↑Z ?,!?X↑(Y*Z) ?,!!
*
```

A run of this program is:

```
*GO
X:3      Y :5     Z :7

(X*Y)↑Z      =+0.170859E+09
(X↑Z)*(Y↑Z)  =+0.170859E+09

(X↑Y)↑Z =+0.500315E+17
X↑(Y*Z) =+0.500315E+17

*
```

Such a trivial program as this should be used with slower students. Experience indicates that the act of writing out the two theorems and then seeing them typed several times does reinforce their learning.

A somewhat more challenging student assignment is that of writing a program which ASKs for X, Y, and Z and then computes $(X*Y)^{\overline{Z}}$ and $(X^{Y})^{Z}$ without using the ↑ operator. In writing such a program, the students demonstrate not only an understanding of the theorems involved but also reinforce their understanding of the meaning of exponents. A typical student program for this assignment appears as:

```
01.10 ASK ?X ?,?  Y ?,?  Z ?;!
01.20 IF (Z) 1.3,1.3; IF (FITR(Z)-Z) 1.3,1.4,1.3
01.30 TYPE !"Z MUST BE A POSITIVE INTEGER."!!; QUIT
01.40 SET ANS=1; FOR COUNT=1,Z; SET ANS=ANS*(X*Y)
01.50 TYPE %,!"(X*Y)↑Z ",ANS
01.60 IF (Y) 1.7,1.7; IF (FITR(Y)-Y) 1.7,1.8,1.7
01.70 TYPE !"Y MUST BE A POSITIVE INTEGER."!!; QUIT
01.80 SET ANS=1; FOR COUNT=1,Y*Z; SET ANS=ANS*X
01.90 TYPE !"(X↑Y)↑Z ",ANS,!!
*
```

The output of two runs of this program is:

```
*GO
X :3   Y :5   Z :7

(X*Y)↑Z =+0.170859E+09
(X↑Y)↑Z =+0.500315E+17

*GO
X :30   Y :50   Z :70

(X*Y)↑Z =+0.212024E+223
(X↑Y)↑Z =+0.706025E+238

*
```

Students often comment on how "close" these numbers seem to be. This certainly affords an excellent opportunity to discuss magnitude of numbers.

When being introduced to the multiplication of monomial times polynomial and polynomial times polynomial, many students will benefit from an assignment to write a program which ASKs for the variables in an expression like $Ax^n (Bx^m + C)$ and then types the expression representing the product. Such a program is:

```
01.10 TYPE "INPUT THE NUMBERS IN THE EXPRESSION  A*X↑N (B*X↑M + C)"!
01.20 ASK ?A , N , B , M , C?,!
01.30 TYPE !"PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J  WHERE:"!
01.40 SET P=A*B; SET I=M+N; SET Q=A*C; SET J=N
01.50 TYPE %6.02,?P ?,?   I ?,!?Q ?,?   J ?,!!
*
```

Following is the output of a run of this program in which the expression $3x^5 (2x^4 + 7)$ is entered and the product $6x^9 + 21x^5$ is typed out.

```
*GO
INPUT THE NUMBERS IN THE EXPRESSION  A*X↑N  (B*X↑M + C)
A :3 , N :5 , B :2 , M :4 , C:7

PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J  WHERE:
P =+   6.00   I =+   9.00
Q =+  21.00   J =+   5.00

*
```

A similar program for the multiplication of two polynomials, $(Ax^n + B) (Cx^m + D)$, is:

```
01.10 TYPE "INPUT THE NUMBERS IN THE EXPRESSION (A*X↑N + B)(C*X↑M + D)"!
01.20 ASK ?A , N , B , C , M , D?,!
01.30 IF (M-N) 1.4,1.7,1.4
01.40 TYPE !"PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J + R*X↑K + S  WHERE:"!
01.50 SET P=A*C;SET I=M+N;SET Q=A*D;SET J=N;SET R=B*C;SET K=M;SET S=B*D
01.60 TYPE %6.02,?P ?,?   I ?,!?Q ?,?   J ?,!?R ?,?   K ?,!?S ?,!!;QUIT
01.70 TYPE !"PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J + R  WHERE:"!
01.80 SET P=A*C;SET I=N+M;SET Q=A*D+B*C;SET J=N;SET R=B*D
01.90 TYPE %6.02,?P ?,?   I ?,!?Q ?,?   J ?,!?R ?,!!
*
```

The output of two runs of this program is:

```
*GO
INPUT THE NUMBERS IN THE EXPRESSION (A*X↑N + B)(C*X↑M + D)
A :2 , N :3 , B :4 , C :3 , M :3 , D:-1   <-- (2x^3 + 4)(3x^3 - 1)

PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J + R  WHERE:
P =+   6.00   I =+   6.00
Q =+  10.00   J =+   3.00
R =-   4.00
```

```
*GO
INPUT THE NUMBERS IN THE EXPRESSION (A*X↑N + B)(C*X↑M + D)
A :5 , N :2 , B :-6 , C :4 , M :3 , D:1
PRODUCT IS IN THE FORM  P*X↑I + Q*X↑J + R*X↑K + S  WHERE:
P =+  20.00    I =+   5.00
Q =+   5.00    J =+   2.00
R =-  24.00    K =+   3.00
S =-   6.00

*
```

$(5x^2-6)(4x^3+1)$

Note that in this program, the student has provided for two different output forms. If m and n are equal, one term is typed rather than two terms of the same degree. Some students usually object to this output form for the cases in which the constant and/or the exponent is zero or one. These students usually agree that each term typed out ought to be in one of the following six formats:

0 A x Ax X^n Ax^n

Certainly these students should be encouraged in attempting to write a program which will use the formats they desire. Experience with this problem, however, has shown students are more successful if they first write a separate program which ASKs for A and N (Ax^n) and then types out the same number using the appropriate format. Their experience with this shorter program will then allow them to incorporate its ideas into other programs without much difficulty. To write this short program the student must first define the conditions which determine each case, as in

Output	Occurs When
0	A = 0, N = any value
A	A ≠ 0, N = 0
X	A = 1, N = 1
Ax	A ≠ 1 or 0, N = 1
X^n	A = 1, N ≠ 0 or 1
Ax^n	A ≠ 0 or 1, N ≠ 0 or 1

Then, he must identify with a series of IF statements, the case being examined. Such a program is:

```
01.10 TYPE "INPUT AN EXPRESSION OF THE FORM  A*X↑N"!
01.20 ASK ?A , N ?,!!"THIS CAN BE WRITTEN AS:  "
01.30 IF (A) 1.4,2.1,1.4
01.40 IF (N) 1.5,3.1,1.5
01.50 IF (N-1) 1.6,1.7,1.6
01.60 IF (A-1) 7.1,6.1,7.1
01.70 IF (A-1) 5.1,4.1,5.1

02.10 TYPE "0"!!; QUIT

03.10 TYPE %4.0,A,!!; QUIT

04.10 TYPE "X"!!; QUIT
```

```
05.10 TYPE %4.0,A,"*X"!!; QUIT
06.10 TYPE "X↑",%2.0,N,!!; QUIT
07.10 TYPE %4.0,A,"*X↑",%2.0,N,!!; QUIT
*
```

The output of two runs of this program is:

```
*GO
INPUT AN EXPRESSION OF THE FORM  A*X↑N
A :3 , N :0

THIS CAN BE WRITTEN AS:  =↑   3

*GO
INPUT AN EXPRESSION OF THE FORM  A*X↑N
A :5 , N :1

THIS CAN BE WRITTEN AS:  =↑   5*X
```

There are at least two good reasons for assigning the apparently easy programs shown on pages 88 and 89. First, the students must determine and write the general expression for the product. This expression should not be given in class. Second, the program both motivates and forces students to work several numerical examples, as this is the only way in which they can check that their progroms ore working properly.

Programs for multiplication of monomials and polynomials also can be written using the techniques shown in the program on page 87. These programs should probably be used as an assignment for better students or as teacher written progroms to provide individual drill for those students having difficulty, since the writing of the programs would require more time than the topic justifies for the average student.

Although the two special products $(A \pm B)^n$ can be handled with programs similar to those already shown in this section, students are often more interested in pursuing these cases using Pascal's triangle. Of course the development of Pascal's triangle requires the students to multiply the expressions in a traditional fashion, thus this practice is not lost. An approach that has been successfully used with students is to have them evaluate $(A + B)^n$ for $N = 1,2,3,4,5$ by multiplying the appropriate polynomials using pencil and paper. The resulting coefficients are then arranged by the teacher to form the first five lines of Pascal's triangle. The students are then asked to write a program which will ASK N, then print the first N lines of this triangle. Note that the students are left with the task of finding an appropriate algorithm. A typical student solution to this problem is:

```
01.01 C PROGRAM TO GENERATE N LINES OF PASCAL'S TRIANGLE (1<=N<=11)
01.10 ASK ?N ?
01.20 SET D=2; SET P[1]=1; SET P[2]=1; TYPE %3.0
01.30 TYPE !!; FOR J=1,N; DO 2
01.40 QUIT

02.10 FOR I=1,D; TYPE P[I]," "
02.15 SET D=D+1; TYPE !
02.20 FOR I=2,D-1; SET Q[I]=P[I]+P[I-1]
02.30 SET P[D]=1; FOR I=2,D-1; SET P[I]=Q[I]
*
```

The output of a run of this program is:

```
*GO
N :11
=+  1 =+  1
=+  1 =+  2 =+  1
=+  1 =+  3 =+  3 =+  1
=+  1 =+  4 =+  6 =+  4 =+  1
=+  1 =+  5 =+ 10 =+ 10 =+  5 =+  1
=+  1 =+  6 =+ 15 =+ 20 =+ 15 =+  6 =+  1
=+  1 =+  7 =+ 21 =+ 35 =+ 35 =+ 21 =+  7 =+  1
=+  1 =+  8 =+ 28 =+ 56 =+ 70 =+ 56 =+ 28 =+  8 =+  1
=+  1 =+  9 =+ 36 =+ 84 =+126 =+126 =+ 84 =+ 36 =+  9 =+  1
=+  1 =+ 10 =+ 45 =+120 =+210 =+252 =+210 =+120 =+ 45 =+ 10 =+  1
=+  1 =+ 11 =+ 55 =+165 =+330 =+462 =+462 =+330 =+165 =+ 55 =+ 11 =+  1
*
```

Note that the algorithm used in the program will correctly compute the terms of the Nth line for a very large N, but the type format used will only allow 12 terms (N=11) to be typed on a single line. This is not, however, much of a disadvantage as the main purpose of this assignment is to develop an algorithm for use in other programs.

After writing a program which generates Pascal's triangle, the students can be assigned the problem of writing a program which will ASK N, then type all the terms in the expansion of $(A+B)^n$. Such a program is:

```
01.01 C PROGRAM TO COMPUTE (A+B)↑N FOR N>=0
01.10 ASK ?N ?,!!,"THE RESULT IS THE SUM OF THE FOLLOWING TERMS:"!
01.20 SET D=2; SET P[1]=1; SET P[2]=1; IF (N-2) 1.3; FOR J=2,N; DO 2
01.30 FOR J=0,N; DO 3
01.40 QUIT

02.10 SET D=D+1; FOR I=2,D-1; SET Q[I]=P[I]+P[I-1]
02.20 SET P[D]=1; FOR I=2,D-1; SET P[I]=Q[I]

03.10 TYPE %8.0,P[J+1]," *A↑ ",%2.0,N-J," * B↑",J,!
*
```

Note that step 1.2 and part 2 are identical to the steps in the previous program which compute the terms in each line of Pascal's triangle. This technique of using part or all of one program when writing another is often useful, and this two-program sequence provides an excellent way to introduce this idea if it has not been previously discussed. The output of two runs of this program is:

```
*GO
N :3

THE RESULT IS THE SUM OF THE FOLLOWING TERMS:
=+       1 * A↑=+ 3 * B↑=+ 0
=+       3 * A↑=+ 2 * B↑=+ 1
=+       3 * A↑=+ 1 * B↑=+ 2
=+       1 * A↑=+ 0 * B↑=+ 3
*


*GO
N :20

THE RESULT IS THE SUM OF THE FOLLOWING TERMS:
=+       1 * A↑=+20 * B↑=+ 0
=+      20 * A↑=+19 * B↑=+ 1
=+     190 * A↑=+18 * B↑=+ 2
=+    1140 * A↑=+17 * B↑=+ 3
=+    4845 * A↑=+16 * B↑=+ 4
=+   15504 * A↑=+15 * B↑=+ 5
=+   38760 * A↑=+14 * B↑=+ 6
=+   77520 * A↑=+13 * B↑=+ 7
=+  125970 * A↑=+12 * B↑=+ 8
=+  167960 * A↑=+11 * B↑=+ 9
=+  184756 * A↑=+10 * B↑=+10
=+  167960 * A↑=+ 9 * A↑=+11
=+  125970 * A↑=+ 8 * B↑=+12
=+   77520 * A↑=+ 7 * B↑=+13
=+   38760 * A↑=+ 6 * B↑=+14
=+   15504 * A↑=+ 5 * B↑=+15
=+    4845 * A↑=+ 4 * B↑=+16
=+    1140 * A↑=+ 3 * B↑=+17
=+     190 * A↑=+ 2 * B↑=+18
=+      20 * A↑=+ 1 * B↑=+19
=+       1 * A↑=+ 0 * B↑=+20
*
```

If class time permits, the students can be asked to modify their programs for the expansion of $(A+B)^n$ so that the expansion of $(A-B)^n$ is given. The following program is identical to the previous program except for changes in steps 1.01, 1.30, and 3.10.

```
01.01 C PROGRAM TO COMPUTE (A-B)↑N FOR N>=0
01.10 ASK ?N ?,!!,"THE RESULT IS THE SUM OF THE FOLLOWING TERMS:"!
01.20 SET D=2; SET P[1]=1; SET P[2]=1; IF (N-2) 1.3; FOR J=2,N; DO 2
01.30 SET SIGN=1; FOR J=0,N; DO 3
01.40 QUIT

02.10 SET D=D+1; FOR I=2,D-1; SET Q[I]=P[I]+P[I-1]
02.20 SET P[D]=1; FOR I=2,D-1; SET P[I]=Q[I]

03.10 TYPE %8.0,SIGN*P[J+1]," * A↑",%2.0,N-J," * B↑",J,!; SET SIGN=-SIGN
*
```

The output of a run of this program is:

```
*GO
N :5

THE RESULT IS THE SUM OF THE FOLLOWING TERMS:
=+       1 * A↑=+ 5 * B↑=+ 0
=-       5 * A↑=+ 4 * B↑=+ 1
=+      10 * A↑=+ 3 * B↑=+ 2
=-      10 * A↑=+ 2 * B↑=+ 3
=+       5 * A↑=+ 1 * B↑=+ 4
=-       1 * A↑=+ 0 * B↑=+ 5
*
```

Having students alter their previous programs in this way strongly reinforces the similarity between the expansion of the expressions $(A+B)^n$ and $(A-B)^n$.

As the final example in this section, consider a program which types an expression of the form $(Cx + Dy)^n$ for the user to solve and then verifies this user's solution. This problem can be used as an assignment for better students or as a teacher written exericse which provides extra practice for students having difficulty. Although the following program is written so that it can provide extra practice for slow students, the program is more instructive for the student who writes it than it is for other students who use it. The program is:

```
01.01 C PROGRAM TO ASSIST EVALUATION OF THE EXPRESSION (C*X + D*Y)↑N
01.10 SET C=FITR(FRAN()*10); SET D=FITR(FRAN()*10)
01.15 SET N=FITR(FABS(FRAN())*5)
01.20 TYPE "THE PROBLEM IS: (",%3.0,C,"*X   ",D,"*Y )↑",%2.0,N,!!
01.30 TYPE "NOW INPUT THE COEFFICIENTS OF EACH TERM."!
01.40 SET DD=2; SET P[1]=1; SET P[2]=1; IF (N-2) 1.5; FOR J=2,N; DO 2
01.50 FOR J=0,N; DO 3
01.60 TYPE !!; QUIT

      Here again is the algorithm for constructing
                                  Pascal's triangle.
                                  ↓
02.10 SET DD=DD+1; FOR I=2,DD-1; SET Q[I]=P[I]+P[I-1]
02.20 SET P[DD]=1; FOR I=2,DD-1; SET P[I]=Q[I]

03.10 T "COEFFICIENT OF X↑",%1.0,N-J," * Y↑",J," TERM IS "; ASK ANS,!
03.20 IF (ANS-P[J+1]*C↑(N-J)*D↑J) 3.3,3.4,3.3
03.30 ASK "TRY THAT AGAIN                    ",ANS,!; GOTO 3.2
03.40 CONTINUE
*
```

The output of two runs of this program is:

```
*GO
THE PROBLEM IS: (=+    4*X    =-    2*Y )+=+ 3

NOW INPUT THE COEFFICIENTS OF EACH TERM.
COEFFICIENT OF X+=+3 * Y+=+0 TERM IS :64
COEFFICIENT OF X+=+2 * Y+=+1 TERM IS :-96
COEFFICIENT OF X+=+1 * Y+=+2 TERM IS :48
COEFFICIENT OF X+=+0 * Y+=+3 TERM IS :-8

*


*GO
THE PROBLEM IS:  (=+    9*X    =+  2*Y )+=+ 4

NOT INPUT THE COEFFICIENTS OF EACH TERM.
COEFFICIENT OF X+=+4 * Y+=+0 TERM IS :7561
TRY THAT AGAIN                       :6561
COEFFICIENT OF X+=+3 * Y+=+1 TERM IS :5832
COEFFICIENT OF X+=+2 * Y+=+2 TERM IS :1944
COEFFICIENT OF X+=+1 * Y+=+3 TERM IS :288
COEFFICIENT OF X+=+0 * Y+=+4 TERM IS :16

*
```

The techniques used in this program can be used with many different expressions. Class time might be used more efficiently if groups of four or five students are each given the task of writing a program similar to the example for a different expression. After writing their own program, the students can then use programs written by others, thus gaining practice with all of the different expressions.

C. Division

Programs similar to those on pages 88, 90 and 95 can be assigned while teaching division of polynomials. Since the basic formats and algorithms are very much like those already shown in the programs for polynomial multiplication, similar sample programs are not included in this text.

The following two programs are typical of problems that can be assigned for students to program. The first program ASKs for values in the expression $\frac{Ax^n}{Bx^m}$, and then simplifies the expression.

```
01.01 C PROGRAM TO EVALUATE THE EXPRESSION  A*X+N/B*X+M
01.10 TYPE "INPUT NUMBERS IN THE FORM A*X+N/B*X+M"!
01.15 ASK ?A ?,? N ?,?·B ?,? M ?,!
01.20 TYPE "RESULT IS  ",%6.02,A/B; IF (N-M) 1.3,1.4,1.5
01.30 TYPE " / X+",%2.0,M-N,!!; QUIT
01.40 TYPE !!; QUIT
01.50 TYPE " * X+",%2.0,N-M,!!
*
```

The output of three runs of this program is:

```
*GO
INPUT NUMBERS IN THE FORM A*X↑N/B*X↑M
A :4  N :5  B :2  M :8
RESULT IS  =+   2.00 / X↑=+ 3

*GO
INPUT NUMBERS IN THE FORM A*X↑N/B*X↑M
A :2  N :8  B :4  M :5
RESULT IS  =+   0.50 * X↑=+ 3

*GO
INPUT NUMBERS IN THE FORM A*X↑N/B*X↑M
A :2  N :7  B :8  M :7
RESULT IS  =+   0.25

*
```

$\frac{4x^5}{2x^8}$ $\frac{2x^8}{4x^5}$ $\frac{2x^7}{8x^7}$

Note that this student's program has included three different formats for typing the results. Using the three formats, the exponent of x is always positive and is not typed when equal to one or zero. By writing a program such as this, the student has demonstrated an ability to properly evaluate any similar expression with or without computer assistance.

A similar program which evaluates expressions of the form $\frac{Ax^n + Bx^m}{Cx^p}$ is:

```
01.01 C PROGRAM TO EVALUATE THE EXPRESSION (A*X↑N + B*X↑M) / (C*X↑P)
01.10 TYPE "INPUT NUMBERS IN THE FORM (A*X↑N + B*X↑M) / (C*X↑P)"!
01.15 ASK ?A ?,? N ?,?   B ?,? M ?,?   C ?,? P ?,!
01.20 TYPE !"RESULT IS ",%6.02,A/C; SET E=N-P; DO 2
01.30 TYPE "    ",%6.02,B/C; SET E=M-P; DO 2
01.40 IF (N-M) 1.6,1.5,1.6
01.50 TYPE !!"THIS CAN BE REDUCED TO ",%6.02,(A+B)/C; DO 2
01.60 TYPE !!; QUIT

