THE COMMODORE 64

IDEA BOOK

INCLUDES 50 READY-TO-RUN EDUCATIONAL PROGRAMS

DAVID H. AHL

The Commodore 64 Ideabook

Includes 50 Ready-to-Run Programs

David H. Ahl

Illustrations: Wayne Kaneshiro

Creative Computing Press
Morris Plains, New Jersey

Library of Congress Number 83-73076
ISBN 0-916688-68-2

Printed in the United States of America
First printing October 1983
10 9 8 7 6 5 4 3 2 1

Creative Computing Press
39 E. Hanover Avenue
Morris Plains, NJ 07950

Dedication: To Betsy, for her friendship, encouragement and understanding.

About the Author

David H. Ahl has a BEE from Cornell University, MBA from Carnegie-Mellon University and has done further work in educational psychology at the University of Pittsburgh.

He served in the Army Security Agency, was a consultant with Management Science Associates and a senior research fellow with Educational Systems Research Institute.

In early 1970, he joined Digital Equipment Corporation. As education product line manager, he formulated the concept of an educational computer system consisting of hardware, software and courseware and helped guide DEC into a leading position in the education market.

Mr. Ahl joined AT&T in 1974 as education marketing manager and was later promoted to manager of marketing communications for the unit later to become American Bell. Concurrent with this move, he started *Creative Computing* as a hobby in late 1974.

As *Creative Computing* grew, Mr. Ahl left AT&T in 1978 to devote full time to it. *Creative Computing* magazine today is Number 1 in software and applications. In January 1980, Ahl founded *SYNC* magazine and, over the years, has acquired or started several other publications.

Mr. Ahl is the author or editor of 16 books and over 150 articles about the use of computers. He is a frequent lecturer and workshop leader at educational and professional conferences.

Contents

Preface

Although Charles Babbage laid down several ideas for computing "engines," the forerunners of today's computers were largely developed in the early 40's as part of the war effort. The Robinson series cryptoanalytic machines developed in England in 1941 spawned many families of computers still in use today. The MIT differential analyzer and real-time aircraft simulation project led to the Whirlwind, and eventually to the immensely successful DEC family of PDP computers (Programmed Data Processors). And, of course, Eniac built at the University of Pennsylvania, was the guiding light behind Univac, IBM and many other successful manufacturers.

Many people today poke fun at these early machines and regard them as dinosaur-like relics. However, it is interesting to consider that a large-scale computer of about 25 to 30 years ago had about the same amount of power as a typical personal computer of today. It was generally not as reliable or user-friendly as a personal computer, and, of course, cost tens of thousands times as much. Why bring this up?

Because a computer of the 50's required that the programmer be very clever and resourceful to solve problems within the capabilities of the computer. He did not have vast gobs of memory available, blinding quick calculation speed, or random disk access. In other words, he had about the same problem to face as you do with your personal computer.

I do not mean to imply that your personal computer is not a full-fledged computer. It certainly is just as much a computer as a room-filling giant of today. However, because of the relatively small memory, it cannot store a large data base. Nor is it suitable for extensive word processing or massive calculations. Highly detailed graphics are best left to other machines as well.

What can we learn from the computing pioneers of the 50's that will help us today? Perhaps most important is the discipline of thoroughly analyzing a problem, breaking it down into manageable steps, and solving it a step at a time. It is also important to determine what can be done "off line" and what must be done on the computer.

That is what this book is all about. While it has more than 50 ready-to-run programs, the main thing you should look for from the book is an approach to solving problems—big and small. Some of the problems demonstrate the capability of the computer; others identify its shortcomings. It is important to be familiar with both the strengths and weaknesses of your tools so you can recognize the types of jobs for which they are suitable (and not suitable).

This book focuses primarily on mathematical and educational applications for the computer. There are many other excellent sources of information about other applications and for making best use of your personal computer. Using the approaches described in this book should enable you to easily convert programs and use applications from other books and magazines such as *Creative Computing.*

This book is designed to be read with a working computer at hand. While there is textual material to be read, the most important things are the experiments and problems to be tried with your own computer. The book raises many questions for which you should try to find answers. There are no answers to these questions and problems in the back of the book; you should be able to discover the answers as you work the problems out on your computer.

You will be able to incorporate many of the routines and approaches in the book into programs of your own as you use your computer to deal with "real world" problems. Other programs will simply point you in the right direction. And some of the programs in the book are just plain fun. Learn. Experiment. Have fun!

Morristown, New Jersey
August 1983

David H. Ahl

The Commodore 64 Ideabook

1

Drill and Practice

Throughout life, there are certain things that simply must be memorized. Obvious things that fall into this category are the addition and multiplication tables, the spelling and meaning of words, how to tell time, and the monetary system.

However, depending upon one's chosen profession, there are many other things to be memorized. A doctor must know what diseases match what symptoms. A chemist must know the gas laws, the properties of elements and so on. A pilot must instantaneously know the meaning of readings on scores of instruments.

To memorize a set of facts, you must go over them again and again and keep trying different variations. Here is where the computer comes in. It is able to present randomly scores of different problems to you for as long as you wish. Some programs will automatically adjust to your level of competence and will grade you; other programs simply present the problems and leave the grading up to you.

There are four programs in this chapter and two in the Science chapter which present material in a drill and practice format. Examine the methods used in these programs and then make up some drill programs of your own for subjects with which you are having trouble, or make up programs for other members of your family.

Addition Practice

This program demonstrates a simplified addition drill and practice routine. This type of drill is sometimes called computer assisted instruction (CAI), although CAI can also apply to tutorial and other approaches as well.

When the program is run, it will first ask "No. of digits?" You enter a number and each addend will contain that many or fewer digits.

The program will present any number of practice problems that you specify. The program presents each problem in turn. The program will not proceed to the next problem until the current one has been answered correctly. After the last problem is answered correctly, the score is printed with an appropriate comment.

There are many improvements and extensions possible in a program like this one. For example, you might want to modify the program so it tells the user the correct answer after a problem has been answered incorrectly two (or three) times.

A more complicated modification would be to change the program to present different kinds of arithmetic problems such as subtraction, multiplication, and division.

Some of these modifications have been made in the next program.

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147);"Addition Practice"
20 inPut "No. of di9its";n
30 inPut "No. of Problems";P
40 w=0
50 x=x+1
60 9=0
70 a=int(10↑n*rnd(1))+1
80 b=int(10↑n*rnd(1))+1
90 Print
100 Print tab(8-int(lo9(a)/lo9(10)+1));a
110 Print tab(7-int(lo9(b)/lo9(10)+1));"+";b
120 Print " --------"
130 c=a+b
140 Print tab(7-int(lo9(c)/lo9(10)+1));
150 inPut9
160 if 9=c then 230
170 9=9+1
180 if 9>1 then 200
190 w=w+1
200 Print:Print "What?  Try a9ain."
220 9oto 90
230 Print:Print "Ri9ht! ";
240 if x>=P then 280
250 Print "Here's another"
270 9oto 50
280 Print:Print:Print "You 9ot";P-w;"correct the first time."
290 if w<.22*P then 320
300 Print "And you missed";w;
310 end
320 Print:Print "Good work!";:end
```

```
Addition Practice
No. of di9its? 3
No. of Problems? 3

     189
   + 973
 --------
   ? 1162

Ri9ht! Here's another

     777
   + 418
 --------
   ? 1365

What?  Try a9ain.

     777
   + 418
 --------
   ? 1195

Ri9ht! Here's another

     830
   + 810
 --------
   ? 1640

Ri9ht!

You 9ot 2 correct the first time.
And you missed 1
```

Addition Practice, Adjusted by Grade Level

One of the major disadvantages with many drill and practice exercises is that they tend to be either boring or frustrating, depending upon the ability of the user relative to the level of the material. To compensate for this, a method is needed which will adjust the difficulty of the problems to the ability of the user.

Ideally, such a system would weigh the most recent performance most heavily but would not ignore previous performance. It should allow a user to advance to more difficult problems than his current mastery level. It should also continue to give some practice on problems already mastered.

Some commercial software packages approach these goals along traditional lines, i.e., determine in which type of problems a student should receive practice by using a complicated computer managed instruction score recording and adjustment system.

The approach here is more innovative; it uses a single measure for each type of problem—call it "estimated grade level"—which meets all of the objectives stated above.

How does it work? The most recent problem presented counts 10% of the overall score if it was answered correctly and was over the current user grade level, or if it was answered incorrectly and was under the current user grade level. Otherwise it is ignored. This may be easier to visualize in the form of a chart:

		Answer	
		Right	*Wrong*
Problem	*Higher than grade level*	Raise student grade level	Ignore
	Lower than grade level	Ignore	Lower student grade level

At first glance this might look complex and somewhat goofy, however, what it really means is that a student is rewarded for doing a problem beyond his grade level but he is not penalized if he cannot do it. On the other hand he is penalized if he cannot do a problem lower than his grade level, but is not rewarded for doing one lower.

Each problem affects the estimated grade level a little bit, with the most recent problems being weighed the most heavily. If the current grade level of a student is L and the level of the most recent problem to be averaged in is P,

then the averaging formula is simply:

$$L = .9L + .1P$$

The remaining task before a program can be written is to assign a grade level to each problem presented. Unfortunately, this will vary depending upon the local school system, the textbook used, and the teaching method. Also a huge data base can not be stored in a small computer, so it is desirable to devise a simple method of determining grade level for different problems. One straightforward approach is to present problems up to one-half a grade level over and under where the student currently is. Thus the overall range of problems for a student at grade level 3.2 would be 2.7 to 3.7.

How do we generate the right problems? Consider one type of skill, vertical addition. It is normally introduced in the first grade and continues through Grade 4 (actually 4.9). The simplest problem in this program is 1 + 1 and the most difficult is 999 + 999. Since learning is not a linear process (it is slow at first, and then progresses rapidly), an exponential formula can be used. For example:

$$\text{Addend} = 1.73 \times (\text{Grade level})^4$$

or

$$\text{Grade level} = \sqrt[4]{\text{Addend}/1.73}$$

This means that students at various grade levels will be working with the following maximum addends:

Grade level	*Addend*
1.0	1
2.0	27
3.0	140
4.0	442
4.9	997

Now it is a relatively straightforward, although somewhat tedious, matter to tie all these elements together in a computer program.

A few notes about the program. The variable G2 is the problem grade level that is always within one-half of a grade level of the current student level, G1. The complicated mess in Statement 290 produces a moving grade level average.

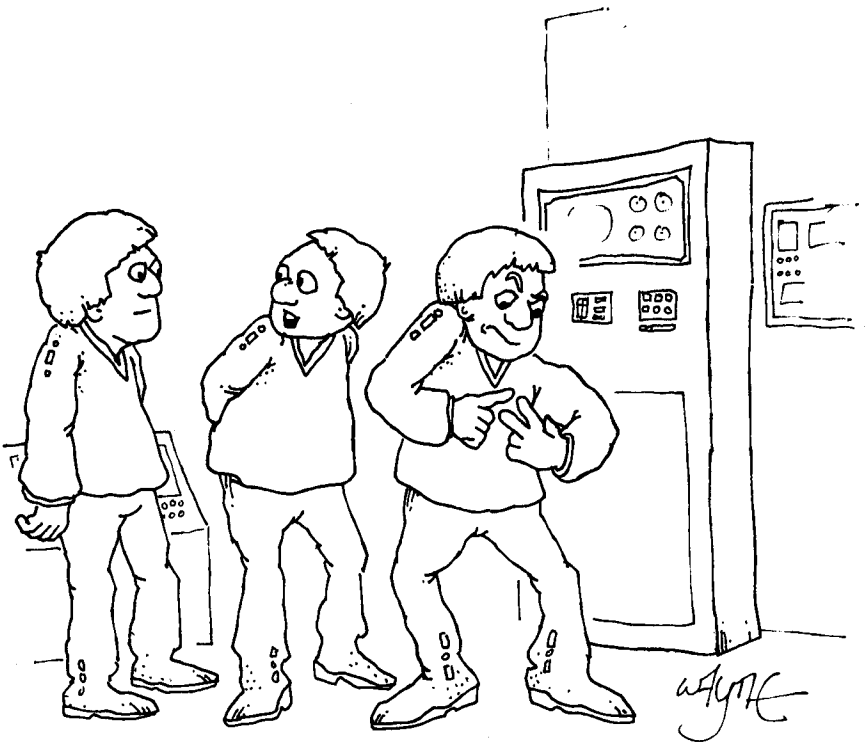
The recording of the grade level and carrying it over to the next lesson is a manual process. On a computer system with a permanent mass storage device, this would be kept on the system.

There are many possible changes and extensions to this program. For example it could present different types of problems such as horizontal addition, vertical and horizontal subtraction, as well as multiplication, division and fraction problems.

```
5 print chr$(14):rem switch lower case
10 w=0
20 print chr$(147);"Hi.  To stop enter 9999 as answer."
30 input "Grade level";g1
40 g2=g1-.5*rnd(0)
50 r=int(2+1.73*g2^4)+1
60 a=int(100*g2*rnd(1))+1
70 if a>r then 60
80 b=r-a
90 print:print tab(8-int(log(b+1)/log(10)+1));b
100 print tab(7-int(log(a+1)/log(10)+1));"+";a
110 print "  --------"
120 r=a+b
130 print tab(7-int(log(r)/log(10)+1));
140 inputg
150 if g=9999 then 320
160 if g=r then 260
170 w=w+1
180 if w>1 then 210
190 print:print "Wrong, try again"
200 goto 90
210 print:print "You missed that 2 times.  The answer is";r
230 w=0
240 if g2>g1 then 300
250 goto 290
260 w=0
270 print:print "Correct!";
280 ifg2<g1 then 300
290 g1=.9*g1+.1*sqr(sqr(r/2/1.73))
300 print:print "Here's another...":print
310 goto 40
320 print chr$(147)
330 print "Okay.  So long."
340 g1=(int(100*g1))/100
350 print "New Grade level";g1
```

```
Hi.  To stop enter 9999 as answer.
Grade level? 4

     349
   +  73
----------
   ? 422

Correct!

     280
   + 125
----------
   ? 405

Correct!

     200
   + 177
----------
   ? 377

Correct!
```

```
     296
   +  55
----------
   ? 341
Wrong, try again

     296
   +  55
----------
   ? 351

Correct!

      76
   + 211
----------
   ? 9999

Okay.  So long.
New Grade level 3.93
```



"You won't believe this, but Zarg can actually add without a computer."

Time/Speed/Distance Problems

As well as being able to present simple numerical drill and practice problems, the computer can present word problems for solution as well.

In this program the formula relating time, speed and distance was applied to a problem involving both a car and train.

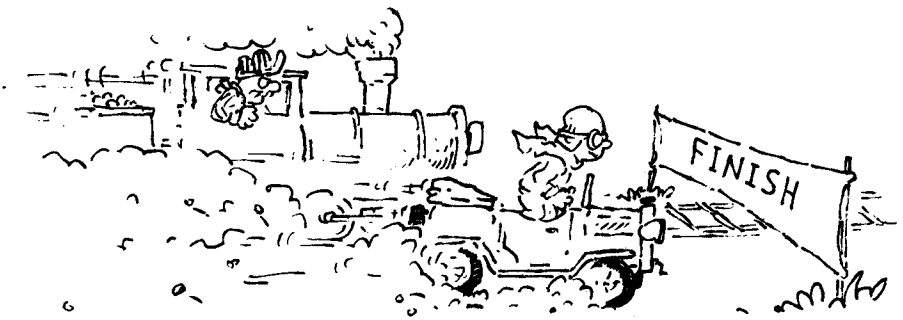
The problem can be stated as follows. A car traveling C miles per hour (computer generates an integer 40 through 65) can make a certain trip in D hours (computer generates an integer 5 through 20) less than a train traveling at T mph (computer generates an integer 20 through 39). How long does the trip take by car? When the two simultaneous equations are solved they produce the single equation for the answer shown in Line 60.

Notice the calculation in Line 70. This calculates the percent difference between the actual answer and the one entered by the user.

Notice also that the computer calculates the correct answer in Line 60 and prints it (on the screen) in Line 130. This answer may have many decimal places; as the program is written it is rounded off to two decimal places. If the 0.5 was not added in Line 60, the number would be truncated and not rounded off. This is an important calculation and one which you will find in many other programs throughout the book.

Consider other problems that can be used as the basis for this kind of drill and practice exercise. Teachers, for example, might wish to have students write drill and practice programs on their own. Different problems could be given to one or a small group of students to serve as the basis for a program.

After the programs are written, students can try out the programs of other class members. This approach ensures that each student not only understands the type of problem assigned to him, but also gets practice in solving other problem types as well. This is an effective technique for stimulating interest as well as for learning how to solve word problems.



```
5 Print chr$(14):rem switch lower case
10 c=int(rnd(1)*25)+40
20 d=int(rnd(1)*15)+5
30 t=int(rnd(1)*19)+20
35 Print chr$(147);
40 Print "A car at";c;"mPh makes a triP in";d;"hrs   less ";
45 Print "than a train at";t;"mPh
50 Print:inPut "Car triP hrs";a
60 v=int(.5+100*(d*t/(c-t)))/100
70 e=int(abs((v-a)*100/a)+.5)
80 Print
90 if e>5 then 120
100 Print "Good.  You were within";e;"Percent."
110 goto 130
120 Print "Sorry.  You were off by";e;"Percent."
130 Print "Correct time=";v
140 Print:inPut "Another (y,n)";z$
150 if z$="y" or z$="Y" then 10
160 Print:Print "Okay.  So long."
170 end
```

```
A car at 53 mPh makes a triP in 5 hrs less
than a train at 23 mPh
Car triP hrs?  4

Good.  You were within 4 Percent.
Correct time= 3.83

Another (y,n)? y

A car at 58 mPh makes a triP in 14 hrs less
than a train at 31 mPh
Car triP hrs?  13

Sorry.  You were off by 24 Percent.
Correct time= 16.07

Another (y,n)? y

A car at 56 mPh makes a triP in 11 hrs less
than a train at 26 mPh
Car triP hrs?  7.25

Sorry.  You were off by 31 Percent.
Correct time= 9.53

Another (y,n)? n

Okay.  So long.
```

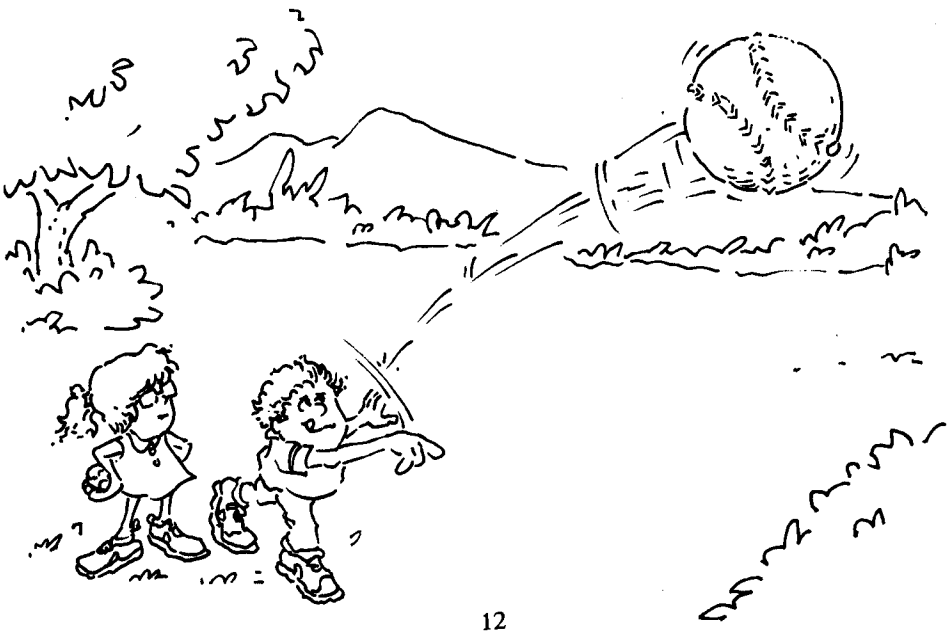
Kinematics Problems

Kinematics relates to the dynamics of motion of bodies apart from considerations of mass and force. Like many other types of problems, these can be generated and presented by the computer to provide practice in solving them.

This program presents a simple kinematics problem for solution. The computer generates a new value for each problem. The problem is as follows. A ball (or any other object) is thrown up at velocity V meters per second (computer generates an integer between 5 and 40). The user must then calculate three factors about the resulting flight of the ball: maximum height, time until it returns, and velocity after T seconds (computer generates a time less than the total flight time).

The key benefit in using a computer to present problems of this type is motivation. The calculations required of the user are no different than those in the back of a chapter in a book or those on homework assignments. However, answering them when they are presented by the computer seems to make it more of a challenge and, frankly, more fun.

The computer program checks each of your responses to see if it is within 15% of the correct answer. If it is, your answer is considered correct. You may wish to change this percentage to require more or less accurate calculations. You may also wish to change the computer calculations to round off to one or two decimal places.



```
5 Print chr$(14):rem switch lower case
10 q=0
20 v=6+int(35*rnd(1))
30 Print chr$(147);"Ball is thrown up at";v;"meters/sec"
40 a=.05*v^2
50 input "What is the maximum height";g
60 gosub 210
70 a=v/5
80 Print:input "When will it come down (sec)";g
90 gosub 210
100 t=2+int(2*v*rnd(1))/10
110 if t>= v/5 then 100
120 a=v-10*t
130 Print:Print "What is velocity at";t;"secs";
140 inputg
150 gosub 210
160 Print:Printq;"right out of 3"
170 if q>1 then Print "Not bad."
180 Print:input "Another (y,n)";z$
190 if z$="y" or z$="Y" then 10
200 end
210 if abs((g-a)/a) <.15 then 240
220 Print "Not even close..."
230 goto 260
240 Print "Pretty good."
250 q=q+1
260 Print "Correct answer =";a
270 return
```

```
Ball is thrown up at 24 meters/sec
What is the maximum height? 30
Pretty good.
Correct answer = 28.8

When will it come down (sec)? 5
Pretty good.
Correct answer = 4.8

What is velocity at 2.3 secs? 2.3
Not even close...
Correct answer = 1

 2 right out of 3
Not bad.

Another (y,n)? y
```

```
Ball is thrown up at 13 meters/sec
What is the maximum height? 8
Pretty good.
Correct answer = 8.45

When will it come down (sec)? 2.5
Pretty good.
Correct answer = 2.6

What is velocity at 2.3 secs?-9
Pretty good.
Correct answer =-10

 3 right out of 3
Not bad.

Another (y,n)? n
```

"This nano-computer is great, but working the keyboard is a real problem!"

2

Problem Solving

In many courses in school, the textbooks present a great variety of devices for solving the problems that have been neatly grouped together at the end of each chapter. Typically these devices consist of formulae, equations, rules and theorems. After a careful study of these devices, teachers give exams which test your ability to recall them.

But what do you do if you are faced with the more realistic situation of not being told what device is likely to solve which problem or, worse yet, of having forgotten how to use a technique altogether? Is all hope lost? Of course not, although some people seem to believe that it is.

In this chapter, several of these nasty devices mentioned above are presented. For example, there are devices written into computer programs that will solve a quadratic or exponential equation and others that will calculate the roots and draw a plot of any function.

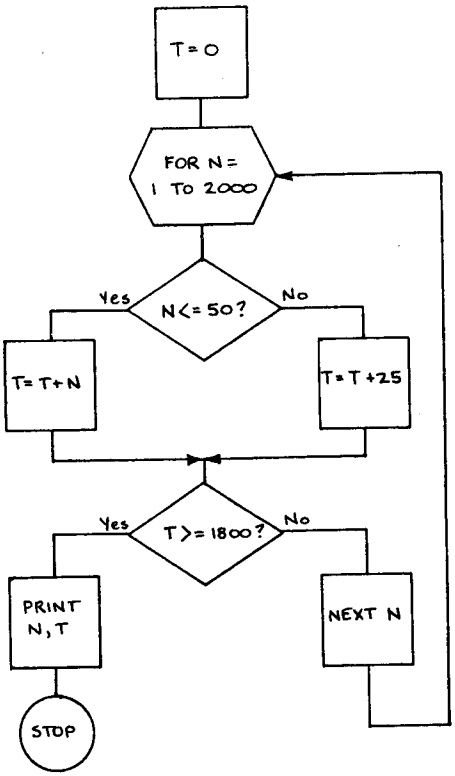
However, you must remember that while these devices are useful in solving certain problems, the really important thing is to understand the underlying logic and approach. Then when it comes time to solve real world problems you will be better prepared to face them. Incidentally, many of the methods and devices presented in this chapter are used in later chapters to solve other kinds of problems.

How Many Tickets?

Here is a problem. At a school raffle to raise money, the organizers have as a prize an electronic game for which they paid $18.00. To add interest to the raffle, the organizers have decided to sell tickets for an amount (in cents) equal to the number on the ticket for tickets numbered 1 to 50. For ticket numbers over 50, the price is 25 cents each. The organizers want to know how many tickets they must sell to exactly break even.

It is probably easiest to visualize a problem of this sort with a flowchart. In the flowchart, T will equal the total money collected and will increase as more tickets are sold. The ticket number is N. When T equals or exceeds $18.00, N will be the answer.

Note that the flowchart has two logical branching points (IF statements in the program). The first compares the current ticket number to 50; if it is less, the ticket number is added to the total whereas if it is greater than 50, the total is increased by 25 cents.



The second branch point compares the total amount collected, T, to $18.00 (actually 1800 cents). If T is equal to or greater than 1800, the break even point has been reached and the values of N and T are printed (on the screen).

A problem of this kind can be done by hand, however, because of the repetitive additions it is quite tedious. Also, doing it by hand frequently leads to an answer of 72 rather than the correct answer of 71. Try it yourself and see what you get.

