ENiX 1

INSTALLATION
```
# gcc ENiX-1.c -o enix
```

EXECUTION
```
# ./enix <COMMAND> <data>
```

“Entities should not be multiplied beyond necessity.”
- William of Ockham

COMMAND LIST
These commands must be uppercase and preceeding the data that they use.
ANALYZER, AUTO, CHOCKY, DUALSCAN, EQUALIZER, EQUATION, FULLAUTO, HELP, IL-CO1, IL-CO2, ILEARN, IL-SC, KINDRED, ODDMAN, PRODIGY, ROTAX.

DATA RULES
Data can be numeric in the interval 0-255 inc. If the data is unknown for a particular value then it can be represented as '?'. Negative numbers will be treated by ENiX as unknown values, '?', and numbers larger than 255 will cause the ENiX application to stop with an error message. Up to 256 numbers or '?'s can be entered.

LOGIC ANALYSER
First of all the numeric data sequence is taken and unknowns, '?'s are represented by negative values to be skipped over. If the data including unknowns is N entries long, then we break up the series into log (2,MAX(data)) independent sequences of 1's, 0's and -1's for unknowns. E.g. “./enix <COMMAND> 0 2 4 6 8 10 12 14 ? ? 20”, becomes:
```
VALUE:     0 2 4 6 8  10  12  14  ?  ?  20
INDEX:     0 1 2 3 4   5   6   7  8  9  10
BIT 4    [0 0 0 0 0   0   0   0 -1 -1   1 ]
BIT 3    [0 0 0 0 1   1   1   1 -1 -1   0 ]
BIT 2    [0 0 1 1 0   0   1   1 -1 -1   1 ]
BIT 1    [0 1 0 1 0   1   0   1 -1 -1   0 ]
BIT 0    [0 0 0 0 0   0   0   0 -1 -1   0 ]
```
Now, since the object of ENiX is to find a relationship between the INDEX and the VALUE, one of the things we can do is look at the association between binary expansion of the index and compare this to the binary expansion of the value. Relate: “Find the relationship between INDEX and VALUE” to “Find the relationship between binary

matrices of INDEX and VALUE” is equivalent to:
Relate:

$$\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0 \\ 0&0&1&1&0&0&1&1&0&0&1 \\ 0&1&0&1&0&1&0&1&0&1&0 \end{bmatrix} \text{ to } \begin{bmatrix} 0&0&0&0&0&0&0&-1&-1&1 \\ 0&0&0&0&1&1&1&-1&-1&0 \\ 0&0&1&1&0&0&1&-1&-1&1 \\ 0&1&0&1&0&1&1&-1&-1&0 \\ 0&0&0&0&0&0&0&-1&-1&0 \end{bmatrix}$$

Already we can see a pattern emerging between the two matrices. So what we are trying to do is to find a way of computing the individual elements in the vertical columns on the rhs from all of the entries in the corresponding column from the lhs. But how do we do this exclusively using the GATE, AND, OR, NOR, XOR, NOT, NAND, RAND and ONE logic operations ?

1) Setup: create a logic operation, connect the first input to the first bit in the index matrix (lhs matrix, bottom row) and connect the second input (if applicable)* to the second bit in the index matrix (lhs matrix, bottom row) and connect the output to the output connects to the first output bit (rhs matrix, bottom row). Execute 2) for the output bit set to each row in the rhs matrix.
2) 1st Input Bit: move the 1st input bit to accept the information from the row in the lhs just above where it was previously accepting input from. If 1st input cannot be moved to the next row up because the matrix row doesn't exist, then it is moved to accept input from the bottom row and step 3) is executed. Test the model to see if it generates the corresponding row in the rhs matrix ignoring unknowns. If it does, then stop program otherwise return to 2).
3) 2nd Input Bit (if applicable)*: move the 2nd input bit to accept the information from the row in the lhs just above where it was previously accepting input from. If 2nd input cannot be moved to the next row up because the matrix row doesn't exist, then it is moved to accept input from the bottom row and step 4) is executed else return to step 2).
4) Logic Operation: change the logic operation to the next logic operation in the list. If all logic operations have been exhausted, then it is changed to the first logic operation in the list and step 5) is executed else return to step 2).
5) Modify Next Logic Operation: make exactly one modification to the next logic operation using methods 2)-5) in order remembering that the stages 2) and 3) would be adjusted by the addition of the new row(s) of buffered previous output (necessary to ensure that computations can be many layers deep). Once this modification has been made return to the first logic operation modifying it to the conditions of step 2) onwards. If the last logic operation cannot be modified, then execute 6).
6) Add Logic Operation: add a new row ontop of the lhs INDEX matrix to buffer output from previous modified logic operation during execution, so that the newly created logic operation can use it.

Create a new logic operation following the rules laid out in 1) then return to stage 2) modifying the first logic operation.

*Some logic operations only have one bit input and therefore it would be a waste of time executing modifications to a non-existant second input.

If ENiX exits successfully from this algorithm, it would have found a successful logical operation to map the INDEX matrix to the VALUE matrix, and therefore it has a guarenteed method for producing a value in the sequence data at position index. At the indices were the data is known the results are guarenteed to be accurate, which means that there is a significant probability that the unknown data could be reproduced by the newly created logic operations. Unfortunately this can not be a guarenteed thing, it does depend on the sequence provided, whether it is modelled well with logic operations, the ratio of knowns to unknowns, and whether the predictions are extrapolated or interpolated from the boundaries of defined values at lowest and highest positions in the data sequence.

EQUATION ANALYSER (METHOD OF FINITE DIFFERENCES)
Because ENiX logic-based pattern analyser is stronger at predicting values from within the limits bounded by the highest and lowest position of knowns, and is very weak and predicting extrapolated results (mainly due, it must be said to an inherent inability to create new rows in the rhs, VALUE matrix) it makes sense to look at an alternative that is stronger with extrapolated predictions than interpolated predictions. For this for the time being we will resort to using the finite difference method for predicting the polynomial based on provided data and skipping over unknowns, making it unreliable for interpolated predictions but fairly reliable for extrapolated indices. Enough said.

CHOCKY
This is a very primative attempt at association ENiX logic analyser with language. It works as follows: Given a sequence of circumstances we should be able to predict the scenerio via ENiX using the following algorithm:

1) Set problem list, P, to be all unknowns.
2) Count the number of unique symbols in circumstances and scenerios, designated A and B resp.
3) Designate powers of two as a primary key to represent each unique symbol in circumstances and scenerios, saving in lists L1 and L2,
4) Set C=0, sum the corresponding primary keys representing the occurance of a symbol in circumstances at max once per symbol,
5) At position C, set the value, S, to the sum of the primary keys of the corresponding scenerio symbols at max once,

6) Put S in position C in the problem list, P,
7) IF all scenerios in P, GOTO 8) ELSE load next scenerio, GOTO 4),
8) Pass the problem, P, to ENiX pattern analyser,
9) To see what the outcome of circumstances are: sum the primary keys of occuring symbols in the circumstances at max once, Store as E,
10) Execute ENiX at E,
11) Translate ENiX output into binary and display corresponding scenerio symbols.

Note: order and identical reoccurring symbols are ignored.

ODDMAN OUT
Principles are quite simple. Uses the logic pattern analyser explained above to find the simplist possible sequence out of several sequences derived from the original sequence by masking out each successive term in the data sequence. ENiX's logic pattern analyser has a small variable which records the total number of times that the logic operator network has had to modify it's own shape to locate the solution. Using the underlying principle of ENiX, utilising Ockham's Razor, we conclude that the sequence with the least required number of iterations to formulate a method for explaining the VALUE provided at location INDEX reliably for all knowns, is the best sequence. This sequence is the sequence with the oddman out masked out by an unknown. The ENiX ODDMAN function does just this and uses the information to find the oddman out and correct the sequence to what it thinks (“thinks” being loosely defined here as the algorithm ENiX follows to reach the solution) would be best based on the explanation's simplicity.

