Chapter 2
Beginning Forth Tutorial

Intended Audience

The intent of this tutorial is to provide a series of experiments that will introduce you to the major concepts of Forth. It is only a starting point. Feel free to deviate from the sequences I provide. A free form investigation that is based on your curiosity is probably the best way to learn any language. Forth is especially well adapted to this type of learning. There are a number of excellent text books on Forth that you may want to use along with this tutorial. Look in the bibliography for a list of books. We recommend either *Starting Forth* or *Forth: A Text and Reference*.

In the tutorials, I will print the things you need to type in upper case, and indent them. You can enter them in upper or lower case. At the end of each line, press the RETURN (or ENTER) key; this causes Forth to interpret what you've entered.

Forth Syntax

Forth has one of the simplest syntaxes, or formats, of any computer language. The syntax can be stated as follows, "Forth code is a bunch of words with spaces between them." This is even simpler than English! Each *word* is equivalent to a function or subroutine in a language like 'C'. They are executed in the order they appear in the code. The following statement, for example, could appear in a Forth program:

```
WAKE.UP  EAT.BREAKFAST  WORK  EAT.DINNER  PLAY  SLEEP
```

Notice that WAKE.UP has a dot between the WAKE and UP. This is because it is *one* Forth word with a specific meaning. Forth word names can have any combination of letters, numbers, or punctuation. We will encounter words with names like:

```
." #S SWAP MOVE ! @ . *
```

They are all called *words*. The word **$%%-GL7OP** is a legal Forth name, although not a very good one. It is up to the programmer to name words in a sensible manner.

The Stack

The Forth language is based on the concept of a *stack*. Imagine a stack of blocks with numbers on them. You can add or remove numbers from the top of the stack. You can also rearrange the order of the numbers. Forth uses several stacks. The *Parameter Stack* is the one used for passing data between Forth words so we will concentrate our attention there. The Parameter Stack is also known as the Data Stack. The *Return Stack* is another Forth stack that is primarily for internal system use. In this tutorial, when we refer to the "stack," we will be referring to the Parameter Stack.

The stack is initially empty. To put some numbers on the stack, enter:

```
23 7 9182
```

You will notice that the OK message now has a '3' in front of it. This tells you that there are three numbers on the stack.

Let's now print the number on top of the stack using the Forth word '**.**', which is pronounced " dot ". This is a hard word to write about in a manual because it is a single period.

Enter: **.**

You should see the last number you entered, 9,182 , printed. Forth has a very handy word for showing you what's on the stack. It is **.S** , which is pronounced "dot S". The name was constructed from "dot" for print, and "S" for stack. If you enter:

```
.S
```

you will see your numbers in a list. The number at the far right is the one on top of the stack.

You will notice that the 9182 is not on the stack. The word ' . ' removes the number on top of the stack before printing it. In contrast, ' .S ' leaves the stack untouched.

We have a way of documenting the effect of words on the stack with a *stack diagram*. A stack diagram is contained in parentheses. In Forth, the parentheses indicate a comment. In the examples that follow, you do not need to type in the comments. When you are programming, of course, we encourage the use of comments and stack diagrams to make your code more readable. In this manual, we often indicate stack diagrams in **bold text** like the one that follows. Do not type these in. The stack diagram for a word like ' . ' would be:

```
. ( N -- , print number on top of stack )
```

The symbols to the left of -- describe the parameters that a word expects to process. In this example, N stands for any integer number. To the right of --, up to the comma, is a description of the stack parameters when the word is finished, in this case there are none because 'dot' "eats" the N that was passed in. (Note that the stack descriptions are not necessary, but they are a great help when learning other peoples programs.)

The text following the comma is an English description of the word. You will note that after the -- , N is gone. You may be concerned about the fact that there were other numbers on the stack, namely 23 and 7 . The stack diagram, however, only describes the portion of the stack that is affected by the word. For a more detailed description of the stack diagrams, there is a special section on them in this manual right before the main glossary section.

Between examples, you will probably want to clear the stack. If you enter **0SP**, pronounced "zero S P", then the stack will be cleared.

