Introduction to pbForth

- Brief History of Forth Systems
- Fundamental Principles of Forth
- Basic Syntax
- The Stack(s)
- The Dictionary
- Basic Math
- Managing the Data Stack
- Comparisons
- Logical Expressions
- Conditional Execution
- Repeated Execution
- Variables, Constants, Arrays
- RCX Specific Words
- Online Resources

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Developed by Charles Moore in the 60's
- First Forth system released in early 70's
- Early application controlled radio telescopes
- Multitasking and realtime support on single CPU
- FORTH Inc formed with Elizabeth Rather
- Forth Interest Group formed and FIG Forth released in late 70's
- Forth's extensibility leads to fragmentation
- Too many "flavours" leads to push for standard
- Forth-83 standard adopted – each vendor still has peculiarities
- ANSI Committee formed to standardize again
- Draft available online
- pbForth is developed in late 1998

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Forth programming is unlike any other language
- Moore's Principles
  1. Keep it Simple
  2. Do not Speculate
  3. Do it Yourself
- General Programming Practice
  1. Keep it Simple
  2. Anticipate Needs
  3. Work as a Group
- What makes Forth unique – according to Leo Brodie
  1. Inplicit Calls
  2. Implicit Data Passing
  3. Direct Access to Memory
- Forth is an interpreter *and* a compiler

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Every group of symbols separated by white space is either a word or number
- pbForth is case sensitive
- Input is totally free format
- Every line ends with a carriage return
- You are responsible for file management
- Work through Forth problems in front of your computer and RCX
- If you don't have an RCX – use hForth

```
This is what you type
This is what Forth types back ok
hi
hForth H8/300 for RCX RAM Model V1.0.9 by Ralph Hempel, 1998
All noncommercial uses are granted.
Please send comments, bug reports and suggestions to:
 rhempel@bmts.com
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Forth has implicit parameter stack and return stacks
- Other languages intermingle their data and return addresses on one stack
- Think of the stack as a pile of cards – Last On First Off (or LIFO)
- Words may take parameters off the stack or put them on
- Numbers leave their value on the stack

```
1 2 3 . . .
3 2 1 ok
```

- The word "`.`" (dot) prints the top of the stack as a signed value
- The word `DUP` (dupe) makes a copy of the top of the stack

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

- The word `DROP` (drop) gets rid of the top of the stack

```
1 2 3 DROP . . .
2 1 xxyy . ? stack underflow
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Forth has a “dictionary” of words it understands
- You can extend this dictionary from simple words to the compiler itself
- Words you (and Forth) know:
- Any number as well as `. DUP DROP`
- What is punctuation in other languages are words in Forth
- You extend the dictionary using `:` (colon) and `;` (semicolon)

```
: PRINT_TWO . . ;
1 2 PRINT_TWO
2 1 ok
```

- That’s all there is to making new words for your dictionary

```
: name put_your_definition_here :
```

- You tell Forth you are starting a new definition with “`:`” then you give your new word a `name`, then you define it in terms of words Forth already knows, and finally you tell Forth that you are “`;`” done.

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Forth uses “Reverse Polish Notation” or “postfix” operators – parameters, then operator

```
1 2 +
3 ok
```

- Forth has 16 bit fixed point math (signed and unsigned) – and some 32 bit math
- You can add floating point – but why bother?
- The basic 16 bit operators are:

`+`	`( n1 n2 -- sum )`	(plus)
`-`	`( n1 n2 -- diff )`	(minus)
`*`	`( n1 n2 -- prod )`	(star)
`/`	`( n1 n2 -- quot )`	(slash)

- There are some extras that come in handy

`MOD`	`( n1 n2 -- rem )`	(mod)
`/MOD`	`( n1 n2 -- rem quot )`	(slash-mod)
`ABS`	`( n1 -- absval )`	(abs)

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Values can be printed in signed or unsigned form

`.`	`( n1 -- )`	(dot)
`U.`	`( n1-- )`	(u-dot)

- You can change the displayed base of numbers you enter and print using

`HEX`	`( -- )`	(hex)
`DECIMAL`	`( -- )`	(decimal)

- You enter double precision (32 bit) numbers by following them with a decimal point
- Each single precision number takes up 1 cell on the stack
- Each double precision number takes up 2 cells on the stack

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- You must make sure your parameters are in the right order

```
foo ( n1 n2 n3 – r1 r2 r3 )
```

- `n3` is the top of the stack before `foo` is called, `r3` is the top of the stack after it returns
- Try and keep the variables you need on the stack, and use variables sparingly
- Here are some stack management words