E.g. taking the sequence: 0 1 2 3 4 0 6 7, ENiX breaks this down into:

{?,1,2,3,4,0,6,7}, {0,?,2,3,4,0,6,7}, {0,1,?,3,4,0,6,7},
{0,1,2,?,4,0,6,7}, {0,1,2,3,?,0,6,7}, {0,1,2,3,4,?,6,7},
{0,1,2,3,4,0,?,7}, {0,1,2,3,4,0,6,?},

Because the 2nd zero really does take quite a lot of computational explaining, all sequences with the 2nd zero are ignored and the sequence with the ? masking the 2nd zero, is considered the oddman out. Therefore the system concludes that the 2nd zero is the oddman out and runs the corresponding sequence through the logic pattern analyser to provide the solution: 0 1 2 3 4 5 6 7.

Equalisation
While building the ODDMAN feature of ENiX, I realised that when the method was of optimal simplicity, the complexity of each of the oddman out sequences was the same independently of which number in the sequence was masked out with an unknown. This property of “equalization” seems to reflect a rigourous depth of understanding of the property. So using oddman out and keeping in mind “equalisation”,

ENiX ANALYZER module takes a sequence of numbers, locates the most incorrect number, using oddman out, corrects it and repeats this process until the sequence is “equalised”. This process usually oversimplifies the sequence greatly, however it shows limited abilities to correct a sequence and to associate sequences through this simplification algorithm.

ERROR CORRECTION AND PATTERN ASSOCIATION
“Equalisation” and “problem complexity” turn out to be very useful tools in correcting errors in input (EQUALIZER, ANALYZER and PRODIGY) and generating similar sequences to a given, possibly incorrect, sequence (KINDRED). The error correcting algorithms strive in various ways to “equalize” the sequence and reduce it to what it perceives as it's simplist form and therefore the best. The generation of similar sequences comes from listing all sequences generated by the “equalisation” of various combinations of maskings by unknowns of the original data sequence. This results in a print out of a list of the most likely similar sequences that ENiX thinks that the user could have ment instead of the sequence which was provided.

ORTHOGNAL PATTERN RECOGNITION
DUALSCAN is an implimentation of this concept. Basically the idea is to accept a [possibly incorrect] sequence of numbers from the user and then using ENiX logic analyser to estimate the error. Then, computing the dual of the problem by computing a matrix t-transform on the lhs and rhs matrices and then parsing to ENiX logic analyser to estimate the error in the orthognal sequence. The final result is expressed as a percentage of agreement between the two orthognal data sets. The idea came about through the realisation that the problems orthognal problem maybe easier to solve.

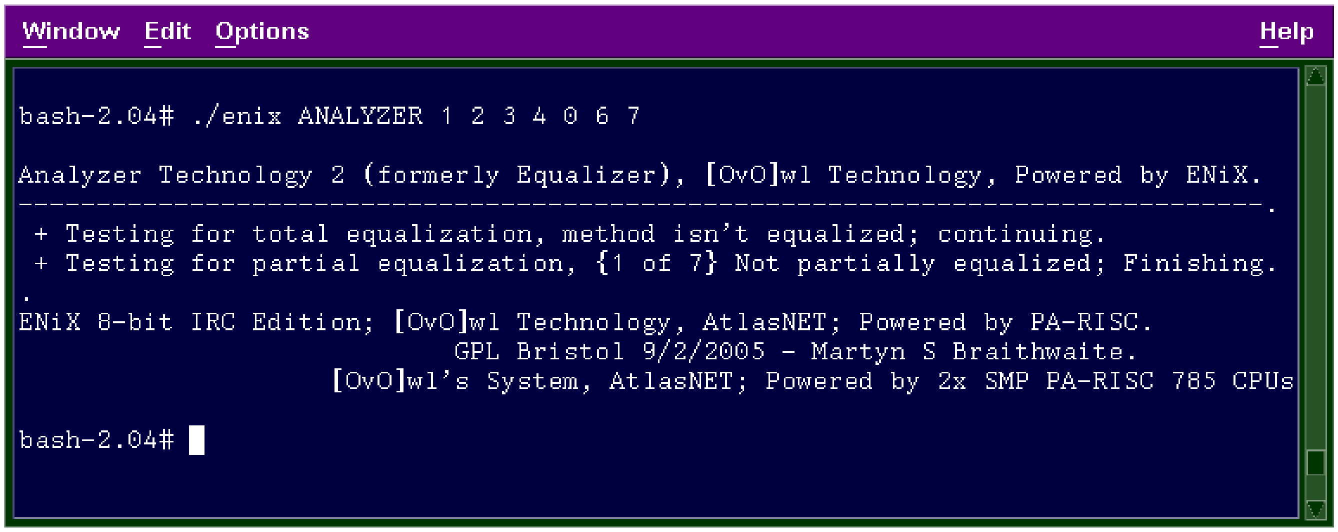
AUTOMATIC METHOD SELECTION
FULLAUTO feature streams the sequence through the equation solver before the logic core, due to it being a lot faster and being a direct mathematical computation before wasting any unnecessary clock cycles on finding the logical solution exhaustively. Because the equation solver knows when it's unable to compute the solution, it will pass the problem on to the logic core if it is given something it cannot handle, while aborting the computation and providing the user with the answer if it succeeds without further time wasting.

LOGIC NETWORK BOOSTERS
AUTO, IL-CO1, IL-CO2, IL-SC were experiments. The idea was that ENiX is far faster at computing the solution to a canonical sequence that started with 0 as the first term and terms varied by unit multiples, rather than if the sequence started with a non-zero term and used non unit multiples. These commands are more an area of interest rather than something that is practical to the function of ENiX IL-SC scales

the data sequence into units, IL-CO1 & IL-CO2 code these so that the first term is zero. AUTO endevours to reach a compremise automatically. These features don't work well or reliably but are left in for future reference.

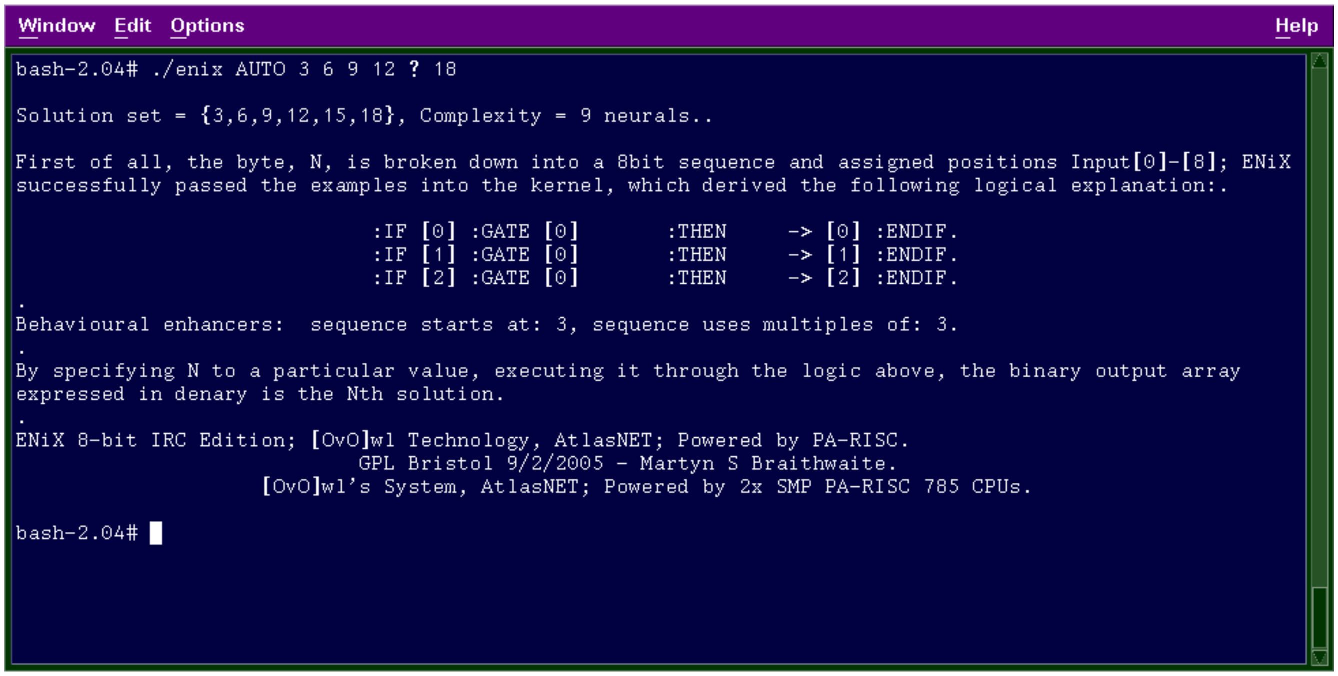
EXAMPLES

```
bash-2.04# ./enix ANALYZER 0 1 2 3 4 0 6 7
```



