2 HOW TO GET RESULTS

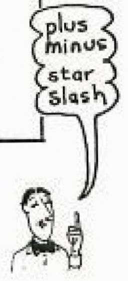
In this chapter, we'll dive right into some specifics that you need to know before we go on. Specifically, we'll introduce some of the arithmetic instructions besides $\boxed{+}$ and some special operators for rearranging the order of numbers on the stack, so that you'll be able to write mathematical equations in FORTH.

FORTH Arithmetic — Calculator Style

Here are the four simplest integer-arithmetic operators in FORTH:†

pronounced:

+	(n1 n2 — sum)	Adds.
-	(n1 n2 — diff)	Subtracts (n1-n2).
*	(n1 n2 — prod)	Multiplies.
/	(n1 n2 — quot)	Divides (n1/n2).



Unlike calculators, computer terminals don't have special keys for multiplication and division. Instead we use $\boxed{*}$ and $\boxed{/}$.

†If Math Is Not Your Thing

Don't worry if this chapter looks a little like an algebra textbook. Solving math problems is only one of the things you can do with FORTH. Later we'll explore some of the other things FORTH can do.

Meanwhile, we'd like to remind you that integers are whole numbers, such as:

... -3, -2, -1, 0, 1, 2, 3, ...

Integer arithmetic (logically enough) is arithmetic that concerns itself only with integers, not with decimal-point numbers, such as 2.71.

In the first chapter, we learned that we can add two numbers by putting them both on the stack, then executing the word $\boxed{+}$, then finally executing the word $\boxed{.}$ (dot) to get the result printed at our terminal.

```
17 5 + . 22 ok
```

We can use this method with all of FORTH's arithmetic operators. In other words, we can use FORTH like a calculator to get answers, even without writing a "program." Try a multiplication problem:

```
7 8 * . 56 ok
```

By now we've seen that the operator comes after the numbers. In the case of subtraction and division, though, we must also consider the order of numbers ("7 - 4" is not the same as "4 - 7").

Just remember this rule:

> To convert to postfix, simply move the operator to the end of the expression:

Infix	Postfix
3 + 4	$3\ \overset{\curvearrowright}{\ 4\ }\ +$
500 - 300	$500\ \overset{\curvearrowright}{\ 300\ }\ -$
6 X 5	$6\ \overset{\curvearrowright}{\ 5\ }\ *$
20 / 4	$20\ \overset{\curvearrowright}{\ 4\ }\ /$

So to do this subtraction problem:

```
7 - 4 =
```

simply type in

```
7 4 - . 3 ok
```

For Adventuresome Newcomers Sitting at a Terminal

If you're one of those people who likes to fool around and figure things out for themselves without reading the book, then you're bound to discover a couple of weird things. First off, as we told you, these operators are integer operators. That not only means that you can't do calculations with decimal values, like

```
10.00 2.25 +
```

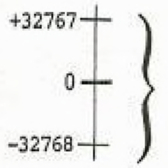
it also means that you can only get integer results, as in

```
21 4 / . 5 ok instead of 5.25 ok
```

Another thing is that if you try to multiply:

```
10000 10 *
```

or some such large numbers, you'll get a crazy answer. So we're telling you up front that with the operators introduced so far and with $\boxed{.}$ to print the results, you can't have any numbers that are higher than 32767 or lower than -32768. Numbers within this range are called "single-length signed numbers."



Allowable range of single-length signed numbers.