02.10 IF (E) 2.2,2.3,2.4
02.20 TYPE " / X↑",%2.0,-E
02.30 RETURN
02.40 TYPE " * X↑",%2.0,E
*
```

The output of two runs of this program is:

```
*GO
INPUT NUMBERS IN THE FORM (A*X↑N + B*X↑M) / (C*X↑P)
A :4  N :7    B :3  M :2    C :5  P :4

RESULT IS =+   0.80 * X↑=+ 3   =+   0.60 / X↑=+ 2
```

$\frac{4x^7 + 3x^2}{5x^4}$

```
*GO
INPUT NUMBERS IN THE FORM (A*X↑N + B*X↑M) / (C*X↑P)
A :4  N :7    B :3  M :7    C :5  P :4  <-------- 4x^7+3x^7 / 5x^4

RESULT IS =+   0.80 * X↑=+ 3   =+   0.60 * X↑=+ 3

THIS CAN BE REDUCED TO =+   1.40 * X↑=+ 3

*
```

Here again there are three different formats for typing the answer as well as two steps which combine terms whenever possible. By using several programs like the preceding two with different groups of students, as discussed for the program on page 95, each student can be assigned a problem appropriate to his ability level.

Although the topic of synthetic division is usually reserved for the third year of algebra, there seems little reason for not presenting this technique when polynomial division is first introduced in Algebra I. Once the methods for synthetic division have been presented in class, the students can be assigned the task of writing a program which will divide polynomials using synthetic division. By successfully writing such a program, the students demonstrate a complete understanding of the required algorithm. This program is not a difficult one for most students to write. Since most are quite interested in the technique of synthetic division, even poor students have achieved complete success with this topic. A program which performs synthetic division is:

```
01.01 C PROGRAM FOR SYNTHETIC DIVISION
01.10 ASK "WHAT IS THE HIGHEST POWER OF X IN THE DIVIDEND ",N,!!
01.11 TYPE "INPUT COEFFICIENTS FOR EACH TERM"
01.12 FOR I=0,N; TYPE !"COEF. OF X↑",%2.0,N-I," TERM IS ";ASK C[I]
01.20 ASK !!"DIVISOR IS X MINUS ",D
01.30 SET A[0]=C[0]; FOR I=1,N; SET A[I]=C[I]+D*A[I-1]
01.40 TYPE !!"RESULT IS THE SUM OF THE TERMS:"
01.50 FOR I=0,N-1; TYPE !,%4.0,A[I],"*X↑",%2.0,N-1-I
01.60 IF (A[N]) 1.7,1.8,1.7
01.70 TYPE !"REMAINDER IS ",%4.0,A[N]
01.80 TYPE !!
*
```

The output of two runs of this program is:

```
*GO
WHAT IS THE HIGHEST POWER OF X IN THE DIVIDEND :3

INPUT COEFFICIENTS FOR EACH TERM
COEF. OF X↑=+ 3 TERM IS :1
COEF. OF X↑=+ 2 TERM IS :-6
COEF. OF X↑=+ 1 TERM IS :3
COEF. OF X↑=+ 0 TERM IS :10

DIVISOR IS X MINUS :2

RESULT IS THE SUM OF THE TERMS:
=+    1*X↑=+ 2
=-    4*X↑=+ 1
=-    5*X↑=+ 0
```

Problem is:

$$\frac{x^3-6x^2+3x+10}{x-2}$$

```
*GO
WHAT IS THE HIGHEST POWER OF X IN THE DIVIDEND :4

INPUT COEFFICIENTS FOR EACH TERM
COEF. OF X↑=+ 4 TERM IS :2
COEF. OF X↑=+ 3 TERM IS :3
COEF. OF X↑=+ 2 TERM IS :-4
COEF. OF X↑=+ 1 TERM IS :5
COEF. OF X↑=+ 0 TERM IS :6

DIVISOR IS X MINUS :-3

RESULT IS THE SUM OF THE TERMS:
=+    2*X↑=+ 3
=-    3*X↑=+ 2
=+    5*X↑=+ 1
=-   10*X↑=+ 0
REMAINDER IS =+   36

*
```

Problem is:

$$\frac{2x^4+3x^3-4x^2+5x+6}{x+3}$$

Although not a part of studying division of polynomials, students are often intrigued by the possibility of obtaining greater than six digit accuracy in the output. A good introduction to techniques for obtaining N place accuracy is an assignment to write a program which ASKs N, numerator (NU), and denominator (DE), then types N places of the decimal form of NU/DE. If this idea is new to the students, this assignment should be restricted to proper fractions expressed as NU/DE. Such a program is:

```
01.01 C PROGRAM TO CONVERT PROPER FRACTION TO DECIMAL FORM
01.02 C CONTAINING N DIGITS
01.10 ASK ?N ?!,"NUMERATOR IS ",NU,!"DENOMINATOR IS ",DE,!!
01.20 TYPE "RESULT IS  .",%1.0
01.30 FOR I=1,N; DO 2
01.40 TYPE !!!; QUIT

02.10 SET NU=NU*10; SET Q=FITR(NU/DE); TYPE Q; SET NU=NU-Q*DE
*
```

The output of two runs of this program is:

```
*GO
N :17
NUMERATOR IS :123
DENOMINATOR IS :1001

RESULT IS  .=+1=+2=+2=+8=+7=+7=+1=+2=+2=+8=+7=+7=+1=+2=+2=+8=+7


*GO
N :20
NUMERATOR IS :1
DENOMINATOR IS :17

RESULT IS  .=+0=+5=+8=+8=+2=+3=+5=+2=+9=+4=+1=+1=+7=+6=+4=+7=+0=+5=+8=+8


*
```

The techniques used in this program can also be used when the topic of repeating and terminating decimals is presented. Note that the computations of this program (step 2.10) involve little more than retaining the integral remainder after performing the operation of division, and this procedure should be familiar to many students.

An instructive problem to follow the writing of a program which converts proper fractions to decimal form is the writing of a program to convert proper and improper fractions to decimal form. The first problem might be skipped with better students, but average students will have more success if it is used as the initial step. A program which correctly converts both proper and improper fractions is:

```
01.01 C PROGRAM TO CONVERT PROPER OR IMPROPER FRACTION TO
01.02 C DECIMAL FORM CONTAINING N DIGITS
01.10 ASK ?N ?!,"NUMERATOR IS ",NU,!"DENOMINATOR IS ",DE,!!
01.12 TYPE "RESULT IS ",%1.0; SET E=-1
01.13 SET E=E+1; IF (10↑E-NU) 1.13, 1.13; FOR I=1,E; DO 3
01.20 TYPE " . "; FOR I=1,N-E; DO 2
01.40 TYPE !!!; QUIT

02.10 SET NU=NU*10; SET Q=FITR(NU/DE); TYPE Q; SET NU=NU-Q*DE

03.10 SET NN=NU/10↑FABS(E-I); SET Q=FITR(NN/DE); TYPE Q
03.20 SET NU=NU-Q*DE*10↑FABS(E-I)
*
```

The output of two runs of this program is:

```
*GO
N :12
NUMERATOR IS :1234
DENOMINATOR IS :7

RESULT IS =+0=+1=+7=+6 . =+2=+8=+5=+7=+1=+4=+2=+8


*GO
N :16
NUMERATOR IS :3
DENOMINATOR IS :17

RESULT IS =+0 . =+1=+7=+6=+4=+7=+0=+5=+8=+8=+2=+3=+5=+2=+9=+4


*
```

Note that this program is quite similar to the preceding one. The major difference is the inclusion of part 3 and step 1.13. Step 1.13 determines the order of magnitude of the numerator and part 3 determines the portion of the quotient which lies to the left of the decimal point.

Although the program already shown satisfies the requirements of the assignment, there are two drawbacks to its use that will concern many students. First, only 19 digits of a quotient can be typed without the occurrence of overprinting in the last print postion. Second, the algorithm counts leading zeroes to the left of the decimal point as part of the N digits to be typed out. The majority of students will be more successful if they first write a working program without worrying about these problems, and then modify this program to eliminate the problems. A modification of the previous example which eliminates both of these handicaps is:

```
01.01 C PROGRAM TO CONVERT PROPER OR IMPROPER FRACTION TO
01.02 C DECIMAL FORM CONTAINING N DIGITS (WITH MODIFIED TYPE FORMAT)
01.10 ASK ?N ?!,"NUMERATOR IS ",NU,!"DENOMINATOR IS ",DE,!!
01.12 TYPE "RESULT IS "!,%1.0; SET E=-1; SET C=0; SET SIG=0
01.13 SET E=E+1; IF (10↑E-NU) 1.13, 1.13; FOR I=1,E; DO 3
01.20 TYPE " . "; SET C=C+1;  FOR I=1,N+SIG; DO 2
01.40 TYPE !!!; QUIT

02.10 SET NU=NU*10; SET Q=FITR(NU/DE); TYPE Q; SET NU=NU-Q*DE; DO 4

03.10 SET NN=NU/10↑FABS(E-I); SET Q=FITR(NN/DE); IF (Q) 3.2,3.15,3.2
03.15 IF (SIG) 3.2; RETURN
03.20 SET SIG=SIG-1; TYPE Q; SET NU=NU-Q*DE*10↑FABS(E-I); DO 4