```
10 t=0
20 for n=1 to 2000
30 if n<=50 then 60
40 t=t+25
50 goto 70
60 t=t+n
70 if t>=1800 then 90
80 next n
90 print n;"TICKETS SOLD"
100 print "COLLECTED $";t/100


 71 TICKETS SOLD
COLLECTED $ 18
```

Many problems can be solved quickly and correctly with a computer using logical analysis and a flowchart. More complex problems may have to be broken down into additional steps and require a longer flowchart, but the approach is fundamentally the same.

Here are two problems for you to solve.

The diameter of a long-playing record is 12 inches. The unused center has a diameter of 4 inches and there is a smooth outer edge 1/2 inch wide around the recording. If there are 91 grooves to the inch, how far does the needle move during the actual playing of the recording?

A movie theater charges $2.50 for an adult admission and $1.00 for a child. At closing, the cashier counted 385 ticket stubs and had $626.50 in cash. How many children entered?

Drinking and High Blood Pressure

This program illustrates how several simple equations can be put in a computer program to solve a more difficult overall problem.

Here is the problem. In a survey of 1000 adults, it was found that 35 had high blood pressure. Of those with high blood pressure, 80% drink 15 oz. or more of alcohol per week. Of those without high blood pressure, 60% drink a similar amount. What percent of drinkers and non-drinkers have high blood pressure?

This problem requires the solution of several simple equations. They could all be combined into one large equation, but it may be easier to understand the approach (and change variables later on) by writing a program with five separate equations.

If H equals the number of people with high blood pressure, then H = 35 (Line 10). Letting H1 equal the number of people with high blood pressure who drink leads to H1 = .8 x H (Line 20).

Letting L1 equal the number of people with low blood pressure who drink yields L1 = .6 x (1000 - H). Then, the total number of drinkers, D = H1 + L1.

Finally, the percentage of drinkers with high blood pressure is X = H1 x 100 / D.

The program solves the problem in a jiffy. The solution for this type of problem can be easily written directly in Basic without any need for a flowchart or detailed analysis. Recognizing this type of problem readily will save a great deal of pencil pushing time.

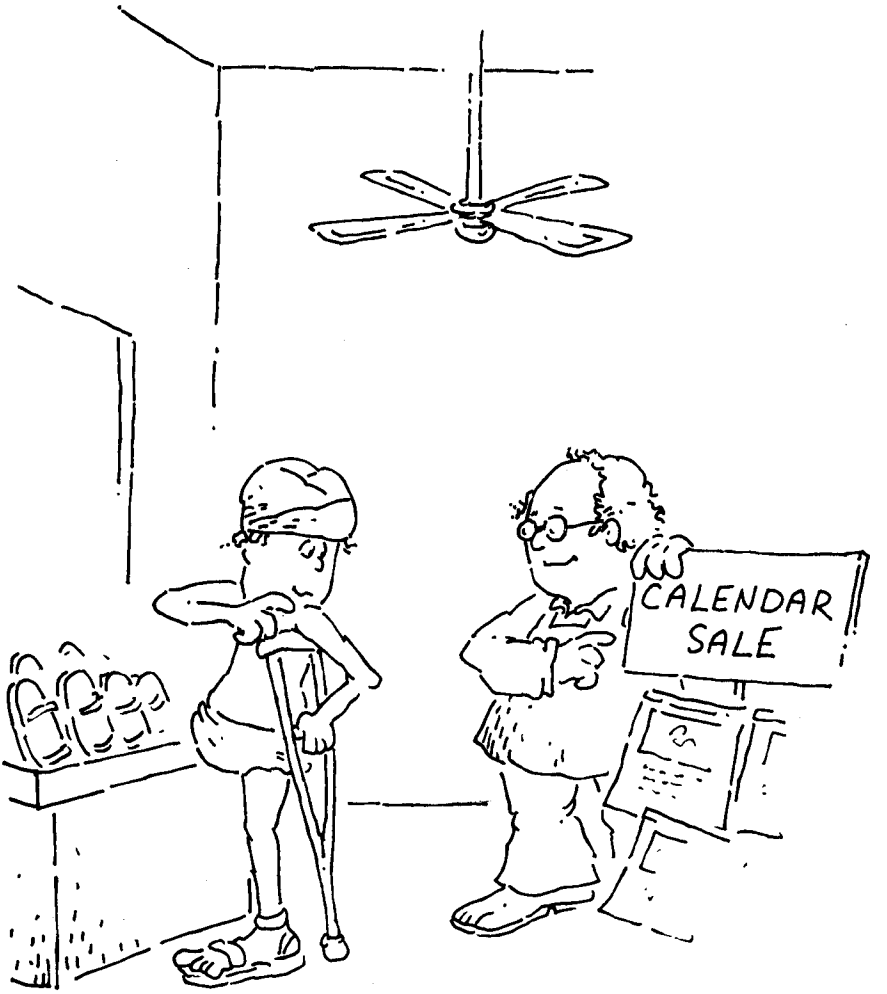
```
10 h=35
20 h1=.8*h
30 l1=.6*(1000-h)
40 d=h1+l1
50 x=h*100/d
60 x1=h*100/1000
70 print "HIGH BLOOD PRESSURE"
80 print "POPULATION";x1;"%"
90 print "DRINKERS  ";x;"%"
```

```
HIGH BLOOD PRESSURE
POPULATION 3.5 %
DRINKERS  5.76606261 %
```

Here is a problem that doesn't require a single equation but makes use of the approaches discussed so far in this chapter. Can you solve it with three Basic statements?

In early January, a shopkeeper marked down some calendars from $2.00 to a lower price. He sold his entire stock in one day for $603.77. How many did he have?

Here's another easy one. A town in India has a population of 20,000 people. Five percent of them are one-legged and half of the others go barefoot. How many sandals are worn in the town?



Two Simultaneous Equations

So far in this chapter, only problems with linear equations have been considered. But the computer can be used to solve much more difficult equations. In fact, it is in problems involving second and third degree equations, exponentials, and the like where the computer really starts to pay off. Consider the following two simultaneous equations:

$$2^x = \frac{16y}{3} \qquad 3^x = 27y$$

It is not at all easy to solve these two equations by hand. But a simple Basic program can be written to solve the equations using trial and error. This is sometimes referred to as a brute force approach because every possible combination of numbers between an upper and lower limit is tried until a solution is reached or until the program runs out of values.

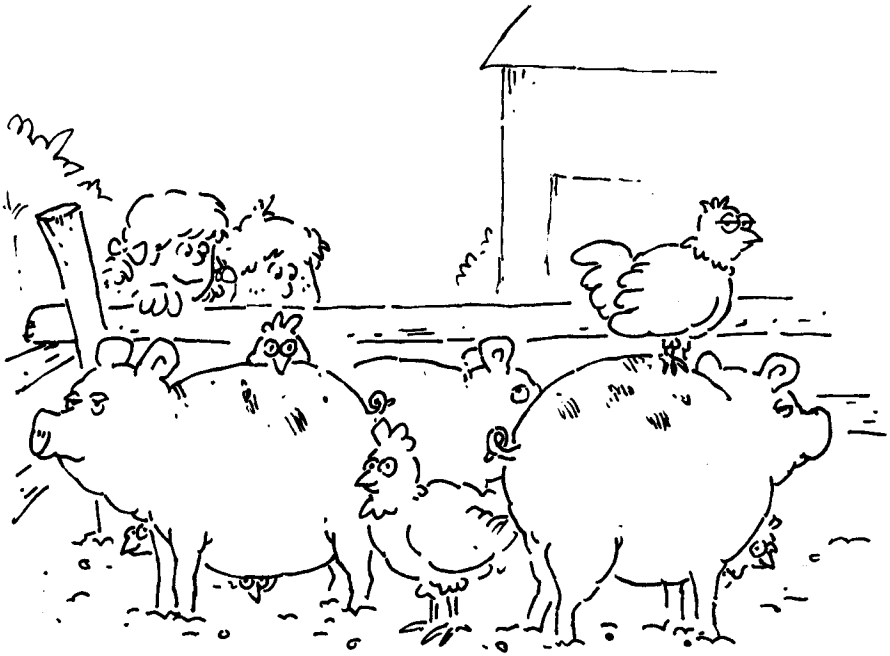
```
5 print "SOLVES TWO SIMULTANEOUS EQUATIONS"
10 for x=1 to 4
15 for x=1 to 4
20 for y=1 to 4
30 if 2↑x<>16*y/3 or int(3↑x)<>27*y then 60
40 print "x=";x,"y=";y
50 stop
60 next y
70 next x
80 print "NO INTEGER SOLUTION"


SOLVES TWO SIMULTANEOUS EQUATIONS
x= 4           y= 3
```

In this particular case, a solution is reached rather quickly with x = 4 and y = 3. However, if in the second equation the y coefficient is changed slightly from 27 to 28, the computer will try 10,000 possible solutions before finally concluding that no integer solution exists—at least within the range of 0 to 100. Warning: this will run for a *very* long time.

```
30 if2↑x<>16*y/3 or int(3↑x)<>28*y then 60


SOLVES TWO SIMULTANEOUS EQUATIONS
NO INTEGER SOLUTION
```

Although the trial and error (brute force) approach is widely used, it is highly inefficient. In general, a systematic or guided trial and error approach is preferable to one that simply tries every possible solution. However, for some problems the simple "try every value" approach may be appropriate. (A comprehensive discussion of trial and error approaches can be found on pp 36-40 of *Computers in Mathematics: A Sourcebook of Ideas.)*

Three problem solving approaches have been discussed so far. Remembering them, how would you do this problem? A boy and his sister visited a farm where they saw a pen filled with pigs and chickens. When they returned home, the boy observed that there were 18 animals in all, and his sister reported that she had counted a total of 50 legs. How many pigs were there in the pen?



Quadratic Equation Solver

For any values A, B, and C of a first degree quadratic equation ($Ax^2 + Bx + C = 0$), this program will compute the roots of the equation. The solution is based on the quadratic theorem which solves for roots with the following formula:

$$X = \frac{-B \pm \sqrt{B^2-4AC}}{2A}$$

Assuming that A, B, and C are real numbers, the following principles apply:

1. If B^2 - 4AC is positive, then the roots are real and unequal.
2. If B^2 - 4AC equals 0, then the roots are real and equal.
3. If B^2 - 4AC is negative, then the roots are imaginary and unequal.

The program takes into account all these possibilities and correctly identifies the type of roots along with their values for any set of coefficients.

Is this program useful by itself? Except for solving quadratic equations for algebra class, probably not. However, as a routine in a larger program to solve quadratic equations that might be encountered, it could be very useful.

```
10 Print:Print "QUADRATIC SOLVER"
20 inPut "A,B,C";a,b,c
30 r=b↑2-4*a*c
40 if a<>0 then 60
50 Print "FIRST DEGREE EQUATION":9oto 150
60 Print "ROOTS ARE ";
70 if r<0 then 120
80 if r=0 then 110
90 Print (-b+sqr(r))/2*a;(-b-sqr(r))/2*a
100 9oto 150
110 Print -b/2*a:9oto 150
120 Print "IMAGINARY"
130 Print (-b/2*a);"+";sqr(-r)/2*a;"*i"
140 Print (-b/2*a);"-";sqr(-r)/2*a;"*i"
150 inPut "ANOTHER (Y,N)";a$
160 if a$="y" then 10
170 stoP
```

```
QUADRATIC SOLVER
A,B,C? 1 , 2 , 1
ROOTS ARE -1
ANOTHER (Y,N)? Y

QUADRATIC SOLVER
A,B,C? 0 , 2 , 4
FIRST DEGREE EQUATION
ANOTHER (Y,N)? Y
```

```
QUADRATIC SOLVER
A,B,C? 2 , 5 , 2
ROOTS ARE -2 -0
ANOTHER (Y,N)? Y

QUADRATIC SOLVER
A,B,C? 4 , 2 , 4
ROOTS ARE IMAGINARY
-4 + 15.4919334 *I
-4 - 15.4919334 *I
ANOTHER (Y,N)? N
```

"According to the computer simulation, it should hit the earth in 0.00298 seconds."

Exponential Equation Solver

Another general routine for solving a particular type of equation is this one to solve for an exponent in an exponential equation.

Given the values A, B, m, and n, this program will solve for x in any exponential equation of the form:

$$A^{mx+n} = B$$

For example, the program will solve any of the following problems:

1. $5^X = 40$
2. $5^{3X+1} = 7.6$
3. $17^{X-3} = 8.12$
4. $11^{1-2X} = 247$

If you were to solve an exponential equation by hand, you would probably go through the following steps:

$$5^X = 40$$
$$\log 5^X = \log 40$$
$$x \log 5 = \log 40$$
$$x = \log 40/\log 5$$
$$x = 1.6021/.6990 = 2.292$$

However, in more generalized form, the solution for x is:

$$X = \frac{\frac{\log B}{\log A}}{M-(N/M)}$$

The data for the four problems listed above were entered into the four data Statements (100–130).

The program as it is presented here can be improved in several ways. First, it always solves the same four equations. How can you generalize it to solve for other equations? Second, if you were to make use of this routine in another program, you would probably not be able to use a READ statement; how could you get rid of it?.

```
5 print chr$(14):rem switch lower case
10 print chr$(147);"Exponential Equation Solver"
20 print
30 print " A      B      M      N      X"
40 print "---    ---    ---    ---    ---"
50 read a,b,m,n
60 x=(log(b)/log(a))/m-(n/m)
70 x=int(.5+100*x)/100
80 print a;tab(7);b;tab(14);m;tab(21);n;tab(27);x
90 goto50
100 data 5,40,1,0
110 data 5,7.6,3,1
120 data 17,8.12,1,-3
130 data 11,247,-2,1
```

```
Exponential Equation Solver

 A      B      M      N      X
---    ---    ---    ---    ---
 5      40     1      0     2.29
 5      7.6    3      1      .09
 17     8.12   1     -3     3.74
 11     247   -2      1     -.65
```

Roots of Any Function

This program will find the roots of a function, any function! The function may be linear, quadratic, cubic, trigonometric or any combination as long at it can be represented in the Basic language. The program as it appears here finds the roots between -20 and 20 although you can change these boundaries in Statement 50.

The method used involves evaluating the function at small incremental intervals, finding places where the value of the function changes sign and then, by successive approximations, finding the zero point. This approach borrows from Newton's method in the final narrowing down but, unlike Newton's method, will not fail to converge in the event one makes an unlucky first guess.

Before running the program you must first type in your function in statement 30. For example,

```
DEF FNA (X) = 2*X↑3+11*X↑2-31*X-180
DEF FNA (X) = X-4
DEF FNA (X) = SIN(X)-.5
```

You may have to refer to the Basic manual with your system to see exactly how a function should be stated.

The routine used in this program is very powerful and could possibly be used as a subroutine in many other programs.

How can you use this program to help you solve this problem for x?

$$x = \sqrt{12 + \sqrt{12 + \sqrt{12 + \sqrt{12 + \sqrt{12 + \ldots}}}}}$$

```
5 print chr$(14):rem switch lower case
10 print chr$(147);"Roots of any function.  Define function in line 30"
30 def fna(x)=2*x^3+11*x^2-31*x-180
40 z1=-200
50 for i=-19.9 to 20
60 if sgn(fna(i))=sgn(fna(i+1)) then 230
70 k=i
80 J=i+1
90 if fna(k)<fna(J) then 130
100 z=k
110 k=J
120 J=z
130 z=(k+J)/2
140 if fna(z)<0 then 170
150 J=z
160 goto 180
170 k=z
180 if abs(fna(z))>.00005 then 130
190 z=sgn(z)*int(abs(z)*10000+.05)/10000
200 if z=z1 then 230
210 print "f(";z;")=0"
220 z1=z
230 next i
```

```
Roots of any function.  Define function in line 30
f(-5 )=0
f(-4.5 )=0
f( 4 )=0
```

```
30 def fna(x)=x-4
```

```
Roots of any function.  Define function in line 30
 f( 4 )=0
```

```
def fna(x)=sin(x)-.5
```

```
Roots of any function.  Define function in line 30
f(-18.3259 )=0
f(-16.2315 )=0
f(-12.0428 )=0
f(-9.9483 )=0
f(-5.7596 )=0
f(-3.6651 )=0
f( .5235 )=0
f( 2.6179 )=0
f( 6.8067 )=0
f( 8.9011 )=0
f( 13.0899 )=0
f( 15.1843 )=0
f( 19.3731 )=0
```

Plot Any Function

Here is a nifty program that will produce a plot of any function on the screen or printer.

Before running this program, you must type in your function in line 200. Like the previous program, which finds the roots of any function, this one will plot any function. You must tell the program between what values you want the function plotted, i.e., a minimum and maximum value of the x coordinate. You also input the x plotting increment you wish.

As it appears here, the program does not allow the user to select the y coordinates; the program plots y values between -30 and 30.

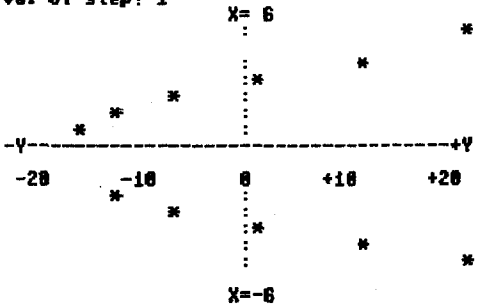
Here are several functions you might want to try plotting:

Function	*X Limits*		*Increment*
DEF FNA(X) = 2*X	-15	15	1
DEF FNA(X) = 30*SIN(X)	-5	5	.25
DEF FNA(X) = X-X*X	-5	6	.5
DEF FNA(X) = 30*EXP(-X*X/100)	-30	30	1.5
DEF FNA(X) = X*X-X	-5	6	1
DEF FNA(X) = X↑2-X-15			

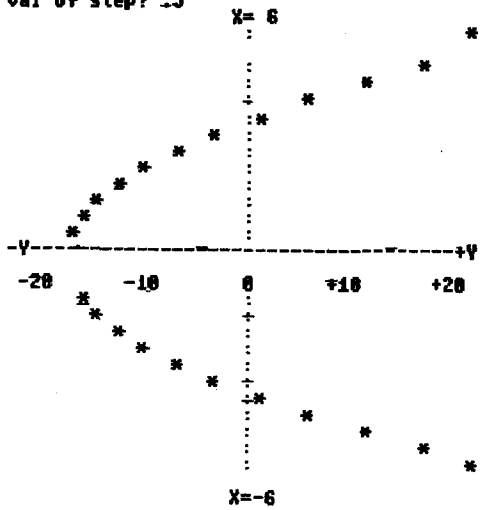
The last function listed is the one plotted in the sample run with the program. Exponential functions are a great deal of fun and sometimes lead to unexpected and interesting results, particularly when combined with trigonometric functions. Experiment! Have fun!

```
5 Printchr$(14):rem switch lower case
10 Print chr$(147);"Plot any function.  Define function in  line 30."
20 Print
30 def fna(x)=x12-15
40 inPut "Initial value of x";x1
50 inPut "Final value of x";x2
60 inPut "Val of steP";s
70 Print tab(19);"X=";x1
80 for x=x1 to x2 steP -s
90 if abs(x)<.00001 then 160
100 y=fna(x)+20
110 if y<40 then 130
120 y=39
130 if y>20 then 200
140 Print tab(y);"*";tab(20)":"
150 goto 210
160 Print "-Y";:for t=1 to 36:Print "-";:next t:Print "+Y"
170 Print " -20      -10       0       +10      +20
180 x=x-s
190 goto 210
200 Print tab(20);":";tab(y);"*"
210 next x
220 Print tab(19);"X=";x2
230 goto230
```

```
Plot any function.  Define function in
line 30.

Initial value of x? 6
Final value of x? -6
Val of step? 1
```




```
Plot any function.  Define function in
line 30.

Initial value of x? 6
Final value of x? -6
Val of step? .5
```



"I'm in the kitchen, dear—using the computer."

3

Sets and Repetitive Trials

For solving relatively simple problems, the computer may not be any help at all. In fact, it may take more time to write a program to solve a problem than it would to solve it by hand or with a calculator. This chapter should help you recognize problems that are suitable for computer solution and those that are not.

In some problems, you may think that the computer will not be any help. However, one thing writing a computer program will always do is force you to reason out the approach to solving the problem logically and precisely. The computer can't solve problems unless it is told exactly how to proceed; hence you must understand a problem completely before you can program it for the computer.

Several of the sections in this chapter discuss sets of data. While the sets used in the examples have relatively few elements or values, you should bear in mind that real world problems often have thousands or millions of pieces of data and the only practical way to solve problems of this size is with a computer. For example, consider how you would most efficiently schedule the shipments on a railroad train leaving Boston with 4000 diverse cargoes bound for Phoenix and 780 points in between. Now consider that there are 200 freight trains per day leaving Boston. Now add to that the 10,000 other trains leaving other cities every day that must use the same network of track and you can see that dealing with real world sets of data is no easy task.

Group of Girls and Boys

In the previous chapter we said that there may be some problems for which brute force trial and error is appropriate. This would be the case if the problems were relatively small and trying every possible solution would not tie up a great deal of valuable computer time. Here is a problem involving two linear equations that lends itself to a trial and error approach.

The problem is as follows: When 15 girls leave a group of boys and girls, there are two boys for every girl (lucky girls). Next, 45 boys decide to leave; then there are 5 girls for every boy (lucky boys!). How many girls were there in the group before anyone left?

Before rushing to the computer, you must recognize that this problem requires the solution of two simultaneous equations. If G equals the original number of girls and B the original number of boys, then the two equations are:

$$(G - 15) \times 2 = B$$

$$(B - 45) \times 5 = (G - 15)$$

```
5 print chr$(14):rem switch lower case
10 i=0
20 print" Program is SLOW, be patient...."
30 for g=1 to 100
40 for b=1 to100
50 i=i+1
60 if 2*(g-15)<>b then 100
70 if 5*(b-45)<>(g-15) then 100
80 print g;"Girls  ";b;"Boys"
90 goto 130
100 next b
110 next g
120 print "No integer solution from 0 to 100."
130 print i;"Trials."
```

```
Program is SLOW, be patient...
40 Girls   50 Boys
3950 Trials.
```

The computer program uses two FOR loops (Statements 30-110 and 40-100) to try every combination of values for B and G between 1 and 100 until a solution is found or until the program runs out of values. The variable I (Statement 50) is a counter which records the number of trials required to reach a solution.

The program is straightforward and finds a solution after 3950 trials. However, it would have been a simple matter to substitute the value of B from the first equation in the second one and quickly solve the problem by hand or with the aid of a calculator. It is important to recognize that if a problem can easily be solved by other methods, the computer offers little or no advantage.

Try this problem. You may or may not want to use your computer. If Matthew can beat Jeff by one-tenth of a mile in a two-mile race and Jeff can beat Steven by one-fifth of a mile in a two-mile race, by what distance could Matthew beat Steven in a two-mile race? (Hint: the answer is not 3/10 mile.)

Brown's Books

The use of a trial and error approach can generally be improved significantly if the combinations to be tried can be narrowed down in some way. The solution to this problem illustrates how the speed of obtaining a solution can be improved well over 100 fold by combining equations and eliminating certain solution possibilities.

Here is the problem. Brown sold 48 books at a flea market, some for $3 each, some for $5 each and others for $8. He collected a total of $175. He remembered having an even number of $5 books. Can you determine how many of each kind of book he had?

The equations for solution are (letting T equal the number of $3 books, F the number of $5 books, and E the number of $8 books):

$$T + F + E = 48$$

$$3*T + 5*F + 8*E = 175$$

The first program was written simply to try all possible combinations of T, F, and E from 1 to 48. It yields three solutions for the problem, although the two solutions with an odd number of $5 books can be eliminated leaving just the one desired solution.

```
5 print chr$(14):rem switch lower case"
10 print "Brown's Books."
20 print "L-O-N-G run time"
40 print "$3    $5    $8"
50 for t=1 to 48
60 for f=1 to 48
70 for e=1 to 48
80 if (t+f+e)<>48 then 110
90 if t*3+f*5+e*8<>175 then 110
100 print t;tab(4);f;tab(10);e
110 next e
120 next f

Brown's Books.
L-O-N-G run time
$3    $5    $8


40    3     5
37    8     3
34    13    1
```

This program took approximately 22 minutes and 13 seconds to run on the Commodore 64 computer. A typical minicomputer (PDP-8/e) could run this problem in about 7.3 seconds. In either case, this is a long time to tie up the computer.

It is rather easy to combine the two equations into one by solving for T. The single equation is then:

$$2*F + 5*E = 31$$

In this equation, the limits can be reduced (from 48 used in the first run) since F cannot possibly be greater than 31/2 or 15.5 and E cannot be greater than 31/5 or 6.2. Making the appropriate program modifications leads to the second program.

```
20
50
60 for f=1 to 15
70 for e=1 to 6
80 if (f*2+e*5)<>31 then 110
90 t=48-f-e
130

Brown's Books.
L-O-N-G run time
#3     #5     #8


40     3      5
37     8      3
34     13     1
```

Using this program produces a dramatic improvement in the time to solution. On the Commodore 64, the time is approximately 2.5 seconds and on the PDP-8, about 0.16 seconds.

Since the problem states that F must be even, a final modification which steps F by two in Statement 20, can be made. This version of the program takes only 1.25 seconds to run on the Commodore 64 and 0.06 seconds to run on the PDP-8.

```
30 Print "F Even"
60 for f=2 to 14 step 2

Brown'S Books.
F Even
#3     #5     #8
37     8      3
```

Notice the enormous improvement in computing time required for a solution, over 1000 fold on the Commodore 64 and 100 fold on a PDP-8. Brute force certainly is inefficient! It is generally worthwhile to think through most problems, particularly big ones, before rushing to the computer. The computer may be fast, but we just improved its performance by 1000 times by using a little common sense.

Intersection of Sets

Two sets of numbers can be combined to yield a third set by the operation of intersection. The intersection of two sets A and B is the set that contains all elements that belong to both A and B. It does not contain any other elements. The intersection is usually written A ∩ B.

For example if M = < 0,2,4,6 > and K = < 1,2,3,4 >, then M ∩ K = < 2,4 >.

This program finds the intersection of two sets of numbers. It has been written to find the intersection of the two repetitive sets described in Statements 30 and 40. In the sample run, Statement 30 describes the set

x = < 1,3,5, . . . 19 > and Statement 40 describes the set

y = < 2,5,8, . . . 29 >.

Notice that successive values of x increase by 2 and y by 3.