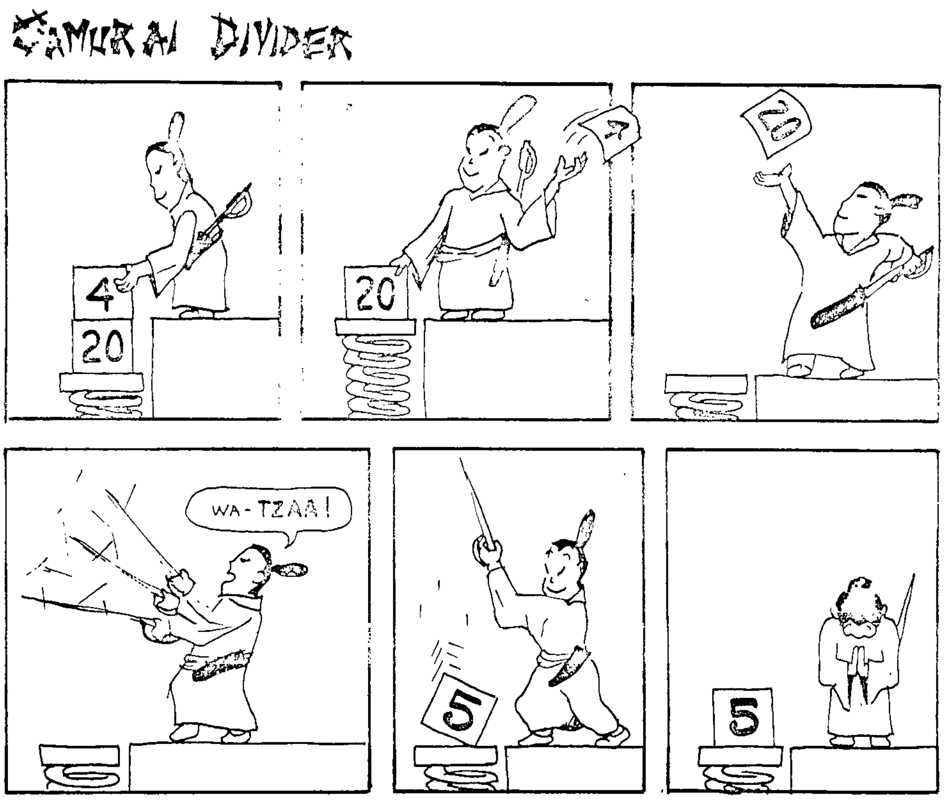
Notice, in the list of FORTH words a few pages back, the letter "n," which stands for "number." Since FORTH uses single-length numbers more often than other types of numbers, the "n" signifies that the number must be single-length. And yes, there are other operators that extend this range ("double-length" operators, which are indicated by "d").

All of these mysteries will be explained in time, so stay tuned.

The order of numbers stays the same. Let's try a division problem:

```
    20 4 / . 5 ok
```

The word $\boxed{/}$ is defined to divide the second number on the stack by the top number:



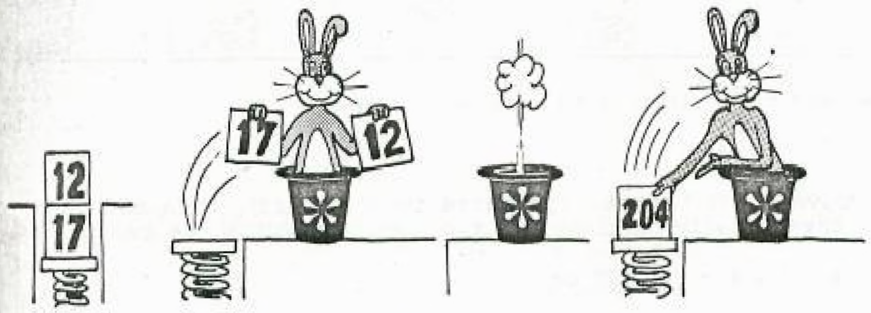
What do you do if you have more than one operator in an expession, like:

```
    4 + (17 * 12)
```

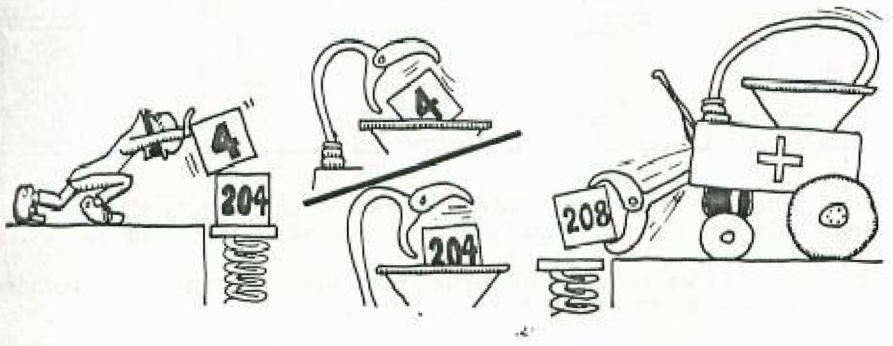
you ask? Let's take it step-by-step: the parentheses tell you to first multiply seventeen by twelve, then add four. So in FORTH you would write:

```
    17 12 * 4 + . 208 ok
```

and here's why:



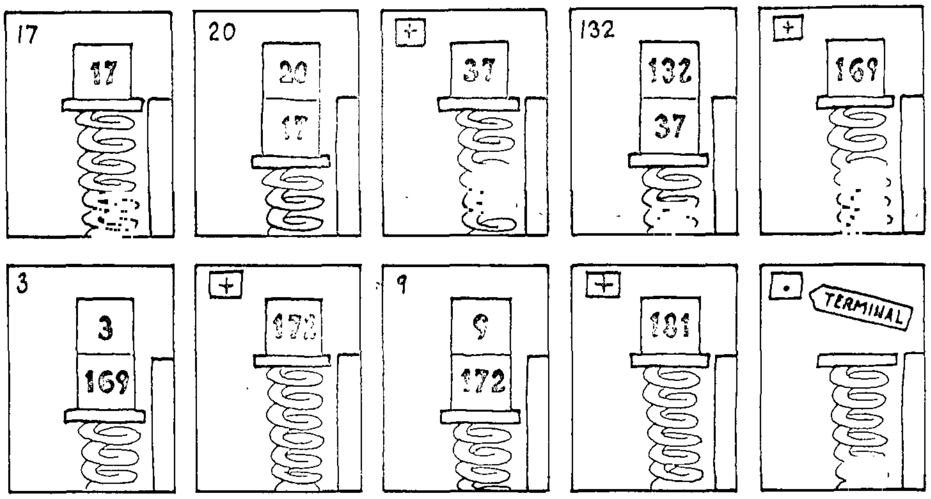
17 and 12 go onto the stack. [*] multiplies them and returns the result.