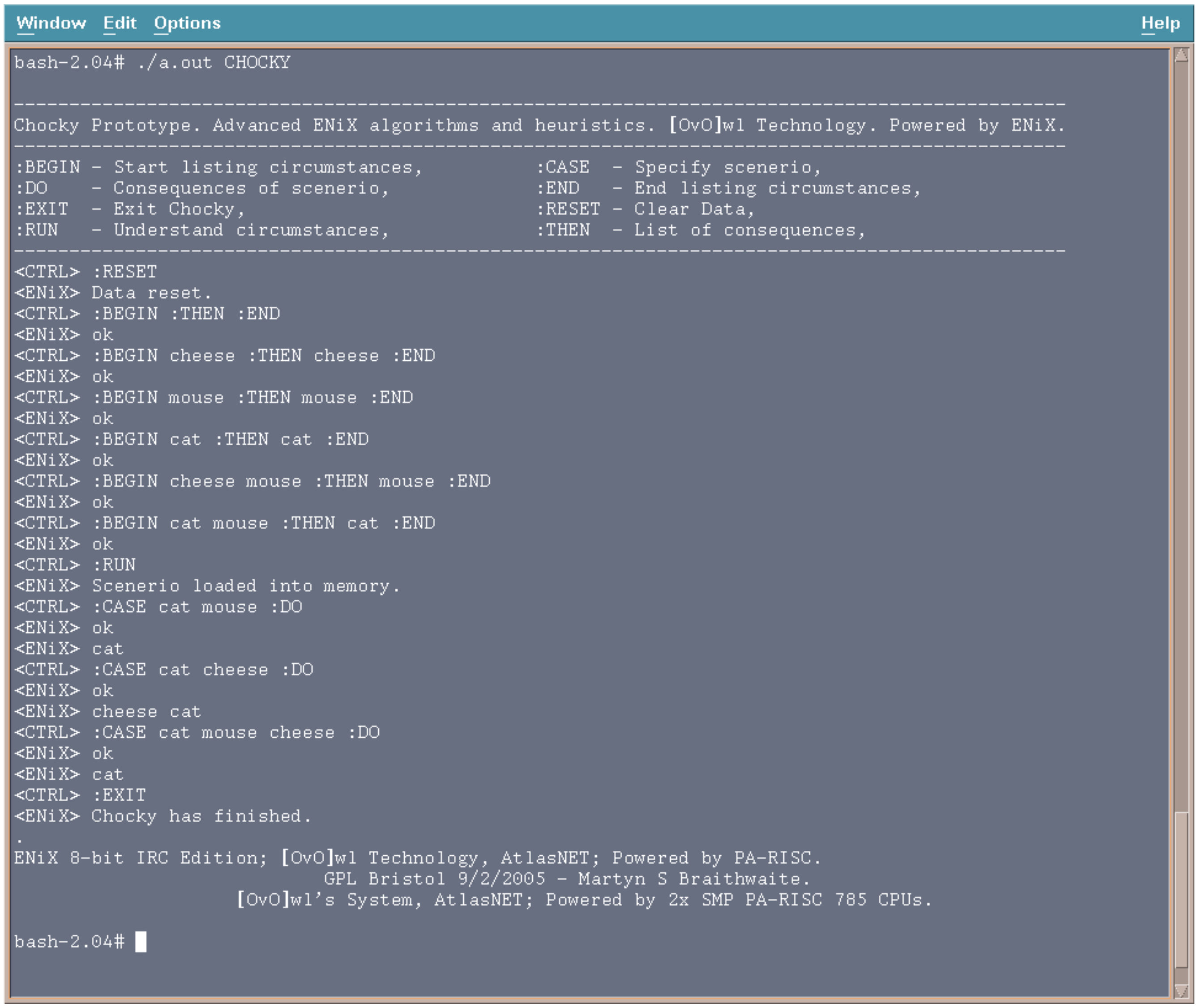
```
bash-2.04# ./enix ANALYZER 1 2 3 4 0 6 7

Analyzer Technology 2 (formerly Equalizer), [OvO]wl Technology, Powered by ENiX.
--------------------------------------------------------------------------------.
 + Testing for total equalization, method isn't equalized; continuing.
 + Testing for partial equalization, {1 of 7} Not partially equalized; Finishing.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs

bash-2.04#
```

```
bash-2.04# ./enix AUTO 3 6 9 12 ? 18
```



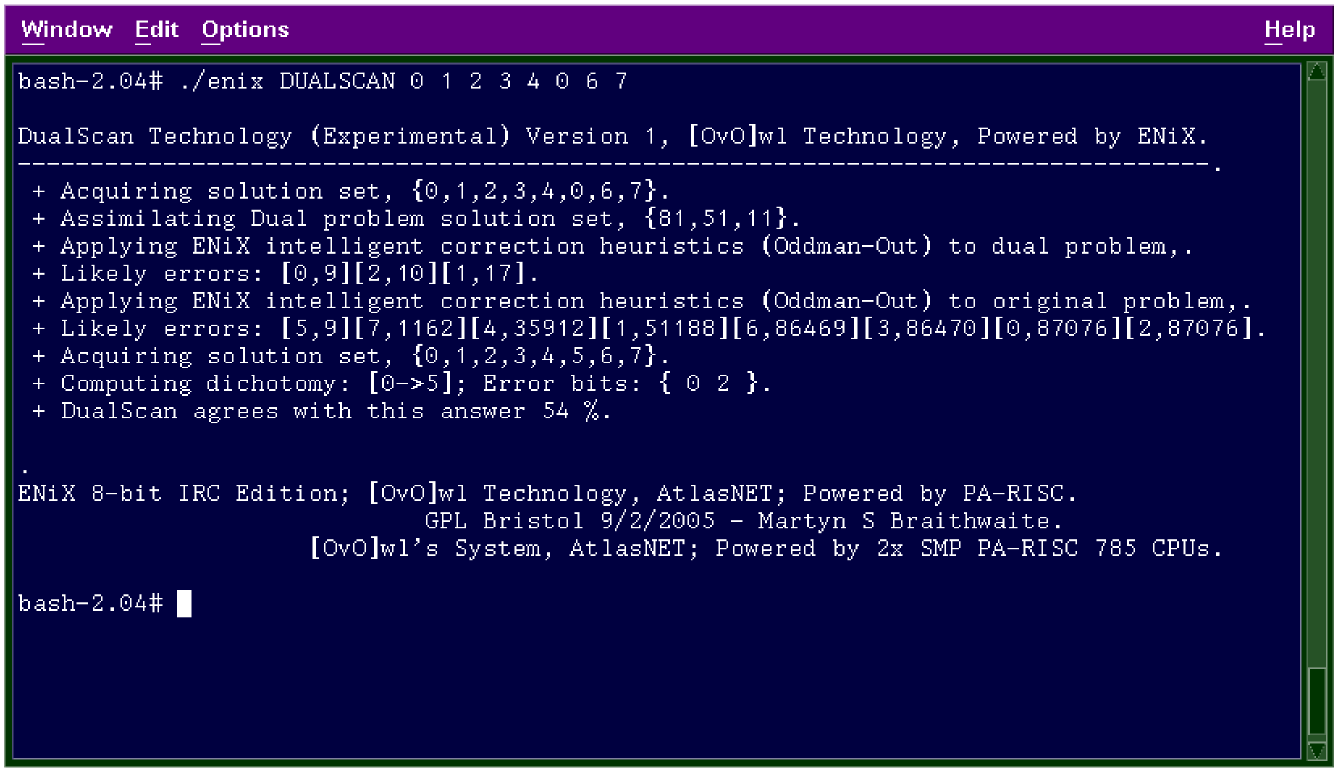
```
bash-2.04# ./enix AUTO 3 6 9 12 ? 18

Solution set = {3,6,9,12,15,18}, Complexity = 9 neurals..

First of all, the byte, N, is broken down into a 8bit sequence and assigned positions Input[0]-[8]; ENiX
successfully passed the examples into the kernel, which derived the following logical explanation:.

                             :IF [0] :GATE [0]      :THEN     -> [0] :ENDIF.
                             :IF [1] :GATE [0]      :THEN     -> [1] :ENDIF.
                             :IF [2] :GATE [0]      :THEN     -> [2] :ENDIF.
.
Behavioural enhancers:  sequence starts at: 3, sequence uses multiples of: 3.
.
By specifying N to a particular value, executing it through the logic above, the binary output array
expressed in denary is the Nth solution.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix CHOCKY
```