```
10 Print chr$(14):rem switch lower case
20 Print "Sets x, y intersect at Points:";
30 for x=1 to 19 step 2
40 for y=2 to 29 step 3
50 if x=y then 90
60 next y
70 next x
80 end
90 Print x;
100 goto 70
```

```
Sets x, y intersect at Points: 5  11  17
```

However, if the set cannot be so neatly described, it may be desirable to rewrite the program to examine any set of data. This is done with the READ statement which reads into x the data points in the DATA statement.The program is set up to use the same y set as the first program, but the x set is defined in the data statement.

You should be able to see from the first combination of sets that if there is a numerical pattern in the sets which intersect, then there is also a pattern in the resulting intersecting set. In the example, the x values increase by 2 and the y values by 3, hence the values in the intersecting set increase by 2 x 3 = 6. Although the intersection of these sets could easily be calculated by hand, the computer can be an aid in evaluating more complicated sets.

```
5 print chr$(14):rem switch lower case
10 print "Sets x, y intersect at Points:";
20 read x
30 for y=2 to 29 step 3
40 if x=y then 80
50 next y
60 goto 20
70 end
80 print x;
90 goto 60
100 data 2,3,8,9,14,15,20,221,26,27


Sets x, y intersect at Points: 2 8 14 20 26
?out of data error in 20
```

Prime Factors

A prime factor is a positive integer that has no factor except itself and one. The first ten prime factors (or numbers) are 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29. The definition gives the basic method for determining whether a number is prime: divide by all smaller integers down to 2, testing whether the remainder is zero for at least one of them. If not, the number is prime.

But this is highly inefficient. It is obvious that a number is not prime if it is any even number greater than 2; hence only odd divisors need to be tried. Also, it is not necessary to try divisors greater than the square root of the number.

Since the division method is inefficient, various schemes have been devised to avoid division. The basic idea underlying all such schemes is called the sieve of Eratosthenes (276 B.C.-195 B.C.). Imagine a list of odd numbers from 3 up. Strike out every third number after 3, every fifth number after 5, and so on. This will leave only prime numbers.

```
5 print chr$(14):rem switch lower case
10 dim a(100)
20 c=0
30 print "Prime Factors"
40 print "Enter 0 to stop"
50 input "Your number";m
60 if m=0 then stop
70 n=m
80 x=0
90 if m>0 then 110
100 print "Invalid.  Again...":goto 50
110 i=1
120 i=i+1
130 if i>m then 190
140 if m/i<>int(m/i) then 120
150 x=x+1
160 a(x)=i
170 m=m/i
180 goto 140
190 if x=1 then 240
200 for l=1 to x
210 print a(l);
220 next l
230 goto 250
240 print n;"is prime.";
260 print
270 goto 50
280 end
```

```
Prime Factors
Enter 0 to stop
Your number 105
 3  5  7
Your number 72
 2  2  2  3  3
Your number 362
 2  181
Your number 89
 89 is prime.
Your number 0
```

The program here finds the prime factors of any integer, or prints out "N is prime" if the integer has no proper divisors.

Run this program for a large number of different integers and see if you can discover relationships between numbers and their prime factors. You should also try to figure out the method employed in the program to find the prime factors of any integer. To do this, you might want to draw a flowchart to show what is happening in the program. This will help you see the method used to find a prime factor and might help you in writing a program to generate primes.

In writing a program to generate prime factors, you can use the sieve method. However, as the numbers become very large, you will have to figure out a way to represent integers with more digits than your computer can handle at one time. (One approach is described on pp. 19-21 of *Computers in Mathematics.*)

Goldbach was a mathematician who made a conjecture that every even number greater than 4 can be written as the sum of two prime numbers ($16 = 11 + 5$, $30 = 17 + 13$, etc.). No one has ever proved it but no one has disproved it either. That is why it is called a conjecture. Can you write a program that will prove or disprove this conjecture? Or how about writing a program to prove Goldbach's conjecture for even numbers up to 50? You should be able to write this program with 12 or fewer statements.

Here is another problem involving prime numbers. Assume a life span of 80 years. In what year of the 20th century (1900-1999) would a person have to be born to have the maximum number of birthdays occurring in prime years? The minimum number?

Greatest Common Divisor

The greatest common divisor of a set of numbers is, as its name implies, the greatest integer that will divide into a set of two or more numbers. For example, the set of numbers 12, 20, and 28 have a greatest common divisor of 4. Nothing larger than 4 will divide evenly into all three numbers.

This program will find the greatest common divisor for any set of integers. To run it, you simply input the number of integers in your set, type them in when requested and let the program calculate the GCD. The heart of the calculation is in Statement 150.

Do you know the meaning of a relatively prime set of numbers? Can you figure out the meaning from the third sample run of the program or from runs of your own? How is a set of relatively prime numbers different from a set of prime factors? Can you find a set of 10 integers that is relatively prime?

```
5 print chr$(14):rem switch lower case
10 dim x(50)
20 print chr$(147)
30 print "Greatest Common Divisor"
35 print
40 input "Numbers in set";n
50 print "Numbers";
60 s=1e25
70 for k=1 to n
80 input x(k)
90 if x(k)>s then 110
100 s=x(k)
110 next k
120 g=0
130 for m=2 to 5
140 for i=1 to n
150 if x(i)/m<>int(x(i)/m) then 180
160 next i
170 g=m
180 next m
190 print "Numbers ";
200 if g>0 then 230
210 print "are relatively prime."
220 goto 240
230 print "GCD is";g
240 input "Another (y,n)";z$
250 if z$="y" or z$="Y" then 35
260 print "Okay.  Bye for now."
```

```
Greatest Commen Divisor


Numbers in set? 3
Numbers? 12
 36
 96
Numbers GCD is 4
Another (y,n)? y

Numbers in set? 3
Numbers? 20
 36
 96
Numbers GCD is 4
Another (y,n)? y

Numbers in set? 3
Numbers? 20
 36
 97
Numbers are relatively prime.
Another (y,n)? n

Okay.  Bye for now.
```

In the last section we discussed prime numbers. Here is an interesting challenge for you involving prime numbers. Until late 1982, the longest progression of prime numbers in which all differed by the same number was 17. Prof. Paul Pritchard in the computer science department at Cornell University wrote a program to determine if there was a longer progression. Using a DEC VAX-11/780, he found the string of 18 numbers shown below. He also discovered fourteen other 17-number progressions and ten 18-number progressions, but none yet with 19 numbers. He believes there is at least one; can you find it?

107928278317	197233324147
117851061187	207156107017
127773844057	217078889887
137696626927	227001672757
147619409797	236924455627
157542192667	246847238497
167464975537	256770021367
177387758407	266692804237
187310541277	276615587107

Cryptarithmetic Problems

Cryptarithmetic or alphametic problems are arithmetic expressions in which the digits are replaced by letters of the alphabet. Each digit is associated with a letter to produce an interesting statement, for example:

```
  SEND
+ MORE
-------
 MONEY
```

If the college student who sent this message to his father needed $106.52 for plane fare home, this was the right message to send since this combination of letters has one unique solution, in particular:

```
  9567
+ 1085
-------
 10652
```

However, if the student let things go to the last moment and was in more of a rush, he might have reworded the message:

```
  WIRE
+ MORE
-------
 MONEY
```

In this case, how much should his dad send? Earlier in the book, trial and error approaches to solving problems were discussed. It was noted that the brute force approach of trying every alternative was sometimes appropriate. Is it in this case?

No! The number of possible alternative solutions is the factorial of the number of different letters in the alphametic expression, i.e., 8! or 40,320. A program to try out every one of these possibilities would run for a l-o-n-g time.

In this case it is much more efficient to apply some common sense to narrow down the number of alternatives. The best approach to this process is to divide up the search space into large classes (or sets), according to a common property shared by members of each class, and then attempt to eliminate entire classes by the method of contradiction.

Consider the "WIRE + MORE = MONEY" problem. Can E = 0? Since E + E = Y, Y must also equal 0, contradicting the fact that Y and E must be different digits. Thus, the entire class of solutions in which E = 0 can be ruled out.

Consider E = 3. Now Y = 6 and there is no carry to the next column. So in this column R + R = E or E + 10 if a carry is involved. But in either case E must be an even number since 2R is always even; this contradicts the assumption that E = 3.

By following this type of classificatory contradiction process for each of the

digits in the order E, R, I, O and N, the computer program will search out all possible solutions to the problem. Unlike the "send more money" problem, the "wire more money" problem has five possible solutions. A smart father would choose the lowest solution and wire his son $103.48.

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "CryPtarithmetic solver...."
30 Print
40 Print "    W  I  R  E"
50 Print "  + M  O  R  E"
60 Print "---------------"
70 Print " M  O  N  E  Y"
90 Print
100 m=1
110 for e=2 to 9
120 y=e+e
130 if y>10 then 160
140 c1=0
150 goto 180
160 c1=1
170 y=y-10
180 for r=0 to 9
190 if r=m or r=e or r=y then 480
200 if r+r+c1=e then 230
210 if r+r+c1=e+10 then 250
220 goto 480
230 c2=0
240 goto 260
250 c2=1
260 for i=1 to 9
270 if i=m or i=e or i=y or i=r then 470
280 for o=0 to 9
290 if o=m or o=e or o=y or o=r or o=i then 460
300 n=i+o+c2
310 if n>=10 then 340
320 c3=0
330 goto 360
340 c3=1
350 n=n-10
360 if n=m or n=e or n=y or n=r or n=i or n=o then 460
370 for w=0 to 9
380 if w=m or w=e or w=y or w=r or w=i or w=o or w=n then 450
390 if o+10<>w+m+c3 then 450
400 Print
410 Print "   ";w;i;r;e
420 Print " + ";m;o;r;e
430 Print " -------------"
440 Print m;o;n;e;y;
445 Print:Print
450 next w
460 next o
470 next i
480 next r
490 next e
500 stop
```

```
Cryptarithmetic solver....

   W I R E
 + M O R E
-------------
 M O N E Y

   9 7 6 2
 + 1 0 6 2
-------------
 1 0 8 2 4

   9 2 7 4
 + 1 0 7 4
-------------
 1 0 3 4 8

   9 5 7 4
 + 1 0 7 4
-------------
 1 0 6 4 8

   9 2 8 7
 + 1 0 8 7
-------------
 1 0 3 7 4

   9 5 8 7
 + 1 0 8 7
-------------
 1 0 6 7 4
```

There are other approaches to solving cryptarithmetic problems, but all of them benefit greatly from reducing the search space as much as possible before putting the problem on the computer. See if you can devise another successful approach and write a program to implement it.

Here are some problems for you to try.

```
   DONALD
 + GERALD
----------
   ROBERT

     ABC
   x DE
-------
     FEC
    DEC
-------
    HGBC

     ONE
     TWO
 + FIVE
--------
   EIGHT

     TWO
   x TWO
-------
   THREE

     THE
   EARTH
   VENUS
  SATURN
+ URANUS
--------
 NEPTUNE

   FORTY
     TEN
 + TEN
-------
   SIXTY

   ABCDE
     x 4
---------
   EDCBA

  SPRING
   RAINS
   BRING
+ GREEN
---------
  PLAINS

    FIVE
   -FOUR
---------
     ONE
 + ONE
---------
     TWO
```

VIOLIN + VIOLIN + VIOLA + CELLO = QUARTET

THREE + NINE = EIGHT + FOUR

Sailors and Monkey Problem

There are many variations of the sailors and monkey problem. Here is one of them.

Five sailors and a monkey were on an island. One evening the sailors rounded up all the coconuts they could find and put them in a large pile. Being exhausted from working so hard, they decided to wait and divide them up equally in the morning. During the night, a sailor awoke and separated the nuts into five equal piles, but had one nut left over which he gave to the monkey. He took one pile, hid it, and pushed the other four together and went back to sleep. He was followed in this action by the other four sailors, each of whom did exactly the same thing. Next morning the remaining nuts were divided equally with one remaining nut going to the monkey. What is the smallest number of coconuts with which they could have begun?

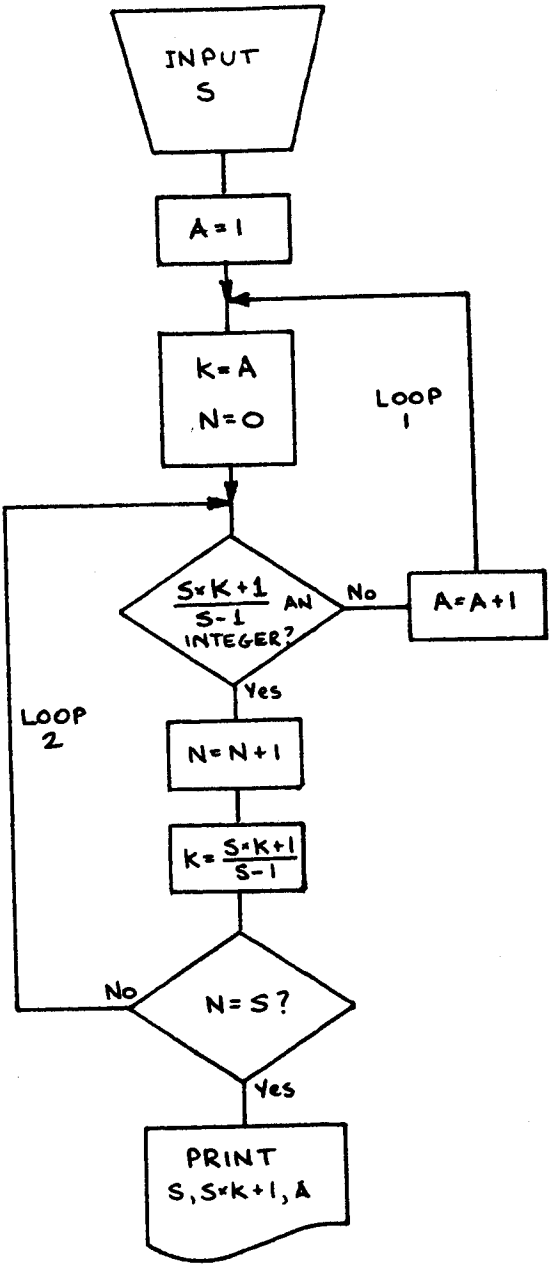
Although there is an elegant algebraic solution to this problem, a more suitable approach for the computer is that of working backwards. A typical solution to a problem can be thought of as a path that leads from the given information to the goal. However, in this case the goal, or final state, is known, thus it is easier to start there and work backwards to the initial state.

As mentioned at the outset, many sailor and monkey problems exist, in fact, an infinite number of them. For example, instead of five sailors, there could be three or six or 14. Thus it is desirable to devise a general solution instead of just one to solve one specific problem.

In the flowchart and computer program, S is the number of sailors and A is the number of coconuts that each sailor received in the final division of the pile. Since one coconut was given to the monkey at each division, the total number of coconuts left in the morning must be S x A + 1. But this pile came from pushing together S - 1 equal piles. Thus, the key condition that must hold for (S x A + 1) / (S - 1) to be an integer K, which represents the number of coconuts that the last sailor stole from a pile of S x K + 1 coconuts. But this pile is the result of pushing together S - 1 equal piles by the previous thief, so again (S x K + 1) / (S - 1) is an integer and so on back through all S raids on the pile.

Note in the flowchart (and program) that the first trial value for A is 1 (Statement 40). In Statement 80 this value is increased by 1 until the value of (S x K + 1)/ (S - 1) is an integer as tested for in Statement 70. This process is then continued until the counter for the second loop, N (nighttime pile divisions) equals the number of sailors.

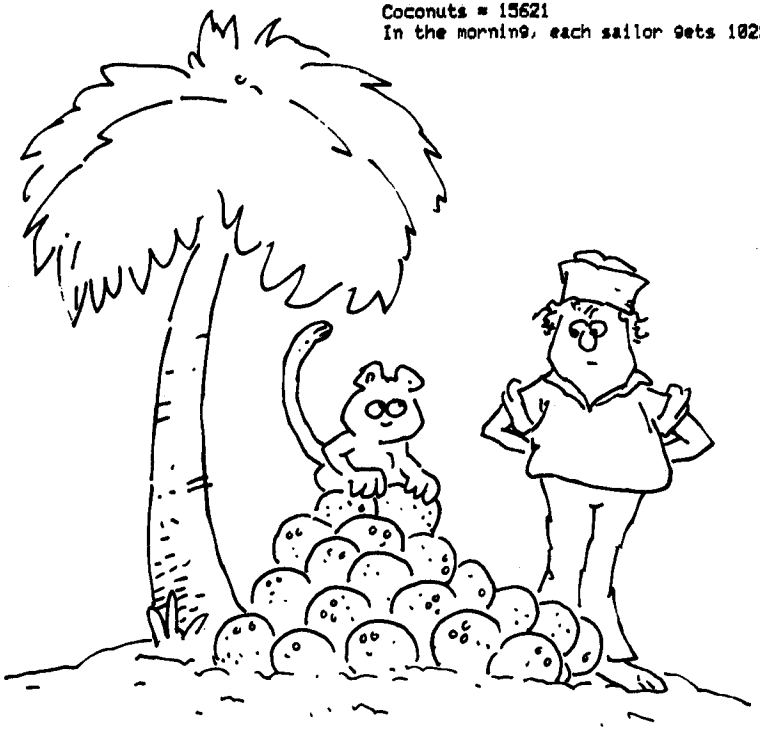
Although the program will work tor any number of sailors, it takes a fairly long time to run for more than five. Remembering what you have learned earlier in this chapter, can you devise a way to make the program more efficient?


INPUT
S
A=1
k=A
N=0
LOOP
1
(S*K+1)/(S-1) AN INTEGER?
No
A=A+1
Yes
LOOP
2
N=N+1
k=(S*K+1)/(S-1)
No
N=S?
Yes
PRINT
S, S*K+1, A

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "Sailors & Monkey"
30 inPut "No. sailors";s
40 a=1
50 k=a
60 n=0
70 if (s^k+1)/(s-1)=int((s^k+1)/(s-1)) then 100
80 a=a+1
90 9oto 50
100 n=n+1
110 k=(s^k+1)/(s-1)
120 if n=s then 140
130 9oto 70
140 Print "Coconuts =";s^k+1
150 Print "In the mornin9, each sailor 9ets";a;
```

```
Sailors & Monkey
No. sailors? 3
Coconuts = 79
In the mornin9, each sailor 9ets 7
```

```
Sailors & Monkey
No. sailors? 5
Coconuts = 15621
In the mornin9, each sailor 9ets 1023
```



Super Accuracy

Under normal circumstances, your computer performs computations to six or seven digits of accuracy. Double precision computations increase accuracy to 13 digits or so.

However, it is possible to do computations one digit at a time and assign each digit to an element in an array. This will achieve virtually any desired accuracy. Any, up to the maximum array size that is.

This example program performs the rather simple operation of successively doubling a number (which is the same as raising 2 to a power).

If the number to be represented is 8192, then:

A(4)	A(3)	A(2)	A(1)
8	1	9	2

To add this to itself, first the rightmost digits are added: A(1) + A(1). If there is a carry, the variable C is set equal to 1, otherwise C is 0. The total is put into B in Line 100.

If B is less than 10, there is no carry (C=0) and the new A(1) equals B. If B is greater than 10 there is a carry (C=1) and the new A(1) equals B-10.

This operation is continued for all the digits (D) of the number and, when it is finished, the new number is printed in Lines 190-210.

If A(N) is printed followed by a semicolon(;) for tight packing, the Basic print routine would leave a space in front of each digit (for the sign) and a space after each digit (for readability). In the program here, these spaces are not wanted, hence the print routine in Line 200 is used which prints the string value of the ASCII value of each digit (which is the same as the digit itself) but without the spaces.

This program, incidentally solves the challenge to calculate the number of moves in the Towers of Brahma problem (see "Change for Any Amount to $5.00). The approach is also used in the next section, "Palindromes."

```
5 Print chr$(14):rem switch lower case
10 Print "ComPutes 2 to Nth to any accuracy":Print
20 dim a(100)
30 m=0:c=0
40 d=0
50 for i=1 to 50:a(i)=0:next i
60 a(1)=1
70 i=0:c=0
80 m=m+1
90 i=i+1
100 b=a(i)+a(i)+c
110 if b<10 then c=0:goto 140
120 b=b-10
130 c=1
140 a(i)=b
150 if i<d then 90
160 if c=1 then 90
170 Print m;
180 d=i
190 for n=i to 1 step -1
200 Print chr$(a(n)+48);
210 next n
220 Print
230 goto 70
```

```
ComPutes 2 to Nth to any accuracy

 1  2                          34  17179869184
 2  4                          35  34359738368
 3  8                          36  68719476736
 4  16                         37  137438953472
 5  32                         38  274877906944
 6  64                         39  549755813888
 7  128                        40  1099511627776
 8  256                        41  2199023255552
 9  512                        42  4398046511104
10  1024                       43  8796093022208
11  2048                       44  17592186044416
12  4096                       45  35184372088832
13  8192                       46  70368744177664
14  16384                      47  140737488355328
15  32768                      48  281474976710656
16  65536                      49  562949953421312
17  131072                     50  1125899906842624
18  262144                     51  2251799813685248
19  524288                     52  4503599627370496
20  1048576                    53  9007199254740992
21  2097152                    54  18014398509481984
22  4194304                    55  36028797018963968
23  8388608                    56  72057594037927936
24  16777216                   57  144115188075855872
25  33554432                   58  288230376151711744
26  67108864                   59  576460752303423488
27  134217728                  60  1152921504606846976
28  268435456                  61  2305843009213693952
29  536870912                  62  4611686018427387904
30  1073741824                 63  9223372036854775808
31  2147483648                 64  18446744073709551616
32  4294967296                 65  36893488147419103232
33  8589934592                 66  73786976294838206464
```

Palindromes

A palindrome is a word, verse, or number that reads the same backwards or forwards. For example, the words "mom" and "eye" are palindromes. So are each of the lines in this verse:

Egad, a base life defiles a bad age
Doom an evil deed, liven a mood
Harass sensuousness, Sarah
Golf; No, sir, prefer prison-flog
Ban campus motto, "Bottoms up, MacNab"

Numeric palindromes are those numbers which read the same backward as forward. The examination of these numbers is a field rich with possibilities for creative computing.

One conjecture concerning palindromes raises an interesting unanswered question. Begin with any positive integer. If it is not a palindrome, reverse its digits and add the two numbers. If the sum is not a palindrome, treat it as the original number and continue. The process stops when a palindrome is obtained. For example, beginning with 78:

```
   78
+  87
-----
  165
+561
-----
  726
+ 627
-----
 1353
+3531
-----
 4884
```

The conjecture, often assumed true, is that this process will always lead to a palindrome. And indeed that is just what usually happens. Most numbers less than 10,000 will produce a palindrome in less than 24 additions. But there is a real thorn in the side of this conjecture, the number 196. Can you determine if a palindrome will ever be produced with a starting number of 196?

The number 196 will produce 1675 after two reversals, but after 100 reversals the resultant sum has 47 digits and is still not palindromic. Why mention 1675? Because ten other numbers under 1000 will also lead to the sum of 1675 and thus may not become palindromic. The first five of these numbers are 196, 295, 394, 493, and 592. What are the other five?

The program here will accept any number as a starting value and complete the process of adding the successive reversals and testing if the sum is a palindrome. Try it with some numbers and see if you can identify any patterns.