Then the four goes onto the stack, on top of the 204. [+] rolls out the adding machine and adds them together, returning only the result.

Or suppose you want to add five numbers. You can do it in FORTH like this:

```
17 20 + 132 + 3 + 9 + . 181 ok
```



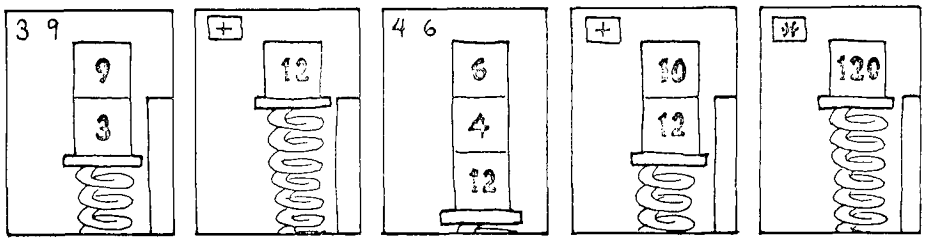
Now here's an interesting problem:

(3+9) * (4+6)

To solve it we have to add three to nine first, then add four to six, then finally multiply the two sums. In FORTH, we can write

```
3 9 + 4 6 + * . 120 ok
```

and here's what happens:



Notice that we very conveniently saved the sum twelve on the stack while we went on about the business of adding four to six.

Remember that we're not concerned yet with writing definitions. We are simply using FORTH as a calculator.

If you're like most beginners, you probably would like to try your hand at a few practice problems until you feel more comfortable with postfix.

Postfix Practice Problems (Quizzie 2-a)

Convert the following infix equations to postfix "calculator style." For example,

ab + c

would become

a b * c +

1. c(a + b)

2. $\frac{3a - b}{4} + c$

†3. $\frac{0.5\ ab}{100}$

4. $\frac{n + 1}{n}$

5. x(7x + 5)

Convert the following postfix expressions to infix:

6. a b - b a + /

7. a b 10 * /

---

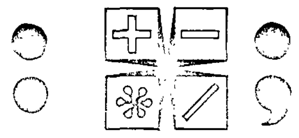
†For Beginners

Remember, we're only using integer arithmetic, so you'll have to be clever.

Answers — Quizzie 2-a

1. a b + c * or c a b + *
2. 3 a * b - 4 / c +
3. a b * 100 / 2 / or a b * 200 / .
4. n 1 + n /
5. 7 x * 5 + x *
6. $\frac{a - b}{b + a}$
7. $\frac{a}{10b}$

FORTH Arithmetic -- Definition Style



In Chap. 1 we saw that we could define new words in terms of numbers and other pre-defined words. Let's explore some further possibilities, using some of our newly-learned math operators.

Let's say we want to convert various measurements to inches. We know that

```
    1 yard = 36 inches
```

and

```
    1 foot = 12 inches
```

so we can define these two words:

```
    : YARDS>IN   36 * ; ok
    : FT>IN   12 * ; ok
```

where the names symbolize "yards-to-inches" and "feet-to-inches." Here's what they do:

```
    10 YARDS>IN . 360 ok
    2 FT>IN . 24 ok
```

If we always want our result to be in inches, we can define:

```
    : YARDS   36 * ; ok
    : FEET   12 * ; ok
    : INCHES ; ok
```

so that we can use the phrase

```
    10 YARDS 2 FEET + 9 INCHES + . 393 ok
```

Notice that the word INCHES doesn't do anything except remind the human user what the nine is there for. If we really want to get fancy, we can add these three definitions:

```
    : YARD   YARDS ; ok
    : FOOT   FEET ; ok
    : INCH ; ok
```

so that the user can enter the singular form of any of these nouns and still get the same result:

```
    1 YARD  2 FEET +  1 INCH + . 61 ok
    2 YARDS  1 FOOT + . 84 ok
```