Since the stack is central to Forth, it is important to be able to alter the stack easily. Let's look at some more words that manipulate the stack. Enter:

```
0SP .S    \ That's a 'zero' 0, not an 'oh' O.
777 DUP .S
```

You will notice that there are two copies of 777 on the stack. The word **DUP** duplicates the top item on the stack. This is useful when you want to use the number on top of the stack and still have a copy. The stack diagram for DUP would be:

```
DUP ( n -- n n , DUPlicate top of stack )
```

Another useful word, is **SWAP**. Enter:

```
0SP
23 7 .S
SWAP .S
SWAP .S
```

The stack diagram for SWAP would be:

```
SWAP ( a b -- b a , swap top two items on stack )
```

Now enter:

```
OVER .S
OVER .S
```

The word **OVER** causes a copy of the second item on the stack to leapfrog over the first. It's stack diagram would be:

```
OVER ( a b -- a b a , copy second item on stack )
```

Here is another commonly used Forth word:

```
DROP  ( a -- , remove item from the stack )
```

Can you guess what we will see if we enter:

```
0SP 11 22 .S
DROP .S
```

Another handy word for manipulating the stack is **ROT**. Enter:

```
0SP
11 22 33 44 .S
ROT .S
```

The stack diagram for ROT is, therefore:

```
ROT  ( a b c -- b c a , ROTate third item to top )
```

You have now learned the more important stack manipulation words. You will see these in almost every Forth program. I should caution you that if you see too many stack manipulation words being used in your code then you may want to reexamine and perhaps reorganize your code. You will often find that you can avoid excessive stack manipulations by using *local or global VARIABLES* which will be discussed later.

If you want to grab any arbitrary item on the stack, use **PICK** . Try entering:

```
0SP
14 13 12 11 10
3 PICK .    ( prints 13 )
0 PICK .    ( prints 10 )
4 PICK .
```

PICK makes a copy of the Nth item on the stack. The numbering starts with zero, therefore:

```
0 PICK   is equivalent to DUP
1 PICK   is equivalent to OVER
```

```
PICK  ( ... v3 v2 v1 v0 N -- ... v3 v2 v1 v0 vN )
```

(Warning. The Forth-79 and FIG Forth standards differ from the Forth '83 standard in that their PICK numbering starts with one, not zero.)

I have included the stack diagrams for some other useful stack manipulation words. Try experimenting with them by putting numbers on the stack and calling them to get a feel for what they do. Again, the text in parentheses is just a comment and need not be entered.

```
DROP  ( n -- , remove top of stack )

?DUP  ( n -- n n | 0 , duplicate only if non-zero, '|' means OR )

-ROT  ( a b c -- c a b , rotate top to third position )

2SWAP ( a b c d -- c d a b , swap pairs )

2OVER ( a b c d -- a b c d a b , leapfrog pair )

2DUP  ( a b -- a b a b , duplicate pair )

2DROP ( a b -- , remove pair )

NIP   ( a b -- b , remove second item from stack )

TUCK  ( a b -- b a b , copy top item to third position )
```

Problems:

Start each problem by entering:

```
0SP 11 22 33
```

Then use the stack manipulation words you have learned to end up with the following numbers on the stack:

```
1)      11 33 22 22

2)      22 33

3)      22 33 11 11 22
```

```
4)      11 33 22 33 11

5)      33 11 22 11 22
```

Answers to the problems can be found at the end of all the tutorials.

Arithmetic

Great joy can be derived from simply moving numbers around on a stack. Eventually, however, you'll want to do something useful with them. This section describes how to perform arithmetic operations in Forth.

The Forth arithmetic operators work on the numbers currently on top of the stack. If you want to add the top two numbers together, use the Forth word + , pronounced "plus". Enter:

```
2 3 + .
2 3 + 10 + .
```

This style of expressing arithmetic operations is called *Reverse Polish Notation,* or *RPN*. It will already be familiar to those of you with HP calculators. In the following examples, I have put the algebraic equivalent representation in a comment.