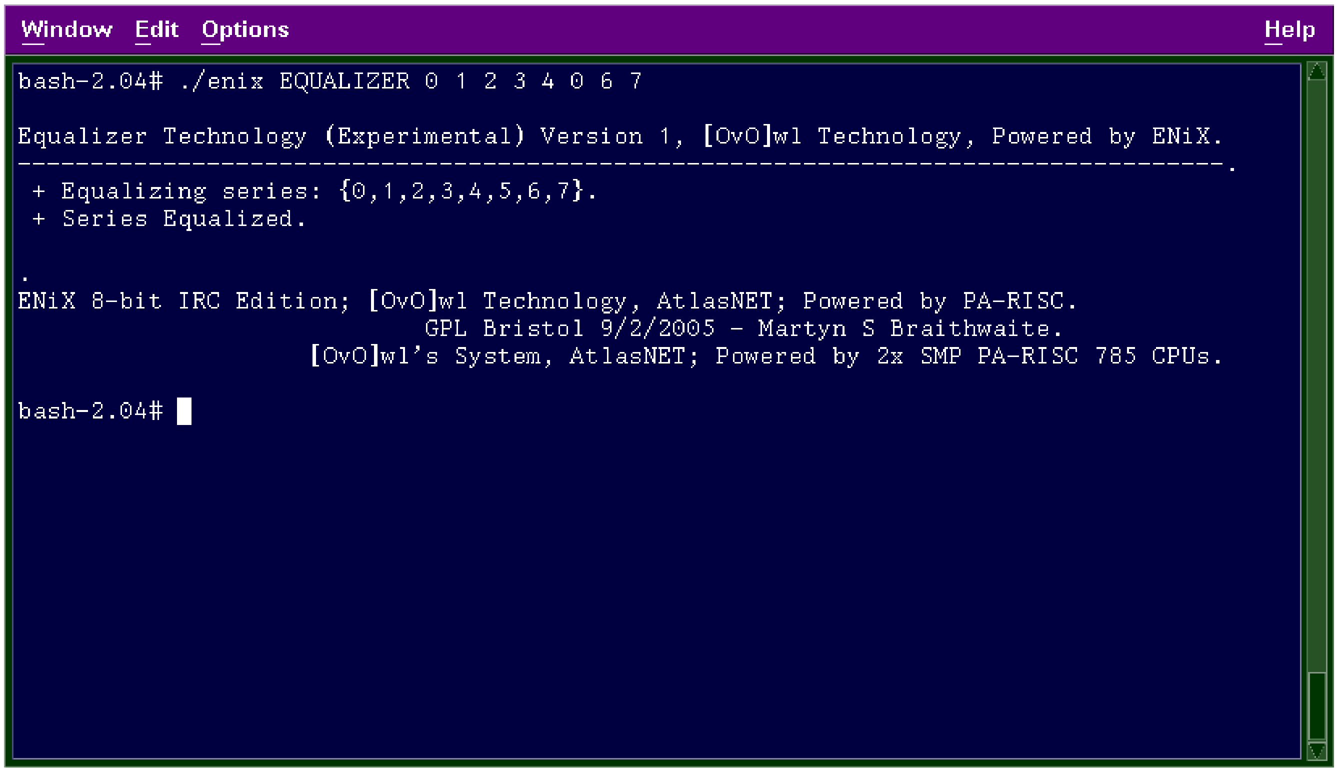
```
bash-2.04# ./a.out CHOCKY

---------------------------------------------------------------------------------------------------
Chocky Prototype. Advanced ENiX algorithms and heuristics. [OvO]wl Technology. Powered by ENiX.
---------------------------------------------------------------------------------------------------
:BEGIN - Start listing circumstances,           :CASE  - Specify scenerio,
:DO    - Consequences of scenerio,              :END   - End listing circumstances,
:EXIT  - Exit Chocky,                           :RESET - Clear Data,
:RUN   - Understand circumstances,              :THEN  - List of consequences,
---------------------------------------------------------------------------------------------------
<CTRL> :RESET
<ENiX> Data reset.
<CTRL> :BEGIN :THEN :END
<ENiX> ok
<CTRL> :BEGIN cheese :THEN cheese :END
<ENiX> ok
<CTRL> :BEGIN mouse :THEN mouse :END
<ENiX> ok
<CTRL> :BEGIN cat :THEN cat :END
<ENiX> ok
<CTRL> :BEGIN cheese mouse :THEN mouse :END
<ENiX> ok
<CTRL> :BEGIN cat mouse :THEN cat :END
<ENiX> ok
<CTRL> :RUN
<ENiX> Scenerio loaded into memory.
<CTRL> :CASE cat mouse :DO
<ENiX> ok
<ENiX> cat
<CTRL> :CASE cat cheese :DO
<ENiX> ok
<ENiX> cheese cat
<CTRL> :CASE cat mouse cheese :DO
<ENiX> ok
<ENiX> cat
<CTRL> :EXIT
<ENiX> Chocky has finished.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                             GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                     [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04# █
```

```
bash-2.04# ./enix DUALSCAN 0 1 2 3 4 0 6 7
```