```
5 Print chr$(14):rem switch lower case
10 Print "Tests conJecture that addin9 reversed di9its will Produce a Palindrome
20 Print
30 dim b(50)
40 Print
50 inPut "Enter number";a
60 e=0
70 e=e+1
80 a=a/10
90 if int(a)>0 then 70
100 for c=e to 1 steP -1
110 a=a%10
120 b(c)=int(a-10%int(a/10))
130 next c
140 d=0
150 for c=1 to int(e/2)
160 if b(c)=b(e+1-c) then 180
170 d=1
180 next c
190 for c=e to 1 steP -1
200 Print chr$(b(c)+48);
210 next c
220 if d=1 then 260
230 Print " Palindrome!"
250 9oto 40
260 Print "   Not yet"
270 if e/2<=int(e/2) then 290
280 b(int(e/2)+1)=2%b(int(e/2)+1)
290 for c=1 to int(e/2)
300 b(c)=b(c)+b(e+1-c)
310 next c
320 for c=1 to int(e/2)
330 b(e+1-c)=b(c)
340 next c
350 b(e+1)=0
360 for c=1 to e
370 b(c+1)=b(c+1)+int(b(c)/10)
380 b(c)=b(c)-10%int(b(c)/10)
390 next c
400 if b(e+1)<=0 then 140
410 e=e+1
420 9oto 140
```

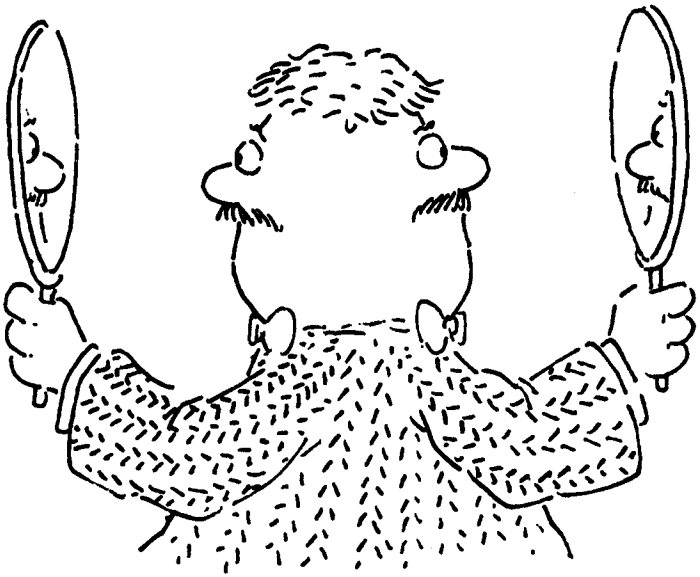
```
Tests conJecture that addin9 reversed di9its will Produce a Palindrome

Enter Number 19
19   Not yet
110   Not yet
121 Palindrome!
```

```
Enter Number 96
96   Not yet
165   Not yet
726   Not yet
1353   Not yet
4884 Palindrome!
```

```
Enter Number 196
196   Not yet
887   Not yet
1675   Not yet
7436   Not yet
13783   Not yet
52514   Not yet
94039   Not yet
187088   Not yet
1067869   Not yet
10755470   Not yet
18211171   Not yet
```

Using this method, write a program that examines all the integers between 1 and 10,000 excluding those that sum to 1675 at any point. What does this show? By the way, you will have to devise a way to deal with 14-digit integers which are larger than your computer can normally handle.

Huh? Is this program "too hot to hoot?"



4

Convergence and Recursion

The computer is especially suitable for doing repetitive and tedious calculations. Two mathematical approaches for solving problems that involve repetitive calculations are convergence and recursion.

Some problems can be reasonably easily stated in words or described with a few simple equations but there are many possible solutions. For example, how many ways can you make change for a dime? It is simply stated and the number of ways can be enumerated fairly easily: two nickels, one nickel and five pennies, or ten pennies, three ways in all. But if you want to solve for all the ways of making change for a dollar or five dollars, it would be nice to have some help.

Help on this kind of problem comes from a class of computer program that simply breaks the problem into smaller ones and counts up all the alternative solutions according to a set of rules. But an even more powerful technique is known as recursion. Using this technique, a simple solving algorithm or routine is set up to solve the smallest subset of the problem. The unique power in a recursive routine comes from the ability of the routine actuallly being able to call itself. This is discussed further in the second program in this section.

Another approach for solving problems that do not have an exact answer is that of successive approximations. For example, the exact value of pi, *e* or the length of an irregular curve cannot be precisely determined. But by means of increasingly accurate approximations, it is possible to approach the desired value from above or below or to converge on it from two directions. The last four programs in this chapter illustrate successive approximations and convergence.

Change For a Dollar

Even though there is very little you can buy for a penny these days, the coin will probably be around for some time to come since it is needed to make change for odd amounts of sales tax and to fill up penny collections.

Today U.S. coinage consists of five coins: penny, nickel, dime, quarter, and half dollar. How many ways can coins of these denominations be used to make change for one dollar? For example, one way is two half dollars, another is one half dollar and two quarters, and so on. Make a best guess now and write it down before you read further.

There are several different ways to approach a problem of this kind. One is to break it down into smaller, more easily solved problems. In other words, how many ways can you make change for a quarter? For a dime? You would solve these subproblems and combine the answers to give the overall solution.

If you were more mathematically inclined, you could write a series of equations relating each piece of change to every other one and to the dollar and solve them.

A third approach is to do the problem by writing down combinations until all the different possibilities are exhausted (or until you are exhausted) and then count them all up. This might be called solving the problem by exhaustion and is a method quite suitable for putting on the computer.

Write a program that uses this approach to solve the problem. If you use loops and count by one, it could take a long time for the computer to run through all the possible combinations, possibly many hours.

Also, if you want to print out all the possible combinations, be warned that the printing could also take quite some time and a fair amount of paper. There are more combinations than you might think!

In fact, most people will not be able to guess the answer to this problem, or even come close. Ask several of your friends how many ways they think a dollar can be changed. Record all the responses and then tabulate them on your computer. What is the mean (average) of all the guesses? The extremes?

The program included here uses the first method discussed to solve the problem, in particular, breaking down the problem into subproblems and then combining the solutions into one final answer.

First, the main problem is broken into the next smaller one of making change for half dollars. There are three such problems: no half dollars (H = 0), one half dollar (H = 1), and two half dollars (H = 2). The last problem is trivial since there is only one way, but the other two need to be broken down further.

This is done by dividing the remaining money into quarters and considering the subproblems on down to the lower denominations. As the number of subproblems is expanded, each one becomes easier to solve. In fact,

subgoals, which can be solved in only one way, are finally reached. For example, if H = 1, Q = 1, D = 2, and N = 0, then the pennies (P) must equal five in order that the total equal 100.

Notice that at the quarter, dime and nickel stages, adjustments are made in the limits of the loops depending upon how much money there is left to change. For example, if H = 1, the only possible subgoals for quarters are 0, 1, and 2, but not 3 or 4. Also notice that there is no need to test combinations of coins to see if they add up to 100, nor is it necessary to include the penny as a variable. Simply counting the number of subgoals is sufficient since each one can be solved in only one way.

```
5 print chr$(14):rem switch lower case
10 c=0
20 for h=0 to 2
30 for q=0 to 4-2*h
40 for d=0 to 10-5*h-2.5*q
50 for n=0 to 20-10*h-5*q-2*d
60 c=c+1
70 next n
80 next d
90 next q
100 next h
110 print "A dollar can be changed in";c;"different ways."

A dollar can be changed in 292 different ways.
```

Try to make some changes in this program or write a new one to solve the following problems. Say you want two quarters in your change to play some video games. In how many ways can a dollar be changed to provide at least two quarters?

Visiting a small town, you find the parking meters still take pennies. In how many ways might you get change so that you had at least three pennies? Is this any different than the number of ways that would give you five pennies? Say you want to make a phone call also; in how many ways can you change a dollar to produce at least four pennies and one dime?

Change For Any Amount to $5.00

Another way to attack the change problem in the previous section is by means of the programming technique called recursion. Get familiar with this one—it is very powerful! Donald Piele and Larry Wood described this method in an issue of *Creative Computing.*

First, define the variables which represent the number of ways to make change for n cents using the coins specified:

A Only pennies
B Nickels and pennies
C Dimes, nickels, and pennies
D Quarters, dimes, nickels, and pennies
E Halves, quarters, dimes, nickels, and pennies.

Initially, there are two subproblems in making change for n cents. In the first, no half dollars are used, and D is the number of ways to change n cents. Second, when one or more halves are used, after one is paid, there remain n–50 cents to pay which can be done in E_{n-50} ways.

Since these two cases are mutually exclusive, it can be inferred that $E_n = D_n + E_{n-50}$. Similarly,

$$D_n = C_n + D_{n-25}$$
$$C_n = B_n + C_{n-10}$$
$$B_n = A_n + B_{n-5}$$

Now, begin with the simplest case and build up to E_{100}. First of all, it is easy to understand why $E_0 = 1$. From above, when $n = 50$, $E_{50} = D_{50} + E_0$, and it is possible to make change for 50 cents only one more way if half dollars are allowed. Therefore $E_0 = 1$. Likewise, $D_0 = C_0 = B_0 = A_0 = 1$. It is also true that $A_n = 1$ for all values of n since there is only one way to make change using only pennies. Now the recursive relationships can be used to solve the original problem.

This is the strategy used in the program. It also has the added advantage that it can count the number of ways of making change (with coins) for any specified amount.

Now it is your turn. Can you modify the program here to include dollar bills so it could count the number of ways to make change for any amount up to $10.00?

Using any method of change making you prefer, write a program to make change for one ruble. Russian coins come in denominations of 1, 2, 3, 5, 10, 15, 20, 50, and 100 kopecks. There are 100 kopecks in one ruble.

```
5 Print chr$(14):rem switch lower case
10 dim a(101),b(101),c(101),d(101),e(101)
15 Print
20 inPut "Amount to be chan9ed";x
30 m=int(20#x)+1
40 a(1)=1:b(1)=1:c(1)=1:d(1)=1:e(1)=1
50 for J=2 to m
60 a(J)=1
70 b(J)=a(J)+b(J-1)
80 c(J)=b(J)
90 if J<=2 then 110
100 c(J)=b(J)+c(J-2)
110 d(J)=c(J)
120 if J<=5 then 140
130 d(J)=c(J)+d(J-5)
140 e(J)=d(J)
150 if J<=10 then 170
160 e(J)=d(J)+e(J-10)
170 next J
180 Print "You can make chan9e in";e(m);"ways."
190 9oto 15
```

```
Amount to be chan9ed? 1.13
You can make chan9e in 384 ways.

Amount to be chan9ed? 4.25
You can make chan9e in 32985 ways.

Amount to be chan9ed? .1
You can make chan9e in 4 ways.

Amount to be chan9ed? .12
You can make chan9e in 4 ways.
```

Perhaps the most famous problem used to demonstrate the principles of recursion is the Towers of Brahma. It is sometimes called the Towers of Hanoi or Pharoah's Needles. Here is the problem in as close to original form as possible. You should be able to solve it with a relatively short program using recursion.

In the great temple at Benares beneath the dome which marks the center of the world rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the Creation, God placed 64 discs of pure gold, the largest disc resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma.

Day and night unceasingly, the priests transfer the discs from one needle to another, according to the fixed and immutable laws of Brahma. These laws require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so there is no smaller disc below it. When the 64 discs shall have been thus transferred from the needle which, at the Creation, God placed them, to one of the other needles; tower, temple, and Brahmans alike will crumble into dust, and with a thunderclap, the world will vanish.

If the priests were to effect one transfer every second, and work 24 hours per day for each day of the year, it would take them 58,454,204,609 decades plus slightly more than six years to perform the feat, assuming they never made a mistake—for one small slip would undo all the work.

How many transfers are required to fulfill the prophecy? Try out your program with fewer discs than 64 to make sure you are on the right track. Here is a table of the first few transfers:

Discs	*Moves*	*Discs*	*Moves*
1	1	6	63
2	3	7	127
3	7	8	255
4	15	9	511
5	31	10	1023

Converge on e and Pi

An incredibly important mathematical constant is designated by the small letter *e*. This constant is both irrational and transcendental. Look up those terms in a dictionary or math book if you wish, or just plunge on to the next paragraph.

The constant *e* was first derived by John Napier, also the inventor of logarithms, to whom we owe an eternal debt of gratitude. Why? If *e* had never been discovered, advances in mathematics, physics, and astronomy would have lagged a century or more, because *e* is the base of all natural logarithms and these logarithms are the basis for many branches of science and mathematics.

How is *e* calculated? The constant *e* is the limiting value of this expression as n approaches infinity:

$$e = (1+1/n)^n$$

Its exact value can never be found, but to 15 places *e* equals 2.718281828459045 . . . How can *e* be calculated? First take 1-½ and square it; that equals 2-¼. Then cube 1-⅓ and you get 2.3686. Raising 1-¼ to the fourth power gives 2.414, and so on. Write a program for this method and have it print out the initial value of *e* and each value after each additional fraction is added on.

Another approach is to expand the expression above using the binominal theorem and, again, letting n approach infinity. The expression for the expansion is:

$$e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} \cdots \frac{1}{n!}$$

Now here is where the computer can again be of some assistance. Since 3! is 3*2! and 4! is 4*3!, all the calculations need not be done for each additional fraction. Look at the program and particularly note the calculations in Statements 70 and 90.

```
5 Print chr$(14):rem switch lower case
10 Print "Converse on e"
20 e=1
30 i=0
40 i=i+1
50 d=1
60 forJ=1 to i
70 d=d*J
80 next J
90 e=e+1/d
100 Print e
110 Goto 40
```

```
Converse on e
 2
 2.5
 2.66666667
 2.70833334
 2.71666667
 2.71805556
 2.71825397
 2.71827877
 2.71828153
 2.7182818
 2.71828183
 2.71828183
 2.71828183
 2.71828183
 2.71828183
```

Pi is another important mathematical constant that is irrational (meaning its exact value can never be determined) and transcendental (meaning it is not the solution to any algebraic equation). Interestingly, pi was known and used by the ancients. Archimedes, who lived in the second century B.C., by using a regular polygon of 96 sides (nearly a circle), proved that the value of pi was less than 22/7 and greater than $3\text{-}^{10}/_{71}$, a remarkable achievement for the mathematics of his day.

Ptolemy in 150 A.D. used the value of 3.1416 for pi and in the middle of the sixteenth century the amazing fraction 355/113 was discovered, giving the value of pi accurately to six decimal places.

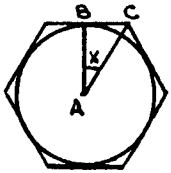
Incidentally, in 1897, the General Assembly of Indiana passed a bill ruling that the value of pi was four.

Several infinite series can be used to grind out increasingly accurate values for pi. One such series is $(1\text{-}^1/_3 + {}^1/_5 - {}^1/_7 + {}^1/_9 \ldots)$. This series is called an arithmetic series and converges very slowly. The program here displays only every 500th value of the series. Don't be alarmed if the program does not seem to be running very fast; a fair amount of calculating is going on between each value that is printed.

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "Converse on Pi Arithmetic series";
30 s=1
40 i=1
50 q=0
60 P=0
70 q=q+1
80 P=P+s/i
90 i=i+2
100 s=-s
110 if q<499 then 70
120 q=0
130 Print:Print P#4;
140 goto 70
```

```
Converse on Pi Arithmetic series
 3.14359667
 3.14059066
 3.14226067
 3.14109167
 3.14199346
 3.14125867
 3.14187896
 3.14134218
 3.14181534
 3.14139227
 3.14177485
 3.14142568
```

Actually, the approach used by Archimedes converges much more quickly and, with the aid of a computer, it is possible to go far beyond the 96-sided polygon used by Archimedes.



His approach was to construct inscribed and circumscribed polygons and measure the perimeters to approximate the circumference of a circle. Consider a polygon circumscribed around a circle of radius 1. The perimeter equals the length of one side times the number of sides. Since the tangent of x = AB/BC, but BC = 1, then tan (x) = AB and the length of a side = 2 tan (x). Since the circumference = 2 pi r and r = 1, then pi is the circumference (or perimeter of an n-sided polygon) divided by 2.

Similar trigonometry leads to the perimeter of an inscribed polygon being equal to the number of sides times sin (x) * cos (x).

The second program produces values for pi using inscribed and circumscribed polygons. Unfortunately, there is one large flaw in the program, because degrees must be converted into radians in Statement 40. This means, of course, that you must already know the value of pi, since the conversion factor is 360 degrees divided by 2 pi.

Setting this flaw aside, it is interesting to note how quickly this program converges on the value of pi compared to the preceding one. That is because this one converges geometrically rather than arithmetically.

Can you figure out a geometric convergence to the value of pi that does not require that you know it (or a conversion factor) before you start?

```
5 Print chr$(14):rem switch lower case
10 Print "Converse on Pi by Poly9ons.":Print
15 Print" Inscribed    Circumscribed":Print
20 n=6
30 n=2*n
40 x=360/(n*57.29578)
50 Print n*sin(x)*cos(x)/2;
60 Print tab(15);n*tan(x)/2
70 9oto 30
```

```
Converse on Pi by Poly9ons.

 Inscribed      Circumscribed

2.5980762        3.46410158
2.99999998       3.21539028
3.10582851       3.15965992
3.13262859       3.14608619
3.13935018       3.14271458
3.14103193       3.14187303
3.14145245       3.14166272
```

Convergence on Pi Revisited

In answer to the question posed in the last paragraph of the preceding section, here is a way to converge on pi without knowing its value beforehand.

As in the previous program, the basic approach is to add up the length of the sides on an inscribed polygon and divide by 2r to obtain a value for pi. The program starts with a square (four sides) and doubles the number of sides each time. If the old side length is S, then the length of S' of a side of a new polygon with twice as many sides is obtained by applying the Pythagorean theorem. In particular,

$$X^2 + (S/2)^2 = R^2$$

$$(R-X)^2 + (S/2)^2 = (S')^2$$

Thus,

$$S' = \sqrt{(R-\sqrt{R^2 - (S/2)^2})^2 + (S/2)^2}$$

It is easy to reduce this formula algebraically, but accuracy suffers if this is done. Also, you will find that S*S is slightly more accurate than S↑2.

Unfortunately, striving for maximum accuracy is somewhat moot on a computer that does not have double precision arithmetic. Notice that accuracy does not improve with more than 1024 sides and, indeed, as numbers in the calculations start to exceed the capacity of the computer (4194304 sides), the accuracy starts to deteriorate badly.

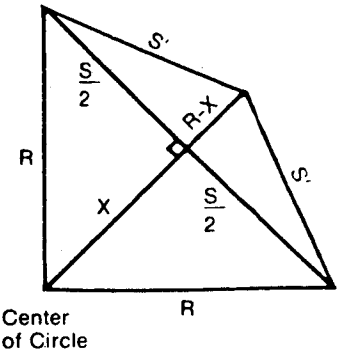
There is yet another method to compute pi by convergence. It uses discoveries of Gregory and Euler. Gregory discovered the formula for arctangent:

$$\text{Arctan } x = x - x^3/3 + x^5/5 - x^7/7 + \ldots$$

Euler came up with a rather interesting formula for pi:

$$\text{pi} = 4\,(\arctan(1/2) + \arctan(1/3))$$

See if you can combine these two formulas to calculate pi. If you are very clever, you can do your calculation to yield far more than the seven decimal place accuracy obtained by the programs presented here so far.



```
5 Print chr$(14):rem switch lower case
10 Print "Converse on Pi by inscribed Poly9ons"
20 Print "Sides      Perimeter"
30 r=10
40 z=2
50 n=z+z
60 s=r*sqr(z)
70 for k=1 to 28
80 Print n;tab(10);(s*n)/(z*r)
90 y=s*s/(z*z)
100 x=r-sqr(r*r-y)
110 s=sqr(x*x+y)
120 n=n*z
130 next k
```

```
Converse on Pi by inscribed Poly9ons
Sides           Perimeter
 4             2.82842713
 8             3.06146746
 16            3.12144515
 32            3.13654849
 64            3.14033116
 128           3.14127725
 256           3.1415138
 512           3.14157294
 1024          3.14158773
 2048          3.14159142
 4096          3.14159235
 8192          3.14159258
 16384         3.14159264
```

Length of Any Curve

The previous programs have demonstrated how it is possible to compute a very accurate value of pi by adding together the length of the sides of a polygon as it approaches a circle and dividing by 2r.

Using a similar approach, it should be possible to inscribe a polygon, or portions of a polygon, inside any regular curve and thus determine the length of the curve. This program approximates the length of any curve as defined in Statement 100 by dividing it into an increasing number of subintervals and computing the sum of the secants (a straight line that cuts a curve at two or more points).

To run the program, you must enter the formula or equation describing your curve in Statement 100 in the form:

100 DEF FNA(X) = your function of X

You then type RUN and enter the end points of the curve you want to use in your calculation. These points are entered in the form of the abscissa, which means the horizontal (or y) coordinate of the point.

The program is written to sum the successive secant lengths and to calculate the percent of change in each summation compared to the preceding one. The sample run uses the function $2x^3 + 3x^2 - 2x + 3$. Note the substantial improvements in the length calculation as the number of intervals increases from 2 to 16, but the rather slight improvements beyond 16.

Try this program with different curves and functions. It might help to plot the function first (remember the program to do that?) and then compute its length. Is this method of length calculation more accurate for a function with no changes in direction of the curve within the interval selected or for a function with one or more changes? Why?

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "Curve length."
30 Print "Define function in line 100"
40 Print
50 Print
60 inPut "Abscissas of end Points";P,q
70 Print
80 Print "Inter  Secant     % change"
85 Print "Vals   Length     in length"
90 Print
95 s1=0
100 def fna(x)=2*x^3+3*x^2-2*x+3
110 for n=1 to 9
120 e=2^(n-1)
130 h=(q-P)/e
140 s=0
150 for i=0 to e-1
160 l=sqr((fna(P+i*h+h)-fna(P+i*h))^2+h*h)
170 s=s+l
180 next i
190 if s1>0 then 220
200 Print e;tab(5);s;" No Previous value"
210 goto 240
220 P5=((abs(s1-s))/s1)*100
230 Print e;tab(5);s;tab(7);P5
240 s1=s
250 next n
```

```
Curve length.
Define function in line 100


Abscissas of end Points? -1 , 6

Inter  Secant          % change
Vals   Length          in length

 1     525.046666      No Previous value
 2     525.158263        .0212546484
 4     529.652248        .855739211
 8     531.017134        .257694752
 16    531.964256        .178360033
 32    532.016566       9.83336535e-03
 64    532.041679       4.72022717e-03
 128   532.048562       1.29376759e-03
```

Converge on a Square Root

Since most square roots are irrational, methods used to calculate them usually involve successive approximations. Although you can simply call up the square root function in Basic or on many pocket calculators, it is interesting to explore various methods of calculating square roots without these built-in functions. After all, these built-in functions are nothing more than successive approximation routines already installed in the machine.

Obviously, a square root is the inverse of the operation of squaring a number. All of the methods of calculating square roots use this fact, but the way in which it is employed is quite different in various calculators and computers.

The program here calculates an upper and lower limit for the square root of a number and, by successive approximations, pinches the root to within a smaller and smaller interval until it reaches the desired level of accuracy.

The starting value for the lower limit is 0 and for the upper limit the number, Z, whose square root is sought. The program then divides this interval into ten steps by simply dividing the difference between the numbers by 10. The variable I is increased from the lower limit to the upper one by the value of the step, S. At any point, if I squared becomes greater than Z, a new upper limit is set to I and a new lower limit is set to I - S.

This method converges very quickly and adds approximately one decimal place of accuracy with each pass beyond the third. What happens when you enter into the program a number that has an exact square root such as 25 or 49? Why?

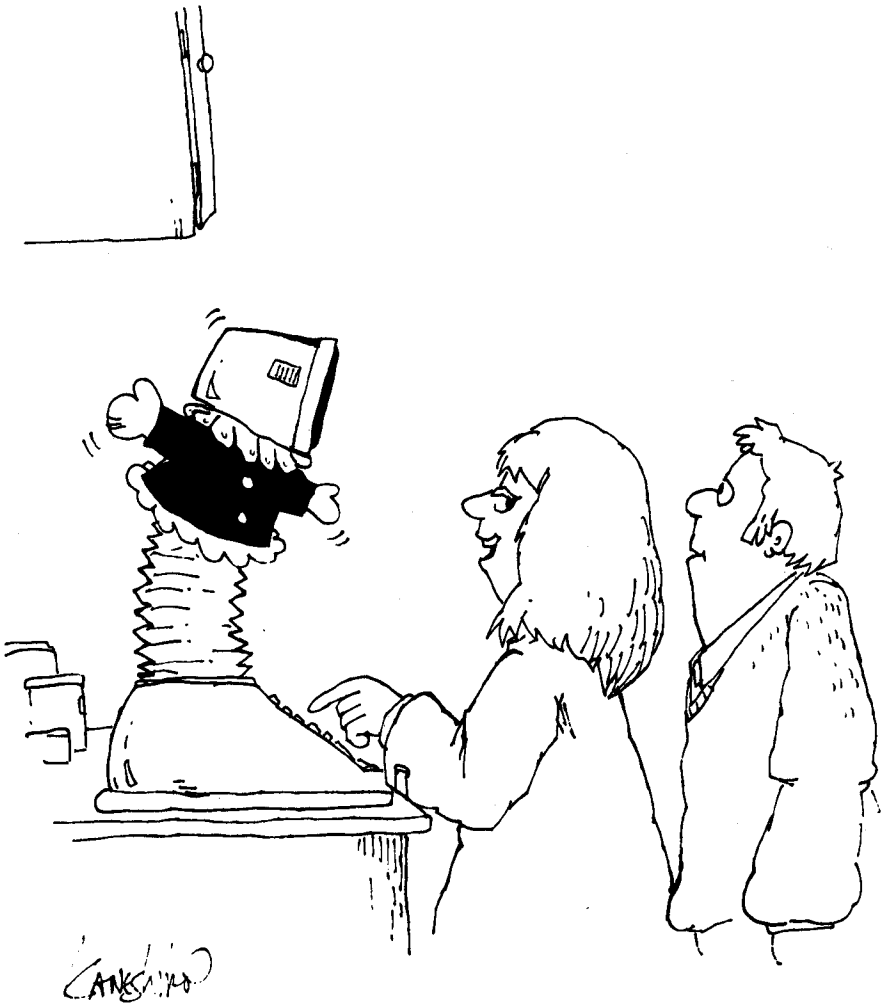
Another approach to calculating square roots by successive approximations is to start with a trial root, X. If X * X is less than the original number N, then increase the trial value by a 0.1. If X * X is greater than N, return to the previous value. This is the first digit of the root. Now, start advancing by 0.01. Continuing in this way, one digit is developed at a time until the desired precision is reached.

This method is quite suitable and fast for square roots of numbers less than 1. A good first trial root value is 0.1. But is it suitable for larger numbers? How should a starting trial value be determined? In this method, especially for numbers greater than 10, the initial trial value for the root matters a great deal in determining the length of time it will take for the calculation to converge. Write a program using this method and compare the speed and accuracy with the program in the book.