`DUP`	`( n1 –- n1 n1 )`	(dupe)
`DROP`	`( n1 n2 -- n1 )`	(drop)
`SWAP`	`( n1 n2 –- n2 n1 )`	(swap)
`OVER`	`( n1 n2 –- n1 n2 n1 )`	(over)
`ROT`	`( n1 n2 n3–- n2 n3 n1 )`	(rote)
`2DUP`	`( n1 n2 –- n1 n2 n1 n2 )`	(two- dupe)
`2DROP`	`( n1 n2 –- )`	(two-drop)
`2SWAP`	`( n1 n2 n3 n4 –- n3 n4 n1 n2 )`	(two-swap)
`2OVER`	`( n1 n2 n3 n4 –- n1 n2 n3 n4 n1 n2 )`	(two-over)
`DEPTH`	`( … –- … n1 )`	(depth)

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- FALSE is defines as zero, TRUE is non-zero
- Conditional expressions use postfix notation too

```
1 2 > U.
0 ok
2 1 > U.
65535 ok
```

- To make things clearer, imagine putting the operator *between* the parameters
- When Forth returns a TRUE value, all of the bits are set
- Here are the signed comparison words

`=`	`( n1 n2 –- f )`	(equal)
`<`	`( n1 n2 –- f )`	(less-than)
`>`	`( n1 n2 –- f )`	(greater-than)
`0=`	`( n1 –- f )`	(zero-equal)
`0<`	`( n1 –- f )`	(zero-less)

- There is no `0>` so make your own or use `0< 0=`

- Here are the unsigned comparison words

`U<`	`( n1 n2 -- f )`	(u-less-than)
`U>`	`( n1 n2 -- f )`	(u-greater-than)
`0=`	`( n1 -- f )`	(zero-equal)
`0<`	`( n1 -- f )`	(zero-less)

- Finally, here are some non-logical comparisons

`MIN`	`( n1 n2 -- minval )`	(min)
`MAX`	`( n1 n2 -- maxval )`	(max)

- And a stack manipulation that uses a conditional

`?DUP`	`( n1 -- n1 | n1 n1 )`	(question-dupe)

- `?DUP` only copies the top item if it's non-zero

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Here are the logical operators you can use in Forth

`AND`	`( u1 u2 –- andval )`	(and)
`OR`	`( u1 u2 –- orval )`	(or)
`XOR`	`( u1 u2 –- xorval )`	(xor)
`INVERT`	`( u1 u2 –- invert )`	(invert)

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Just like other languages, Forth allows you to do things based on conditions
- You can only use conditional execution *inside* a definition

```
: MY_MINa ( n1 n2 –- minval )
  > IF SWAP DROP ELSE DROP THEN ;
```

- Or you could save a step and write

```
: MY_MINb ( n1 n2 –- minval )
  > IF SWAP THEN DROP ;
```

- The basic form is

```
: … flag IF do_if_true ELSE do_if_false THEN … ;
```

- Remember, the `IF` word uses up the value on the top of the stack

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Just like other languages, Forth allows you to repeat things based on
- You can only use repeated execution (loops) *inside* a definition

```
: .S ( … –- … )
  DEPTH 0 DO I PICK . LOOP ;
```

- The general form of a counted do loop is

```
: … limit index DO do_stuff_here LOOP … ;
```

- The loop continues to run as long as `limit` is less than `index`
- To get the index of the current loop you use `I`
- Here's a new stack manipulation word

  `PICK ( … n1 –- n1 | n1 ni )` (pick)

- The `PICK` word grabs the indexed item off the stack.
- `0 PICK` is the same as `DUP`
- `1 PICK` is the same as `OVER`
- There is a bug in the previous code … can you find it?

- What happens when `limit` is equal to `index` ?
- Here's how to fix the problem

```
: .S ( … -- … )
  DEPTH ?DUP IF 0 DO I PICK . LOOP THEN ;
```

- For incrementing a loop index by a value other than 1, use `+LOOP`

```
: BY5 ( n –- )
  ?DUP IF 0 DO I . 5 +LOOP THEN ;
24 BY5 0 5 10 15 20 ok
```

- To get out of a loop early, just `LEAVE`

```
: BY5to10 ( n –- )
  ?DUP IF 0 DO I DUP 10 > LEAVE . 5 +LOOP THEN ;
24 BY5to10 0 5 10 ok
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- You can make a traditional do loop like this

```
: … BEGIN do_loop_stuff flag UNTIL … ;
```

- The `BEGIN UNTIL` loop executes at least once, and runs as long as the flag is FALSE. In other words, it runs *until* the flag is TRUE

- You can also make a loop only execute under certain conditions - a while loop

```
: … BEGIN do_check flag WHILE do_loop_stuff REPEAT … ;
```

- The optional code after `BEGIN` is always executed, the code between `WHILE` and `UNTIL` executes only if the flag is TRUE. The `REPEAT` takes us back to the `BEGIN`

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Forth *does* support variables, constants, and arrays, just like other languages
- Here's how you make a variable that can store a single-celled value