04.10 SET C=C+1; IF (FITR(C/20)-C/20) 4.2; TYPE !
04.20 CONTINUE
*
```

The output of two runs of this program is:

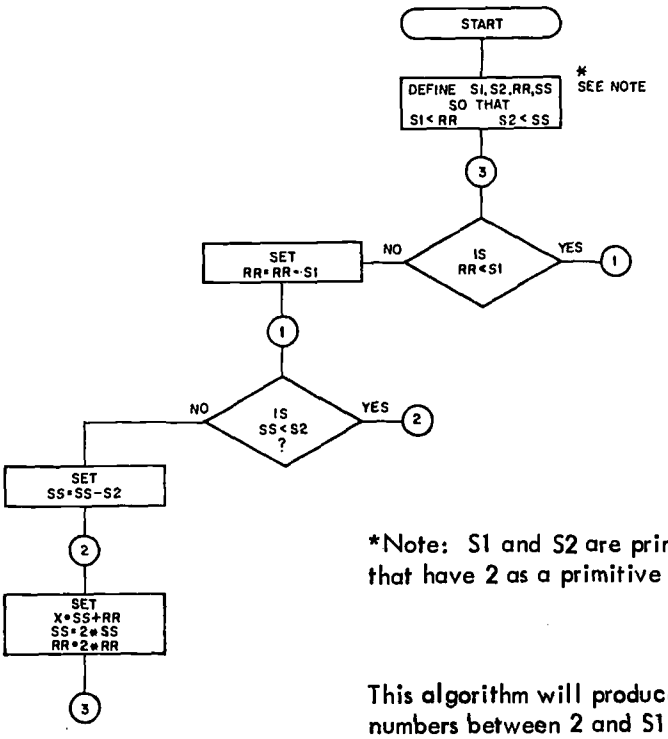
```
*GO
N :48
NUMERATOR IS :3
DENOMINATOR IS :17

RESULT IS
 . =+1=+7=+6=+4=+7=+0=+5=+8=+8=+2=+3=+5=+2=+9=+4=+1=+1=+7=+6
=+4=+7=+0=+5=+8=+8=+2=+3=+5=+2=+9=+4=+1=+1=+7=+6=+4=+7=+0=+5
=+8=+8=+2=+3=+5=+2=+9=+4=+1 <-------- There is no upper limit to the
                                      number of digits that can be typed.

*GO
N :12
NUMERATOR IS :923E+6
DENOMINATOR IS :18004

RESULT IS
=+5=+1=+2=+6=+6 . =+4=+2=+0=+7=+9=+5=+3
        <-- leading zeroes have been suppressed
*
```

In this program, part 4 is used to cause a carriage return and line feed after each 20 digits of the quotient have been typed. The variable SIG is used as a flag to indicate when one or more non-zero digits have been typed to the left of the decimal point. If students are not asked to write a program incorporating these modifications, the teacher might discuss methods for including them, since both methods involve programming techniques that can be used in many other situations as well.

D. Pascal's Triangle Using Random Numbers (Supplementary)

The use of random numbers is a topic that seems to fascinate students at all ability levels. Although the FRAN ( ) function has been used in several programs, none of the applications previously discussed have done any more than select a few random numbers to be used as coefficients or exponents in an algebraic expression. FOCAL's random function is satisfactory for the applications shown so far, but when statistical data is to be derived using random numbers, the FOCAL function is not satisfactory. This means that students must first write (if time permits) or be given a program to generate pseudo-random numbers.

There are several methods for generating pseudo-random numbers. The text Mathematics and Computing with FORTRAN Programming; by Dorn and Greenberg; John Wiley & Sons, 1967, presents several techniques, all suitable for presentation to high school students. One method for generating pseudo-random numbers that works very well in statistical applications is that shown in the following flow chart. (This method is discussed in the Dorn and Greenberg text.)



*Note: S1 and S2 are prime numbers that have 2 as a primitive root.

This algorithm will produce random numbers between 2 and S1 + S2 - 2 with a cycle length that is the least common multiple of S1 - 1 and S2 - 1.

The following program is written according to this algorithm. The FOR statement is used simply to cause the program to generate 10 numbers.

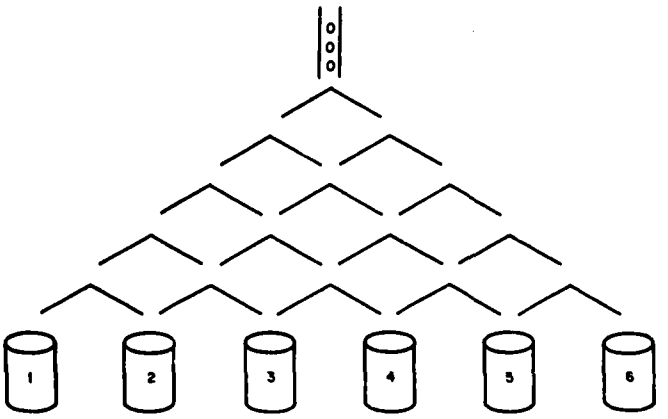
```
01.10 SET S1=2005; SET S2=2003; SET RR=800; SET SS=801
01.20 FOR I=1,10; DO 2
01.30 QUIT

02.10 IF (RR-S1) 2.2; SET RR=RR-S1
02.20 IF (SS-S2) 2.3; SET SS=SS-S2
02.30 SET X=SS+RR; SET SS=2*SS; SET RR=2*RR
02.40 TYPE %5.0,X,!
*
```

```
*GO
=+ 1601
=+ 3202
=+ 2396
=+  784
=+ 1568
=+ 3136
=+ 2264
=+  520
=+ 1040
=+ 2080
*
```

Using these values for S1, S2, RR, and SS, the program generates pseudo-random integers in the interval 2 through 4006 with a cycle length of 2,006,004.

A problem that has proven very successful when used as an assignment for students is:

Write a program which will simulate the dropping of N balls through the following array. Assume that each ball is equally likely to fall to the left or right at each point in the array.



Your program should count the number of balls that land in each of the P baskets (P = 6 in the diagram) and then typeout the number in each basket after every TO balls have dropped.

A student written solution to this problem is:

```
01.10 ASK "NUMBER OF PILES ",P,!"NUMBER OF BALLS TO BE DROPPED ",N
01.15 ASK !"PRINT TOTAL EVERY ",TO," BALLS."!
01.20 FOR I=1,P; SET P[I]=0
01.30 FOR C=1,N; DO 2
01.40 TYPE !!; QUIT

02.10 SET POT=1; FOR B=1,P-1; DO 3
02.20 SET P[POT]=P[POT]+1
02.30 IF (FITR(C/TO)-C/TO) 2.4; TYPE !,%5.0,C; FOR I=1,P; TYPE P[I]
02.40 CONTINUE

03.10 DO 10; SET X=FITR(X/10); IF (FITR(X/2)-X/2) 3.2; SET POT=POT+1
03.20 CONTINUE

10.10 IF (S1) 10.3,10.2,10.3
10.20 SET S1=2005; SET S2=2003; SET RR=800; SET SS=801
10.30 IF (RR-S1) 10.4; SET RR=RR-S1
10.40 IF (SS-S2) 10.5; SET SS=SS-S2
10.50 SET X=SS+RR; SET SS=2*SS; SET RR=2*RR
*
```

Note that part 10 is the pseudo-random number generator already discussed. Part 10 can be used in this form to generate a random number x in any program. For this application, the 10 s position of the random number is checked for evenness to determine whether the ball drops left or right. If used in this way, good statistical results are achieved. The output of a run of this program is:

```
*GO
NUMBER OF PILES :6
NUMBER OF BALLS TO BE DROPPED :5000
PRINT TOTAL EVERY :100  BALLS.

=+  100=+    4=+   31=+   20=+   28=+   13=+    4
=+  200=+   10=+   44=+   55=+   58=+   25=+    8
=+  300=+   18=+   58=+   86=+   88=+   40=+   10
=+  400=+   24=+   79=+  115=+  113=+   56=+   13
=+  500=+   27=+  101=+  143=+  141=+   70=+   18
=+  600=+   29=+  112=+  183=+  165=+   89=+   22
=+  700=+   34=+  132=+  225=+  181=+  101=+   27
=+  800=+   34=+  147=+  261=+  215=+  115=+   28
=+  900=+   38=+  162=+  295=+  240=+  135=+   30
=+ 1000=+   43=+  178=+  322=+  274=+  152=+   31
=+ 1100=+   47=+  191=+  349=+  308=+  171=+   34
=+ 1200=+   51=+  205=+  384=+  335=+  186=+   39
=+ 1300=+   56=+  223=+  407=+  370=+  202=+   42
=+ 1400=+   59=+  245=+  437=+  398=+  216=+   45
=+ 1500=+   61=+  257=+  466=+  437=+  232=+   47
=+ 1600=+   64=+  276=+  500=+  463=+  243=+   54
=+ 1700=+   67=+  292=+  527=+  497=+  259=+   58
=+ 1800=+   69=+  312=+  554=+  526=+  279=+   60
=+ 1900=+   75=+  324=+  593=+  552=+  291=+   65
=+ 2000=+   79=+  339=+  617=+  590=+  308=+   67
```

```
=+ 2100=+   81=+  354=+  645=+  617=+  332=+   71
=+ 2200=+   85=+  370=+  671=+  651=+  348=+   75
=+ 2300=+   88=+  388=+  697=+  688=+  362=+   77
=+ 2400=+   95=+  406=+  726=+  717=+  377=+   79
=+ 2500=+   96=+  425=+  757=+  750=+  390=+   82
=+ 2600=+   98=+  448=+  786=+  783=+  400=+   85
=+ 2700=+  101=+  461=+  821=+  815=+  412=+   90
=+ 2800=+  103=+  475=+  854=+  845=+  428=+   95
=+ 2900=+  104=+  493=+  884=+  874=+  445=+  100
=+ 3000=+  108=+  508=+  915=+  904=+  464=+  101
=+ 3100=+  112=+  522=+  944=+  941=+  478=+  103
=+ 3200=+  118=+  540=+  969=+  975=+  495=+  103
=+ 3300=+  120=+  556=+ 1003=+ 1004=+  513=+  104
=+ 3400=+  121=+  566=+ 1043=+ 1036=+  522=+  112
=+ 3500=+  125=+  575=+ 1077=+ 1070=+  541=+  112
=+ 3600=+  127=+  594=+ 1112=+ 1099=+  554=+  114
=+ 3700=+  130=+  606=+ 1147=+ 1133=+  567=+  117
=+ 3800=+  134=+  619=+ 1177=+ 1163=+  584=+  123
=+ 3900=+  139=+  630=+ 1214=+ 1188=+  604=+  125
=+ 4000=+  144=+  646=+ 1242=+ 1219=+  620=+  129
=+ 4100=+  146=+  660=+ 1267=+ 1254=+  638=+  135
=+ 4200=+  147=+  678=+ 1294=+ 1292=+  651=+  138
=+ 4300=+  147=+  694=+ 1323=+ 1328=+  667=+  141
=+ 4400=+  148=+  712=+ 1362=+ 1354=+  681=+  143
=+ 4500=+  151=+  736=+ 1389=+ 1379=+  699=+  146
=+ 4600=+  152=+  751=+ 1423=+ 1409=+  716=+  149
=+ 4700=+  153=+  769=+ 1455=+ 1441=+  730=+  152
=+ 4800=+  156=+  793=+ 1481=+ 1470=+  745=+  155
=+ 4900=+  159=+  805=+ 1510=+ 1503=+  767=+  156
=+ 5000=+  161=+  823=+ 1534=+ 1537=+  781=+  164

*
```

The output of this run compares quite favorably to the expected theoretical results. The fifth line of Pascal's triangle is 1 5 10 10 5 1. This means the six numbers have the ratio $\frac{1}{32}$ $\frac{5}{32}$ $\frac{10}{32}$ $\frac{10}{32}$ $\frac{5}{32}$ $\frac{1}{32}$. Comparing these ratios to the output yields:

```
*TYPE %,1/32,!,161/5000,!,164/5000,!!
=+0.312500E-01
=+0.322000E-01
=+0.328000E-01

*TYPE 5/32,!,823/5000,!,781/5000,!!
=+0.156250E+00
=+0.164600E+00
=+0.156200E+00

*TYPE 10/32,!,1534/5000,!,1537/5000,!!
=+0.312500E+00
=+0.306800E+00
=+0.307400E+00

*
```

The output of another run of this program is:

```
*GO
NUMBER OF PILES :4
NUMBER OF BALLS TO BE DROPPED :5000
PRINT TOTAL EVERY :500  BALLS.

=+  500=+   82=+  194=+  163=+   61
=+ 1000=+  149=+  394=+  326=+  131
=+ 1500=+  221=+  587=+  502=+  190
=+ 2000=+  278=+  777=+  690=+  255
=+ 2500=+  345=+  973=+  860=+  322
=+ 3000=+  415=+ 1156=+ 1036=+  393
=+ 3500=+  476=+ 1336=+ 1225=+  463
=+ 4000=+  533=+ 1539=+ 1410=+  518
=+ 4500=+  596=+ 1732=+ 1603=+  569
=+ 5000=+  651=+ 1919=+ 1797=+  633

*
```

And again the results compare favorably to the expected theoretical results.

```
*TYPE %,1/8,!,651/5000,!,633/5000,!!
=+0.125000E+00
=+0.130200E+00
=+0.126600E+00

*TYPE 3/8,!,1919/5000,!,1797/5000,!!
=+0.375000E+00
=+0.383800E+00
=+0.359400E+00

*
```

There is, however, one obstacle to extensive use of this assignment. To run a program a sufficient number of times to obtain acceptable results requires a great deal of computer time. An approach that has been successfully used is to ask each student to write his own program, then choose two or three of the most efficient student programs to obtain data for the entire class to use in developing Pascal's triangle.

VII FACTORING

A. Prime Numbers and Factors of Integers

There are many more program examples in this section than time would permit using in a conventional Algebra I or II curriculum. So many examples are included because, although these programs require very little new mathematics, they do require a wide range of programming techniques and are quite useful when teaching programming early in the course (Algebra I, II, or III). Students can concentrate on learning the programming skills being presented without also having to learn new mathematics.

Certainly, the task of testing one integer for divisibility by another occurs in many different problems. Asking students to write a program such as this allows them to first "discover" the algorithm that will later become second nature to them. For example, consider this solution:

```
Ø1.Ø1 C PROGRAM TO TELL WHETHER OR NOT  N  IS A FACTOR OF  D .
Ø1.1Ø ASK  ?N ?,?  D ?!
Ø1.2Ø IF  (FITR(D/N)-D/N)  1.4,1.3,1.4
Ø1.3Ø TYPE %5.Ø,?D/N ?,!!; QUIT
Ø1.4Ø TYPE "N IS NOT A FACTOR OF D."!!
*
```

The output of two runs of this program is:

```
*GO
N :17   D:12Ø9
N IS NOT A FACTOR OF D.

*GO
N :13   D :1ØØ1
D/N =+   77

*
```

An alternate way of presenting this problem is to ask students to write a program which will tell whether an ASKed number is even or odd. Since the solution algorithm is identical, slower students might benefit if the teacher presents this alternate program in class and then assigns the more general case as a homework problem.

Although a program for computing the factorial of an ASKed number is slightly misplaced in this section on factoring, the following program is included because it is a good student assignment when FOR statements are first introduced.

```
Ø1.Ø1 C PROGRAM TO COMPUTE N FACTORIAL
Ø1.1Ø ASK N
Ø1.2Ø SET ANS=1; FOR I=1,N; SET ANS=ANS*I
Ø1.3Ø TYPE "! ",%,ANS,!!
*


*GO
:5  ! =+Ø.12ØØØØE+Ø3

*GO
:Ø ! =+Ø.1ØØØØØE+Ø1

*GO
:25Ø ! =+Ø.323279E+493

*
```

This program correctly computes factorials for positive integers but does not reject improper input values. Since students should be taught to anticipate all cases, applying corrective modifications to a program such as this is a good exercise. A modified form is:

```
Ø1.Ø1 C PROGRAM TO COMPUTE N FACTORIAL
Ø1.1Ø ASK N
Ø1.14 IF (N) 1.4, 1.2, 1.18
Ø1.18 IF (FITR(N)-N)1.4, 1.2, 1.4
Ø1.2Ø SET ANS=1; FOR I=1,N; SET ANS=ANS*I
Ø1.3Ø TYPE "! ",%,ANS,!!; QUIT
Ø1.4Ø TYPE "  HAS NO DEFINED FACTORIAL."!!
*
```

The outputs of three runs of this program are:

```
*GO
:4.32   HAS NO DEFINED FACTORIAL.

*GO
:-8   HAS NO DEFINED FACTORIAL.

*GO
:1ØØ ! =+Ø.933257E+158

*
```

A program which determines all factors of an ASKed number is useful, first, when teaching techniques of programming, and later, as a part of programs for factoring polynomials. Such a program can also be used with elementary students to help introduce the notion of a prime number. One such program is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE FACTORS OF N
Ø1.1Ø ASK ?N ?!"FACTORS OF N ARE:",%4.0
Ø1.2Ø FOR I=1,N; DO 2
Ø1.3Ø TYPE !!; QUIT

Ø2.1Ø IF (FITR(N/I)-N/I) 2.3,2.2,2.3
Ø2.2Ø TYPE !,I
Ø2.3Ø CONTINUE
*
```

The outputs of two runs of this program are:

```
*GO
N :23
FACTORS OF N ARE:
=+   1
=+  23

*GO
N :24
FACTORS OF N ARE:
=+   1
=+   2
=+   3
=+   4
=+   6
=+   8
=+  12
=+  24

*
```

Although this program works for all positive integers, it does so with little efficiency, as the algorithm simply tries each integer 1 through N as a divisor. Many students will quickly realize that other than N itself, $\sqrt{N}$ is the largest possible divisor. Encouraging students to write this program as efficiently as possible often results in their learning more mathematics than anticipated. One possible program is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE FACTORS OF N
Ø1.1Ø ASK ?N ?!"FACTORS OF N ARE:",%4.Ø
Ø1.2Ø SET E=FSQT(N); FOR I=1,E; DO 2
Ø1.3Ø TYPE !!; QUIT

Ø2.1Ø IF (FITR(N/I)-N/I) 2.3,2.2,2.3
Ø2.2Ø TYPE !,I,"   ",N/I
Ø2.3Ø CONTINUE
*
```

Note that this program is identical to the previous example with the exception of changes in lines 1.2 and 2.2.

The output of a run of this program is:

```
*GO
N :144
FACTORS OF N ARE:
=+    1    =+ 144
=+    2    =+  72
=+    3    =+  48
=+    4    =+  36
=+    6    =+  24
=+    8    =+  18
=+    9    =+  16
=+   12    =+  12

*
```

One word of caution regarding asking students to write programs as efficiently as possible. Although this request is one which most student enjoy fulfilling and one that is quite useful when programming is first introduced, there is no reason for continuing to request more than reasonable efficiency. Since the main purpose of using computer facilities is to better teach mathematics, the students' goal should be to create a mathematically and logically valid algorithm. Often, little additional mathematics is learned when students are asked to create the most efficient algorithm possible, because such algorithms often depend on programming tricks rather than on additional mathematics.

The problem of writing a program to type the prime factors of an ASKed number is useful for introducing both the notion of prime numbers and the techniques of programming. The following program follows an algorithm quite similar to that shown in the previous example for finding all factors.

```
Ø1.Ø1 C PROGRAM TO FIND PRIME FACTORS OF N
Ø1.1Ø ASK ?N ?!"PRIME FACTORS OF N ARE:";TYPE %4.Ø
Ø1.2Ø SET I=2; DO 2
Ø1.3Ø FOR I=3,2,N; DO 2
Ø1.4Ø TYPE !!; QUIT

Ø2.1Ø IF (FITR(N/I)-N/I) 2.3,2.2,2.3
Ø2.2Ø SET N=N/I; TYPE !,I; GOTO 2.1
Ø2.3Ø IF (N-1) 1.4, 1.4; CONTINUE
```

The outputs of two runs of this program are:

```
*GO
N :196Ø
PRIME FACTORS OF N ARE:
=+    2
=+    2
=+    2
=+    5
=+    7
=+    7
```

```
*GO
N :31
PRIME FACTORS OF N ARE:
=+  31

*
```

Note that this algorithm determines the prime factors without previously knowing which numbers are prime. In the sample runs, the divisors used are 2, then all odd integers until all factors are found. Although the program would be even less efficient, all positive integers greater than 1 could be used as trial divisors by students totally unfamiliar with prime numbers. By properly choosing values of N for students to use, the program could be used to develop an intuitive notion as to which numbers are prime and why. This program might be used also to demonstrate the problem in determining the prime factors of zero:

```
*GO
N :Ø
PRIME FACTORS OF N ARE:
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+    2
=+@?Ø1.ØØ
*
```

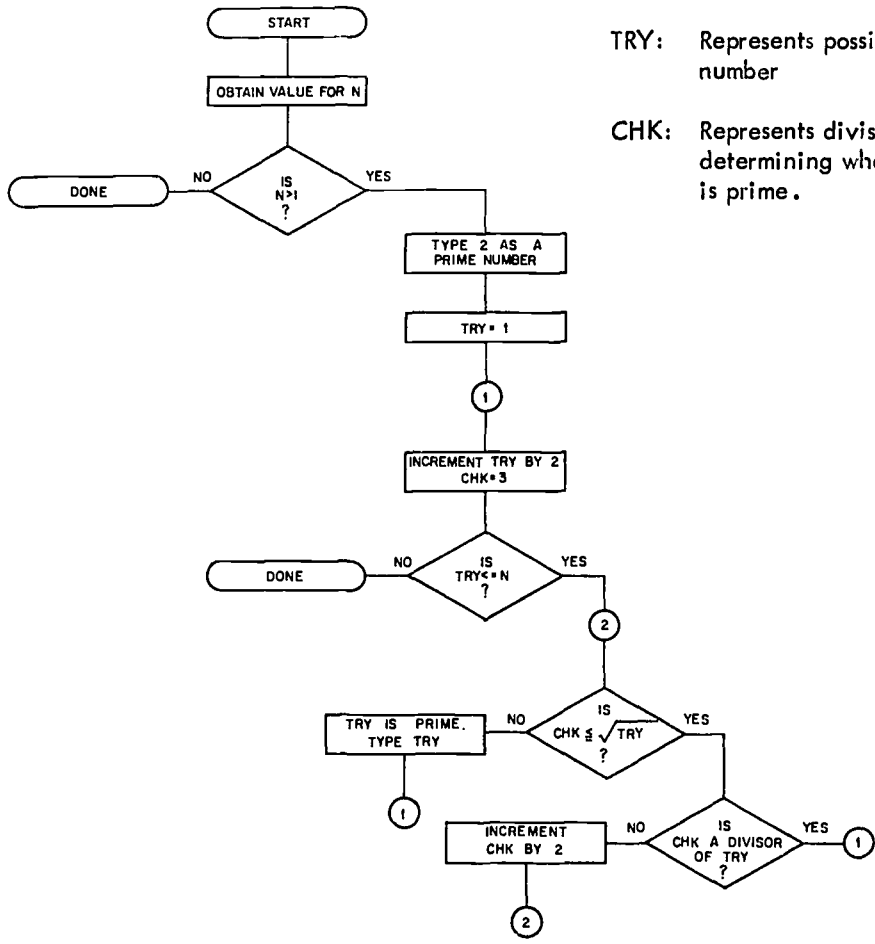
Better students can be given the slightly more difficult problem of having each factor typed only once with the appropriate power. However, since only the programming is more difficult, with little additional mathematics to be learned, the problem is only valuable when teaching programming techniques. A program which does this is:

```
Ø1.Ø1 C PROGRAM TO FIND PRIME FACTORS OF N
Ø1.1Ø ASK ?N ?!"PRIME FACTORS OF N ARE:"
Ø1.2Ø SET I=2; DO 2
Ø1.3Ø FOR I=3,2,N; DO 2
Ø1.4Ø TYPE !!; QUIT

Ø2.Ø5 SET C=Ø
Ø2.1Ø IF (FITR(N/I)-N/I) 2.3,2.2,2.3
Ø2.2Ø SET N=N/I; SET C=C+1; GOTO 2.1
Ø2.3Ø IF (C) 2.5,2.5; TYPE %4.Ø,!,I; IF (C-1) 2.5,2.5,2.4
Ø2.4Ø TYPE|" ↑ ",%2.Ø,C
Ø2.5Ø IF (N-1) 1.4,1.4; CONTINUE
*
```

The outputs of three runs of this program are:

```
*GO
N :196Ø
PRIME FACTORS OF N ARE:
=+   2 ↑ =+  3
=+   5
=+   7 ↑ =+  2

*GO
N :2Ø48
PRIME FACTORS OF N ARE:
=+   2 ↑ =+11

*GO
N :31
PRIME FACTORS OF N ARE:
=+  31

*
```

The task of writing of a problem to generate prime numbers less than or equal to an ASKed N should be given to all students, as the algorithm they develop can be used later in many other programs. The following algorithm is typical of those written by students in Algebra I or II:



TRY: Represents possible prime number

CHK: Represents divisors used in determining whether TRY is prime.

A program using this algorithm is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE PRIMES <=N.
Ø1.1Ø ASK ?N ?!"PRIMES <= N ARE:"; IF (1-N) 1.2; TYPE !!!; QUIT
Ø1.2Ø TYPE %4.0,1,2; SET TRY=1
Ø1.3Ø SET TRY=TRY+2; SET CHK=3; IF (TRY-N) 1.4,1.4; TYPE !!!; QUIT
01.40 IF (CHK-FSQT(TRY)) 1.5,1.5; TYPE !,TRY; GOTO 1.3
01.50 IF (TRY/CHK-FITR(TRY/CHK)) 1.3,1.3; SET CHK=CHK+2; GOTO 1.4
*
```

The output of a run of this program is:

```
*GO
N :5Ø
PRIMES <= N ARE:
=+    2
=+    3
=+    5
=+    7
=+   11
=+   13
=+   17
```

```
=+  19
=+  23
=+  29
=+  31
=+  37
=+  41
=+  43
=+  47

*
```

If time permits, students can be asked to modify their prime generating program so that it generates twin primes or prime triplets. Experience with this assignment shows that it often motivates many students to pursue several areas of number theory with and without computer assistance. A modification of the previous program which types only twin primes is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE TWIN PRIMES <= N.
Ø1.1Ø ASK ?% ?!"TWIN PRIMES <= N ARE:"; IF (1-N) 1.2; TYPE !!!; QUIT
Ø1.2Ø SET TRY=1; SET LTRY=2; GOTO 1.3
Ø1.25 SET LTRY=TRY
Ø1.3Ø SET TRY=TRY+2; SET CHK=3; IF (TRY-N) 1.4,1.4; TYPE !!!; QUIT
Ø1.4Ø IF (CHK-FSQT(TRY)) 1.5,1.5; IF (LTRY+2-TRY) 1.25,1.45,1.25
Ø1.45 TYPE %4.Ø,!,LTRY,"  ",TRY; GOTO 1.25
Ø1.5Ø IF (TRY/CHK-FITR(TRY/CHK)) 1.3,1.3; SET CHK=CHK+2; GOTO 1.4
*
```

The output of a run of this program is:

```
*GO
N :500
TWIN PRIMES <= N ARE:
=+   3    =+   5
=+   5    =+   7
=+  11    =+  13
=+  17    =+  19
=+  29    =+  31
=+  41    =+  43
=+  59    =+  61
=+  71    =+  73
=+ 1Ø1    =+ 1Ø3
=+ 1Ø7    =+ 1Ø9
=+ 137    =+ 139
=+ 149    =+ 151
=+ 167    =+ 169
=+ 179    =+ 181
=+ 191    =+ 193
=+ 197    =+ 199
=+ 227    =+ 229
=+ 239    =+ 241
=+ 269    =+ 271
=+ 281    =+ 283
=+ 311    =+ 313
```

Note: After running their programs for prime triplets, many students will be surprised that only 1 set of triplets is found. Asking students to prove that there are no other triplets is often a productive assignment. Although not all students will be successful in proving the desired theorem, most will discover other useful ideas in the process.

```
=+ 347    =+ 349
=+ 419    =+ 421
=+ 431    =+ 433
=+ 461    =+ 463

*
```

The program on page 115 correctly computes, but does not remember, all primes less than an ASKed N. Since many applications will require a series of primes to be known, the following program is a modification which will remember the primes computed. Note that this program can also be used as an early exercise involving subscripts for students learning to program.

```
Ø1.Ø1 C PROGRAM TO DETERMINE AND REMEMBER PRIMES <=N
Ø1.1Ø ASK ?N ?!"PRIMES <= N ARE:"; IF (1-N) 1.2; TYPE !!!; QUIT
Ø1.2Ø SET I=1; SET P[I]=2; SET TRY=1; TYPE %4.0,!,2
Ø1.3Ø SET TRY=TRY+2; SET J=1; IF (TRY-N) 1.4,1.4; TYPE !!!; QUIT
Ø1.4Ø IF (P[J]-FSQT(TRY)) 1.5,1.5; S I=I+1; S P[I]=TRY;T !,TRY;GOTO 1.3
Ø1.5Ø IF (TRY/P[J]-FITR(TRY/P[J])) 1.2,1.3; SET J=J+1; GOTO 1.4
*
```

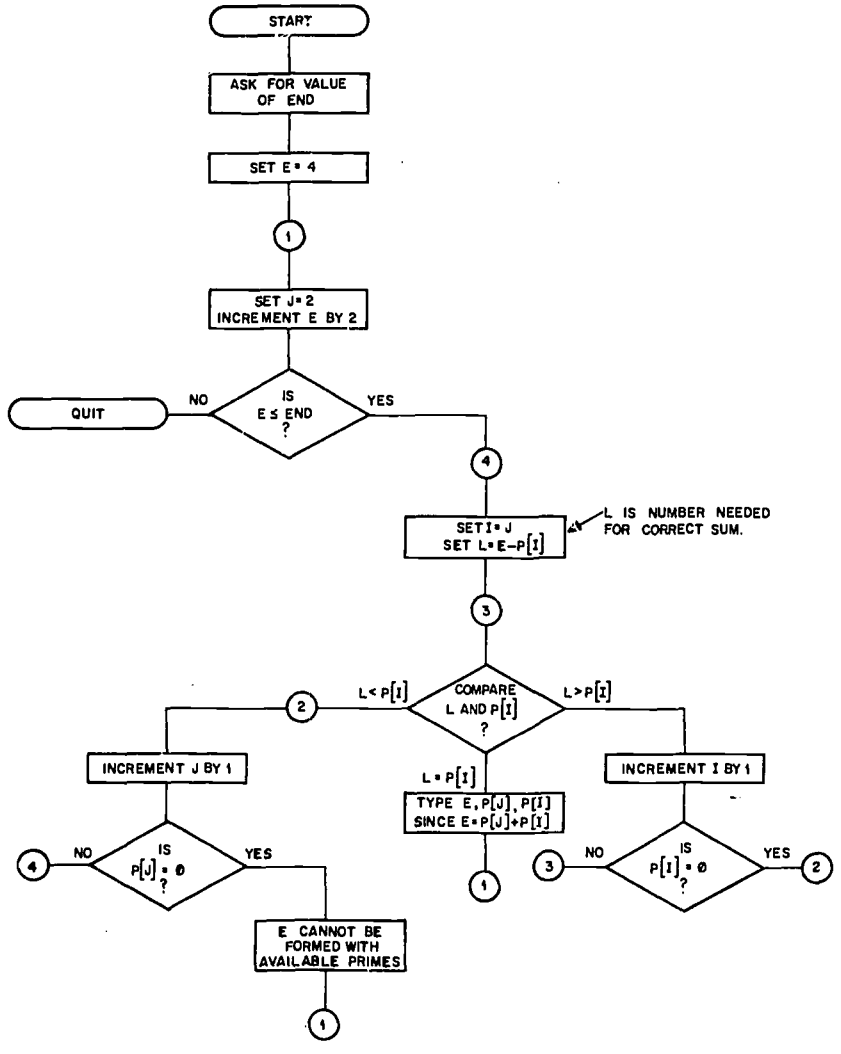
```
*GO
N :53Ø
PRIMES <= N ARE:
=+    2
=+    3
=+    5
=+    7
=+   11
=+   13
=+   17
     .
     .
     .
=+  499
=+  5Ø3
=+  5Ø9
=+  521
=+  523
=+  529

*
```

The program, as shown, will compute and remember up to the first 101 prime numbers (N=530) using 4K FOCAL. This limit does not exist if more memory is available.

Note that this program also uses the primes it computes as trial divisors in determining additional primes. This technique greatly increases the efficiency of the program, and students should be encouraged to develop such an algorithm.

The conjecture that all even numbers greater than 4 can be represented as the sum of two prime numbers appeals to many students, provides an excellent introduction to techniques for writing one program which will utilize the results of another, and provides further work with subscripts. Following

is a flowchart which represents an algorithm for determining which two prime numbers from an arbitrarily long list of primes (Pi) have a sum that equals each of the even numbers 6 through an ASKed END.



A program which follows this flowchart is:

```
Ø2.1Ø ASK ?END ?!; SET E=4
Ø2.2Ø SET E=E+2; SET J=2; IF (E-END) 2.3,2.3; TYPE !!; QUIT
Ø2.3Ø SET I=J; SET L=E-P[I]
Ø2.4Ø IF (L-P[I]) 2.5,2.6,2.7
Ø2.5Ø SET J=J+1; IF (P[J]) 2.3,2.8,2.3
Ø2.6Ø TYPE %4.Ø,E,"     ",P[J],"   ",P[I],!;  GOTO 2.2
Ø2.7Ø SET I=I+1; IF (P[I]) 2.4,2.5,2.4
Ø2.8Ø TYPE %4.Ø,E,"  CAN'T BE FORMED WITH PRIMES CHECKED."!; GOTO 2.2
*
```

Note that the program is written as Part 2 so that the prime generating program shown on page 117 can also be used.

To run this program, one first DOes Part 1, the prime generating program, to obtain the desired list of primes, then DOes Part 2 to form the even number as the sum of two primes.

The output of such a run is:

```
*DO 1
N :1ØØ
PRIMES <= N ARE:
=+    2
=+    3
=+    5
=+    7
=+   11
=+   13
=+   17
=+   19
=+   23
=+   29
=+   31
=+   37
=+   41
=+   43
=+   47
=+   53
=+   59
=+   61
=+   67
=+   71
=+   73
=+   79
=+   83
=+   89
=+   97
```

Part 1 generates the primes which are then used in Part 2.

```
*DO 2
END :1ØØ
=+    6     =+    3     =+    3
=+    8     =+    3     =+    5
=+   1Ø     =+    3     =+    7
=+   12     =+    5     =+    7
=+   14     =+    3     =+   11
=+   16     =+    3     =+   13
=+   18     =+    5     =+   13
=+   2Ø     =+    3     =+   17
=+   22     =+    3     =+   19
=+   24     =+    5     =+   19
=+   26     =+    3     =+   23
=+   28     =+    5     =+   23
=+   3Ø     =+    7     =+   23
=+   32     =+    3     =+   29
=+   34     =+    3     =+   31
=+   36     =+    5     =+   31
=+   38     =+    7     =+   31
=+   4Ø     =+    3     =+   37
=+   42     =+    5     =+   37
=+   44     =+    3     =+   41
=+   46     =+    3     =+   43
=+   48     =+    5     =+   43
=+   5Ø     =+    3     =+   47
=+   52     =+    5     =+   47
=+   54     =+    7     =+   47
=+   56     =+    3     =+   53
=+   58     =+    5     =+   53
=+   6Ø     =+    7     =+   53
=+   62     =+    3     =+   59
=+   64     =+    3     =+   61
=+   66     =+    5     =+   61
=+   68     =+    7     =+   61
=+   7Ø     =+    3     =+   67
=+   72     =+    5     =+   67
=+   74     =+    3     =+   71
=+   76     =+    3     =+   73
=+   78     =+    5     =+   73
=+   8Ø     =+    7     =+   73
=+   82     =+    3     =+   79
=+   84     =+    5     =+   79
=+   86     =+    3     =+   83
=+   88     =+    5     =+   83
=+   9Ø     =+    7     =+   83
=+   92     =+    3     =+   89
=+   94     =+    5     =+   89
=+   96     =+    7     =+   89
=+   98     =+   19     =+   79
=+  1ØØ     =+    3     =+   97

*
```

A more interesting output can be obtained if a shorter list of primes is used. For example:

```
*DO  1
N :24
PRIMES <= N ARE:          Part 1 is used to generate a shorter list of primes.
=+    2
=+    3
=+    5
=+    7
=+   11
=+   13
=+   17
=+   19
=+   23


*DO 2
END :5Ø
=+    6     =+    3     =+    3
=+    8     =+    3     =+    5
=+   1Ø     =+    3     =+    7
=+   12     =+    5     =+    7
=+   14     =+    3     =+   11
=+   16     =+    3     =+   13
=+   18     =+    5     =+   13
=+   2Ø     =+    3     =+   17
=+   22     =+    3     =+   19
=+   24     =+    5     =+   19
=+   26     =+    3     =+   23
=+   28     =+    5     =+   23
=+   3Ø     =+    7     =+   23
=+   32     =+   13     =+   19
=+   34     =+   11     =+   23
=+   36     =+   13     =+   23
=+   38     =+   19     =+   19
=+   4Ø     =+   17     =+   23
=+   42     =+   19     =+   23
=+   44   CAN'T BE FORMED WITH PRIMES CHECKED.
=+   46     =+   23     =+   23
=+   48   CAN'T BE FORMED WITH PRIMES CHECKED.
=+   5Ø   CAN'T BE FORMED WITH PRIMES CHECKED.

*
```

A very good assignment for use when teaching the concepts of programming and/or flowcharting is that of writing a program which will tell whether or not an ASKed number N is prime. Such a program, although brief, involves: the elimination of improper input; the use of functions; use of loops and branching; and usually more than one part. Typical of student written programs which accomplish this task is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE IF GIVEN NUMBER IS PRIME
Ø1.1Ø ASK N
Ø1.15 IF (N-2) 2.2; IF (FITR(N)-N) 2.2,1.18,2.2
Ø1.18 IF (3-N) 1.2,1.4,1.4
Ø1.2Ø SET D=2; DO 2
Ø1.3Ø SET E=FSQT(N); FOR D=3,2,E; DO 2
Ø1.4Ø TYPE " IS PRIME."!!; QUIT

Ø2.1Ø IF (N/D-FITR(N/D)) 2.3,2.2,2.3
Ø2.2Ø TYPE " IS NOT PRIME."!!; QUIT
Ø2.3Ø CONTINUE
*
```

The outputs of two runs of this program are:

```
*GO
:97  IS PRIME.

*GO
:1003  IS NOT PRIME.

*
```

Note that this program will properly identify negative numbers, fractions, and all positive integers 1 through 2047. Since this restriction is imposed by the restriction placed on FOCAL's FITR function, students should be encouraged to "discover" a method for eliminating this problem. A very simple technique for doing this is to repeatedly subtract 2047 from the number to be used in the FITR function until the number lies in the range 0 th ough 2047. A modification of the previous program which reduces all arguments of the FITR function to an acceptable magnitude is:

```
Ø1.Ø1 C PROGRAM TO DETERMINE IF GIVEN NUMBER IS PRIME
Ø1.1Ø ASK N
Ø1.15 IF (N-2) 2.2; SET R=N; IF (R-2047) 1.16,1.16;SET X=R; DO 3; SET R=X
Ø1.16 IF (FITR(R)-R) 2.2,1.18,2.2
Ø1.18 IF (3-N) 1.2,1.4,1.4
Ø1.2Ø SET D=2; DO 2
Ø1.3Ø SET E=FSQT(N); FOR D=3,2,E; DO 2
Ø1.4Ø TYPE " IS PRIME."!!; QUIT

Ø2.Ø5 SET R=N/D; IF (R-2Ø47) 2.1,2.1; SET X=R; DO 3; SET R=X
Ø2.1Ø IF (R-FITR(R)) 2.3,2.2,2.3
Ø2.2Ø TYPE " IS NOT PRIME."!!; QUIT
Ø2.3Ø CONTINUE

Ø3.1Ø FOR II=2Ø47,2Ø47,X; SET X=X-2Ø47
*
```

The output of two runs of this program are:

```
*GO
:1ØØ489  IS NOT PRIME.

*GO
:89899  IS PRIME.

*
```

Note that in this program, Part 3 is used to reduce the argument to be used in the FITR function in Step 1.16 and Step 2.10.