```
5 print chr$(14):rem switch lower case
10 print "Square Roots"
20 inPut "Your number";z
30 e=.00001
40 Print "Low Limit  UP Limit"
50 a=0
60 b=z
70 s=(b-a)/10
80 Print a;tab(10);b
90 if abs(a*b-z)<e then 180
100 for i=a to b steP s
110 if z<i*i then 150
120 next i
130 b=b*10
140 goto 70
150 b=i
160 a=i-s
170 goto 70
180 print "Accurate to .00001"
190 Print "Average =";(a+b)/2
```

```
Square Roots
Your number? 54
Low Limit  UP Limit
 0             54
 5.4           10.8
 7.02          7.56
 7.344         7.398
 7.344         7.3494
 7.34832       7.34886
 7.348428      7.348482
 7.3484658     7.3484712
 7.34846904    7.34846958

Accurate to .00001
Average = 7.34846931
```

```
Square Roots
Your number? 3
Low Limit  UP Limit
0             3
1.5           1.8
1.71          1.74
1.731         1.734
1.7319        1.7322

Accurate to .00001
Average = 1.73205
```



"I've just programmed our computer to give surprise birthday parties."

5

Compounding

As with successive approximations and recursion, compounding requires many repetitive calculations. Some compound interest and growth situations can be represented by a formula, but for many problems, solving by repetitive calculations is an excellent approach.

There is nothing magical about compounding; you generally start with an initial amount of money, number of animals, etc. This quantity then grows or diminishes by a certain percentage at set intervals. The new amount is the old amount increased or decreased by this percentage. Repeat this calculation over and over again, and you have solved the problem.

Five different types of problems which involve compounding of some sort are described in this chapter. Try the extra problems that follow some of the programs; you may be surprised at the results.

Indians and Interest

Here is a simple compound interest problem that produces an astonishing answer. The problem has to do with the sale of Manhattan Island to the Dutch for approximately $24 worth of trinkets and beads.

If the $24 that the Indians received in 1626 had been deposited in a bank paying 5-¾% interest compounded annually, how much would it amount to in 1983?

The program here solves this little problem by making use of the formula to calculate compound interest. In particular, if P dollars are invested at an interest rate of R (expressed as a decimal) and compounded N times, then the total amount A is given by the formula:

$$A = P(1 + R)^N$$

How much was gained in 1983 alone? How much was gained in the decade from 1974 to 1983? You can change the value of N in Statement 30 to get the answers to these questions. But a better way might be to enter the ending year with an INPUT statement.

Can you read a number expressed in the E (exponential format)? In a more conventional format, the number is $11,176,500,000 or $11.2 billion.

There are many other ways you can improve the program and make it more suitable for general purpose compound interest calculations. Modify it to accept any starting and ending year, any rate of interest, and any starting principal amount.

The problem as stated is somewhat unrealistic since banks were not paying interest rates of 5-¾% from 1626 to 1983. Change the program to calculate the total amount based on the following interest rates:

Years	*Interest Rate*
1626-1830	1.5%
1831-1870	2.0%
1871-1910	3.0%
1911-1921	3.5%
1922-1929	6.5%
1930-1940	2.3%
1941-1945	3.5%
1946-1960	5.3%
1961-1980	6.5%
1981-1983	9.5%

```
10 P=24
20 r=.0575
30 n=1983-1626
40 a=P*((1+r)^n)
50 Print "Indians got $24 in 1626 for NY."
60 Print "By 1983 at 5.75% it would be $";a;
```

```
Indians got $24 in 1626 for NY.
By 1983 at 5.75% it would be $ 1.11764926e+10
```

Systematic Savings

In the previous program, compound interest was calculated by means of a formula well known to bankers, money lenders and real estate agents. However, if you did not know the formula, how would you calculate interest on money in a savings account by hand or with a calculator?

You would probably begin by multiplying the principal amount P by the interest rate R and adding that amount to the original principal at the start of the second year. Doing that for all the years the money is in the bank will yield a final amount. Why not write a program to perform the calculations in this manner rather than use a formula? Here is such a program.

It is set up for an initial principal amount of 100 (Statement 50), an interest rate of 10% (R = 0.1 in Statement 60) and ten years (N = 10 in Statement 70). Naturally these values could be read in using INPUT statements. The clever calculation in Statement 100 rounds off the amount to two decimal places (dollars and cents).

In contrast with the program in the previous section, this one does not use a compound interest formula, but simply adds the interest each year to the growing principal amount in Statement 90.

```
5 print chr$(14):rem switch lower case
10 print chr$(147);"Calculates interest on $100 invested at 10% for 10 years"
20 print
30 print "At end    Current"
40 print "of year    balance"
50 p=100
60 r=.1
70 n=10
80 for i=1 to n
90 p=p+p*r
100 print i;tab(10);int(p*100+.5)/100
110 next i
```

```
Calculates interest on $100
invested at 10% for 10 years

At end     Current
of year     balance
 1            110
 2            121
 3            133.1
 4            146.41
 5            161.05
 6            177.16
 7            194.87
 8            214.36
 9            235.79
 10           259.37
```

Now, how could this program be modified to allow for a plan of systematic saving? In other words, instead of letting the $100 lie around all lonely while it is compounding, each year you add another $100 to it. With this program, making the modification for systematic savings is easy: Statement 105 is added to add the new deposit each year to the ever growing principal.

```
5 print chr$(14):rem switch lower case
10 print chr$(147);"Calculates interest on $100 invested at 10% every year";
20 print "for 10 years."
30 print "End        Current"
40 print "Yr Invested Bal"
50 p=100
55 c=100
60 r=.1
70 n=10
80 for i=1 to n
90 p=p+p*r
100 print i;tab(4);i*c;tab(11);int(P*100+.5)/100
105 p=p+c
110 next i
```

```
Calculates interest on $100 invested at 10% every yearfor 10 years.
End                    Current
Yr   Invested            Bal
 1       100             110
 2       200             231
 3       300             364.1
 4       400             510.51
 5       500             671.56
 6       600             848.72
 7       700             1043.59
 8       800             1257.95
 9       900             1493.74
10      1000             1753.12
```

In an effort to attract depositors, many banks over the last 20 years have started offering savings and investment accounts in which the deposits are compounded more often than annually. In the early 50's, many banks went to quarterly compounding; in the late 50's, to daily; and in the early 60's, some "competitive" S&L's went to continuous compounding.

How much does more frequent compounding really mean? Modify either of the two programs to compute the interest for more frequent compounding. What is the difference in interest for one year for annual, quarterly, monthly, daily, and continuous (every second) compounding on a principal amount of $1000 invested at 8%? How about for ten years?

Incidentally, if you want to use the formula method in the previous section for this calculation, the formula for P principal invested at R rate compounded N times per year is:

$$A = P(1 + R/N)^N$$

Try this problem which uses the same principles of compounding. Consumer prices rose an average of 8.8% in 1980. While the government keeps trying to bring inflation under control, they don't seem to be meeting with much success. Assuming that prices continue to go up this much (8.8%) every year, how much will a $6000 economy car (1980 dollars) cost in the year 2000? How much will it cost when you are 65 years old?

Systematic Savings Revisited

Using the formula for compound interest and systematic savings, this program will calculate the amount accumulated after a given period of time.

When you run the program, it will ask how much you wish to save each month, the number of compounding periods in a year, the interest rate, and the length of time you wish to continue your systematic savings program. The program will then calculate the total amount at the end of that period.

From the preceding programs, you should be able to see that a systematic savings program is a very effective way to accumulate a nest egg for the future.

```
5 Print chr$(14):rem switch lower case
10 Print "Calculates interest and balance for a   systematic savings Program."
15 Print
20 inPut "Monthly dePosit $";a
40 inPut "ComPounding Periods Per year";b
80 inPut "Interest rate (xx.x)";r
100 r=r/100
110 inPut "No. of years";n
130 Print
140 Print "End             Current"
150 Print "Year Invested  Balance"
160 Print
170 c=12*a
180 P=12*a
190 for i=1 to n
200 P=P*(1+r/b)^b
210 Print i;tab(6);i*c;tab(14);int(P*100+.5)/100
220 P=P+c
230 next i
```

```
Calculates interest and balance for a systematic savings Program.

Monthly dePosit $? 10
ComPounding Periods Per year? 12
Interest rate (xx.x)? 15
No. of years? 5

End                 Current
Year    Invested    Balance

 1        120       139.29
 2        240       300.97
 3        360       488.65
 4        480       706.49
 5        600       959.35
```

```
Calculates interest and balance for a systematic savings program.

Monthly deposit $? 10
Compounding Periods Per year? 1
Interest rate (xx.x)? 8.5
No. of years? 8

End                          Current
Year     Invested            Balance

 1          120              130.2
 2          240              271.47
 3          360              424.74
 4          480              591.04
 5          600              771.48
 6          720              967.26
 7          840              1179.68
 8          960              1410.15


Calculates interest and balance for a systematic savings program.

Monthly deposit $? 10
Compounding Periods Per year? 365
Interest rate (xx.x)? 8.5
No. of years? 8

End                          Current
Year     Invested            Balance

 1          120              130.64
 2          240              272.98
 3          360              427.73
 4          480              596.32
 5          600              779.86
 6          720              979.68
 7          840              1197.23
 8          960              1434.08
```

Try to combine the things that you have learned from the programs in these sections to write a completely generalized program for systematic savings. It should accept the following information as input:

- Initial deposit
- Frequency of periodic deposits
- Amount of each deposit
- Interest rate
- Interest compounding frequency
- Length of time of savings program

Your program should produce output in tabular form showing years (or other time periods), amount invested without interest, total amount with interest, and the interest alone.

Loan Payments

Although saving money in a systematic way is a noble goal, many people frequently find themselves talking to a different bank officer, namely the one in charge of loans.

This program will calculate the payments for a loan for a period of one year or longer. The program asks you to enter the key facets of the loan in question: amount borrowed, annual rate of interest, interval between payments and term of the loan in years.

The sample run shows a relatively short-term loan (2 years) of $3000 at a bargain basement interest rate of 8.5%. The monthly payment of the loan is computed to be $136.37 and the total interest $272.79. Why is the total interest not $255 (8.5% of $3000)? Instead it is 9.09% Is this a mistake in the program?

Try the program for some longer term loans at real world interest rates. For example, run it for an automobile loan of $8000 at 12% over a 5-year period. Perhaps you can see from this that systematically saving your money and paying cash for an item makes more sense than time payments.

Run the program to calculate the mortgage payments on a $150,000 house with a $40,000 down payment. Try it with a 16% interest rate stretched over a 30-year period. Ouch! Look at all that interest!

At the heart of this program are Statements 180-210. Can you see what is happening in these calculations?

```
5 Print chr$(14):rem switch lower case
10 Print "Calculates Payment schedule on a loan."
20 Print
30 inPut "Amount borrowed";a
40 inPut "Interest rate (xx.x)";i
50 inPut "Payment interval (months)";P
60 inPut "Term (years)";y
70 inPut "OutPut table (1) or totals only (2)";v
80 Print
150 if v=2 then 180
160 Print "Period  Interest   PrinciPal"
170 Print "          Due          Due   "
180 z=(y*12)/P
190 k=(i*(P/12))/100
200 e=a*k/(1-1/(1+k)^z)
210 e=int(e*100+.5)/100
220 c=a
230 f=0
240 d1=0
250 t1=0
260 t1=t1+1
270 if t1>z then 380
280 b=t1
290 c=c-f
300 d=c*k
310 f=e-d
320 d=int(d*100+.5)/100
330 f=int(f*100+.5)/100
340 d1=d1+d
350 if v=2 then 260
360 Print b;tab(8);d;tab(19);f
370 goto 260
380 if v=1 then 430
390 d1=int(d1*100+.5)/100
400 Print "PrinciPal $";a
410 Print "Interest $";d1
420 goto 450
430 Print "         ------      -------"
440 Print "Total    ";d1;tab(20);a:Print
450 e5=int((d1+a)*100+.5)/100
460 e6=e5/((y*12)/P)
470 e6=int(100*e6+.5)/100
480 Print "Total $";e5
490 Print "Each Payment $";e6


Calculates Payment schedule on a loan.

Amount borrowed? 3000
Interest rate (xx.x)? 8.5
Payment interval (months)? 1
Term (years)? 2
OutPut table (1) or totals only (2)? 1


PrinciPal $ 3000

Interest $ 272.79
Total $ 3272.79
Each Payment $ 136.37
```

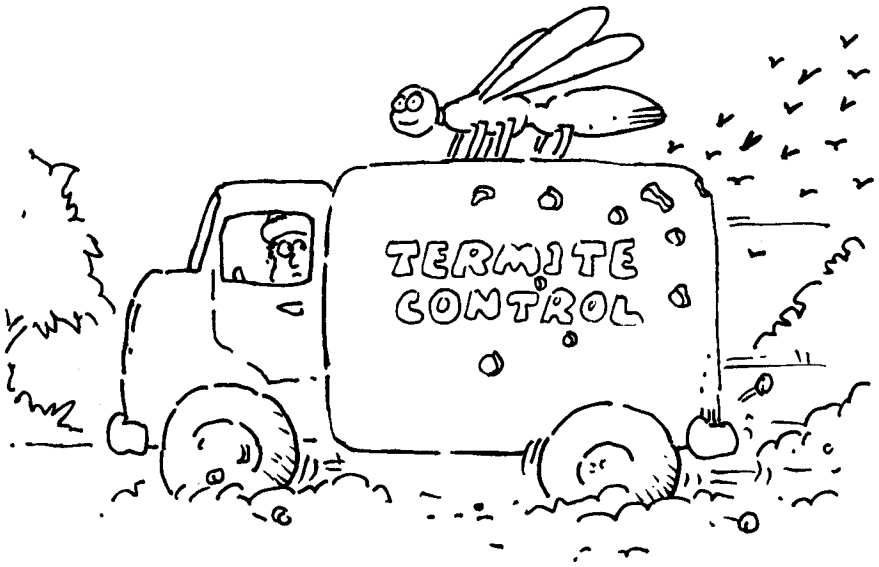
Interest on Credit Purchases

All too frequently, the rate of interest to be charged on a loan or credit purchase is hidden in the small type. After all, the bank or car dealer or finance company wants to convince you that you can afford that new car or home improvement or whatever.

This program calculates the interest rate on a loan given the principal value of the loan, the number of payments, and the amount per payment. To make things even easier, the program will accept the cash purchase price of the article and the down payment and automatically compute the principal value of the loan.

The sample run shows a rather unrealistic loan for an item costing $88.99. The down payment was $10.00 and the loan is over a period of 18 months; each monthly payment is $4.85. The program run shows that the actual interest rate is a modest 7.01%.

Now try the program with a more realistic loan. A termite company in New Jersey advertises a total treatment for your house for only $200; just $29.95 down and 24 monthly payments of $11.95 per month. Is this the bargain that it seems? What is the annual rate of interest?



```
5 Print chr$(14):rem switch lower case
10 Print chr$(147);"Interest rate of a Purchase on credit"
20 inPut "Purchase Price";P
30 inPut "Down Payment";d
40 inPut "Number of Payments";n
50 inPut "Payments Per month";m
60 inPut "Amount Per Payment";a
70 P1=P-d
80 t=a*n-P1
90 y=n/(m*12)
100 r=int(10000*t/P1/y+.5)/100
110 Print:Print "Interest rate is";r;"%"
```

```
Down Payment?Interest rate of a Purchase on credit
Purchase Price? 88.99
Down Payment? 10
Number of Payments? 18
Payments Per month? 1
Amount Per Payment? 4.85

Interest rate is 7.01 %
```

```
Interest rate of a Purchase on credit
Purchase Price? 1000
Down Payment? 0
Number of Payments? 24
Payments Per month? 2
Amount Per Payment? 50

Interest rate is 20 %
```

Population Growth

So far, all the compounding problems in this chapter have involved money—interest, savings, and loans. But many fascinating compounding problems do not involve money. Consider the following problem.

In 1960, the population figures for the United States and Mexico were 180 million and 85 million respectively. The annual population growth rate for the United States was 1.23% and for Mexico 2.23%. If these growth rates remain steady, in some distant year the population of Mexico will exceed that of the United States. In what year will this occur?

The program uses the method of accumulating the principal amount rather than a formula, and applies it to both populations. The sample run reveals that the population of Mexico will overtake that of the U.S. in the not too distant future.

```
5 Print chr$(14):rem switch lower case
10 u=180000000
20 m=85000000
30 r1=1.0123
40 r2=1.0223
50 y=1960
60 u=u*r1
70 m=m*r2
80 y=y+1
90 if m<u then 60
100 Print "Year =";y
110 Print tab(9);"Population"
120 Print "U.S.A. ";u
130 Print "Mexico ";m


Year = 2037
       Population
U.S.A.  461406538
Mexico  464464688
```

But what if the Mexican people were diligently trying to bring their increasing population under control and their annual growth rate was increasing by only 0.001% per year, compared to the increase of the U.S. growth rate of 0.01% per year. If that were the case, would the population of Mexico ever exceed that of the U.S.?

Insert Statements 60, 70, and 85 into the first program, run it and find out the answer to the question.

```
60 u=u*(1.01*r1)
70 m=m*(1.001*r2)
85 if y>9999 then 110
```

Going back to the first program, run it with the population data from the U.S. (180 million) and California (15.7 million) from 1960. At that time, the annual growth rates were 1.23% and 3.7% respectively. Running the program will indicate the year that the population of California will exceed that of the United States. This, of course, is nonsense. Where is the discrepancy?

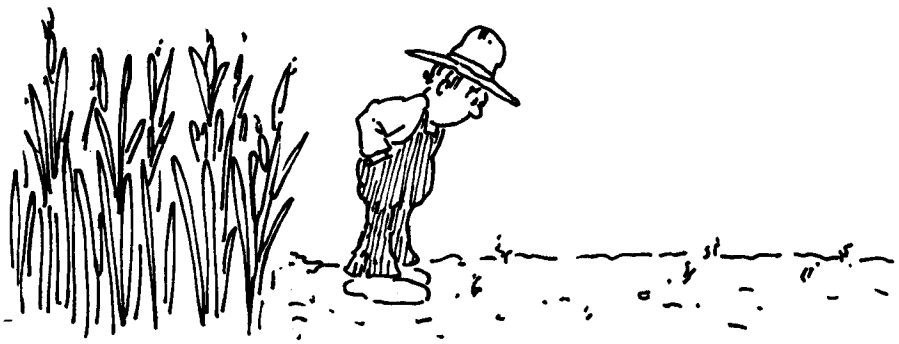
This program might make more sense if it were restated to ask in what year the population of California would exceed that of the remainder of the U.S., if ever?

Compounding can also be used to solve other kinds of population growth problems. Consider the bristleworm. The bristleworm can reproduce by splitting itself into 24 segments, each of which grows a new head and tail. What is the maximum number of bristleworms that could be obtained in this fashion, starting with only one worm, after ten splittings? Assuming a splitting occurs every 22 days, how many offspring will one worm produce in a year? When will the earth be overrun with bristleworms?

Here is another problem for which the principle of compounding is useful. It takes nature about 500 years to produce one inch of topsoil. Many years ago, the United States had an average depth of almost nine inches of this good dirt, but as of 1975, the country was down to about six inches. This type of soil is necessary, of course, for growing food.

Careless management of our soil causes about 1% per year to erode away; it is then lost forever. Once soil depth reaches three inches or less, it is impossible to grow crops on a large scale. Write a program to calculate the year in which the U.S. will have less the 3" of topsoil, assuming that it continues to erode away at 1% per year.

Will you be alive then? Will your children be alive? Will anyone be alive?



6

Probability

Statistics and probability are subjects that bring up all kinds of images. Some people who have not had a pleasant time in a required college statistics course keep as far away from the subject as possible. To other people, statistics is something with which devious manufacturers can mislead you about their products.

Not long ago, one foreign automobile manufacturer boasted that 90% of their cars sold in the United States in the last seven years were still on the road. This sounds like excellent reliability. But consider the fact that this manufacturer was rapidly expanding in the U.S. market and 65% of the cars they had sold in the U.S. had been sold in just the previous two years. You would expect that all of these would still be running.

If the manufacturer had sold 10% of their 7-year volume in Years 1 to 3, and most of these cars were not running, then the advertising claim loses much of its meaning.

When approached in a logical, step-by-step manner, statistics and probability are not difficult. In fact, they can be a great deal of fun.

Some of the programs use a formula while others simulate the event in which we are interested. The results are the same, but you may find that the simulations help you to understand exactly what is happening.

Pascal's Triangle—Calculated

Pascal's Triangle is quite fascinating. As you can see from the portion of one reproduced below, each row is symmetrical. Each row also contains the coefficients for a binomial expansion. The sums across the ascending diagonals form the Fibonacci sequence. The sums across the rows are all powers of two. Each row corresponds to the digits of a power of 11. Every element is the sum of the two above it. And, in case you care, all the elements in it are identities in combinatorial theory.

The ways this marvelous triangle can be generated are as varied and interesting as its properties, though perhaps more difficult to figure out. Here is one way to use a computer to generate the triangle.

Any element can be found be adding together the two elements immediately above it. The program uses this principle to produce a triangle eight levels deep. Lines 10-30 set the rightmost diagonal to 1. Each element is stored in the two-dimensional variable P(R,C) with R denoting the row and C denoting the "column." The variable T (Line 50) simply leaves some blank spaces so the output resembles a triangle. The crux of the calculation is in Line 70 in which the value of each new element is calculated.

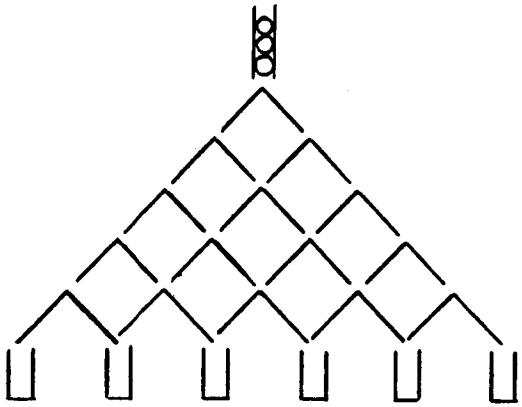
There is another interesting way to calculate Pascal's Triangle in even fewer statements than the program here. It generates the triangle one element at a time and does not use any arrays or two-dimensional variables. Can you figure out how to write such a program?

```
10 for c=1 to 8
20 P(c,c)=1
30 next c
40 for r=1 to 8
50 t=20-r*2
60 for c=2 to r+1
70 P(r+1,c)=P(r,c)+P(r,c-1)
80 if P(r,c)=0 then 110
90 Print tab(t);P(r,c);
100 t=t+4
110 next c
120 Print
130 Print
140 next r
```

```
                    1
                  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
```

Pascal's Triangle—by Probability

This program simulates the dropping of balls through a triangular array shown below. At each level, a ball is equally likely to fall either to the left or right. At the bottom of the array a cup is placed at each end point; these cups collect the balls. After each group of balls has been dropped, the number of balls in each cup is tallied and displayed.



```
5 Print chr$(14):rem switch lower case
10 Print "Pascal's trainGle by Probability"
20 inPut "Balls to droP";m
30 inPut "No. of levels";k
40 Print "Position    Balls"
50 for n=1 to m
60 t=0
70 for l=1 to k
80 if rnd(1)>.5 then 100
90 t=t+1
100 next l
110 b(t+1)=b(t+1)+1
120 next n
130 for l=1 to k+1
140 Print l;tab(12);b(l)
150 b(l)=0
160 next l
```

```
Pascal's trainGle by Probability
Balls to droP 1000
No. of levels 2
Position    Balls
 1              249
 2              516
 3              235
```

```
Pascal's trainGle by Probability
Balls to droP 1000
No. of levels 4
Position    Balls
 1              64
 2              241
 3              390
 4              241
 5              64
```

```
Pascal's triangle by Probability
Balls to drop 160
No. of levels 4
Position     Balls
 1             9
 2            45
 3            56
 4            40
 5            10
```

```
Pascal's triangle by Probability
Balls to drop 640
No. of levels 6
Position     Balls
 1             9
 2            70
 3           142
 4           200
 5           154
 6            51
 7             6
```

The number of balls landing in the various cups at each level, when reduced to the lowest common denominator, should approximate the numbers in Pascal's Triangle. Do you know why they should?

Try a few runs of the program with a different number of balls. Do the numbers obtained really approximate those in Pascal's Triangle? If, at each dividing point in the array, every other ball went in the opposite direction, then Pascal's Triangle would be produced. Try running the program with 4 balls at Level 2; with 8 balls at Level 3; and with 16 balls at Level 4. Do your results look like those of the previous program which calculated the triangle? Does the approximation come closer as the number of balls is increased?

How can you determine how close your results from this program are to the exact value of Pascal's Triangle? One way is to take the number of balls that dropped into each cup on a given level and divide that by the total number of balls divided by the theoretical sum of the row. So, in the sample run at Level 4, you would divide the balls in each cup by 62.5 (1000/16). You then compare that to the "correct" number and compute the percent difference.

Here is the result of this procedure for Level 4 (actually the fifth row) in the sample run:

Cup	*Balls*	*÷62.5*	*Correct*	*Deviation*
1	64	1.02	1	2.00%
2	241	3.86	4	3.50%
3	390	6.24	6	4.00%
4	241	3.86	4	3.50%
5	64	1.-2	1	2.00%

Common Birthdays

Here is an interesting little problem in probability. In a group of ten people selected at random, what is the probability that any of them will share the same birthday? How about a group of 20 people? Of 50 people?

Conversely, how many people would you need in a group such that there is a 50% probability that at least two of them have the same birthday? How many people would be needed for a 90% probability of an overlap?

Try to answer these questions, either by guess or by calculation, before you look at the output from the program.

This program provides a painless introduction to the world of statistics. The calculation is actually quite trivial. The probability that any person in a group has a birthday on January 1 is 1/365. If our group has only two people in it, the probability that both of them have a birthday on January 1 is 1/365 times 1/365, or a very small number indeed. The probability that they have a common birthday is 365 times the very small number just obtained, or about 0.27%.

However, if there are three people in the group, the probability goes up slightly. Call the three people Betsy, Ken, and Larry. From the reasoning above, we know that there is a 0.27% probability that Betsy and Ken have the same birthday; also a 0.27 % probability that Betsy and Larry share a birthday; and finally, a 0.27% chance that Larry and Ken have a common birthday. Hence, the total probability for a group of three people is three times the probability for just two people.

A group of four increases the probability over that of two people by a factor of six, five people by a factor of 10, six people by 15, seven people by 21, and so on. There is a progression here, but the probability can also be calculated by the formula:

$$P = 1 - \frac{365\text{-}N}{365}$$

Can you see why this formula produces the same result as the description in words and associated progression above?

Consider all the presidents of the United States. (How many have there been to date?) Two of them, James Polk and Warren Harding, were born on November 2. Is this to be expected given the size of the group?

Since birthdays can be predicted, at least statistically, it ought to be possible to predict deaths as well. It is interesting too that John Adams, James Monroe, and Thomas Jefferson all died on July 4th. Millard Fillmore and William Taft both died on March 8. What is the probability of that set of events?