```
VARIABLE FOO
```

- When you execute the `FOO` word later, the *address* of the cell in memory is returned
- Here are the words that let us read and write arbitrary addresses in memory:

`@`	`( … addr –- n1 )`	(fetch)
`!`	`( … n1 addr –- )`	(store)

- This is pretty easy stuff – but watch out! Storing to invalid addresses will probably crash your system!
- Here's how you fetch and store values out of `FOO`

```
46 FOO !
ok
FOO @ .
46 ok
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Here's how you make a coonstant that is a single-celled value

```
2362 CONSTANT BAR
```

- When you execute the BAR word later, the *value* of the cell in memory is returned
- There is no (easy) way to change the value of a constant
- Here's how you use constants

```
BAR .
2362 ok
```

- What `VARIABLE` and `CONSTANT` do is add words to the dictionary and allocate space for the values that they represent.
- `VARIABLE` and `CONSTANT` are defining words since they alter the dictionary.
- The other defining words we have seen so far are `:` and `;`

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- A new defining word is introduced at this point `CREATE`.
- This word makes a new name in the dictionary. When you execute this word, the address returned is where `VARIABLE` would store its value.
- `CREATE` does not allocate space for you – you have complete control.

```
CREATE FOOARRAY 32 CELLS ALLOT
ok
```

- We have just created a 32 cell array, so here are some words we can define to get and set values in the array…I've omitted the ok from pbForth

```
: ARRAY@ ( addr n2 –- an ) CELLS + @ ;
: ARRAY! ( n1 addr n2 -- ) CELLS + ! ;
34 FOOARRAY 12 ARRAY!
12 ARRAY@ U.
34 ok
```

- There are two words for reading and writing at the byte level as well

`C@`	`( … addr –- c1 )`	(c-fetch)
`C!`	`( … c1 addr –- )`	(c-store)

- pbForth is distinguished from other Forths by having a few words in its dictionary which are only useful on RCX systems.
- The words use the fully tested software that is in the ROM of the RCX.
- The calling conventions for the words closely mimic those of the ROM.
- The following groups of words will be discussed…
    1. RCX and Power Control
    2. Display Control
    3. Motor Control
    4. Button Control
    5. Sound Control
    6. Sensor Control
    7. Timer Control

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Before using the other words to control the RCX, it must be initialized.
- The ROM routines handle sampling the A/D, motor driving, button sensing etc
- Interrupts and data areas must be ininitialized
- Use the following words to set up the RCX system

```
RCX_INIT        ( -- )
RCX_SHUTDOWN    ( -- )
```

- The power can be turned off and on using the following words

```
POWER_INIT      ( -- )
RCX_POWER       ( -- addr )
POWER_GET       ( addr code -- )
POWER_OFF       ( -- )
```

- The `RCX_POWER` word returns the address of the variable that hold the result of `POWER_GET`
- The `code` parameter (hex) values for `POWER_GET` can be:

```
4000    power key status – 0 if pressed
4001    battery voltage – multiply by 43998 then divide by 1560
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The display control words can be used at any time.

```
LCD_SHOW        ( segment -- )
LCD_HIDE        ( segment -- )
LCD_NUMBER      ( comma number int -- )
LCD_CLEAR       ( -- )
LCD_REFRESH     ( -- )
```

- The `DISPLAY_REFRESH` word must be called to actually change the display
- Here are the legal (hex) values for the `segment` parameter

```
3006 standing figure
3007 walking figure
3008 sensor 0 view selected
3009 sensor 0 active
300a sensor 1 view selected
300b sensor 1 active
300c sensor 2 view selected
300d sensor 2 active
300e motor 0 view selected
300f motor 0 backward arrow
3010 motor 0 forward arrow
3011 motor 1 view selected
3012 motor 1 backward arrow
3013 motor 1 forward arrow
3014 motor 2 view selected
3015 motor 2 backward arrow
3016 motor 2 forward arrow
3018 datalog indicator, multiple calls add
     4 quarters clockwise
3019 download in progress, multiple calls
     adds up to 5 dots to right
301a upload in progress, multiple calls
     removes up to 5 dots from left
301b battery low
301c short range indicator
301d long range indicator
3020 all segments
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The point codes for the `LCD_NUMBER` word are a bit confusing
- The `comma` parameter can take the following (hex) values

```
3002      no decimal point
3003      000.0 format
3004      00.00 format
3005      0.000 format
```

- The `int` parameter can take the following (hex) values

```
3001      Set main number on display to signed value, with no leading zeros
          If value > 9999, displayed value is 9999
          If value < -9999, displayed value is -9999
3017      Set lcd program number
          Set program number on display to value, use pointcode=0
          If value < 0, no value is displayed
          If value > 0, no value is displayed
          Pointcode is ignored, no real need to set to zero
301f      Set lcd main number unsigned
          Set main number on display to unsigned value, with leading zeros
          Value is unsigned, so it is never less than 0
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Motor control is very simple, there is only one word to control them