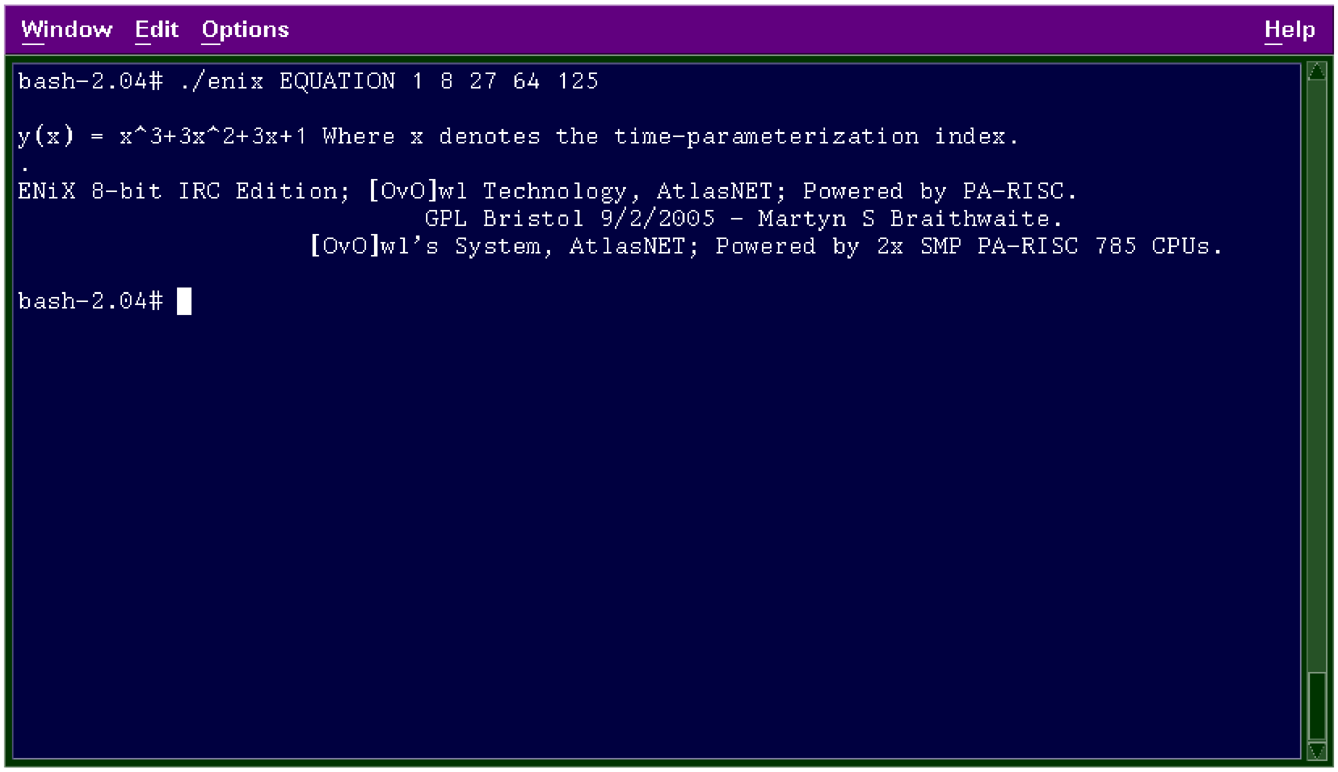
```
bash-2.04# ./enix DUALSCAN 0 1 2 3 4 0 6 7

DualScan Technology (Experimental) Version 1, [OvO]wl Technology, Powered by ENiX.
---------------------------------------------------------------------------------.
 + Acquiring solution set, {0,1,2,3,4,0,6,7}.
 + Assimilating Dual problem solution set, {81,51,11}.
 + Applying ENiX intelligent correction heuristics (Oddman-Out) to dual problem,.
 + Likely errors: [0,9][2,10][1,17].
 + Applying ENiX intelligent correction heuristics (Oddman-Out) to original problem,.
 + Likely errors: [5,9][7,1162][4,35912][1,51188][6,86469][3,86470][0,87076][2,87076].
 + Acquiring solution set, {0,1,2,3,4,5,6,7}.
 + Computing dichotomy: [0->5]; Error bits: { 0 2 }.
 + DualScan agrees with this answer 54 %.

.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix EQUALIZER 0 1 2 3 4 0 6 7
```



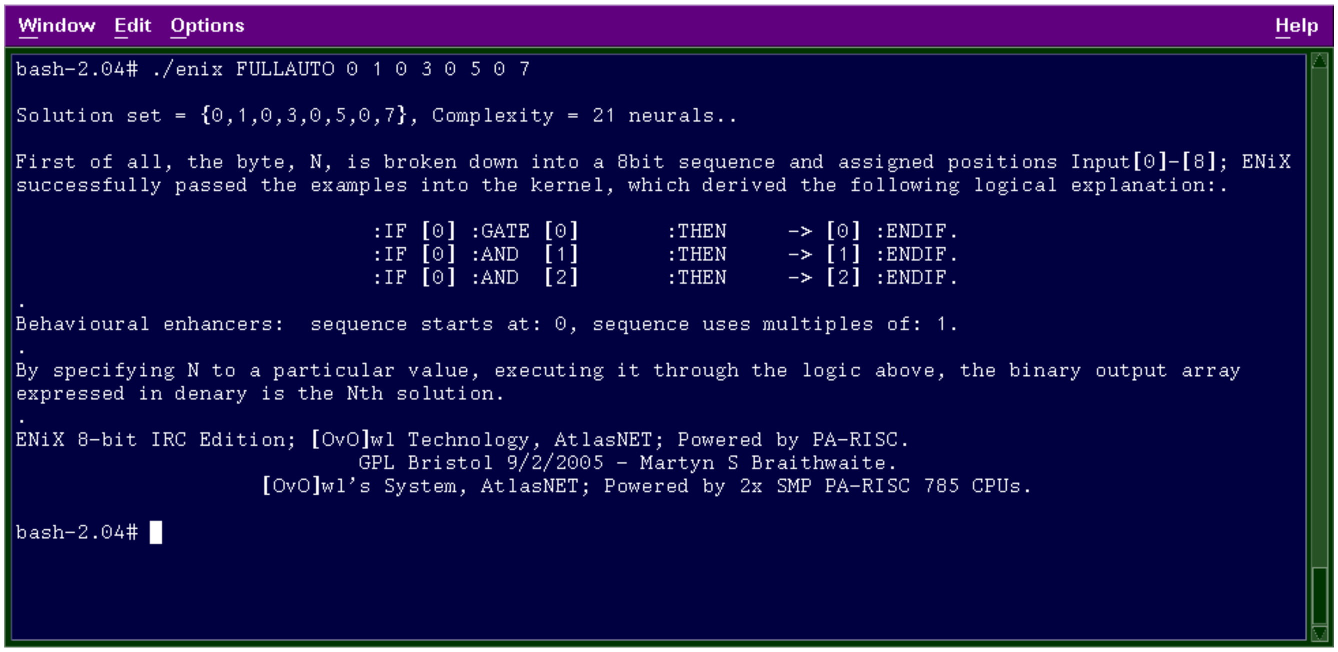
```
bash-2.04# ./enix EQUALIZER 0 1 2 3 4 0 6 7

Equalizer Technology (Experimental) Version 1, [OvO]wl Technology, Powered by ENiX.
----------------------------------------------------------------------------------.
 + Equalizing series: {0,1,2,3,4,5,6,7}.
 + Series Equalized.

.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix EQUATION 1 8 27 64 125
```



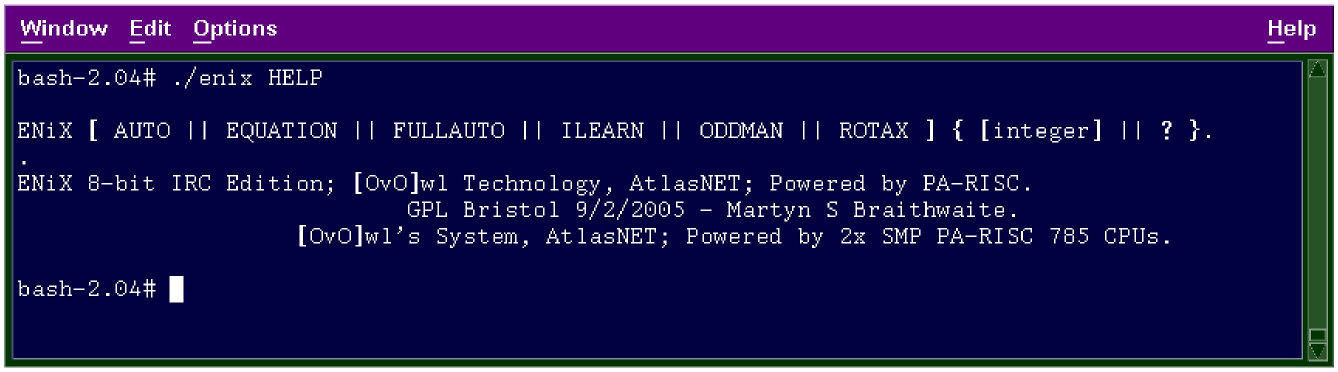
```
bash-2.04# ./enix EQUATION 1 8 27 64 125

y(x) = x^3+3x^2+3x+1 Where x denotes the time-parameterization index.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix FULLAUTO 0 1 0 3 0 5 0 7
```