```
5 Print chr$(14):rem switch lower case
10 Print "People      Same birthday Probability"
20 q=364/365
30 for n=2 to 40
40 P=100*(1-q)
50 Print n;tab(20);int(P*100+.5)/100;"%"
60 q=q*(365-n)/365
70 next n
```

```
People      Same birthday Probability
 2                    .27 %
 3                    .82 %
 4                   1.64 %
 5                   2.71 %
 6                   4.05 %
 7                   5.62 %
 8                   7.43 %
 9                   9.46 %
10                  11.69 %
11                  14.11 %
12                  16.7 %
13                  19.44 %
14                  22.31 %
15                  25.29 %
16                  28.36 %
17                  31.5 %
18                  34.69 %
19                  37.91 %
20                  41.14 %
21                  44.37 %
22                  47.57 %
23                  50.73 %
24                  53.83 %
25                  56.87 %
26                  59.82 %
27                  62.69 %
28                  65.45 %
29                  68.1 %
30                  70.63 %
31                  73.05 %
32                  75.33 %
33                  77.5 %
34                  79.53 %
35                  81.44 %
36                  83.22 %
37                  84.87 %
38                  86.41 %
39                  87.82 %
40                  89.12 %
```

Now it is your turn to write a program. Here is a game to be played among your friends. You make a bet that they have similar preferences in colors. Each person in the group puts up one penny and you, because you are so sure of yourself (and probability), put up an amount in cents equal to the number of people in the group. If you win, you get to keep all of the money; if your friends win, each of them gets double his original bet, or two cents.

Then each person selects a color (from a larger list than there are people, of course). If any two have picked the same color, you win; if all have picked different colors, they win.

Since you would like to win, you need to know how many colors should be on the list for different size groups to give you a better than 50-50 chance of winning. You might like the probability to be up around 70% or so. You can produce the necessary table with a five-line program. Go to it!

Coins in a Pocket

The next two programs were originally written by Glenda Lappan and M.J. Winter and appeared in *Creative Computing* magazine. They are marvelous simulations for illustrating various aspects of probability.

Coins in a Pocket is a simulation of the following situation. A newspaper costs 5 cents. A customer has 5 pennies and a dime in his pocket and offers to pay for his paper by letting you, the vendor, select at random two of the six coins. If you and this customer repeat this procedure for the 20 working days of a month, how much more or less than $1.00 (20 days x 5 cents) are you likely to have collected?

The program below solves this little problem. The first random coin is selected from a group of six (Line 50 in which X = INT(RND*6)). The value of X can be 0, 1, 2, 3, 4, or 5. If it is 0, we assume the dime was selected and a second pick is not made, because it will surely be a penny. Thus 11 cents is added to the running total in Line 110.

If the first coin is a penny (X = 1, 2, 3, 4, or 5) then we make a second pick from the five remaining coins. Again, if the dime is chosen (Y = 0), 11 cents is added to the total; otherwise 2 cents is added.

As you can see from the sample runs, after a great number of trials, the average amount of money collected each day seems to be close to 5 cents. If you would like to convince yourself that the answer is, in fact, 5 cents, consider the following. Assign a letter to each coin, A - F. Let A be the dime and B through F be the pennies. Since all combinations are equally probable, here are all the possible combinations:

AB
AC BC
AD BD CD
AE BE CE DE
AF BF CF DF EF

There are five combinations with a dime (5 x 11 cents) and ten combinations of only pennies (10 x 2 cents). Add it up and you have 55 cents plus 20 cents divided by a total of 15 combinations which equals an average value of 75/15 or 5 cents.

```
5 print chr$(14):rem lower case
10 Print "Simulates takin9 2 coins at random from a dime and 5 Pennies"
20 inPut "No. trials";n
30 u=0
40 for k=1 to n
50 x=int(rnd(1)*6)
60 if x=0 then 110
70 y=int(rnd(1)*6)
80 if y=0 then 110
90 u=u+2
100 9oto 120
110 u=u+1i
120 next k
130 Print "Aver9e value is";u/n;"cents."
```

```
Simulates takin9 2 coins at random from a dime and 5 Pennies
No. trials? 100
Aver9e value is 4.16 cents.
```

```
Simulates takin9 2 coins at random from a dime and 5 Pennies
No. trials? 1000
Aver9e value is 4.7 cents.
```

Baseball Cards

In statistics and probability, it is frequently necessary to predict the number of trials on average until one is successful. For example, if you are trying to roll a four with one die, on average how many tries will it take until you do so?

Since a die has six sides, the probability of rolling any number between 1 and 6 is $p = 1/6$. Thus, on any given roll, you have a $1/6$ chance of rolling a four and $5/6$ chance of rolling something else. This failure is designated q. So to be successful in rolling a four, we have a $1/6$ chance on the first roll. On two rolls, the probability is two trials x the probability of one failure, then success ($2 \times 5/6 \times 1/6$). On the third roll, the probability of success is three trials x the probability of two failures followed by one success ($3 \times 5/6 \times 5/6 \times 1/6$).

Continuing this reasoning leads to the formula for the expected number of trials to success:

$$E = 1p + 2q \times p + 3q \times p + 4q \times p + \ldots$$

This series can be reduced and solved for E which leads to:

$$E = \frac{1}{1-q} = \frac{1}{p}$$

So now the answer to the original problem can be calculated. The expected number of trials to roll a four is $1/p = 6$ tries until success.

But there is another way to approach this type of problem with the assistance of the computer. Consider the problem. Assume there are ten different prizes in Crunchies cereal boxes. How many boxes of cereal would you have to buy to obtain the complete set? The formula above can be expanded to solve this problem:

$$\frac{N}{N} + \frac{N}{N-1} + \frac{N}{N-2} + \cdots + \frac{N}{1} = N(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}$$

For a value of ten, you can solve this formula by hand. However, let's say you would like to solve this problem for baseball cards as found in packs of bubble gum. If there are 50 cards in a complete set, how many packs of gum must you buy, on average, to get a complete set. Write a computer program using the general formula above to solve for a set of any size. The answer for 50 cards is 224.96 packs of gum; how many would you have to buy for a set of 100 cards?

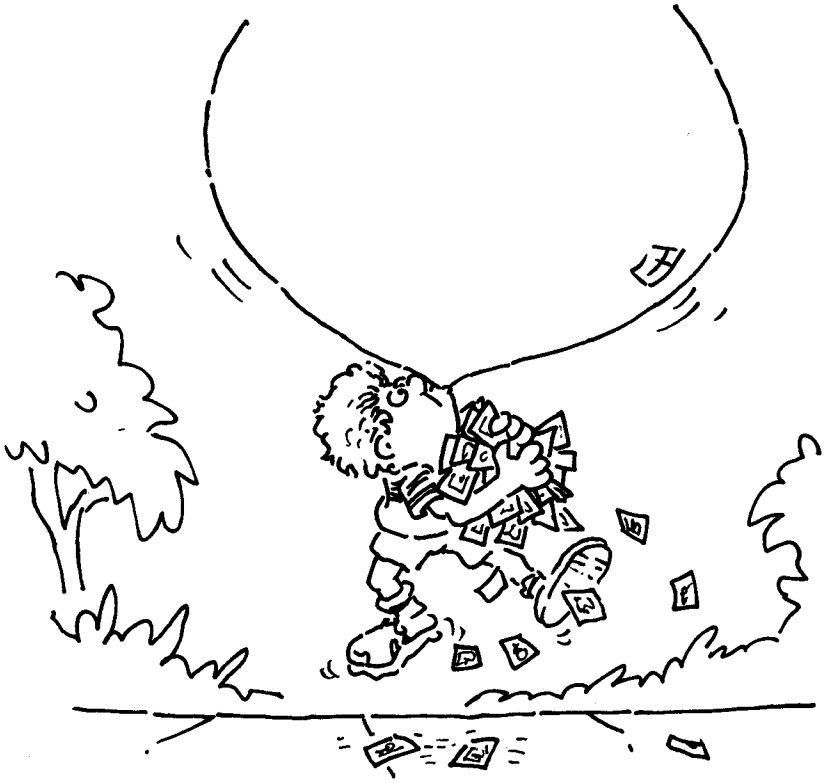
There is another way of approaching this problem. This method is similar to that used to randomly select coins out of the pocket. In this case, the random group is set equal to the total number of cards; this value is accepted as input in Statement 20. Each purchase is set equal to a random number between 1 and N (Statement 120). This card is then put into its proper place in the collection (Statement 130). However, if there is already a card there from a previous purchase, we simply increment the purchase counter (Statement 160) but do not get any closer to obtaining a complete set. After each purchase, we test to see if the set is complete (Statement 170), otherwise we go on buying more packs of bubble gum.

When the set is complete, the number of packs of gum are tallied up and printed out. After a set of trials, the average is computed. This averaging value should be reasonably close to the value obtained by the formula although it will take a great number of trials before the two numbers are within 1% of each other.

```
5 Print chr$(14):rem switch lower case
10 Print "Simulates buying gum with baseball      cards."
20 inPut "Cards in series";n
30 inPut "No. similar cards";k
40 dim c(100)
50 Print "Trial       Packs"
60 s=0
70 for i=1 to k
80 d=0
90 for J=1 to n
100 c(J)=0
110 next J
120 x=int(rnd(1)*n)+1
130 c(x)=c(x)+1
140 if c(x)=1 then 160
150 goto 120
160 d=d+1
170 if d=n then 190
180 goto 120
190 t=0
200 for J=1 to n
210 t=t+c(J)
220 next J
230 Print i;tab(11);t
240 s=s+t
250 next i
260 Print "Average   ";s/k
```

```
Simulates buying gum with baseball
cards.
Cards in series? 10
No. similar cards? 5
Trial        Packs
 1             36
 2             25
 3             21
 4             20
 5             19
Average    24.2
```

If you run this second program for a set of 100 cards, it will sometimes run for a very long time before arriving at the answer. Try it with a set of 500 cards as are used by some real bubble gum manufacturers. You can go have your dinner while the program computes just the first value. Be sure to change the dimension statement in Line 40 to C(500).



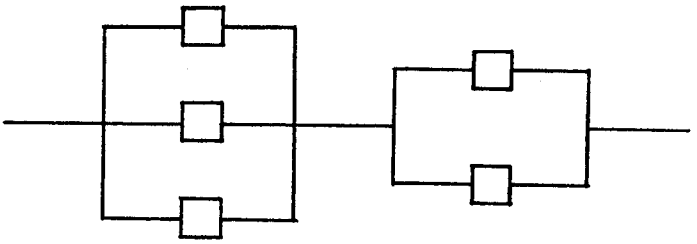
System Reliability

As more and more people in the world come to depend upon mechanical and electronic devices in a myriad of different ways, it is important that these technological devices continue to function.

Some years ago, the military came up with a measure that could be applied to all kinds of systems, big and small, to measure reliability. It is called "mean time between failures." What this means is the length of time, on average, between breakdowns. For a tank, this may be 100 hours, while for a spacecraft, the MTBF must be considerably longer than the planned mission.

As we saw in an earlier section, it is frequently desirable to break down a large problem into smaller subproblems. To calculate the MTBF for a spacecraft would be quite impossible. Instead, it is necessary to start with smaller systems and build up to the whole.

Consider one of the electrical subsystems of a spacecraft. It uses five components as shown below, two parallel systems A and B, arranged in series. The parallel subsystems are said to be "redundant." This is one method of increasing reliability since the system will continue to work if at least one of the components works.



If the manufacturer of the components stated that each one has a 60% probability of lasting 1000 hours, what is the chance that the entire system will last 1000 hours? Take a guess before reading on to the solution below. Is your guess greater than 60% or less? Why?

The program below is a simulation of this system. Remember in subsystem A, it will continue to work if any of the three components works and in B if either of the two components works. However, the system will fail if all three of the A components or both of the B components fail.

In Statement 30, the program is set up to make 500 trials of the system. Statement 50 selects a random integer between 1 and 10 for each component for each trial. If the value of this integer is 6 or under, the component is okay (60% probability of working at the end of 100 hours). If it is 7, 8, 9, or 10, this means the component has failed, and the program prints "Bad!"

```
5 print chr$(14):rem switch lower case
10 print "Simulates use of a circuit for 1000 hrs":print
20 s=0
30 for i=1 to 500
40 for j=1 to 5
50 c(j)=int(rnd(1)*10)+1
60 if c(j)>6 then 90
70 c(j)=1
80 goto 100
90 c(j)=0
100 next j
110 if c(1)=1 or c(2)=1 or c(3)=1 then 140
120 print "Bad!";
130 goto 180
140 if c(4)=1 or c(5)=1 then 170
150 print "Bad!";
160 goto 180
170 s=s+1
180 next i
190 print:print "Reliability is approx.";int(10000*(s/500)+.5)/100;"%"
```

```
Simulates use of a circuit for 1000 hrs

Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Bad!
Reliability is approx. 81.8 %
```

Statement 110 tests whether subsystem A works and Statement 140 checks out subsystem B. If both work, a success is counted in Statement 170.

There are, of course, shorter ways to do this. For example, all the individual tests (Statements 110 to 140) can be replaced by one overall test (new Statement 110).

```
110 if (c(1)+c(2)+c(3))*(c(4)+c(5))=0 then 140
130 s=s+1
140 next I

Reliability is approx. 78.9 %
```

Now it is your turn to use what you have learned in this chapter and write a program to determine the mean time between failures of the system above.

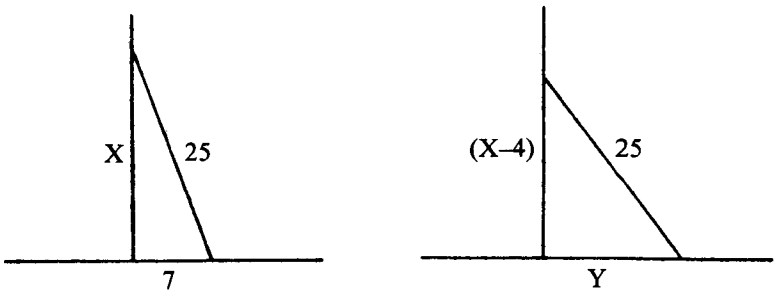
7

Geometry and Calculus

In this chapter several of the problem solving approaches from earlier chapters are used to solve geometric problems. You will find that many problems can be solved in a variety of different ways—by applying a formula, by trial and error, by successive approximations, and, in some cases, by common sense. Perhaps the most important thing to learn from these problems and programs is how to analyze a problem to reach the solution quickly and painlessly.

Crossed and Slipping Ladders

Here is a simple problem of a slipping ladder. A ladder 25 feet long is placed so its foot is 7 feet from the base of a building. The base of the ladder slipped on some loose gravel so that the top is 4 feet lower than where it was to start. How far did the foot of the ladder slip?



The diagrams show the two positions of the ladder. By the Pythagorean theorem we know that $a^2 + b^2 = c^2$, hence, the equations needed to solve the problem are:

$$x = \sqrt{25^2 - 7^2}$$

$$y = \sqrt{25^2 - (x-4)^2}$$

and the amount of slippage of the base is $z = y - 7$. These three equations are put into a computer program which quickly calculates an answer of 8 feet.

```
10 Print "Slipping Ladder"
20 c=25
30 b=7
40 x=sqr(c↑2-b↑2)
50 y=sqr(c↑2-(x-4)↑2)
60 z=y-b
70 Print "Base slipped";z;"ft"
```

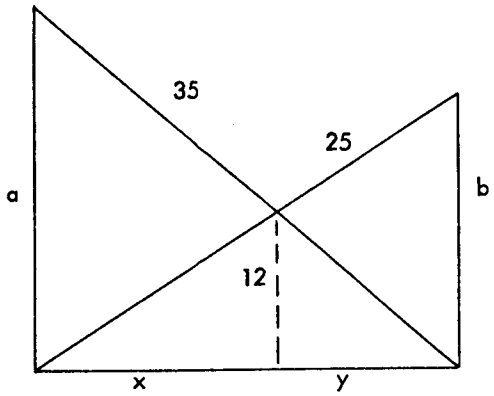
```
Slipping Ladder
Base slipped 7.99999999 ft
```

While the aritmetic in the above problem is not particularly messy, it is still no great joy to solve by hand. However, the computer is just as happy to do the problem with really messy dimensions, say a ladder length of 27.83 feet and a distance from the wall of 7.62 feet.

```
20 c=27.83
30 b=7.62
```

```
Slipping Ladder
Base slipped 8.38612 ft
```

Let's consider an old problem found in many classic collections. Two ladders, one 25 feet long and the other 35 feet long lean against buildings on opposite sides of an alley as shown below. The point at which the ladders cross is 12 feet above the ground. How wide is the alley?



By using similar triangles twice we find that

$$12/a + 12/b = 1$$

Then, by applying the Pythagorean theorem and reducing, we obtain:

$$a^2 - b^2 = 600$$

Using one of the methods described in the Problem Solving chapter, you can solve these two simultaneous equations. Or they can be combined into one equation in which z, the width of the alley is

$$z = \sqrt{35^2 - a^2}$$

By eliminating b, the following fouth degree equation is obtained

$$a^4 - 24a^3 - 600a^2 + 14400a - 86400 = 0$$

Again, this can be solved using one of the methods in the chapter on Problem Solving. This is the traditional approach, but there is another. The solution is the intersection of the curves described by the two original equations. By using successive approximations, say in steps of 1 for a, you could solve for b_1 and b_2 in the following restated original equations

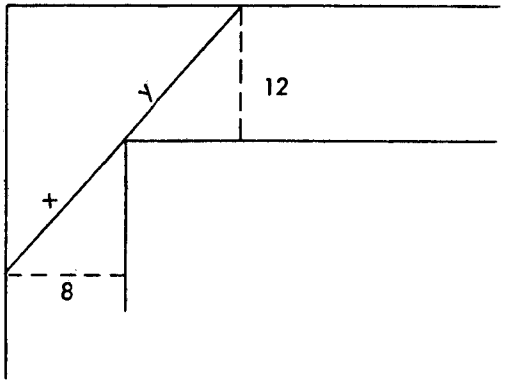
$$b_1 = \frac{12a}{a - 12}$$

$$b_2 = \sqrt{600 - a^2}$$

When b_1 falls below b_2, reduce the step to 0.1 to obtain a closer approximation. By continuing this method of successive approximations, it is possible to obtain a very accurate solution.

Is there another approach? Yes, there is and it also avoids the quartic equation. It uses the original equations in a trial and error procedure as described in the chapter on Sets and Repetitive Trials. See if you can write a program using this approach.

Here is another classic problem for you to solve in any way that you like. What is the longest ladder that can be carried in a horizontal position around the corner made where a 12-foot wide alley meets one that is 8 feet wide. The diagram shows the problem.



Distance Between Coordinate Points

This program solves for the distance between any two points in three-dimensional space defined by their x, y, and z coordinates.

The formula for the distance between two points in three-dimensional space is:

$$d = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$$

It would be desirable for the program to be able to solve any problem of this kind. One approach would be to use INPUT statements to accept the point coordinates. Another is to use a DATA statement to define the points. Then only this one statement has to be changed for a new problem or set of problems.

The program is written to calculate the distance between three sets of points. The points used were:

0, 0, 0	and 3, 4, 5
3.5, -4.7, 6.2	and -0.9, 3.0, 4.4
67, 36, 82	and 54, 25, 90

This calculation is used extensively in aerospace navigation. Can you determine the angle or "compass heading" of the resulting flight path? Better start with just two dimensions.

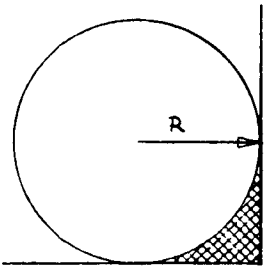
```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "Calculates distance between Points in  3-D sPace"
30 Print
40 Print "Coordinates       Distance"
50 read a,b,c,d,e,f
60 l=sqr((a-d)↑2+(b-e)↑2+(c-f)↑2)
70 Print a;b;c
80 Print d;e;f;tab(18);int(l*1000+.5)/1000
90 Print
100 goto 40
110 data 0,0,0,3,4,5
120 data 3.5,-4.7,6.2,-.9,3,4.4
130 data 67,36,82,54,25,90
```

```
Calculates distance between Points in  3-D sPace

Coordinates          Distance
 0  0  0
 3  4  5              7.071

Coordinates          Distance
 3.5 -4.7  6.2
-.9  3  4.4             9.049

Coordinates          Distance
 67  36  82
 54  25  90             18.815
```

Area—by Calculation

It is a simple matter to calculate the area of regular geometric figures by using the usual formulae. However, the problem becomes more difficult when it is necessary to calculate the area of two combined regular shapes or the area of irregular shapes. This program shows the method of analysis for solving a problem of the first type, while the next program demonstrates four methods of dealing with irregular areas.

The problem is to solve for the shaded area of the figure below for any value of the radius, R.



The first step is to recall the formulae for calculating the area of a circle and square:

$$A\ (circle) = pi * R^2$$

$$A\ (square) = Side^2 \text{ or } (2 * R)^2$$

The difference in area between a square and a circle inscribed within its borders is:

$$A\ (difference) = A\ (square) - A\ (circle)$$

and the area of one corner is the difference divided by 4. The program below will calculate the area for any radius.

Now it is your turn. Write a program to calculate the difference in area between a circle and square in which the square is inscribed within the circle. Now, change your program to calculate the difference in area for a triangle, a hexagon and an octagon inscribed within a circle.

```
5 Print chr$(14):rem switch lower case
10 Print "Area between circle inscribed in square"
20 inPut "Radius";r
30 c=3.14159*r↑2
40 s=(2*r)↑2
50 d=s-c
60 d1=d/4
70 Print "TraPPed area";d
80 Print "One corner";d1

Area between circle inscribed in square
Radius 10
TraPPed area 85.8410001
One corner 21.46025

Area between circle inscribed in square
Radius 1
TraPPed area .858409999
One corner .2146025
```

Extend your program to calculate the difference in area for any regular figure inscribed within a circle of radius 1.0. Set up a table or results as follows:

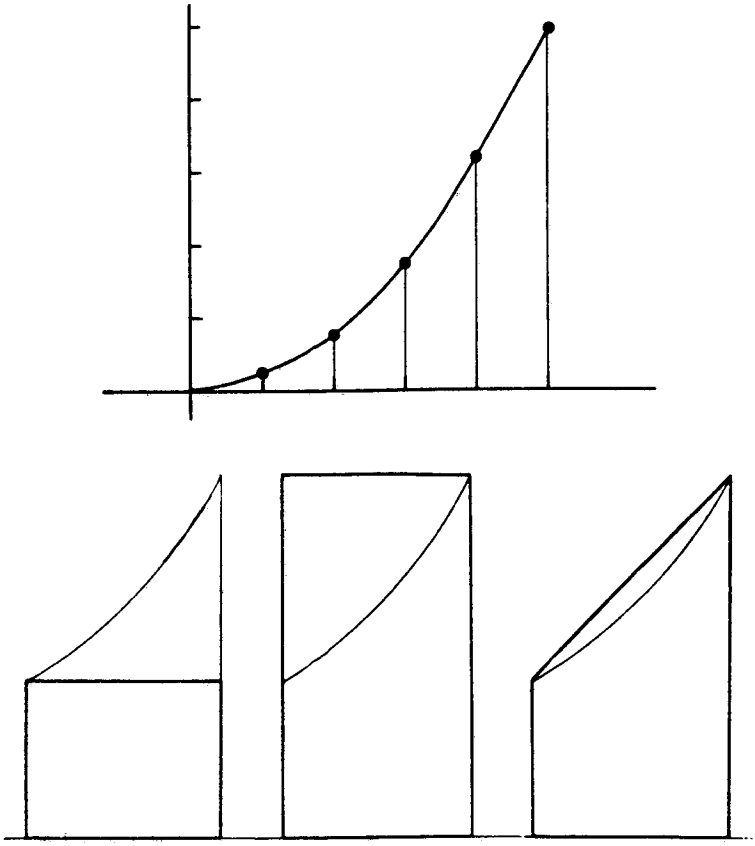
Number of sides	*Difference in Area*
3	
4	
5	
6	
7	
8	

What can you conclude from these results? Do these areas follow in some sort of progression?

Area—by Integration

In many cases it is necessary to determine the area of an irregular figure or the area under a curve where an exact formula is not available. The approach most commonly used is to divide up the enclosed area into small regularly shaped pieces and sum up the areas of all of these pieces.

The easiest shape to use in these calculations is a rectangle. A group of rectangles can either be inscribed within the irregular figure or curve, or circumscribed around it. The first two diagrams show these two methods being used to find the area under a curve. A third method is to use trapezoids which, depending upon the direction of curvature, will either be inscribed or circumscribed automatically.



Inscribed Rectangle | Circumscribed Rectangle | Trapezoid

A fourth method, known as Simpson's Rule, essentially fits a series of parabolas between the points of the curve and calculates the average area. It requires that the area be divided into an even number of parallel slits. Let us call the total number of divisions 2m which are h distance apart. The point on the curve where the first line intersects is y_0, the second y_1, and so on until y_{2m}. The area is then given by the formula:

$$A = \tfrac{1}{3}h[(y_0 + y_{2m}) + 4(y_1 + y_3 + \ldots + y_{2m-1}) + 2(y_2 + y_4 + \ldots y_{2m-2})]$$

The area as calculated by Simpson's Rule converges extremely quickly compared to the other methods. Nevertheless, as the number of intervals increases, all the methods approach the same limit. As might be expected, the trapezoidal approach converges more quickly than either of the approaches using rectangles. Compute the average of the two methods using rectangles. What do you get? Does this suggest another method?

The methods used in this program involve the calculus. And you thought the calculus was difficult! Now you know otherwise.

