Ev

COLLABORATORS			
	TITLE : Ev		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 14, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Ev

1.1 Ev - Math tool for its users

```
                Ev v1.3

A replacement for Eval, distributed with Amiga OS.


The Usual Menu


                INTRODUCTION
                - Differences between Ev and Eval

                FUNCTIONS
                - Functions implemented

                USAGE
                - How to express yourself

                BASES
                - Using different bases

                ADVANCED USAGE
                - Multipel expression evaluation

                SCRIPT USAGE
                - How to use Ev in batch scripts

                BUGS
                - Nah..!

                HISTORY
                - What has happened?

                INDEX
                - Ev commands


                FREEWARE!
                - What else?
```

```
                AUTHOR
                - It's me!

                COMMERCIAL BREAK
                - Other software


Greetings to:

       Matthias Bethke for testing out the program and reporting bugs
       Michaela Prüß for great ideas and tons of help
```

1.2 Why does this program exist?

```
What is this?


I have for some time been very unsatisfied with the functions of the tool
eval, included in Amiga OS. I was hoping for an update in Amiga OS 3.5 but
nothing has been done to this tool.
Therefor I have written my own tool for mathematical evaluations.

The worst things about the old eval:

* It can't count! 1+2*3 should equal 7 not 9!! (no priorities?)
* No suport for binary numbers
* No support for decimal numbers
* Too few functions
* Not flexible enough to be userfriendly
* No bugfixes or upgrades are made from the developer

Ev is improved in theese areas and some other:

* The prioritys are correct! 1+2*3 = 7
* Support for binary numbers added in input and output
* The constants pi and e added
* Support for decimal numbers added, both 0.1 and .1 are accepted
* Correct rounding of decimals

* New flexibility added with support for:
  * the words and (&), or (|), not (~)
  * both <<, >> and lsh, rsh (and l, r) for left/right shift
  * d## and @d for decimal input and output
  * several new ESC-codes in output (tab, bell, formfeed etc.)
  * number of digits in octal and hexadecimal output is optional
  * multiplication without '*'-token in obvious places
~ * reading expressions from a file or stdin
~ * multiple expression calculation
  * variables

* Several new functions:
  * absolute value (abs)
  * square root (sqrt)
```

```
  * faculty (!)
  * powered numbers (^)
  * trigonometric functions (sin, cos, tan, asin, acon, atan, sinh, cosh, tanh)
  * random number (rnd)
  * logarithmic functions (ln, log, log2, logx)

+ I will keep upgrading it if there is a need for that.

Downside of the new version:

* Bigger executable. I've used the tools bison and flex to create the parser
  and theese tools do not create small files.

* Not 100% compatible with Eval, e does not mean eqv any more since the
  number e = 2.71828182845904.
  From v1.2 the %n, %o and %x has changed to @n, @o and @x to make Ev
  preform better in scripts, this means more compability to Eval is lost.
  On the other hand it was Amiga Inc. who told me to make the changes so I
  guess it's allright.

* The argument management seems to be different in some ways which seems
  to have a negative effect on the behaviour of Ev in scripts. For example
  the test script in the manual for OS 3.5 with Eval used as a counter will
  not work with Ev. Other scripts I have seen works fine though.

I hope thats it.
```

1.3 What can it do?

```
                Functions suported by Ev


All input values are interpreted as decimal numbers by default. If you
need to specify other forms of input see
                Using different bases
                Arithmetic operations

The four rules of arithmetics, +, -, * and / are of course supported,
and I don't see any point in explaining them any futher.
NOTE that the times sign, '*', can be omitted in the usual cases,
4sin(#) = 4*sin(#), 2pi = 2*pi, 5(4+2) = 5*(4+2) etc.

mod     Modulo. Returns remainder from division.
                Can also be written as: % or m

^       Power function.   Eg: 4^2 = 4*4 = 16
                              4^3 = 4*4*4 = 64

!       Faculty. 5! = 1*2*3*4*5 = 120
                 n! = 1*2*3*4*...*(n-1)*n

Sqrt()  The square root. Sqrt(16) = 4 Because 4*4 = 16
                         Sqrt(25) = 5 (5*5 = 25)

Abs()   Absolute value. Abs(-3) = 3
```

Abs(3) = 3 ie Makes numbers positive.