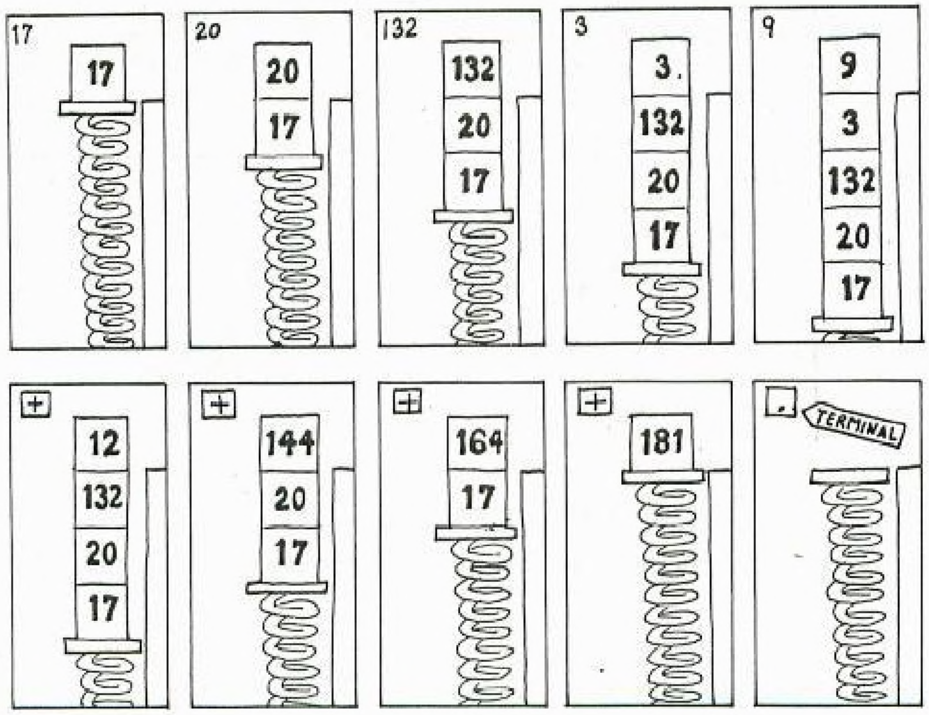
So far we have only defined words whose definitions contain a single math operator. But it's perfectly possible to put many operators inside a definition, if that's what you need to do.

Let's say we want a word that computes the sum of five numbers on the stack. A few pages back we summed five numbers like this:

```
    17 20 + 132 + 3 + 9 + . 181 ok
```

But we can also enter

```
    17 20 132 3 9 + + + + . 181 ok
```



We get the same answer, even though we've clustered all the numbers into one group and all the operators into another group. We can write our definition like this:

```
    : 5#SUM   + + + + ; ok
```

and execute it like this:

```
    17 20 132 3 9 5#SUM . 181 ok
```

If we were going to keep 5#SUM for future use, we could enter it into our ever-growing glossary, along with a note that it "expects five arguments"† on the stack, which it will add together.

Here's another equation to write a definition for:‡

```
    (a + b) * c
```

As we saw in Quizzie 2-a, this expression can be written in postfix as

```
    c a b + *
```

Thus we could write our definition

```
    : SOLUTION   + * ; ok
```

as long as we make sure that we enter the arguments in the proper order:

```
    c a b SOLUTION
```

---

†For Semantic Freaks

In mathematics, the word "argument" refers to an independent variable of a function. Computer linguists have borrowed this term to refer to numbers being operated on by operators. They have also borrowed the word "parameters" to describe pretty much the same thing.

‡For Beginners Who Like Word-problems

If a jet plane flies at an average air speed of 600 mph and if it flies with a tail wind of 25 mph, how far will it travel in five hours?

If we define

```
    : FLIGHT-DISTANCE   + * ;
```

we could enter

```
    5 600 25 FLIGHT-DISTANCE . 3125 ok
```

Try it with different values, including head winds (negative values).

Definition-style Practice Problems (Quizzie 2-b)

Convert the following infix expressions into FORTH definitions and show the stack order required by your definitions. Since this is Quizzie 2-b, you can name your definitions 2B1, 2B2, etc. For example,

1. ab + c would become `: 2B1 * + ;`

   which expects this stack order: (c b a -- result)

2. $\frac{a - 4b}{6} + c$

3. $\frac{a}{8b}$

4. $\frac{0.5\ ab}{100}$

5. $a(2a + 3)$

6. $\frac{a - b}{c}$

Answers — Quizzie 2-b

2.
```
: 2B2   4 * - 6 / + ;
(c a b -- result)
```

3.
```
: 2B3   8 * / ;
(a b -- result)
```

4.
```
: 2B4   * 200 / ;
(a b -- result)
```

5.
```
: 2B5   2 * 3 +  * ;
(a a -- result)
```

6. If you said this one's impossible, you're right!--at least without the stack manipulation operators which we'll introduce very shortly.