```
5 Print chr$(14):rem switch lower case
10 Print chr$(147)
20 Print "ComPutes are under a curve by 4 methods of inteGration."
40 Print
50 Print "Enter start and end values for x (small one first)"
60 inPut a,b
70 Print "RanGe of x=";a;",";b
80 Print:Print "No. intervals
90 Print "Inscribed Rects          TraPezoids"
100 Print "Circumscribed Rects     Parabolas"
110 m=-2
120 s=0
130 def fna(x)=x↑3
140 m=m+3
150 for n=m to m+2
160 c=s
170 q=0
180 P=0
190 d=2↑n
200 h=(b-a)/d
210 for i=0 to d-1
220 x=a+i*h
230 P=P+h*fna(x)
240 q=q+h*fna(x+h)
250 next i
260 t=(P+q)/2
270 u=fna(a)+fna(x+h)
280 for J=2 to (d-2) steP 2
290 u=u+2*fna(a+J*h)
300 next J
310 v=0
320 for k=1 to (d-1) steP 2
330 v=v+4*fna(a+k*h)
340 next k
350 s=(u+v)*(h/3)
```

```
360 Print:Print d:Print:Print p;tab(12);t:Print q;tab(12);s
370 next n
380 if d<64 then 140
390 if abs((c-s)/((c+s)/2))>.0001 then 140
400 stop


Computes are under a curve by 4 methods of integration.

Enter start and end values for x (small one first)
? 1
?? 10
Range of x= 1 , 10

No. intervals
Inscribed Rects          Trapezoids
Circumscribed Rects      Parabolas

 2
 753.187501               3000.9375
 5248.6875               5499.75001

 4
 1501.17188               2625.04688
 3748.92188               2499.75

 8
 1969.13672               2531.07422
 3093.01172               2499.75

 16
 2226.61231               2507.58106
 2788.54981               2499.75

 32
 2361.22339               2501.70776
 2642.19214               2499.75

 64
 2429.99726               2500.23944
 2570.48163               2499.75
```

8

Science

Using techniques and approaches presented in the previous chapters, this chapter contains five programs in the area of science. One uses a formula to solve simple gas problems, two are drill exercises on the gas laws of Boyle and Charles, and the last two are simulations. You will see that the simulations draw upon many previous techniques such as progressions and repetitive calculations.

Gas Volume

Here is a simple program to produce a not-so-simple table of values for gas volumes.

The volume of a gas varies directly with the absolute temperature T (Kelvin) and inversely with the pressure P. If a certain quantity of gas occupies 500 cubic feet at a pressure of 53 pounds per square foot and an absolute temperature of 500 degrees, what volume will it occupy at 600 degrees absolute temperature and pressures from 100 to 1000 pounds per square foot in increments of 50 pounds?

The original conditions are used to solve for the constant K in Statement 50 (K = V*P/T). Then new volumes are computed for varying pressures with T equal to 600 degrees. The formula used is V = K*T/P.

In the second part of the program, Lines 70 to 100 are replaced to produce a plot of the gas volume for the various pressures.

How would you modify the program to deal with a more general case (i.e., other gasses and different temperatures)? Second, can you write a program that produces a table of values for a gas at different pressures and temperatures?

```
5 Print chr$(14):rem switch lower case
10 Print "Gas volumes at different Pressures":Print
20 v=500
30 P=53
40 t=500
50 k=v*P/t
60 inPut "TemPerature (k)";t1
70 Print "Pressure Volume"
80 for P=100 to 1000 steP 50
90 v=k*t1/P
100 Print P;tab(10);v
110 next P
```

```
Gas volumes at different Pressures

TemPerature (k)? 600
Pressure      Volume
 100           318
 150           212
 200           159
 250           127.2
 300           106
 350           90.8571429
 400           79.5
 450           70.6666667
 500           63.6
 550           57.8181818
 600           53
 650           48.9230769
 700           45.4285715
 750           42.4
 800           39.75
 850           37.4117647
 900           35.3333333
 950           33.4736842
 1000          31.8
```

```
5 print chr$(14):rem switch lower case
10 print "Gas volumes at different Pressures":print
20 v=500
30 p=53
40 t=500
50 k=v*p/t
60 input "Temperature (k)";ti
65 print
70 print "Pressure    Volume Plot"
80 for p=100 to 1000 step 50
90 v=k*ti/p
100 print p;tab(4+v/17);"*"
110 next p
```

```
Gas volumes at different pressures

Temperature (k)? 600

Pressure    Volume Plot
 100                    *
 150                *
 200              *
 250            *
 300          *
 350        *
 400      *
 450      *
 500    *
 550    *
 600    *
 650  *
 700  *
 750  *
 800  *
 850  *
 900  *
 950  *
1000  *
```

Charles' Law Drill

In the previous program, gas volumes were calculated using Charles' Law and Boyle's Law. Here is a drill and practice program (remember Chapter 1?) that produces problems relating the volume and temperature of a gas. When pressure is constant, the volume/temperature relationships can be stated as follows:

$$\frac{V_0}{T_0} = \frac{V_1}{T_1}$$

The program presents four problems, one to solve for each of the four variables in the equation above.

How can you make the program more efficient? More interesting?

```
5 Print chr$(14):rem switch lower case
10 Print "Charles' Law drill"
20 Print "Volume in milliliters"
30 Print "Temperature in degrees Kelvin"
40 Print
50 v=int(rnd(1)*50+50)*100
60 v1=int(rnd(1)*50+50)*100
70 t=int(rnd(1)*125+125)
80 t1=int(rnd(1)*125+125)
90 c=c+1
100 on c goto 110,150,190,230,270
110 v=0
120 Print:Print "Solve for v when"
130 q1=v1*t/t1
140 goto 280
150 v1=0
160 Print " What is v1 given"
170 q1=v*t1/t
180 goto 280
190 t=0
200 Print:Print "Calculate value of T"
210 q1=v*t1/v1
220 goto 280
230 t1=0
240 Print:Print "Solve for t1 given"
250 q1=t*v1/v
260 goto 280
270 stop
280 Print "v= ";v;tab(11);"t=";t
290 Print "v1= ";v1;tab(11);"t1=";t1
300 input "Your answer";q
310 Print "Correct value";q1
320 Print
330 goto50
```

```
Charles' Law drill
Volume in milliliters
Temperature in degrees Kelvin


Solve for v when
v=  0            t= 217
v1=  6000         t1= 196
Your answer? 114418
Correct value 6642.85715

What is v1 given
v=  5000          t= 153
v1=  0           t1= 243
Your answer? 523821
Correct value 7941.17647


Calculate value of T
v=  7100          t= 0
v1=  9800         t1= 162
Your answer? 18494
Correct value 117.367347


Solve for t1 given
v=  9500          t= 222
v1=  8300         t1= 0
Your answer? 75000
Correct value 193.957895
```

Boyle's Law Drill

Boyle's Law describes the behavior of gases under ideal conditions when pressure and volume are varied. When the temperature is constant, Boyle found that:

$$P_0 * V_0 = P_1 * V_1$$

Pressure is normally measured in centimeters of mercury while volume is measured in millilitres. As with the drill on Charles' Law, this program presents four problems, one to solve for each of the four variables in the equation above.

Unlike many other drill and practice programs, these two do not compare your answer with the correct one. Instead, they leave that up to you to do. Is this desirable? Why or why not? If you feel it is undesirable, change the programs so they do compare answers and calculate a score.

```
5 Print chr$(14):rem switch lower case
10 Print "Boyle's Law drill"
20 Print "Volume in milliliters"
30 Print "Pressure in cmmer";
40 Print
50 v=int(rnd(1)*50+50)*100
60 v1=int(rnd(1)*50+50)*100
70 P=int(rnd(1)*150+150)
80 P1=int(rnd(1)*150+150)
90 c=c+1
100 on c goto 110,150,190,230,270
110 v=0
120 Print:Print "Solve for v when"
130 q1=P1*v1/P
140 goto 280
150 v1=0
160 Print "What is v1 given"
170 q1=P*v/P1
180 goto 280
190 P=0
200 Print:Print "Calculate value of P"
210 q1=P1*v1/v
220 goto 280
230 P1=0
240 Print:Print "Solve for P1 given"
250 q1=P*v/v1
260 goto 280
270 stop
280 Print "v= ";v;tab(11);"P=";P
290 Print "v1= ";v1;tab(11);"P1=";P1
300 inPut "Your answer";q
310 Print "Correct value";q1
320 Print
330 goto50
```

```
Boyle's Law drill
Volume in milliliters
Pressure in cm mer

Solve for v when
v=  0              p= 294
v1=  9500           p1= 273
Your answer? 114418
Correct value 8821.42857

What is v1 given
v=  6400           p= 276
v1=  0            p1= 190
Your answer? 523821
Correct value 9296.84211


Calculate value of p
v=  7700           p= 0
v1=  7400          p1= 255
Your answer? 18494
Correct value 245.064935

v=  7000           p= 165
v1=  9100          p1= 0
Your answer? 75000
Correct value 126.923077
```

Photoelectric Emissions

When light of a short wavelength falls on a metal surface, electrons are emitted from the metal. According to the description of this phenomenon by Einstein, there is a maximum wavelength for every metal above which no electrons are emitted. This is called the critical wavelength of the metal.

This program simulates a laboratory experiment in which a metal is placed in a vacuum and bombarded with soft X-rays. The number of electrons emitted is collected and measured with a microammeter. The program simulates three trials at each of nine wave lenghts.

After each set of experimental data, the program asks if you would like another run at a higher light intensity. The reason for doing this is that sometimes at low light intensities, not enough electrons have been emitted for meaningful measurements.

You can increase the precision of the experiment by increasing the number of wavelengths at which it is run. This value can be changed in Statement 60; note that the variable L is divided into 1000 in order to express the wavelength in Angstroms. One Angstrom (A) equals 10^{-8} centimeters or 10^{-4} microns.

Here are the coefficients for several metals:

Silver	.308
Bismuth	.338
Cadmium	.318
Lead	.340
Platinum	.385

It is a rare physics laboratory in a high school or college today that has the experimental apparatus to run this experiment, yet with a small computer the equipment can be simulated. There are many other things with which you would not normally be able to experiment that can be simulated with a computer. Things such as a nuclear power plant, a malaria epidemic, an urban mass transit system, and a bicycle factory.

Can you write a simulation for a real world system? You will find sections on simulations in the books *Computers in Mathematics* and *Computers in Science and Social Studies.* These, along with articles in *Creative Computing* magazine, might be of some help in writing a simulation of your own.

```
5 Print chr$(14):rem switch lower case
10 Print "Bombardment of metal with soft X-rays."
20 inPut "Coefficient";v
30 k=int(1+2*rnd(1))
35 Print
40 Print "OutPut in microamPs"
45 Print
50 Print " Wave  Trial   Trial"
55 Print "Len9th   1       2"
56 Print "------ -----   -----"
60 for l=.42 to .25 steP -.02
70 m=int(1000/l)
80 Print m;
90 for J=1 to 2
100 if l>v then 130
110 i=sqr(int(25*rnd(1)))
120 9oto 140
130 i=sqr(k*k*100+int(35*rnd(1)))
140 n=int(10*i+.5)/10
150 Print tab(J*7);n;
160 next J
170 Print
180 next l
190 Print:inPut "Increase intensity of li9ht (y,n)";a$
210 if a$="n" or a$="N" then 270
220 inPut "By what factor (1 to 10)";f
240 k=k*f
250 Print
260 9oto 50
270 stoP
```

```
Bombardment of metal with soft X-rays.
Coefficient? .34
OutPut in microamPs
  Wave  Trial   Trial
Len9th   1       2
------ -----   -----
 2380   10.3    10.2
 2500   10.3    10.9
 2631   11.2    11
 2777   11      11.1
 2941   3.3     2.8
 3125   3.6     4.2
 3333   1       4.9
 3571   2.4     1.7
 3846   3.5     1.7

Increase intensity of li9ht (y,n)? y
By what factor (1 to 10)? 5

  Wave  Trial   Trial
Len9th   1       2
------ -----   -----
 2380   50.1    50
 2500   50.2    50.3
 2631   50.1    50.1
 2777   50.3    50.3
 2941   4.7     2.4
 3125   4.1     3.5
 3333   3.9     4.7
 3571   2       4.6
 3846   1.7     1.7

Increase intensity of li9ht (y,n)? n
```

Mutation of Moths

This program is a simulation of the growth of a colony of pepper moths. The program allows a genetic mutation to be introduced in some year between 1 and 30. The mutation can favor either dark or light colored moths.

The program as it appears starts with a total colony of light moths; you could add an initial group of dark colored moths in Statement 45 as P1.

The sample run shows a mutation which favors dark moths occurring in Year 3. The mutation affects about 2% of the light moths each year and causes them to become dark. The program displays the number of moths of each color over a 30-year period.

Obviously, because of the long time period involved it would be difficult to carry out this experiment in school, so the computer again is of real benefit.

```
5 Print chr$(14):rem switch lower case
10 Print "Mutation of a colony of moths."
20 inPut "Rate of mutation (1 to 10)";m
40 P0=10000
50 z=P0
60 inPut "Environmental change occurs in year";x
80 l=1
90 d=2
100 inPut "Change favors light or dark (L/D)";e$
120 Print:Print "Year  Dark   Light"
125 Print "----  ----   -----"
130 for t=1 to 30
140 if t>=x then 170
150 P1=0
160 goto 230
170 if e$<>"D" and e$<>"d" then 150
180 P1=int(P1+.01*m*P0+.5)
190 P0=int(z-P1+.5)
200 if P1<z then 230
210 P1=z
220 P0=0
230 Print t;tab(5);P1;tab(12);P0
240 next t
```

The so-called "killer bees" that were introduced into South America in the mid 70's were supposed to help honey production because they were much more energetic than the lazy honey bees in Brazil. The idea was that they would mate with the existing honey bees and produce a more productive strain. However, they went on a rampage killing people and terrorizing the country. Then they started to migrate north. U.S. agricultural officials became alarmed that they would invade this country and bring death and destruction.

```
Mutation of a colony of moths.
Rate of mutation (1 to 10)? 2
Environmental change occurs in year? 3
Change favors light or dark (L,D)? d

Year        Dark        Light
----        ----        -----
 1           0          10000
 2           0          10000
 3         200           9800
 4         396           9604
 5         588           9412
 6         776           9224
 7         960           9040
 8        1141           8859
 9        1318           8682
10        1492           8508
11        1662           8338
12        1829           8171
13        1992           8008
14        2152           7848
15        2309           7691
16        2463           7537
17        2614           7386
18        2762           7238
19        2907           7093
20        3049           6951
21        3188           6812
22        3324           6676
23        3458           6542
24        3589           6411
25        3717           6283
26        3843           6157
27        3966           6034
28        4087           5913
29        4205           5795
30        4321           5679
```

Over the years, the killer bees have started to mate with the more docile South American honey bees, but only at the rate of 3% per year. Assuming they pose a danger to the U.S. until their numbers are reduced to 15% of their original quantity, how many years will it take for this to occur? The present migration patterns will bring them to the U.S. by 1989; will they still be dangerous (assume that the year of introduction was 1974)?

Projectile Motion

The path followed by a projectile is called its trajectory. The trajectory is affected to a large extent by air resistance, which makes an exact analysis of the motion extremely complex. We shall, however, neglect the effects of air resistance and assume the motion takes place in empty space.

In the general case of projectile motion, the body (bullet, rocket, mortar, etc.) is given an initial velocity at some angle θ above (or below) the horizontal.

If V_0 represents the initial velocity (muzzle velocity), the horizontal and vertical components are:

$$V_{ox} = V_0 \cos\theta \ , \ V_{oy} = V_0 \sin\theta$$

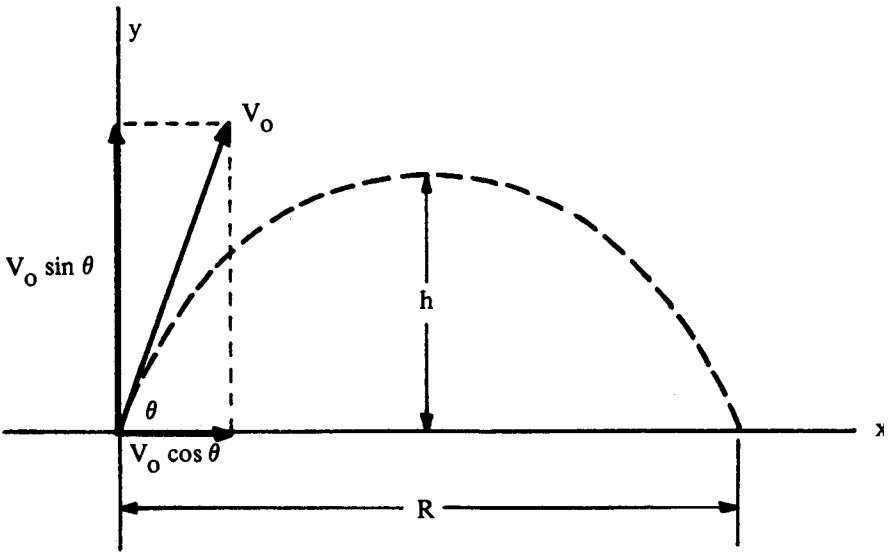
Since we are neglecting air resistance, the horizontal velocity component remains constant throughout the motion. At any time, it is:

$$V_x = V_{ox} = V_0 \cos\theta = \text{constant} \quad (1)$$

The vertical part of the motion is one of constant downward acceleration due to gravity. It is the same as for a body projected straight upward with an initial velocity $V_0 \sin\theta$. At a time "t" after the start, the vertical velocity is:

$$V_y = V_{oy} - gt = V_0 \sin\theta - gt \quad (2)$$

where "g" is the acceleration due to gravity.



The horizontal distance is given by:

$$x = V_{ox}t = (V_o \cos\theta)\, t \qquad (3)$$

and the vertical distance by:

$$y = V_{oy}t - 1/2\, gt^2$$
$$= (V_o \sin\theta)\, t - 1/2\, gt^2 \qquad (4)$$

The time for the projectile to return to its initial elevation is found from Equation (4) by setting $y = 0$. This gives

$$t = \frac{2\, V_o \sin\theta}{g} \qquad (5)$$

The horizontal distance when the projectile returns to its initial elevation is called the horizontal range. "R." Introducing the time to reach the point in Equation (3), we find:

$$R = \frac{2\, V_o^2 \sin\theta \cos\theta}{g} \qquad (6)$$

Since $2\sin\theta \cos\theta = \sin 2\theta$, Equation 6 becomes:

$$R = \frac{V_o^{\,2} \sin 2\theta}{g} \qquad (7)$$

The horizontal range is thus proportional to the square of the initial velocity for a given angle of elevation. Since the maximum value of $\sin 2\theta$ is 1, the maximum horizontal range, Rmax is V_o^2/g. But if $\sin 2\theta = 1$, then $2\theta = 90°$ and $\theta = 45°$. Hence the maximum horizontal range, in the absence of air resistance, is attained with angle of elevation of 45°.

From the standpoint of gunnery, what one usually wishes to know is what the angle of elevation should be for a given muzzle velocity v_o in order to hit a target whose position is known. Assuming the target and gun are at the same elevation and the target is at a distance R, Equation (7) may be solved for θ.

$$\theta = 1/2 \sin^{-1}\left(\frac{Rg}{V_o^{\,2}}\right)$$
$$= 1/2 \sin^{-1}\left(\frac{R}{R_{max}}\right) \qquad (8)$$

Provided R is less than the maximum range, this equation has two solutions for values of θ between 0° and 90°. Either of the angles gives the same range. Of course, time of flight and maximum height reached are both greater for the high angle trajectory.

For example, say the maximum range of our gun is 10,000 yards and the target is at 5,900 yards:

$$\theta = 1/2 \sin^{-1} \frac{5,900}{10,000}$$

$$= 1/2\ 36^\circ$$

$$= 18^\circ, \text{ or } 90^\circ - 18^\circ = 72^\circ$$

Try the computer game Gunner which appears below. Use trial and error to try and destroy the target (see sample run). You get five chances per target. How many shots did it take to destroy all five targets? Did you ever fail to destroy a target in five trials?

From the discussion above, you should realize that 45 degrees of elevation provides maximum range with values over or under 45° providing less range.

The maximum range of the gun will vary between 20,000 and 60,000 yards and the burst radius of a shell is 100 yards.

You can also determine the correct firing angle from sine tables, or a slide rule, a scientific calculator or a computer. You should be able to destroy every target with just one shot. What happens when the target is very close? Can you always use whole angles?

Now write a computer program to accept the maximum range of your gun and the range to the target and then calculate the correct firing angle. You will have to solve two problems to write such a program:

1. The Basic language does not have an ARCSIN function. However, the following formula may help.

$$\sin^{-1} x = \tan^{-1} \left(\frac{x}{\sqrt{1-x2}} \right)$$

2. You must convert from radians to degrees.

```
5 Print chr$(14):rem switch lower case
8 Print chr$(147)
10 Print "You are commanding a gun crew.  Hit within 100 yards of the target."
50 Print
70 r=int(40000*rnd(1)+20000)
80 Print "Max range";r;"yards"
90 z=0
100 s1=0
110 t=int(r*(.1+.8*rnd(1)))
120 s=0
130 goto 310
140 Print "Min = 1 degree"
150 goto 320
160 Print "Max = 89 degrees"
170 goto 320
180 Print "Over by";abs(e);"yards"
190 goto 320
200 Print "Short by";abs(e)"yards"
210 goto 320
220 Print "Target destroyed!"
230 Print s"Rounds used"
240 Print
250 s1=s1+s
260 if z=4 then 450
270 z=z+1
280 Print
290 Print "More enemy activity"
300 goto 110
310 Print "Target at";t;"yards"
320 rem
330 Print:input "Elevation";b
340 if b>89 then 160
350 if b<1 then  140
360 s=s+1
370 if s<6 then 410
380 Print chr$(147)
390 Print tab(12)"Boom!":Print tab(5)"Boom!":Print tab(14)"Boom!":Print
400 Print "Enemy got you first":goto 480
410 b2=.035*b:i=r*sin(b2):x=t-i:e=int(x)
420 if abs(e)<100 then 220
430 if e>100 then 200
440 goto 100
450 Print chr$(147);"Total rounds =";s1
460 if s1>18 then 480
470 Print "Nice shooting!";:goto 490
480 Print:Print "Go back to Fort Sill for more training.";
490 Print:input "Try again (y,n)";z$
500 ifz$="y" or z$="Y" then 50
510 Print:Print "OK.  Return to camp."
520 stop
550 end
```

```
You are commanding a gun crew.  Hit within 100 yards of the target.
Max range 53171 yards
Target at 39743 yards

Elevation? 28.5
Over by 4927 yards

Elevation? 25
Over by 1069 yards

Elevation? 23
Short by 1415 yards

Elevation? 23.7
Short by 524 yards

Elevation? 23.9
Short by 274 yards

Elevation? 24
           Boom!
     Boom!
                Boom!

Enemy got you first

Go back to Fort Sill for more training.
Try again (y,n)? y


Max range 58685 yards
Target at 25871 yards

Elevation? 15
Over by 3543 yards

Elevation? 12
Short by 1941 yards

Elevation? 13.8
Over by 1385 yards

Elevation? 12.6
Short by 821 yards

Elevation? 13
Target destroyed!
 5 Rounds used

Try again (y,n)? n

OK.  Return to camp.
```

9

Potpourri

Here are five programs that didn't seem to fit anywhere else. The first and the last are games, although you may come to think of the lunar landing simulation as a game also. One is a nifty simulation of smog, and the other calculates depreciation by three different methods.

Number Guessing Game

Here is a computer program that plays the popular number guessing game. In it the computer picks a secret number between 1 and 100. You attempt to guess that number in as few tries as possible.

There are many ways to go about guessing the secret number. Let some of your friends play this game and see what approaches they use to find the secret number. One approach is to start with a guess of 10. If this is too low, increase each guess by 10 until the computer says that a guess is too high. When this point is reached, start from the previous guess and increase each guess by 1. This is the method used to solve some of the problems in the chapter on convergence.

Another approach is to try to bracket the number between upper and lower limits and reduce the limits by steps until the number is finally found. Two of the convergence programs used this approach.

Is one of these the best way? Well, these methods are not bad, particularly compared to starting with 1 and simply counting to 100 until the solution is found. But there is a better way. It is known as binary search.

This technique involves dividing the search domain, in this case 1 to 100, in half, and then in half again, and so on until the secret number is found. Play the game many times using different approaches. In the long run you should find that the binary search approach is the most efficient.

In Line 230, the program contains the statement that "you should not need more than 7 guesses." Why? If the secret number was between 1 and 128, what is the maximum number of guesses that would be necessary to find it? What if the number range were 1 to 130; then what would be the maximum number of guesses?

Revise the program to choose a secret number between 1 and 10,000. Now the upper limit is 100 times the 1 to 100 game here which requires a maximum of seven guesses to find the secret number; how many guesses will now be required?

Can you write a program in which the roles of the computer and player are reversed? In other words, the computer will try to guess your secret number between 1 and some upper limit. After each guess, you enter L for low, H for high, or C for correct. Can you write this program so the computer can tell if you are cheating, i.e., giving it inconsistent clues?

```
5 print chr$(14):rem switch lower case
10 Print "I've a secret number between 1 & 100.   Try to guess it.";
20 Print "  I'll give you clues."
30 Print
40 Print
50 c=0
60 n=int(rnd(1)*100)+1
70 c=c+1
80 Print tab(10);"Guess";
90 inPut g
100 if n=g then 160
110 if g>n then 140
120 Print "Too low."
130 goto 70
140 Print "Too high."
150 goto 70
160 Print:Print "Correct in";c;"tries.";
170 if c>3 then 200
180 Print "  You were lucky."
190 goto 240
200 if c>7 then 230
210 Print "  Good Job."
220 goto 240
230 Print "  You shouldn't need more than 7 tries."
240 Print
250 Print
260 inPut "Play another game (y,n)";a$
270 if a$="y" or a$="y" then 30
280 stoP
```

```
I've a secret number between 1 & 100.
Try to guess it.
I'll give you clues.