Bitwise operations

and Can also be written as: &

or Can also be written as: |

xor Can also be written as: x

not Can also be written as: ~

lsh Left shift (of bits). Can also be written as: << or l

rsh Right shift (of bits). Can also be written as: >> or r

eqv Bitwise equvialence. (XNOR)

Trigonometric functions

The trigonometric functions are used with the aid of the unit circle.
The angle is measured in radians. One rotation of 360\textdegree{} is 2*pi radians ↩
.

sin() Sine

cos() Cosine

tan() Tangent

asin() Arcsine, the inverse of sine.

acos() Arccosine, the inverse of cosine.

atan() Arctangent, the inverse of tangent.

Hyperbolic functions

Any function defined on the real line can be expressed (in an unique way)
as the sum of an even function and an odd function. The hyperbolic functions
cosh(x) and sinh(x) are respectively the even and odd functions whose sum is
the exponential function e^x.

sinh() Hyperbolic sine

cosh() Hyperbolic cosine

tanh() Hyperbolic tangent

Logarithmic functions

log() The ten-logarithm function. log(x)=n <=> 10^n=x

```
ln()    The natural logarithm.      ln(x)=n     <=> e^n=x

log2()  The two-logarithm function. log2(x)=n   <=> 2^n=x

logx()  The any-logarithm function. logx(x,b)=n <=> b^n=x


Misc constants and other stuff

pi      3.141592653589793    NOTE: pi and e only have an accuracy of
                                   fifteen decimals.
e       2.718281828459045

rnd     A random number between 0 and 1.

Parenthesis, '(' and ')', can be included in the expressions.
```

1.4 Ten is boring!

```
                Using different bases


The most common base in modern mathematics is ten, (probably because
the number of fingers on the average human). It is however possible to
use any number as a base.
Ev supports the most common bases two, eight,ten and sixteen, ten is
used by default. To use some of the other bases you use a prefix.

-----------------------------------


Binary

The base two is used much in lowlevel comuting and digital electronics.
These numbers are called binary numbers and are typed as folows:

 b0000 = 0      b0100 = 4       b10011010 = 154
 b0001 = 1      b0101 = 5       b00010110 = 22
 b0010 = 2      b0110 = 6       b00010101 = 21
 b0011 = 3      b0111 = 7       b00101010 = 42

The 'b' is used to tell Ev that it is a binary number. There is no need
to write the zeros before the number, but the most common way to write
binary numbers are in groups of four or eight digits.

-----------------------------------


Octal

The base eight is allso pretty common in electronics, and the numbers
are called octal numbers. Ev expects you to write octal numbers
with the prefix 'o', '0' or '#':
```

```
 o5  = 5        027 = 23        #31 = 25
 o10 = 8        030 = 24        #52 = 42
```

As you might have notised there are no digits higher than 7. Octal numbers can not contain the digits eight or nine. An attempt to write o9 will result in a parse error.

Decimal

The decimal numbers are used by default, but the prefix 'd' can be used if you think it looks nice.

The decimal numbers can contain decimals, with a preceding number or without.

```
Eg: 47.11
     0.42
      .4711
```

Hexadecimal

The most used base in the computer world is sixteen. The numbers are called hexadecimal numbers or just hex for short. If you've looked in to the memory of your computer you have most likely seen hexadecimal numbers before.
Ev handles several different prefixes for hex: $, 0x and #x

```
 $7  = 7        0xf  = 15
 $a  = 10       0x10 = 16
 $b  = 11       #x11 = 17
 $c  = 12       #x2a = 42
```

As you can see there are some letters involved here. Allowed letters are: a, b, c, d, e, f. Since there are only ten figures, one have to use some letters to be able to express the hexadecimal numbers.