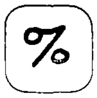
The Division Operators

The word / is FORTH's simplest division operator. Slash supplies only the quotient; any remainder is lost. If you type:

```
     22 4 / . 5 ok
```

you get only the quotient five, not the remainder two.

If you're thinking of a pocket calculator's per-cent operator, then five is not the full answer.

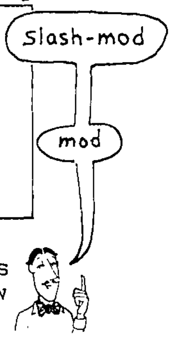


But / is only one of several division operators supplied by FORTH to give you the flexibility to tell the computer exactly what you want it to do.

For example, let's say you want to solve this problem: "How many dollar bills can I get in exchange for 22 quarters?" The real answer, of course, is exactly 5, not 5.5. A computerized money changer, for example, would not know how to give you 5.5 dollar bills.

Here are two more FORTH division operators:

			pronounced:
/MOD	(u1 u2 -- u-rem u-quot)	Divides. Returns the remainder and quotient.	slash-mod
MOD	(u1 u2 -- u-rem)	Returns the remainder from division.	mod



The "u" stands for "unsigned." We'll see what this means in the chapter on computer numbers. For now though, it means that the numbers can't be negative.

/MOD gives both the remainder and the quotient; MOD gives the remainder only.† (For /MOD, the stack notation in the table indicates that the quotient will be on the top of the stack, and the remainder below. Remember, the rightmost represents the topmost.)

†For the Curious

MOD refers to the term "modulo," which basically means "remainder."

Let's try the first one:

```
        22 4 /MOD . . 5 2 ok
```

Here $\boxed{\text{/MOD}}$ performs the division and puts both the quotient and the remainder on the stack. The first dot prints the quotient because the quotient was on top.

SAMURAI $\boxed{\text{/MOD}}$ (slash's older brother)

4
22
HA-DAA!
5
2

With what we've learned so far, we can easily define this word:

```
   : QUARTERS   4 /MOD .  ." ONES AND "  .  ." QUARTERS " ;
```

So that you can type:

```
    22 QUARTERS
```

with this result:

```
    22 QUARTERS 5 ONES AND 2 QUARTERS ok
```

The second word in the table, $\boxed{\text{MOD}}$, leaves only the remainder. For example in:

```
    22 4 MOD . 2 ok
```

the two is the remainder.

Stack Maneuvers

If you worked Prob. 6 in the last set, you discovered that the infix equation

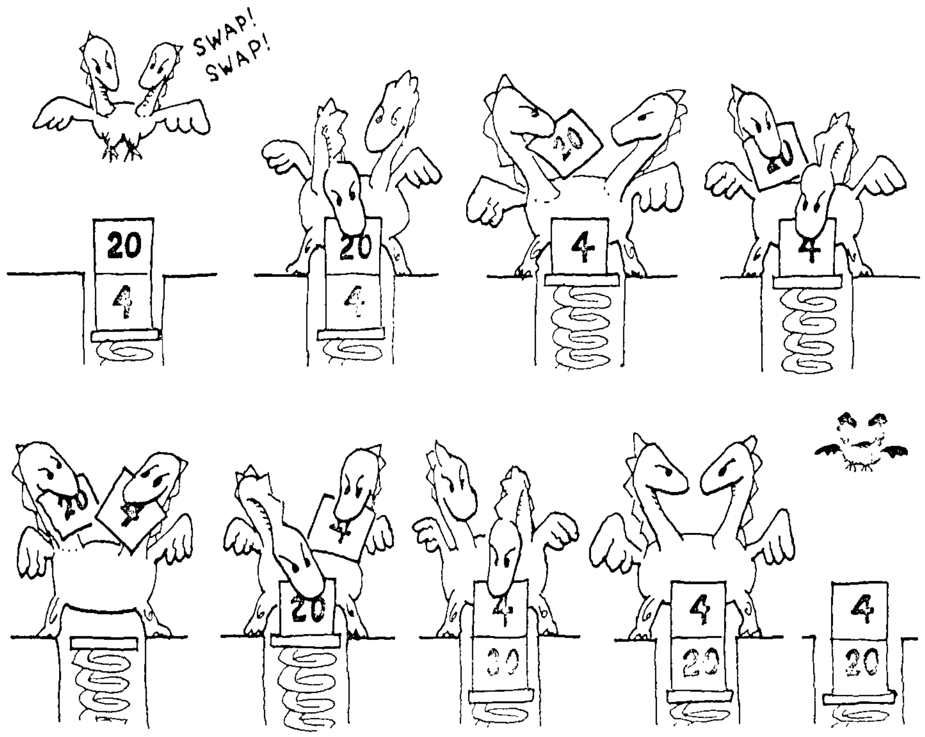
$$\frac{a - b}{c}$$

cannot be solved with a definition unless there is some way to rearrange values on the stack.