          Guess? 50
 Too low.
          Guess? 75
 Too low.
          Guess? 85
 Too low.
          Guess? 90
 Too low.
          Guess? 95

 Correct in 5 tries.  Good Job.


 Play another game (y,n)? n
```

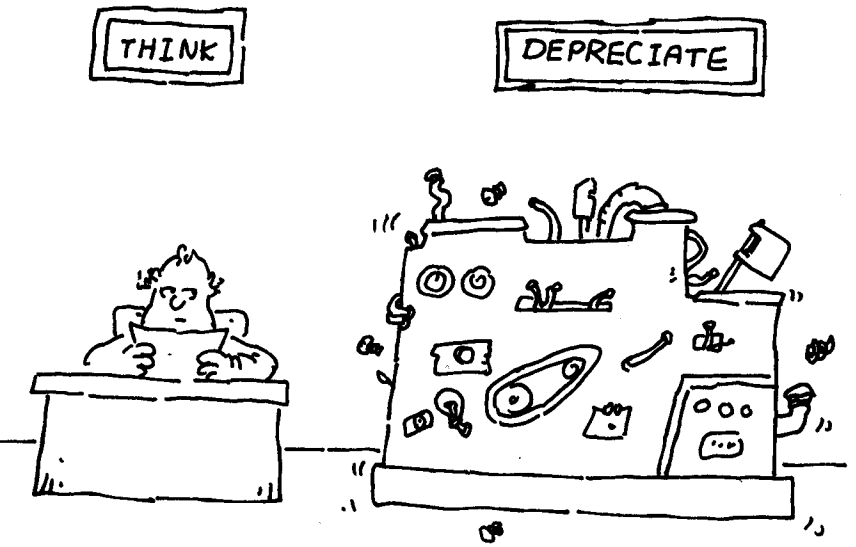
Depreciation—Three Methods

This program shows how a piece of capital equipment depreciates according to three commonly used methods of depreciation: straight line, sum of the year digits, and double declining.

The program asks for the original cost of the item, its expected life in years (the period of time over which it is to be depreciated), and its expected scrap (or sale) value at the end of that time. A table showing the annual depreciation for each of the three methods is then displayed.

```
5 Print chr$(14):rem switch lower case
10 Print "DePreciation by 3 methods."
20 Print
30 inPut "OriGonal cost";c
50 inPut "Life (years)";l
70 inPut "ScraP value";s
90 Print
100 v=c-s
110 d1=v/l
120 Print "StraiGht line dePreciation is $";d1;"Per year.":Print
130 y=((l+1)/2)*l
140 z=l
150 Print "      Sum of     Double"
160 Print "Year  DiGits     DeclininG
165 Print "----  ------     ----------"
170 for x=1 to l
180 d2=v*(z/y)
190 d2=int(d2*100+.5)/100
200 z=z-1
210 d3=2*c/l
220 d3=int(d3*100+.5)/100
230 c=c-d3
240 Print x;tab(5);d2;tab(17);d3
250 next x
```

```
DePreciation by 3 methods.

OriGonal cost? 8000
Life (years)? 8
ScraP value? 500

StraiGht line dePreciation is $ 937.5 Per year.

      Sum of       Double
Year  DiGits       DeclininG
----  ------       ----------
 1    1666.67        2000
 2    1458.33        1500
 3    1250           1125
 4    1041.67         843.75
 5     833.33         632.81
 6     625            474.61
 7     416.67         355.96
 8     208.33         266.97
```


THINK
DEPRECIATE

Smog Simulation

This program is an adaptation of the smog model originally written by Herbert Peckham. The model assumes that vehicular traffic is the sole producer of smog, a somewhat poor assumption. It also assumes that most automobile traffic occurs during the daylight hours and that traffic volume is very low (actually, zero) at night. The smog generated by the cars is dissipated by atmospheric conditions which vary depending upon sunlight, temperature, and weather. All of these conditions may be specified by the user.

The model could be improved significantly by taking into account other sources of smog, by varying vehicular traffic according to the hour of the day, and by allowing daily variation of weather factors. Nevertheless, even in this rudimentary form it is interesting and instructive.

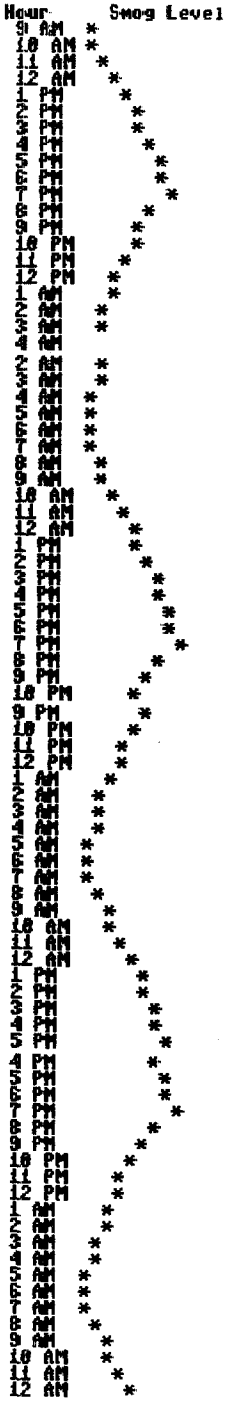
A plot of the smog level is produced in Statement 400. Under some conditions, the smog level reaches a value that cannot be plotted because it is greater than the width of the screen (and printer). For runs with conditions like this, you might want to delete the plotting routine. A more elegant solution would be to estimate the maximum value of the smog level from the input factors and calculate an appropriate plotting multiplication factor.

```
10 Printchr$(147);"Smog Simulation"
20 Print
30 t=7
35 Print
40 Print "Cars Per road mile":Print "Los Angeles    200":Print "Detroit        1
00"
45 Print "Tulsa           25"
60 Print "Smog City    ";:inputc
65 Print
70 Print "Smog Generation factor":Print "1950 Auto    .005":Print"1975 Auto    .
001"
75 Print "Bus          .010"
80 Print "Smog City  ";:inputl1
90 Print
100 Print "Daytime smog breakdown":Print "in Percent Per hour"
105 Print "Clear       .02":Print "Cloudy       .01"
110 Print "Smog City ";:inputri
120 Print
130 Print "Nighttime breakdown":Print "Hot          .05":Print "Cool         .10"
140 Print "Smog City ";:inputr2
150 Print
160 Print "Dispersion (% Per hour)":Print "High wind & rain  1.00"
165 Print "No wind and dry    0.01"
170 Print "Smog City        ";:inputr3
180 Print
190 input "Table or Plot  (t or P)";a$
200 Print "Hour     Smog Level"
210 r=ri
220 k1=l1
230 t1=int((t-6)/12)
240 if t1/2=int(t1/2) then 270
```

```
250 k1=0
260 r=r2
270 s=s+k1*c*10-r*s-r3*s
280 if s<=0 then s=0
290 t=t+1
300 t3=int(t/12)
310 t2=t-t3*12+1
320 x=int(1000*s+.5)/1000
330 if t3/2=int(t3/2) then 350
340 Print t2;"PM";tab(7);:goto 360
350 Print t2;"AM";tab(7);
360 if a$="P" or a$="P" then 390
370 Print x
380 goto 210
390 if x>17 then x=17
400 Print tab(6+x);"*"
410 goto 210
```

```
Smog Simulation


Cars Per road mile
Los Angeles    200
Detroit        100
Tulsa           25
Smog City   ?  100

Smog Generation factor
1950 Auto    .005
1975 Auto    .001
Bus          .010
Smog City ?  .001

Daytime smog breakdown
in Percent Per hour
Clear        .02
Cloudy       .01
Smog City?   .01

Nighttime breakdown
Hot          .05
Cool         .1
Smog City?   .1

Dispersion (% Per hour)
High wind & rain  1.00
No wind and dry   0.01
Smog City      ?   .05
```

```
Table or Plot  (t or P)? t
Hour      Smog Level
 9 AM          1
 10 AM         1.94
 11 AM         2.824
 12 AM         3.654
 1 PM          4.435
 2 PM          5.169
 3 PM          5.859
 4 PM          6.507
 5 PM          7.117
 6 PM          7.69
 7 PM          8.228
 8 PM          6.994
 9 PM          5.945
 10 PM         5.053
 11 PM         4.295
 12 PM         3.651
 1 AM          3.103
 2 AM          2.638
 3 AM          2.242
 4 AM          1.906
 5 AM          1.62
 6 AM          1.377
 7 AM          1.17
 8 AM          2.1
 9 AM          2.974
 10 AM         3.796
 11 AM         4.568
 12 AM         5.294
 1 PM          5.976
```



Lunar Lander Simulation

This program is one of the most popular computer simulations around. It is available in many versions; this one is adapted from the original program written in 1969.

The program represents an exact simulation of an Apollo lunar landing module during the final descent. This portion of the descent would normally be controlled by the on-board computer backed up by another computer in the lunar orbiter, and still another computer on Earth. However, to exercise your knowledge of physics and to make an interesting game, we will assume that all three computers have had a simultaneous malfunction. Hence, it is up to you to land the spacecraft safely.

To make a soft landing, you may change the burn rate of the retro rockets every ten seconds. You have a choice of not firing at all (burn rate of 0) or of firing at a fuel rate of between 8 and 200 pounds per second. Engine ignition occurs at 8 pounds, hence values between 1 and 7 pounds are not possible. You have 16,500 pounds of fuel. This is 500 pounds more than an actual LEM has, which will give you a little margin for error. When you get proficient, change Statement 130 to N = 16000 to simulate the real thing more closely. The capsule weight is 33,000 pounds.

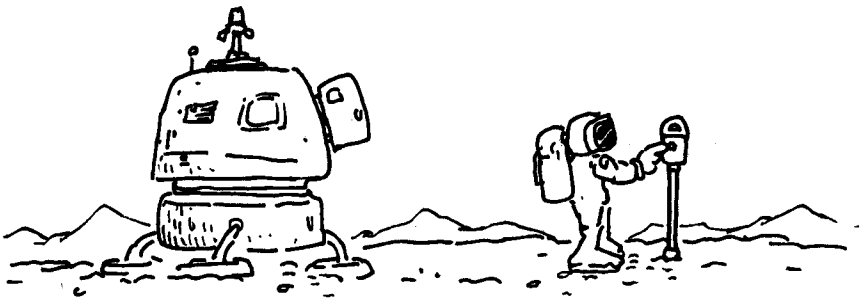
Not that this is the way to come in, but if you did not fire the rockets at all, the estimated time for a free fall descent is 120 seconds to impact (and a huge splat).

Good luck!

```
5 print chr$(14):rem switch lower case
10 print chr$(147);"         Lunar Lander"
15 print "          by David Ahl
20 print:print "Computer malfunction - take manual control of Lunar capsule."
30 print
40 print "Capsule weight -- 32500 lbs"
45 print "Fuel weight  --   16500 lbs"
50 print
60 print "Set retro burn rate every 10 secs to 0 or a value between";
70 print "8 & 200 lbs per sec."
80 print
90 print "Miles     MPH     Fuel     Time     Burn"
95 print "-----     ---     ----     ----     ----"
100 a=120
110 v=1
120 m=33000
130 n=16500
140 g=.001
150 z=1.8
160 p=int(3600*v+.5)
170 if abs(p)>=1 then 190
180 p=int((3600*v+.5)*1000)/1000
190 if a<10 then 210
200 d=int(a+.5):goto 240
210 if a<1 then 230
```

```
220 d=(int(10*a+.5))/10:goto 240
230 d=(int(1000*a+.5))/1000
240 Print d;tab(8);p;tab(17);m-n;tab(26);l;tab(34);:inPut k
260 if k>200 then 290
270 if k>7 then 300
280 if k=0 then 300
290 Print "Can't do--again Please":goto 240
300 t=10
320 if m-n<.001 then 430
330 if t<.001 then 160
340 s=t
350 if m>n+s*k then 370
360 s=(m-n)/k
370 gosub 770
380 if i<=0 then 630
390 if v<=0 then 410
400 if J<0 then 690
410 gosub 570
420 goto 320
430 Print "You ran out of fuel at";l;"seconds."
440 s=(-v+sqr(v*v+2*a*g))/g
450 v=v+g*s
460 l=l+s
470 w=3600*v
480 Print "On the moon at";int(l);"sec.  Impact velocity was";
485 Print int(100*w+.5)/100;"MPh"
490 if w>1.2 then 510
500 Print "Perfect landing!":end
510 if w>10 then 530
520 Print "Doooomph!  That's rough on the shocks.  You need more practice!":end
530 if w>60 then 550
540 Print "Severe craft damage!  You're stranded   until help arrives.":end
550 Print "Sorry, there were no survivors.  You    blew it."
560 Print "In fact, you blasted a new lunar crater";int(w*.278);"feet deep!":end
570 l=l+s
580 t=t-s
590 m=m-s*k
600 a=i
610 v=j
620 return
630 if s<.005 then 470
640 d=v+sqr(v*v+2*a*(g-z*k/m))
650 s=2*a/d
660 gosub 770
670 gosub 570
680 goto 630
690 w=(1-m*g/(z*k))/2
700 s=m*v/(z*k*(w+sqr(w*w+v/z)))+.05
710 gosub 770
720 if i<=0 then 630
730 gosub 570
740 if J>0 then 320
750 if v>0 then 690
760 goto 320
770 q=s*k/m
780 J=v+g*s-z*q*(1+q*(.5+q*(1/3+q*(.25+q/5))))
790 i=a-g*s*s/2-v*s+z*s*q*(.5+q*(1/6+q*(1/12+q/20)))
800 return
```

```
        Lunar Lander
        by David Ahl

Computer malfunction - take manual control of Lunar capsule.

Capsule weight -- 32500 lbs
Fuel weight  --   16500 lbs

Set retro burn rate every 10 secs to 0 or a value between 8 & 200 lbs per sec.

Miles     MPH     Fuel    Time    Burn
-----     ---     ----    ----    ----
 120     3600    16500     0    ? 0
 110     3636    16500     10   ? 0
 100     3672    16500     20   ? 0
 90      3708    16500     30   ? 50
 79      3645    16000     40   ? 50
 69      3581    15500     50   ? 50
 59      3515    15000     60   ? 100
 50      3341    14000     70   ? 100
 41      3161    13000     80   ? 100
 32      2974    12000     90   ? 100
 24      2779    11000     100  ? 200
 17      2325    9000      110  ? 200
 11      1832    7000      120  ? 200
 7.1     1292    5000      130  ? 200
 4.4     695     3000      140  ? 200
 3.3     30      1000      150  ? 150
You ran out of fuel at 156.666667 seconds.
On the moon at 372 sec.  Impact velocity was 448.42 MPh
Sorry, there were no survivors.  You blew it.
In fact, you blasted a new lunar crater 124 feet deep!
```



Hammurabi

Hammurabi is one of the all-time favorite computer games. On the one hand, it may be considered a game, but on the other it is an intriguing simulation of barter and management.

Hammurabi is your servant as you try to manage the ancient city-state of Sumeria. The economy of the city-state revolves around just one thing—the annual crop of grain (probably soybeans).

Each year, you must determine how many bushels of grain you wish to feed to your people (you'll quickly discover how much a person needs to survive), how much you wish to use as seed in planting crops for the coming year, how much you wish to use for the purchase of additional land from your neighboring city-state, and how much you wish to put in storage.

Of course, if you have a bad harvest or if rats overrun your grain storage bins, you may have to sell land in order to get enough grain to keep your people from starving, or to plant the land for the coming year. Unfortunately, disasters always seem to strike, forcing you to sell land, when the price is at an all-time low; but that's not any different from the real world.

Most people start to play this game with noble ambitions. However, before long, they start longing for a plague to trim their growing population. Or they deliberately starve some people to keep things in balance (gosh, maybe these zero population growth people have something, after all!).

Over the years, this game more than any other, has spawned a host of look-alikes, extensions, and modifications. Indeed, several manufacturers have taken my original with no changes whatsoever, put it in a fancy box, and charged a handsome price for it. Accept no imitations! Here is the original game (with the dialog shortened slightly) for you to run on your computer.

If you want to experiment with changes, here are some suggestions. In the existing game, plagues randomly occur 15% of the time; lower this to 10% or 5%. People now require a fixed amount of food; vary this amount slightly from year to year. Permit the construction of a rat-proof grain bin, but this must cost a fair amount. Introduce a mining industry as well as agriculture. How about fishing or tourism? Let your imagination run wild. Experiment! Have fun!

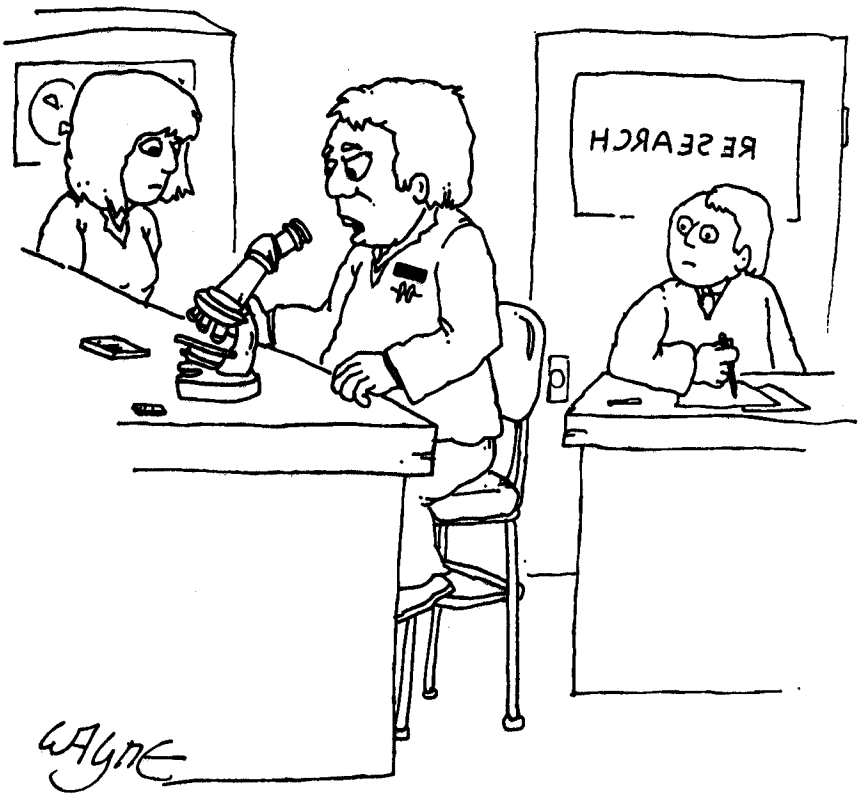
```
5 Print chr$(14):rem switch lower case
10 Print chr$(147);"Try to sovern ancient Sumeria for 10    years."
30 d1=0:p1=0
40 z=0:p=95:s=2800:h=3000:e=h-s
50 y=3:a=h/y:i=5:q=1
60 d=0
70 Print:z=z+1
80 Print " Hammurabi: Year";z
90 Print d;"Starved,";i;"Born"
100 p=p+i
110 if q>0 then 140
120 p=int(p/2)
130 Print " Horrible Plaque!"
140 Print " Population =";p
150 Print " You own";a;"acres which Produced";y;"    bushels Per acre."
160 Print " Rats ate";e;"bushels."
190 Print s;"bushels stored.":Print
200 if z=11 then 840
210 c=int(10*rnd(1)):y=c+17
220 Print "Land =";y;"bushels/acre"
230 input "Acres to buy";q
250 if q<0 then 810
260 if y*q<=s then 290
270 gosub 730
280 goto 230
290 if q=0 then 320
300 a=a+q:s=s-y*q:c=0
310 goto 380
320 input "Acres to sell";q
330 if q<0 then 810
340 if q<a then 370
350 gosub 760
360 goto 320
370 a=a-q:s=s+y*q:c=0
380 Print
390 input "Bushels for feed";q
400 if q<0 then 810
410 if q<=s then 440
420 gosub 730
430 goto 390
440 s=s-q:c=1
450 Print
460 input "Acres to seed";d:if d=0 then 570
470 if d<0 then 810
480 if d<=athen 500
490 goto 460
500 if int(d/2)<=s then 530
510 gosub 730
520 goto 460
530 if d<10*P then 560
540 Print "Not enough People"
550 goto 460
560 s=s-int(d/2)
570 gosub 790
580 y=c:h=d*y:e=0
590 gosub 790
600 if int(c/2)<>c/2 then 620
620 s=s-e+h
630 gosub 790
640 i=int(c*(20*a+s)/P/100+1)
650 c=int(q/20)
660 q=int(10*(2*rnd(1)-.3))
```

```
670 ifP<c then 60
680 d=P-c:if d>.45*P then 710
690 P1=((z-1)*P1+d*100/P)/z
700 P=c:d1=d1+d:goto 70
710 Print:Print d;"People starved!"
720 Print "You are impeached.":goto 1030
730 Print "Think again.  You have just";s;"bushels"
750 return
760 Print "Think again.  You own only";a;"acres"
780 return
790 c=int(rnd(1)*5)+1
800 return
810 Print "Hammurabi: I can't do that.  Get yourself another steward!":goto 1030
840 Print "In 10 years";P1;"%"
850 Print "starved per year.  A total of";d1;"died!"
870 l=a/P
880 Print "You started with 10 acres/person and ended with";l;"acres/person."
910 Print
920 if P1>33 then 720
930 if l<7 then 720
940 if P1>10 then 990
950 if l<9 then 990
960 if P1>3 then 1010
970 if l<10then 1010
980 Print "Fantastic performance!":goto 1030
990 Print "Very heavy handed!  People are rebelling!":goto 1030
1010 Print "Not bad, but could be somewhat beter."
1030 Print:Print "So long for now."
1050 end
```

```
Try to govern ancient Sumeria for 10 years.

 Hammurabi: Year 1
 0 Starved, 5 Born
 Population = 100
 You own 1000 acres which produced 3 bushels per acre.
 Rats ate 200 bushels.
 2800 bushels stored.
Land = 20 bushels/acre
Acres to buy? 0
Acres to sell? 0

Bushels for feed? 1500

Acres to seed? 500

 Hammurabi: Year 2
 25 Starved, 7 Born
 Population = 82
 You own 1000 acres which produced 3 bushels per acre.
 Rats ate 0 bushels.
 2550 bushels stored.
Land = 23 bushels/acre
Acres to buy? 0
Acres to sell? 0

Bushels for feed? 1500
```

```
Acres to seed? 500

 Hammurabi: Year 3
 7 Starved, 8 Born
 Population = 83
 You own 1000 acres which Produced 2 bushels Per acre.
 Rats ate 0 bushels.
 1000 bushels stored.
Land = 24 bushels/acre
Acres to buy? 0
Acres to sell? 200

Bushels for feed? 1500

Acres to seed? 500

 Hammurabi: Year 4
 8 Starved, 6 Born
 Horrible Plague!
 Population = 40
 You own 800 acres which Produced 4 bushels Per acre.
 Rats ate 0 bushels.
 6850 bushels stored.
Land = 20 bushels/acre
Acres to buy? 0
Acres to sell? 0

Bushels for feed? 2000

Acres to seed? 800
Not enough People
Acres to seed? 500
Not enough People
Acres to seed? 300

 Hammurabi: Year 5
 0 Starved, 27 Born
 Population = 67
 You own 800 acres which Produced 1 bushels Per acre.
 Rats ate 0 bushels.
 5000 bushels stored.
Land = 20 bushels/acre
Acres to buy? 0
Acres to sell? 20

Bushels for feed? 2000


Acres to seed? 500

 Hammurabi: Year 6
 0 Starved, 9 Born
 Population = 76
 You own 780 acres which Produced 2 bushels Per acre.
 Rats ate 0 bushels.
 4150 bushels stored.
Land = 20 bushels/acre
Acres to buy? 0
Acres to sell? 80

Bushels for feed? 2500
```

```
Acres to seed? 500

 Hammurabi: Year 7
 0 Starved, 8 Born
 Population = 84
 You own 700 acres which Produced 2 bushels Per acre.
 Rats ate 0 bushels.
 4000 bushels stored.
Land = 22 bushels/acre
Acres to buy? 0
Acres to sell? 0

Bushels for feed? 2000

Acres to seed? 800
Acres to seed? 30

 Hammurabi: Year 8
 0 Starved, 6 Born
 Population = 90
 You own 700 acres which Produced 3 bushels Per acre.
 Rats ate 0 bushels.
 2075 bushels stored.
Land = 21 bushels/acre
Acres to buy? 0
Acres to sell? 300

Bushels for feed? 3000

Acres to seed? 300

 Hammurabi: Year 9
 0 Starved, 5 Born
 Population = 95
 You own 400 acres which Produced 1 bushels Per acre.
 Rats ate 0 bushels.
 5525 bushels stored.
Land = 18 bushels/acre
Acres to buy? 0
Acres to sell? 400
Think a9ain.  You own only 400 acres
Acres to sell? 40

Bushels for feed? 2000

Acres to seed? 500
Acres to seed? 300

 Hammurabi: Year 10
 0 Starved, 7 Born
 Population = 102
 You own 360 acres which Produced 2 bushels Per acre.
 Rats ate 0 bushels.
 4695 bushels stored.
Land = 19 bushels/acre
Acres to buy? 100

Bushels for feed? 0

Acres to seed? 0
 102 People starved!
You are impeached.

So lon9 for now.
```



"Rats! A bacterium just ate the new micro-mini computer."

References

The magazine referred to in the text is:

• *Creative Computing.* This is the leading magazine of software and applications for all small computers. It carries articles, tutorials, how-to applications, and extensive in-depth evaluations.

Books referred to in the text include:

• *Computers in Mathematics: A Sourcebook of Ideas.* Hundreds of classroom-tested ideas for using computers to learn about mathematics.

• *Computers in Science and Social Studies.* Scores of simulation programs in biology, ecology, physics and management of real world systems.

All of these books and magazines are available from *Creative Computing.* Write or call for the current price:

Creative Computing
39 E. Hanover Ave.
Morris Plains, NJ 07950
(800) 631-8112
In NJ (201) 540-0445

Notes