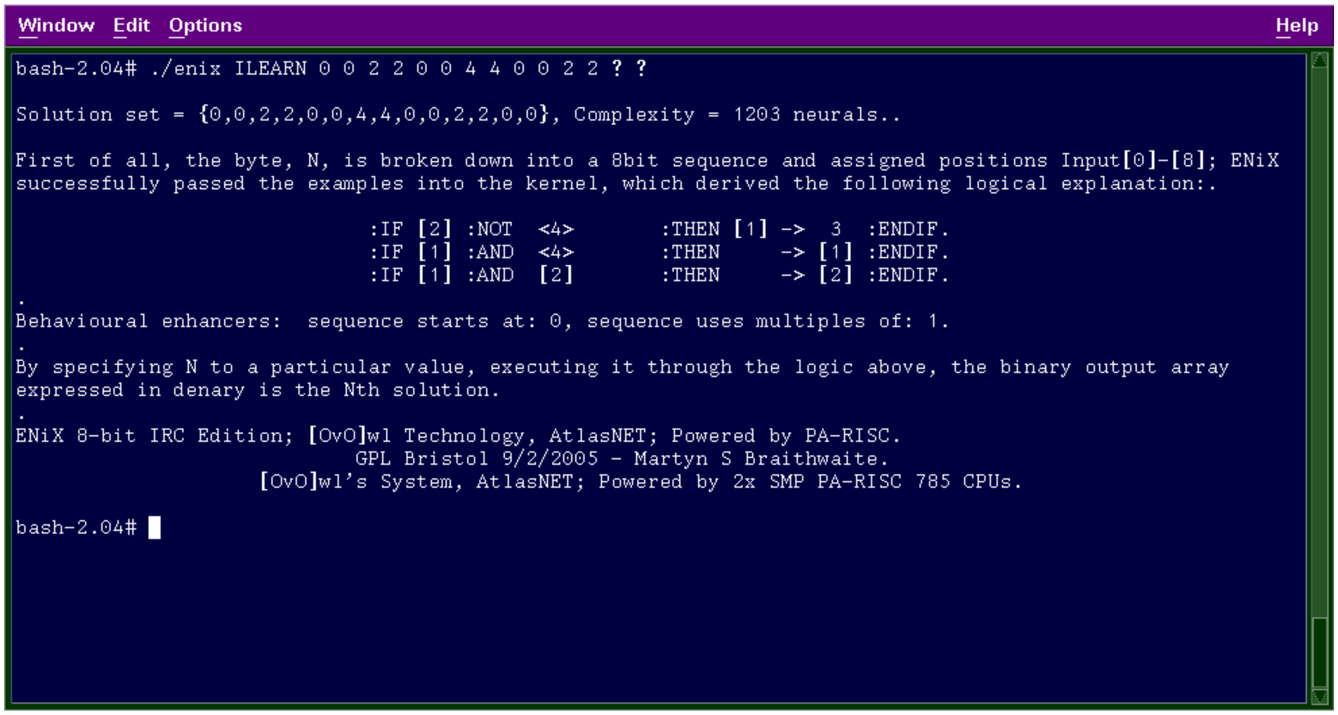
```
bash-2.04# ./enix FULLAUTO 0 1 0 3 0 5 0 7

Solution set = {0,1,0,3,0,5,0,7}, Complexity = 21 neurals..

First of all, the byte, N, is broken down into a 8bit sequence and assigned positions Input[0]-[8]; ENiX
successfully passed the examples into the kernel, which derived the following logical explanation:.

                             :IF [0] :GATE [0]       :THEN     -> [0] :ENDIF.
                             :IF [0] :AND  [1]       :THEN     -> [1] :ENDIF.
                             :IF [0] :AND  [2]       :THEN     -> [2] :ENDIF.
.
Behavioural enhancers:  sequence starts at: 0, sequence uses multiples of: 1.
.
By specifying N to a particular value, executing it through the logic above, the binary output array
expressed in denary is the Nth solution.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix HELP
```



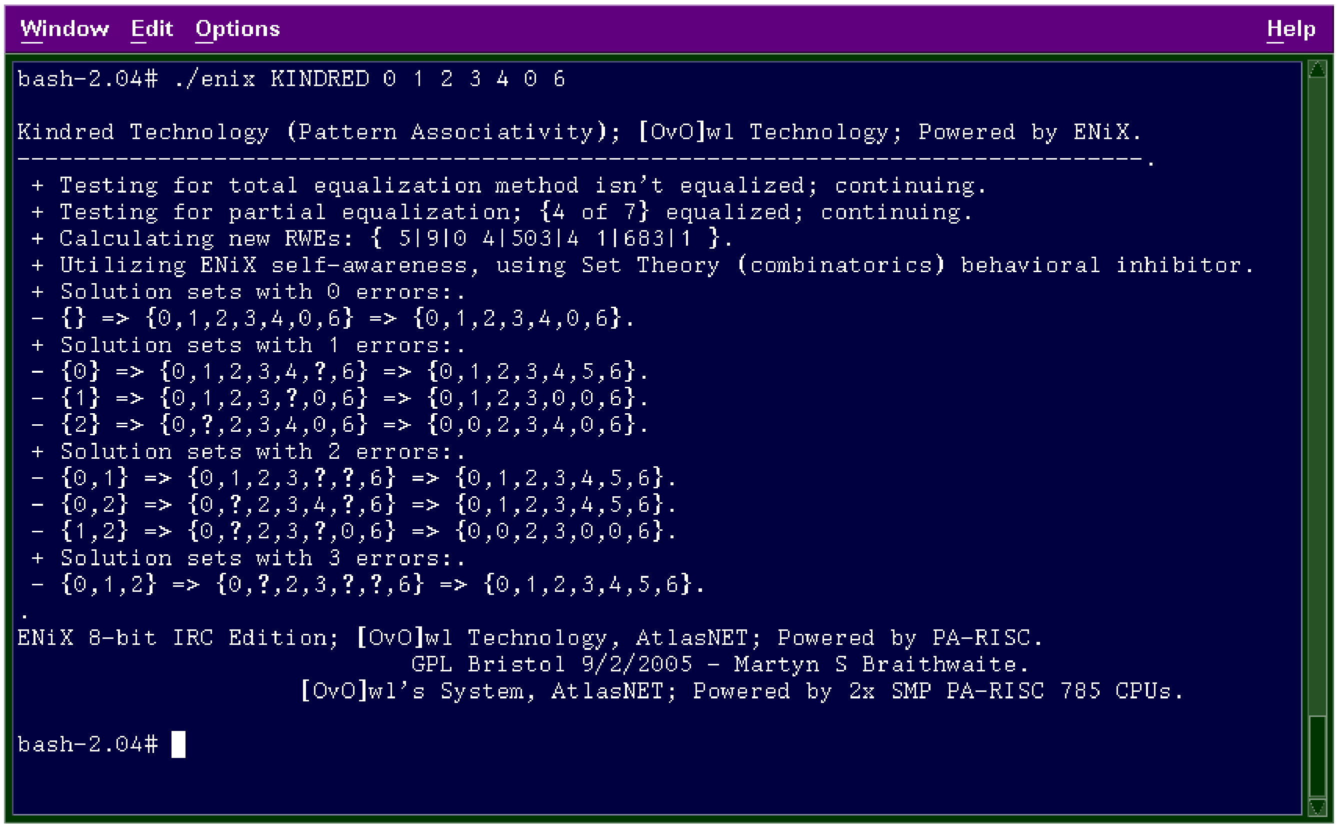
```
bash-2.04# ./enix HELP

ENiX [ AUTO || EQUATION || FULLAUTO || ILEARN || ODDMAN || ROTAX ] { [integer] || ? }.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                                   GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                         [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix ILEARN 0 0 2 2 0 0 4 4 0 0 2 2 ? ?
```