Some other arithmetic operators are - * / . Enter:

```
30 5 - .  ( 25=30-5 )
30 5 / .  ( 6=30/5 )
30 5 * .  ( 150=30*5 )
30 5 + 7 / .  \  5=(30+5)/7
```

Some combinations of operations are very common and have been coded in assembly language for speed. For example, **2*** is short for 2 * . You should use these whenever possible to increase the speed of your program. These include:

```
1+  1-  2+  2-  2*  2/
```

Try entering:

```
10 1- .
7 2* 1+ .  ( 15=7*2+1 )
```

One thing that you should be aware of is that when you are doing division with integers using / , the remainder is lost. Enter:

```
15 5 / .
17 5 / .
```

This is true in all languages on all computers. Later we will examine **/MOD** and **MOD** which do give the remainder.

Defining a New Word

It's now time to write a *small program* in Forth. You can do this by defining a new word that is a combination of words we have already learned. Let's define and test a new word that takes the average of two numbers.

We will make use of two new words, **:** ("colon"), and **;** ("semicolon") . These words start and end a typical *Forth definition*. When you type this in, you may want to time how long it takes between hitting the RETURN key and the appearance of the OK prompt. This will be how long it takes Forth to compile and link this small program. Enter:

```
: AVERAGE  ( a b -- avg )  + 2/   ;
```

Congratulations. You have just written a Forth program. If you are used to 'C', you were probably wondering what to get from the kitchen while compiling. One danger of programming in Forth is that, once hooked, you may never take time to eat.

Let's look more closely at what just happened. The colon told Forth to add a new word to its list of words. This list is called the Forth dictionary. The name of the new word will be whatever name follows the colon. Any Forth words entered after this will be compiled into the new word. This continues until the semicolon is reached which finishes the definition.

Let's test this word by entering:

```
10 20 AVERAGE .  ( should print 15 )
```

Once a word has been defined, it can be used to define more words. Let's write a word that tests our word.. Enter:

```
: TEST ( --)  50 60 AVERAGE . ;
TEST
```

Try combining some of the words you have learned into new Forth definitions of your choice. If you promise not to be overwhelmed, you can get a list of the words that are available for programming by entering:

```
WORDS
```

You can stop this listing by hitting the space bar. To continue, hit the return key. To stop, type:

```
QUIT
```

Don't worry, only a small fraction of these will be used directly in your programs.

More Arithmetic

When you need to know the remainder of a divide operation. /MOD will return the remainder as well as the quotient. the word MOD will only return the remainder. Enter:

```
0SP
53 10 /MOD .S
0SP
7 5 MOD .S
```

Two other handy words are **MIN** and **MAX** . They accept two numbers and return the MINimum or MAXimum value respectively. Try entering the following:

```
56 34 MAX .
56 34 MIN .
-17 0 MIN .
```

Some other useful words are:

```
ABS    ( n -- abs(n) , absolute value of n )

NEGATE ( n -- -n , negate value, faster then -1 * )

ASHIFT ( n c -- n*(2**c) , arithmetic shift of n )

SHIFT  ( n c -- n' , logical shift of n )
```

ASHIFT can be used if you have to multiply quickly by a power of 2 . A negative count is like doing a divide. This is much faster than doing a regular multiply and should be used whenever possible. Try entering:

```
: 256*   8 ASHIFT ;
3 256* .
```

Arithmetic Overflow

If you are having problems with your calculation overflowing the 32-bit precision of the stack, then you can use */ . This produces an intermediate result that is 64 bits long. Try the following three methods of doing the same calculation. Only the one using */ will yield the correct answer, 5197799.

```
34867312 99154 * 665134 / .
34867312 665134 / 99154 * .
34867312 99154 665134 */ .
```

Convert Algebraic Expressions to Forth

How do we express complex algebraic expressions in Forth? For example: 20 + (3 * 4)

To convert this to Forth you must order the operations in the order of evaluation. In Forth, therefore, this would look like:

```
3 4 *  20 +
```

Evaluation proceeds from left to right in Forth so there is no ambiguity. Compare the following algebraic expressions and their Forth equivalents: (Do **not** enter these!)

(100+50)/2 ==> 100 50 + 2/
((2*7) + (13*5)) ==> 2 7 * 13 5 * +

If any of these expressions puzzle you, try entering them one word at a time, while viewing the stack with .S .