Characters

Ev can also read characters. It will use the ASCII values to calculate with. The prefix is: '

This means that Ev can be used to find the ASCII value for different characters:

```
Ev 'A   -> 65
Ev 'a   -> 97
```

Unfortunally the argument handling won't let you give a SPACE char as input, but the ASCII value for SPACE is 32 so now you won't have to.

```
-----------------------------------


All prefixes must be typed BEFORE their number, as showed in the examles
above. It is allowed to mix different bases in the same expression, the
answer will be decimal anyway. If you want to get the answer in binary,
octal or hexadecimal form you can use the appropriate
                LFORMAT string
                .
```

1.5 How do I use this thing?

```
                Usage


TEMPLATE

V1=VALUE1,OP,V2=VALUE2/M,FROM/K,TO/K,A=APPEND/S,LF=LFORMAT/K,S=SCRIPT/S,RF= ↩
    READFILES/S

Just type the expression you want to evaluate, it's as simple as that!

Ev 1+2*3

Will give you the answer: 7

Ev (2(3+5)^2)/2+4
-> 68

You can use parenthesis with all functions, but they are only required
in some special cases.

For instance if you want to calculate 2+3*4 it will give the answer 14.
This might be what you wanted (if you know how to express yourself),
but if you want the answer 20, you have to use parenthesis around 2+3.

Eg: (2+3)*4 or 4(2+3)

Read more about
                functions
                 for detailed information
about each function supported.

-----------------------------------


VALUE1

The VALUE1 keyword will normally hold the entire expression to evaluate.
It consumes the input from the start to the first space character.

-----------------------------------
```

OP

Can be used to set the operator in the expression.
This operator will be concatenated onto the end of the VALUE1 expression and
will result in an parse error if the VALUE2 field is left empty.

VALUE2

The rest of the expression. The contents of this keyword will be concatenated
to the end of the expression built up from VALUE1 and OP.

FROM

Ev can be used to read input from a file or stdin and filter out expressions
to evaluate. The expressions should be encapsulated between @E and @e.
@E tells Ev to start read the expression and @e to end.
When FROM is defined all input in VALUE1, OP and VALUE2 will be ignored.
Allso the keywords READFILES and SCRIPT is ignored.

For more help on this issue see
Advanced Usage

TO

To redirect the output to a file, use the keyword TO <file>

Eg:~Ev 42 TO Work:File

Ev does not warn if the filename given already exists!
So in theory it is possible to overwrite important files with the TO keyword.

APPEND

APPEND tells Ev to add the output to the end of the file given with TO rather
than replacing an existing file.

LFORMAT

All ansvers is by default in decimal form. It is however possible to use a
format string to design the output.