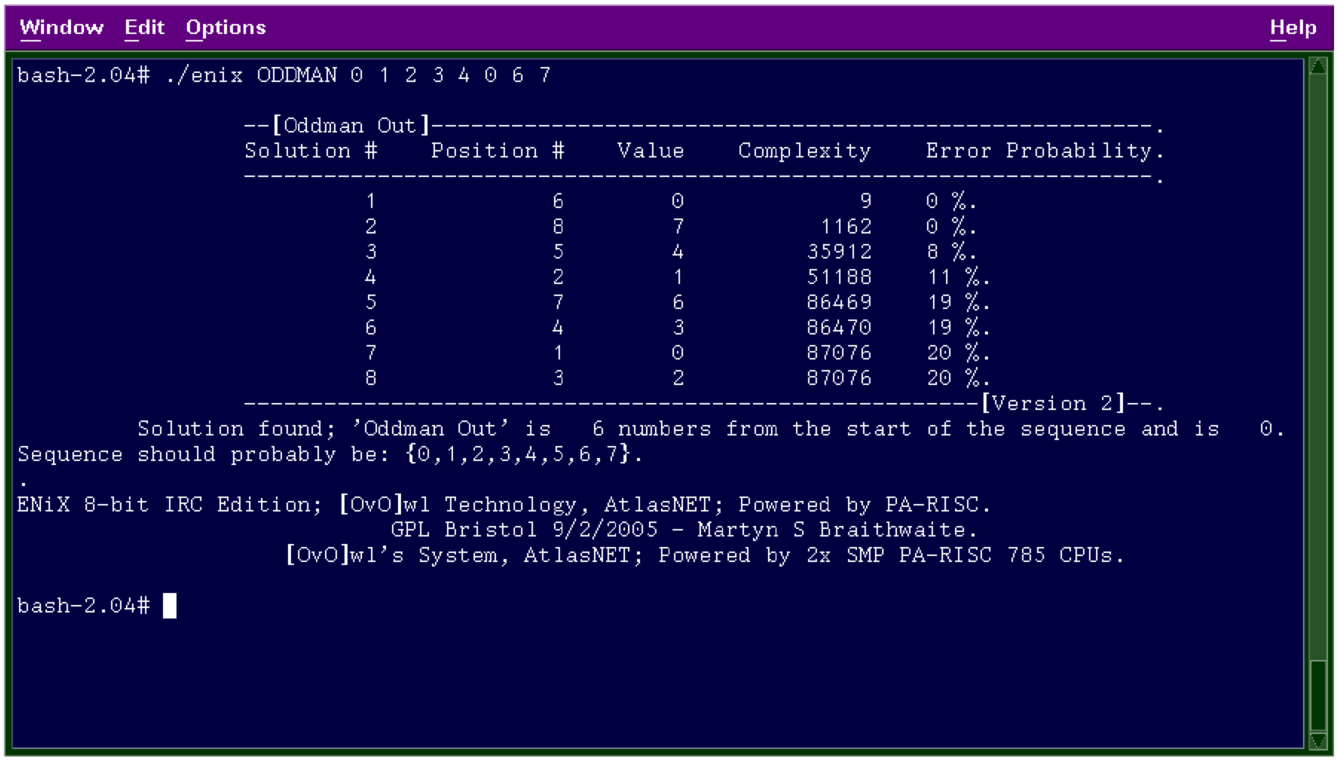
```
bash-2.04# ./enix ILEARN 0 0 2 2 0 0 4 4 0 0 2 2 ? ?

Solution set = {0,0,2,2,0,0,4,4,0,0,2,2,0,0}, Complexity = 1203 neurals..

First of all, the byte, N, is broken down into a 8bit sequence and assigned positions Input[0]-[8]; ENiX
successfully passed the examples into the kernel, which derived the following logical explanation:.

                             :IF [2] :NOT  <4>       :THEN [1] ->  3  :ENDIF.
                             :IF [1] :AND  <4>       :THEN     -> [1] :ENDIF.
                             :IF [1] :AND  [2]       :THEN     -> [2] :ENDIF.
.
Behavioural enhancers:  sequence starts at: 0, sequence uses multiples of: 1.
.
By specifying N to a particular value, executing it through the logic above, the binary output array
expressed in denary is the Nth solution.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                              GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix KINDRED 0 1 2 3 4 0 6 7
```



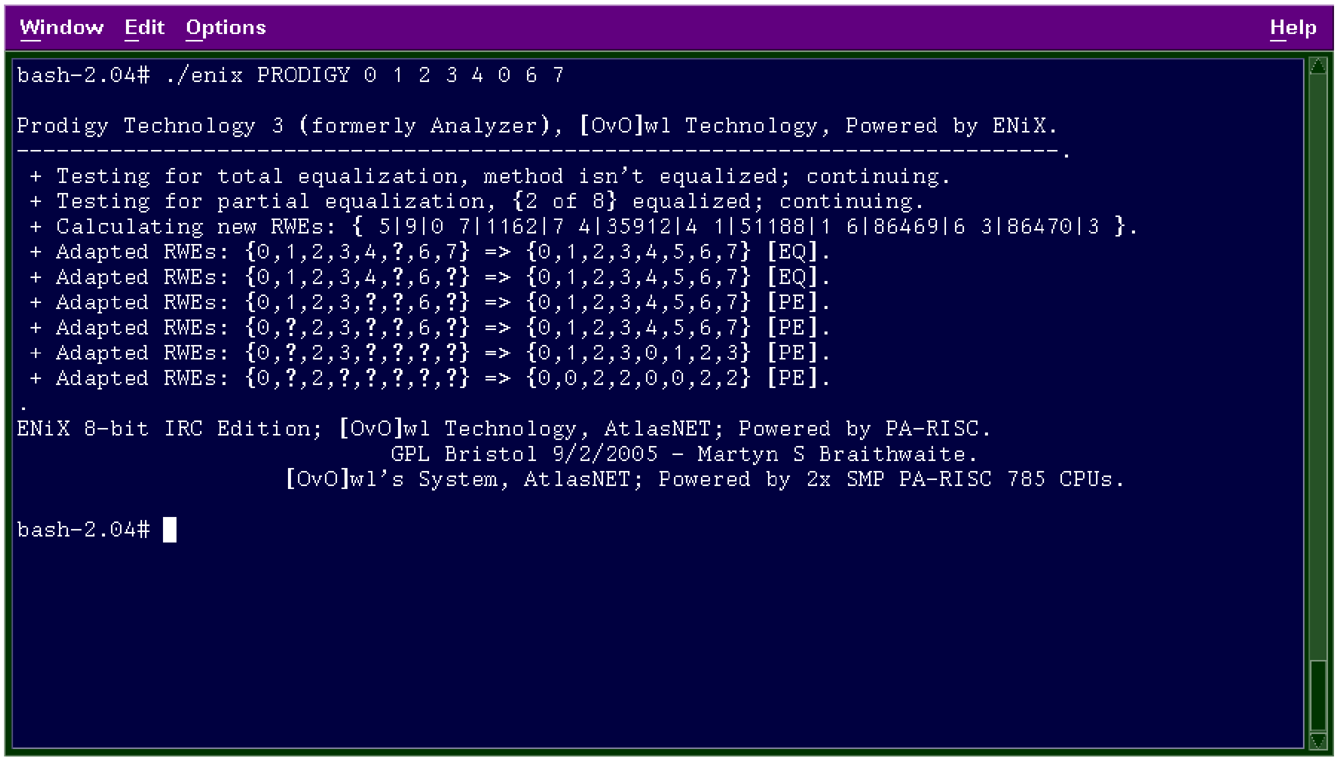
```
bash-2.04# ./enix KINDRED 0 1 2 3 4 0 6

Kindred Technology (Pattern Associativity); [OvO]wl Technology; Powered by ENiX.
--------------------------------------------------------------------------------.
 + Testing for total equalization method isn't equalized; continuing.
 + Testing for partial equalization; {4 of 7} equalized; continuing.
 + Calculating new RWEs: { 5|9|0 4|503|4 1|683|1 }.
 + Utilizing ENiX self-awareness, using Set Theory (combinatorics) behavioral inhibitor.
 + Solution sets with 0 errors:.
 - {} => {0,1,2,3,4,0,6} => {0,1,2,3,4,0,6}.
 + Solution sets with 1 errors:.
 - {0} => {0,1,2,3,4,?,6} => {0,1,2,3,4,5,6}.
 - {1} => {0,1,2,3,?,0,6} => {0,1,2,3,0,0,6}.
 - {2} => {0,?,2,3,4,0,6} => {0,0,2,3,4,0,6}.
 + Solution sets with 2 errors:.
 - {0,1} => {0,1,2,3,?,?,6} => {0,1,2,3,4,5,6}.
 - {0,2} => {0,?,2,3,4,?,6} => {0,1,2,3,4,5,6}.
 - {1,2} => {0,?,2,3,?,0,6} => {0,0,2,3,0,0,6}.
 + Solution sets with 3 errors:.
 - {0,1,2} => {0,?,2,3,?,?,6} => {0,1,2,3,4,5,6}.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04# █
```

```
bash-2.04# ./enix ODDMAN 0 1 2 3 4 0 6 7
```