A program for the computation of the least common multiple (LCM) of three ASKed integers A, B, and C is a good assignment when introducing the techniques of elementary factoring as well as when teaching programming. A straightforward algorithm for computing LCM is to allow the trial LCM to be the largest of the three ASKed integers. If this trial LCM is divisible by A, B, and C, it is the correct answer. If not, a new trial LCM is formed by adding the largest of A, B, and C to the old trial LCM and repeating the divisibility tests. A program using this algorithm is a good exercise in using the IF statement. A typical student program which follows this algorithm is:

```
Ø1.Ø1 C PROGRAM TO FIND LEAST COMMON MULTIPLE OF THREE INTEGERS
Ø1.Ø5 ASK ?A ?,? B ?,? C ?,"          "
Ø1.Ø8 SET A=FABS(A); SET B=FABS(B); SET C=FABS(C)
Ø1.1Ø SET BIG=A; IF (B-BIG) 1.15; SET BIG=B
Ø1.15 IF (C-BIG) 1.2; SET BIG=C
Ø1.2Ø SET LCM=BIG
Ø1.3Ø IF (FITR(LCM/A)-LCM/A) 1.7, 1.4, 1.7
Ø1.4Ø IF (FITR(LCM/B)-LCM/B) 1.7, 1.5, 1.7
Ø1.5Ø IF (FITR(LCM/C)-LCM/C) 1.7, 1.6, 1.7
Ø1.6Ø TYPE %8.Ø, ?LCM ?,!!; QUIT
Ø1.7Ø SET LCM=LCM+BIG; GOTO 1.3
*
```

The outputs of two runs of this program are:

```
*GO
A :5   B :3   C :11          LCM =+      165

*GO
A :12   B :8   C :16          LCM =+       48

*
```

Note that this program will properly handle the entry of negative integers but will not work if fractions or zero are entered. Experience with this assignment has shown that students quickly "discover" or understand this algorithm, and the writing of the program does indeed reinforce their classroom introduction to the use of the IF statement. This same algorithm can be extended to finding the LCM of N ASKed integers by using subscripted variables. Although useful when teaching programming, the more general program is decidedly more difficult for students to write and is probably not useful as an assignment immediately following the version involving only three ASKed numbers. However, this assignment is excellent when subscripts have been introduced and students are gaining confidence in their programming ability. A program which satisfies this assignment is:

```
Ø1.1Ø ASK ?N?,!; FOR I=1,N; ASK A[I]
Ø1.2Ø SET BIG=A[1]; FOR I=2,N; DO 2
Ø1.3Ø SET LCM=BIG
Ø1.4Ø SET SIGNAL=Ø; FOR I=1,N; DO 3
Ø1.5Ø IF (SIGNAL) 1.6,1.7,1.6
Ø1.6Ø SET LCM=LCM+BIG; GOTO 1.4
Ø1.7Ø TYPE %8.Ø,!?LCM ?,!!; QUIT

Ø2.1Ø IF (A[I]-BIG) 2.2,2.2; SET BIG=A[I]
Ø2.2Ø CONTINUE

Ø3.1Ø IF (LCM/A[I]-FITR(LCM/A[I])) 3.2,3.2; SET SIGNAL=1; SET I=N
Ø3.2Ø CONTINUE
*
```

Note that this program assumes that the user will enter all positive integers. The outputs of three runs of this program are:

```
*GO
N:5
:8 :24 :16 :4Ø :36
LCM =+      72Ø

*GO
N:4
:15 :25 :35 :1Ø5
LCM =+      525

*GO
N:3
:13 :11 :15
LCM =+     2145

*
```

Note that to write this program the student had to know how to determine the largest of a list of N numbers (step 1.2 and part 2) as well as how to use a variable as a signal flag. Experience with this

assignment has demonstrated that some students will have difficulty in changing the question, "Is the trial LCM divisible by all input integers?", as used in the algorithm of the previous example, to the question, "Is the trial LCM not divisible by at least one of the input integers?", as used in the algorithm for this program. Although this question need not be changed in order to obtain a valid algorithm, doing so allows the development of a much more efficient algorithm.

The problem of computing the greatest common divisor (GCD) is a natural companion to that of computing the LCM. Programs using algorithms quite similar to the two preceding LCM examples are often very beneficial as assignments. A tutorial tactic that has been very successful with students learning to program is to present the algorithm and program for computing the LCM of three ASKed numbers in class and then to assign, without further discussion, the problem of writing a program to compute the GCD of three ASKed numbers. Even if students have written one or both of these GCD programs successfully, they should be encouraged to write a program which uses the Euclidian algorithm for computing the GCD of two ASKed positive integers. Need for such an algorithm is naturally motivated when the computation of the GCD for some sets of numbers requires more than a brief period of time. Uses for a brief GCD algorithm will also occur when complete polynomial factoring programs are written later in the year. A working program using the Euclidian algorithm is:

```
Ø1.Ø1 C EUCLIDIAN ALGORITHM FOR GREATEST COMMON DIVISOR
Ø1.1Ø ASK "THE TWO NUMBERS ARE ",N," ",D
Ø1.2Ø IF (D-N) 1.3,1.3; SET S=N; SET N=D; SET D=S
Ø1.3Ø SET R=N-D*FITR(N/D); SET N=D; SET D=R; IF (R) 1.3,1.4,1.3
Ø1.4Ø TYPE "     GCD ",%4.Ø,N,!!
*
```

The outputs of two runs of this program are:

```
*GO
THE TWO NUMBERS ARE :544   :5321       GCD =+  17

*GO
THE TWO NUMBERS ARE :19   :31      GCD =+    1

*
```

Note that this program contains no provision for rejecting negative numbers, zero, or fractions.

An effective assignment that can be used when discussing either factoring or multiplication of monomials is the writing of a program which generates a problem of the form $Ax^P * \underline{\quad ? \quad} = Bx^Q$, ASKs the user to solve the problem, and then verifies his solution. If students have not yet been

introduced to the use of random numbers, the assignment might be changed to having the program ASK for the numbers A, P, B, and Q. A program using random numbers which determines the missing term is:

```
Ø1.1Ø SET A=FITR(FRAN()*15); SET P=FITR(FABS(FRAN())
Ø1.15 SET B=FITR(FRAN()*15)*A; SET Q=FITR(FABS(FRAN(
Ø1.2Ø TYPE "PROBLEM IS:     ",%4.Ø,A,"*X↑",%2.Ø,P,"       MES      ?     "
Ø1.25 TYPE "EQUALS     ",%4.Ø,B,"*X↑"%2.Ø,Q,!
Ø1.3Ø ASK !"MISSING TERM IS:  ",C,"*X↑",R,!!
Ø1.4Ø IF (C*A-B) 1.7,1.5,1.7
Ø1.5Ø IF (R+P-Q) 1.7,1.6,1.7
Ø1.6Ø TYPE "CORRECT"!!; QUIT
Ø1.7Ø TYPE "TRY THAT TERM AGAIN"!!; GOTO 1.3
*
```

The outputs of two runs of this program are:

```
*GO
PROBLEM IS:      =+    3*X↑=+ 3      TIMES      ?      EQUALS      =-   12*X↑=+ 6

MISSING TERM IS:  :-4  *X↑:3  <-- Input
CORRECT

*GO
PROBLEM IS:      =+    6*X↑=+13      TIMES      ?      EQUALS      =+   24*X↑=+26

MISSING TERM IS:  :4  *X↑:2
TRY THAT TERM AGAIN

MISSING TERM IS:  :4  *X↑  :13
CORRECT

*
```

Note that the random numbers are chosen in such a way that B is always divisible by A and Q is always greater than P. These restrictions avoid the necessity of negative exponents and decimals in the ASKed answer. If students are familiar with negative exponents or are willing to enter the ASKed coefficient with six significant digits, then these restrictions are not necessary. Since a run of this program appears very tutorial, remember that the suggested use of this problem is as an assignment for students to program. The nature of a run of the program simply facilitates checking the students' work.

Another program which can be used effectively as a student assignment when discussing monomial factors of a polynomial is the writing of a program which ASKs for all constant terms in the expression $A \times {}^{P}Y^{Q} + B \times {}^{R}Y^{S}$, computes the monomial factor (if any), and then types the factored expression.

A program which does this is:

```
Ø1.Ø5 TYPE "EXPRESSION TO BE FACTORED IS:"!
Ø1.1Ø ASK A,"*X↑",P,"*Y↑",Q,"  PLUS     ",B,"*X↑",R,"*Y↑",S,!!
Ø1.2Ø SET N=FABS(A);SET D=FABS(B); IF (D-N) 1.3;SET I=N;SET N=D;SET D=I
Ø1.3Ø SET RE=N-D*FITR(N/D); SET N=D; SET D=RE; IF (RE) 1.3,1.4,1.3
Ø1.4Ø SET XL=P;SET XS=R; IF (XL-XS) 1.5,1.5;SET I=XL;SET XL=XS; SET XS=I
Ø1.5Ø SET YL=Q;SET YS=S; IF (YL-YS) 1.6,1.6;SET I=YL;SET YL=YS; SET YS=I
Ø1.6Ø IF (1-N) 1.7; IF (XL+YL) 1.7,1.65,1.7
Ø1.65 TYPE "EXPRESSION CONTAINS NO MONOMIAL FACTOR."!!; QUIT
Ø1.7Ø TYPE "FACTORED EXPRESSION IS: "!,%3.Ø,N,"*X↑",%2.Ø,XL," *Y↑",YL
Ø1.75 TYPE " (",%3.Ø,A/N," *X↑",%2.Ø,P-XL," *Y↑",Q-YL
Ø1.78 TYPE " PLUS ",%3.Ø,B/N," *X↑",%2.Ø,R-XL," *Y↑",S-YL,")"!!
*
```

The outputs of two runs of this program are:

```
*GO
EXPRESSION TO BE FACTORED IS:
:12 *X↑:0 *Y↑:5     PLUS     :36 *X↑:4 *Y↑:2

FACTORED EXPRESSION IS:
=+ 12 *X↑=+ 0 *Y↑=+ 2 (=+   1 *X↑=+ 0 *Y↑=+  3 PLUS =+ 3*X↑=+4 *Y↑=+  0)

*GO
EXPRESSION TO BE FACTORED IS:
:7 *X↑:5 *Y↑:0     PLUS     :9 *X↑:0 *Y↑:10

EXPRESSION CONTAINS NO MONOMIAL FACTOR

*
```

Note that this program uses the Euclidian algorithm (lines 1.2 and 1.3) for finding the greatest common divisor of A and B. This program does not factor -1 from A and B when appropriate. Certainly, finding the common factor -1 should be expected if students have had sufficient programming experience. Techniques for doing this are shown in subsequent programs. This program also assumes all exponents are entered as positive integers and that the values of A and B are such that $A * B \neq 0$. Elimination of these restrictions may be included as part of the assignment if that will not interfere with the main point of the lesson.

INDEX

digital

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

184 Printed in U.S.A.