Problems:

Convert the following algebraic expressions to their equivalent Forth expressions. (Do **not** enter these because they are not Forth code!)

(12 * (20 - 17))
(1 - (4 * (-18) / 6))
(6 * 13) - (4 * 2 * 7)

Use the words you have learned to write these new words:

```
SQUARE   ( N -- N*N , calculate square )
DIFF.SQUARES ( A B -- A*A-B*B , difference of squares )
AVERAGE4  ( A B C D -- [A+B+C+D]/4 )
HMS>SECONDS ( HOURS MINUTES SECONDS -- TOTAL-SECONDS , convert )
```

Character Input and Output

The numbers on top of the stack can represent anything. The top number might be how many blue whales are left on Earth or your weight in kilograms. It can also be an ASCII character. Try entering the following:

```
72 EMIT  105 EMIT
```

You should see the word "Hi" appear before the OK. The 72 is an ASCII 'H' and 105 is an 'i'. EMIT takes the number on the stack and outputs it as a character. If you want to find the ASCII value for any character, you can use the word ASCII . Enter:

```
ASCII W .
ASCII %   DUP .   EMIT
ASCII A   DUP .
32 + EMIT
```

There is an ASCII chart in the back of this manual for a complete character list.

Notice that the word ASCII is a bit unusual because its input comes not from the stack, but from the following text. In a stack diagram, we represent that by putting the input in angle brackets, <input>. Here is the stack diagram for ASCII.

ASCII (<char> -- char , get ASCII value of a character)

Using EMIT to output character strings would be very tedious. Luckily there is a better way. Enter:

```
: TOFU ." Yummy bean curd!" ;
TOFU
```

The word **."** , pronounced "dot quote", will take everything up to the next quotation mark and print it to the screen. Make sure you leave a space after the first quotation mark. When you want to have text begin on a new line, you can issue a carriage return using the word **CR** . Enter:

```
: SPROUTS ." Miniature vegetables." ;
: MENU
    CR TOFU CR SPROUTS CR
;
```

```
MENU
```

You can emit a blank space with **SPACE** . A number of spaces can be output with SPACES . Enter:

```
CR TOFU SPROUTS
CR TOFU SPACE SPROUTS
CR 10 SPACES TOFU CR 20 SPACES SPROUTS
```

For character input, Forth uses the word **KEY** which corresponds to the word EMIT for output. KEY waits for the user to press a key then leaves its value on the stack. Try the following.

```
: TESTKEY ( -- )
    ." Hit a key: " KEY CR
    ." That = " . CR
;
TESTKEY
```

EMIT (char -- , output character)

KEY (-- char , input character)

SPACE (-- , output a space)

SPACES (n -- , output n spaces)

ASCII (<char> -- char , convert to ASCII)

CR (-- , start new line , carriage return)

." (-- , output " delimited text)

Answers to Problems

If your answer doesn't exactly match these but it works, don't fret. In Forth, there are usually many ways to the same thing.

Stack Manipulations

```
1) SWAP DUP
2) ROT DROP
3) ROT DUP 3 PICK
4) SWAP OVER 3 PICK
5) -ROT 2DUP
```

Arithmetic

(12 * (20 - 17)) ==> `20 17 - 12 *`

(1 - (4 * (-18) / 6)) ==> `1 4 -18 * 6 / -`

(6 * 13) - (4 * 2 * 7) ==> `6 13 * 4 2 * 7 * -`

```
: SQUARE ( N -- N*N )
    DUP *
;
: DIFF.SQUARES ( A B -- A*A-B*B )
    SWAP SQUARE
    SWAP SQUARE -
;
: AVERAGE4 ( A B C D -- [A+B+C+D]/4 )
    + + +   ( add'em up )
    -2 ashift ( divide by four the fast way, or 4 / )
;
: HMS>SECONDS  ( HOURS MINUTES SECONDS -- TOTAL-SECONDS )
    -ROT SWAP ( -- seconds minutes hours )
    60 * +    ( -- seconds total-minutes )
```

```
    60 * +    ( -- seconds )
;
```