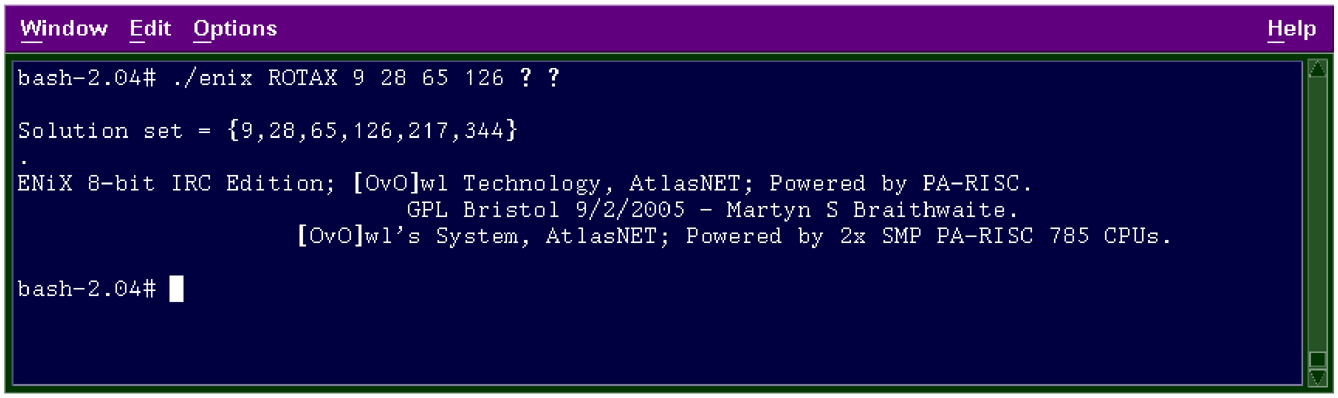
```
bash-2.04# ./enix ODDMAN 0 1 2 3 4 0 6 7

                 --[Oddman Out]----------------------------------------------------.
                 Solution #    Position #    Value    Complexity    Error Probability.
                 ------------------------------------------------------------------.
                          1             6        0             9    0 %.
                          2             8        7          1162    0 %.
                          3             5        4         35912    8 %.
                          4             2        1         51188    11 %.
                          5             7        6         86469    19 %.
                          6             4        3         86470    19 %.
                          7             1        0         87076    20 %.
                          8             3        2         87076    20 %.
                 -------------------------------------------------------[Version 2]--.
         Solution found; 'Oddman Out' is   6 numbers from the start of the sequence and is   0.
Sequence should probably be: {0,1,2,3,4,5,6,7}.
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04# █
```

```
bash-2.04# ./enix PRODIGY 0 1 2 3 4 0 6 7
```



```
bash-2.04# ./enix PRODIGY 0 1 2 3 4 0 6 7

Prodigy Technology 3 (formerly Analyzer), [OvO]wl Technology, Powered by ENiX.
------------------------------------------------------------------------------.
 + Testing for total equalization, method isn't equalized; continuing.
 + Testing for partial equalization, {2 of 8} equalized; continuing.
 + Calculating new RWEs: { 5|9|0 7|1162|7 4|35912|4 1|51188|1 6|86469|6 3|86470|3 }.
 + Adapted RWEs: {0,1,2,3,4,?,6,7} => {0,1,2,3,4,5,6,7} [EQ].
 + Adapted RWEs: {0,1,2,3,4,?,6,?} => {0,1,2,3,4,5,6,7} [EQ].
 + Adapted RWEs: {0,1,2,3,?,?,6,?} => {0,1,2,3,4,5,6,7} [PE].
 + Adapted RWEs: {0,?,2,3,?,?,6,?} => {0,1,2,3,4,5,6,7} [PE].
 + Adapted RWEs: {0,?,2,3,?,?,?,?} => {0,1,2,3,0,1,2,3} [PE].
 + Adapted RWEs: {0,?,2,?,?,?,?,?} => {0,0,2,2,0,0,2,2} [PE].
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

```
bash-2.04# ./enix ROTAX 9 28 65 126 ? ?
```



```
bash-2.04# ./enix ROTAX 9 28 65 126 ? ?

Solution set = {9,28,65,126,217,344}
.
ENiX 8-bit IRC Edition; [OvO]wl Technology, AtlasNET; Powered by PA-RISC.
                            GPL Bristol 9/2/2005 - Martyn S Braithwaite.
                    [OvO]wl's System, AtlasNET; Powered by 2x SMP PA-RISC 785 CPUs.

bash-2.04#
```

ENiX THEORY OF EQUALISATION

Oddman-out shows not only the oddman out but it provides a ranked indication of which values are mostly likely to be correct based on the level of complexity in evaluating the error-masked-sequence(s) divided by the total complexity of all problems with correct-masked-sequences. This is displayed as a percentage of the total complexity of computing all the sequences. Higher percentage means that there is more chance that the value is correct. Interestingly, when a problem is in it's simplist form the probability of these values being “correct” turn out to be all equal. This is because in an equalised sequence, substituting any one of these values with an unknown flag '?', can be reverted back to the intended sequence and that there is no value in the sequence which makes more or less sense than others. This represents a notion of interpolated cognition used within ENiX in modules such as EQUALIZER, ANALYZER and PRODIGY, a notion of completeness of data for the evaluated model to the end of the current power of 2 terms in the sequence.

“Partial equalisation” occurs when some of the terms are considered more “correct” than others while some terms are equal in complexity. The “correct” values have higher percentage of “correctness” and the first layer of partial equalisation occurs around the “correct” values and are equal. These high percentage values that are equal can be the ones that are considered most “correct” because of this. This form of equalisation could represent not only the recognition of an error inside of the sequence but also the recognition that there is a salvagable underlying pattern beneath the error(s).

“Not equalised” occurs when no two values within the sequence have the same “correctness” within the sequence. This typically occurs when either the sequence is insufficiently long (insufficient information) or there are too many errors to find any pattern in the data at all. This is a situation where ENiX has no cognition of the sequence since attempting to “equalize” it would lead to the equalisation of all values about the one which is considered simplist and it will consequently default to the simplist state featuring the “correct” value.

SO WHY DOESN'T ENiX PREDICT THE SAME ANSWER AS ME?

Is 2 4 6 8 etc, any more correct than 2 4 8 16 etc ? Yet both of these examples start with the same two terms. In a system that analyses addition before multiplication, the first sequence would be more “correct” yet to a different system analyzing multiplication above addition, it would be the latter that would seem more “correct”... The notion of “correctness” is only really consistant between pattern analysers that compute the models' complexity in the same, or similar way. That is, assuming that they adhire to Ockham's Razor at all. In other words, “correct” is merely a subjective value.