Well, there is a way: by using a "stack manipulation operator" called SWAP.

SWAP

The word SWAP is defined to switch the order of the top two stack items:



As with the other stack manipulation operators, you can test SWAP at your terminal in "calculator style"; that is, it doesn't have to be contained within a definition.

First enter

```
1 2 . . 2 1 ok
```

then again, this time with SWAP:

```
1 2 SWAP . . 1 2 ok
```

Thus Prob. 6 can be solved with this phrase:

```
- SWAP /
```

with (c a b --) on the stack.

Let's give a, b, and c these test values:

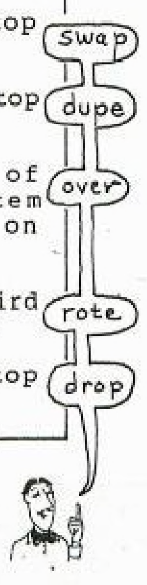
a = 10   b = 4   c = 2

then put them on the stack and execute the phrase, like so:

```
2 10 4 - SWAP / . 3 ok
```

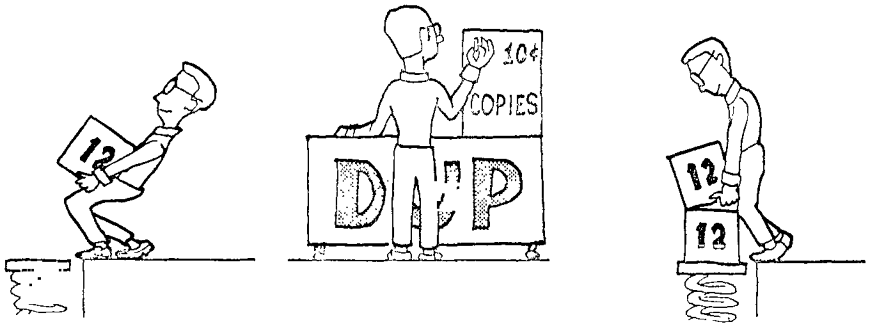
Here is a list of several stack manipulation operators, including SWAP.

SWAP	(n1 n2 -- n2 n1)	Reverses the top two stack items.
DUP	(n -- n n)	Duplicates the top stack item.
OVER	(n1 n2 -- n1 n2 n1)	Makes a copy of the second item and pushes it on top.
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotates the third item to the top.
DROP	(n --)	Discards the top stack item.



DUP

The next stack manipulation operator on the list, DUP, simply makes a second copy (duplicate) of the top stack item.



For example, if we have "a" on the stack, we can compute:

a^2

as follows:

```
DUP *
```

in which the following steps occur:

Operation	Contents of Stack
	a
DUP	a a
*	a^2

$\boxed{\text{OVER}}$

Now somebody tells you to evaluate the expression:

a * (a + b)

given the following stack order:

(a b —)

But, you say, I'm going to need a new manipulation operator: I want two copies of the "a," and the "a" is <u>under</u> the "b." Here's the word you need: $\boxed{\text{OVER}}$. $\boxed{\text{OVER}}$ simply makes a copy of the "a" and leapfrogs it <u>over</u> the "b":

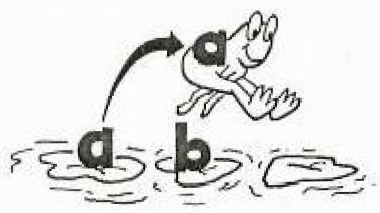
(a b — a b a)

Now the expression:

a * (a + b)

can easily be written:

OVER + *



Here's what happens:

Operation	Contents of Stack
	a b
OVER	a b a
+	a (b+a)
*	a*(b+a)

When writing equations in FORTH, it's best to "factor them out" first. For example, if somebody asks you to evaluate:

$a^2 + ab$

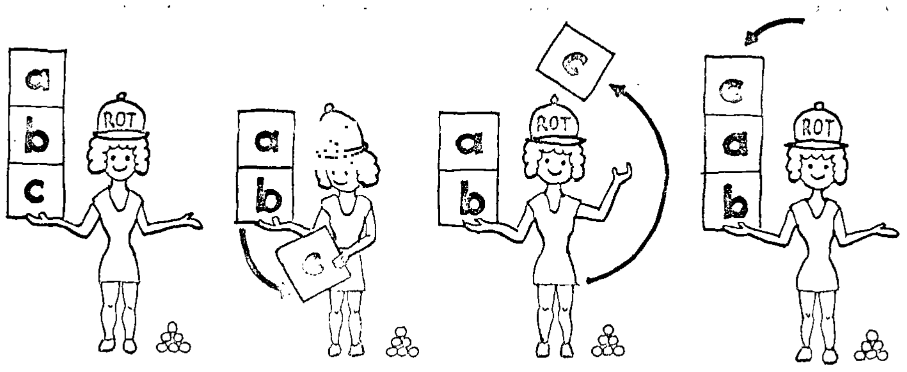
in FORTH, you'll find it quite complicated (and maybe even impossible) using the words we've introduced so far ... unless you factor out the expression to read:

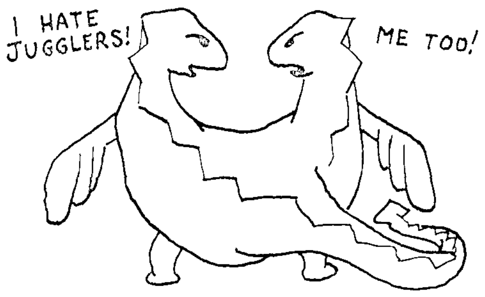
a * (a + b)

which is the expression we just evaluated so easily.

ROT

The fourth stack manipulator on the list is ROT (pronounced rote), which is short for "rotate." Here's what ROT does to the top three stack values:





For example, if we need to evaluate the expression:

    ab - bc

we should first factor out the "b"s:

    b * (a - c)

Now if our starting-stack order is this:

    (c b a --)

we can use:

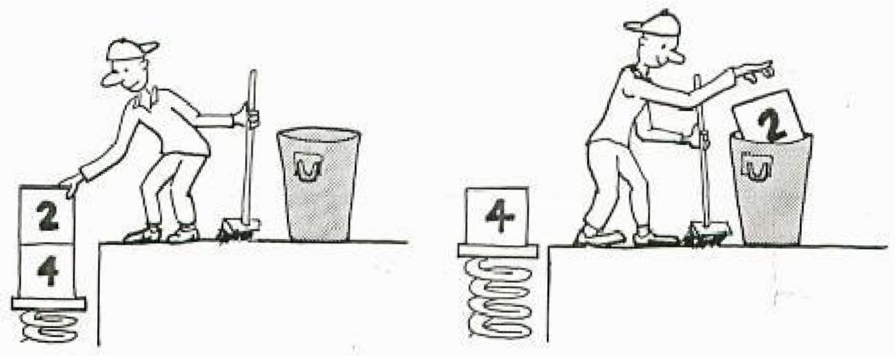
    ROT - *

in which the following steps will occur:

Operation	Contents of Stack
	c b a
ROT	b a c
-	b (a-c)
*	(b*(a-c))

$\boxed{\text{DROP}}$

The final stack manipulation operator on the list is $\boxed{\text{DROP}}$. All it does is discard the top stack value.



Pretty simple, huh? We'll see some good uses for $\boxed{\text{DROP}}$ later on.

A Handy Hint

A Non-destructive Stack Print

Beginners who are just learning to manipulate numbers on the stack in useful ways very often find themselves typing a series of dots to see what's on the stack after their manipulations. The problem with dots, though, is that they don't leave the numbers on the stack for future manipulations.

Here is the definition of a very useful word for such beginners. .S prints out all the values that happen to be on the stack "non-destructively"; that is, without removing them. Type the definition in as shown here, and don't worry about how it works.

```
    : .S   CR  'S S0 @ 2- DO  I @ .  -2 +LOOP ; ok
```

Let's test it, first with nothing on the stack:

```
    .S
    0 ok
```

As you can see, in this version of .S, we always see at least one number as a reference for the bottom of the stack; that is, the same number we see when we type a [.] and get

```
    .RETURN 0 . STACK EMPTY
```

Now let's try it with numbers on the stack:

```
    1 2 3 .S
    0 1 2 3 ok

    ROT .S
    0 2 3 1 ok
```

<u>Stack Manipulation and Math Definitions</u> (Quizzie 2-c)

1. Write a phrase which flips three items on the stack, leaving the middle number in the middle; that is,

   a b c becomes c b a

2. Write a phrase that does what $\boxed{\text{OVER}}$ does, without using $\boxed{\text{OVER}}$.

3. Write a definition called <ROT, which rotates the top three stack items in the opposite direction from $\boxed{\text{ROT}}$; that is,

   a b c becomes c a b

Write definitions for the following equations, given the stack effects shown:

4. $\frac{n + 1}{n}$ (n — result)

5. x(7x + 5) (x — result)

6. $9a^2$ – ba (a b — result)