```
MOTOR_SET        ( power dir idx-- )
```

- The `power` parameter values can range from 0 (off) to 7 (full power).
- The `dir` parameter values can be:

```
1        forward
2        reverse
3        brake
4        float
```

- The `idx` parameter is the motor number

```
0        MOTOR A
1        MOTOR B
2        MOTOR_C
```

- Typical usage is:

```
7 1 0 MOTOR_SET ( turns motor on forward )
7 3 0 MOTOR_SET ( brakes hard )
7 2 0 MOTOR_SET ( turns motor on reverse )
7 4 0 MOTOR_SET ( motor coasts to a stop )
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Before using the button system, it must be initialized
- Here are the button control words

```
BUTTON_INIT     ( -- )
RCX_BUTTON      ( -- addr )
BUTTON_GET      ( addr -- )
```

- The RCX_BUTTON word returns the address of the variable that hold the result of BUTTON_GET
- Typical use of the button system is as follows:

```
RCX_BUTTON DUP BUTTON_GET @ U.
```

- The (hex) values left in the RCX_BUTTON variable are as follows:

```
1   RUN button pressed
2   PRGM button pressed
3   VIEW button pressed
```

- Remember to debounce the button readings to be sure that they stable – this goes for the RCX_POWER function too.

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The sound system for pbForth allows the standard tones to be played
- The tones can queued so you don't have to wait until the current one is done
- The sound control words are:

```
RCX_SOUND        ( -- addr )
SOUND_PLAY       ( sound code -- )
SOUND_GET        ( addr -- )
```

- The `RCX_SOUND` word returns the address of the variable that hold the result of `SOUND_GET`
- The `sound` parameter can have the following values:

```
0    Blip
1    Beep Beep
2    Upward Tones
3    Downward Tones
4    Low Buzz
5    Fast Upward Tones
```

- The `code` parameter is one of the following (hex) values:

```
4003 Sound is not queued
4004 Sound is queued
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The sensor system must be initialized before use – and each sesnor must be initialized too
- Here are the words for the basic sensor control

```
SENSOR_INIT      ( -- )
SENSOR_PASSIVE   ( idx -- )
SENSOR_ACTIVE    ( idx -- )
SENSOR_TYPE      ( type idx -- )
SENSOR_MODE      ( mode idx -- )
```

- The `idx` parameter is a bit confusing because it is 0 based

```
0    Sensor 1
1    Sensor 2
2    Sensor 3
```

- You need to tell the RCX if a sensor is passive or active. Touch and heat sensors are passive. Light and angle sesnors are active

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The `type` parameter for `SENSOR_TYPE` can take on the following (hex) values

```
1    Touch
2    Temperature
3    Light
4    Angle (Rotation)
```

- The `mode` parameter for `SENSOR_MODE` tells the RCX how to process the data from the sensor and has the following (hex) values. Not all modes will not make sense with all types.

```
00   Raw
20   Boolean
40   Edge
60   Pulse
80   Percent
A0   Degrees Celsius
C0   Degrees Fahrenheit
E0   Angle
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- Once the sensor has been initialized it must be read on a regular basis – depending on what you are measuring

```
SENSOR_READ      ( idx -- )
```

- The current value of the sesnor can be read in a number of forms, not all will make sense

```
SENSOR_RAW       ( idx –- raw )
SENSOR_VALUE     ( idx –- val )
SENSOR_BOOL      ( idx -- bool )
```

- Sometimes, you will want to wipe out all of a sensors values, but leave the mode and type as they were.

```
SENSOR_CLEAR     ( idx -- )
```

©1999 Ralph Hempel – Visit us on the Web at <http://www.hempeldesigngroup.com>

- The RCX has two kinds of timers that pbForth can access
- The 4 low resolution timers are incremented every 100 msec and count *up* continuously and roll over at 65535.
- Here are the low resolution timer words, the `idx` parameter specifies the timer number. Be very careful to keep within the proper range!

```
TIMER_SET        ( value idx -- )
TIMER_GET        ( idx –- value )
```

- The 10 high resolution timers are decremented every 10 msec and count *down* to zero and then stop
- Here are the high resolution timer words, the `idx` parameter specifies the timer number. Be very careful to keep within the proper range!

```
timer_SET        ( value idx -- )
timer_GET        ( idx –- value )
```

- The RCX has a lots of on-line resources supported by knowledgable and friendly users
- Here are a few of them:

`www.lugnet.com`	General starting point in your quest for knowledge about LEGO®, Mindtorms®, or the RCX
`www.forth.org`	Good source of on-line documentation for Forth in general
`www.hempeldesigngroup.com`	The official home of pbForth with links to other RCX sites