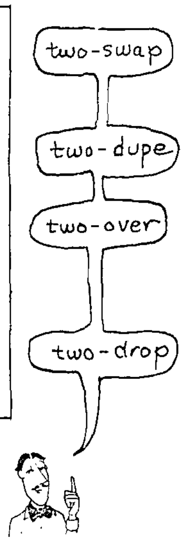
<u>Answers — Quizzie 2-c</u>

```
1.  SWAP ROT
2.  SWAP DUP ROT SWAP
3.  : <ROT   ROT ROT ;
4.  : 2C4    DUP 1 + SWAP / ;      or
    : 2C4    DUP 1+ SWAP / ;
5.  : 2C5    DUP 7 *  5 + * ;
6.  : 2C6    OVER 9 *  SWAP - * ;
```

Playing Doubles[†]

The next four stack manipulation operators should look vaguely familiar:

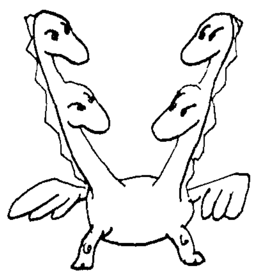
2SWAP	(d1 d2 -- d2 d1)	Reverses the top two pairs of numbers.
2DUP	(d -- d d)	Duplicates the top pair of numbers.
2OVER	(d1 d2 -- d1 d2 d1)	Makes a copy of the second pair of numbers and pushes it on top.
2DROP	(d --)	Discards the top pair of numbers.



The prefix "2" indicates that these stack manipulation operators handle numbers in pairs.[‡] The letter "d" in the stack effects column stands for "double." "Double" has a special significance that we will discuss when we discuss "n" and "u."

The "2"-manipulators listed above are so straightforward, we won't even bore you with examples.

One more thing: there are still some stack manipulators we haven't talked about yet, so don't go crazy by trying too much fancy footwork on the stack.


Guess who.

[†]FORTH-79 Standard

These words are part of the Standard's "Double Number Word Set," which is optional in a Standard system. $\boxed{2\ldots}$ is included.

[‡]For Old Hands

They can also be used to handle double-length (32-bit) numbers.

Here's a list of the FORTH words we've covered in this chapter:

+	(n1 n2 -- sum)	Adds.
-	(n1 n2 -- diff)	Subtracts (n1-n2).
*	(n1 n2 -- prod)	Multiplies.
/	(n1 n2 -- quot)	Divides (n1/n2).
/MOD	(u1 u2 -- u-rem u-quot)	Divides. Returns the remainder and quotient.
MOD	(u1 u2 -- u-rem)	Returns the remainder from division.
SWAP	(n1 n2 -- n2 n1)	Reverses the top two stack items.
DUP	(n -- n n)	Duplicates the top stack item.
OVER	(n1 n2 -- n1 n2 n1)	Makes a copy of the second item and pushes it on top.
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotates the third item to the top.
DROP	(n --)	Discards the top stack item.
2SWAP	(d1 d2 -- d2 d1)	Reverses the top two pairs of numbers.
2DUP	(d -- d d)	Duplicates the top pairs of numbers.
2OVER	(d1 d2 -- d1 d2 d1)	Makes a copy of the second pair of numbers and pushes it on top.
2DROP	(d --)	Discards the top pair of numbers.

Review of Terms

Double-length numbers	integers which encompass a range of over -2 billion to +2 billion (and which we'll introduce officially in Chap. 7).
Single-length numbers	integers which fall within the range of -32768 to +32767: the only numbers which are valid as the arguments or results of any of the operators we've discussed so far. (This seemingly arbitrary range comes from the way computers are designed, as we'll see later on.)

Problems -- Chapter 2

(answers in the back of the book)

1. What is the difference between $\boxed{\text{DUP}}$ $\boxed{\text{DUP}}$ and $\boxed{\text{2DUP}}$?

2. Write a phrase which will reverse the order of the top four items on the stack; that is,

   (1 2 3 4 -- 4 3 2 1)

3. Write a definition called 3DUP which will duplicate the top three numbers on the stack; for example,

   (1 2 3 -- 1 2 3 1 2 3)

Write definitions for the following infix equations, given the stack effects shown:

4. $a^2 + ab + c$ (c a b -- result)

5. $\frac{a - b}{a + b}$ (a b -- result)

6. Write a set of words to compute prison sentences for hardened criminals such that the judge can enter:

   ```
   CONVICTED-OF ARSON HOMICIDE TAX-EVASION ok
   WILL-SERVE 35 YEARS ok
   ```

   or any series of crimes beginning with the word CONVICTED-OF and ending with WILL-SERVE. Use these sentences:

HOMICIDE	20 years
ARSON	10 years
BOOKMAKING	2 years
TAX-EVASION	5 years

7. You're the inventory programmer at Maria's Egg Ranch. Define a word called EGG.CARTONS which expects on the stack the total number of eggs laid by the chickens today and prints out the number of cartons that can be filled with a dozen each, as well as the number of left-over eggs.