Eg: Ev 6*7 LFORMAT="The answer is @d.*n"

```
Will give the result: The answer is 42.

The @d will insert the decimal value in the string. The available output
formats for this string are:

 @b Binary format
 @o Octal format
 @d or @n Decimal format
 @h or @x Hexadecimal format
 @c Display the character vith corresponding ASCII value

 @@ Will give the '@'-character in case there are collisions with codes given.

Binary, octal and hexadecimal values may have an optional value to determine
the least number of digits to display. If the number to display does not fit
into the selected number of digits more digits will be added to fit the number.
If a binary numver is greater than 16 bits a space will be added to separate
each group of eight bits.

Eg: Ev 42 LFORMAT="@b8 @o @d @x4 @c"  -> 00101010 52 42 002a *

    Ev 500 LFORMAT="@b4"              -> 111110100

    Ev 100000 LFORMAT="@b"            -> 1 10000110 10100000



Decimal values may have a value to determine the number of decimals to
display, and a value to fix the total width of the number (including decimals).
Up to fourteen decimals can be displayed.

The notation is @d<width>.<decimals> either can be omitted.

Eg: Ev 100.09 LFORMAT="@d.1"   -> 100.1

    Ev 100.99 LFORMAT="@d3.1"  -> 101.0

    Ev 99.9 LFORMAT="@d2.0"    -> 100


Decimal numbers is padded with the space-character, other bases with '0'.

NOTE: It is only possible to fix a minimum width of a number. If the number
      is greater (or has more decimals) the limit will be ignored.



Ev can read several different ESC-codes in the LFORMAT string which will
give the possibillities to create any form of output.

 *a Alert
 *b Backspace
 *f Formfeed
 *n Newline
 *r Carriage return
 *t Horizontal tab
```

```
 *v Vertical tab

 **~The '*'-character

NOTE: When using a LFORMAT string no newline character will be added to the
      string. You will have to add this yourself with *n if you want it.

-----------------------------------


SCRIPT

The script mode of Ev is slightly different from the rest of the program.
The functionality is cut down to a minimum and only integer input is allowed.

All text given to Ev while in script mode will be fed directly to the chosen
output. Only text within '@' will be processed and only numeric expressions
will be evaluated. The only characters that is interpreted as numeric are:

0 1 2 3 4 5 6 7 8 9 + - * / ^ ( )

All other characters will be treated as text, including space.

If combined with READFILES the filenames will be treated as files, not text.

Eg: Ev 1+2ab@3+4cd@   -> 1+2ab7cd

See
                Script usage
                 for more details.

-----------------------------------


READFILES

This keyword will force Ev to check whether each argument is a filename or not.
If a filename is given, the contents of that file will be included in the
expression. If any  numeric expression given to Ev collides with an existing
filename, the file will be included instead of the numeric expression.

Eg: Ev 42 + ENV:number READFILES

The contents of the file ENV:number will be inserted instead of the filename.
If there is a file named '42', or '+', that file will be inserted as well.

See
                Script usage
                 for more details.

NOTE: The entire file will be included and fed to the parser. If the file
      contains anything else than one single, correct expression a parse error
      will occur.
```

1.6 Script usage

```
How to use Ev in scripts


When Ev is given the keyword SCRIPT it will change form. The input can now
only contain simple expressions, everything else will be treated as text.
Text is directly fed to the chosen output.

The point in this is to make Ev interact with the batch scripts in a polite
and effective manner.

Input will be scanned for @'s and will only evaluate expressions in between
theese @'s.

Only the basic arithmetics are supported. +, -, *, / and ^
The ( ) can be used as usual, but that's it. No functions are available,
no decimal numbers (the . is treated as text) and only base ten input.

LFORMAT can still be used to control the output though. However, the output
will always end with a newline.



Eg: This example will show one of the many posibillities with SCRIPT mode.

Here the command LIST is used to make a batch script for renaming files:

LIST www1??.gif LFORMAT="Ev Rename %n @%m+900@.gif TO b2 APPEND SCRIPT" >batch

There will now be a file named 'batch' containing lines like theese:

...

Ev Rename www144.gif @www144+900@.gif TO b2 APPEND SCRIPT
Ev Rename www145.gif @www145+900@.gif TO b2 APPEND SCRIPT
Ev Rename www146.gif @www146+900@.gif TO b2 APPEND SCRIPT

...

After this file has been executed there will be a new file called 'b2'
containing lines like theese:

...

Rename www144.pic www1044.pic
Rename www145.pic www1045.pic
Rename www146.pic www1046.pic

...

Witch is a simple way of renaming a couple of hundred files at the same time.

-----------------------------------
```

```
Another example:

Setenv A 0

List a#? LFORMAT="Ev ENV:A + 1 TO ENV:A READFILES*nRename %n SF_$A.%e" >batch

Result in 'batch':

...

Ev ENV:A + 1 TO ENV:A READFILES
Rename a1 SF_$A.gif
Ev ENV:A + 1 TO ENV:A READFILES
Rename a11 SF_$A.gif

...


New filenames:

SF_1.gif
SF_2.gif
SF_3.gif
SF_4.gif
...


NOTE: The variable A MUST be initiated with SETENV or a call to Ev like
      Ev 0 TO ENV:A


NOTE 2: Ev can be used to other things than renaming files...
```

1.7 Advanced usage

```
                How to get more out of Ev


Multiple expression evaluation

When the keyword FROM is defined to a file or 'stdin' Ev will search the
input for a '@E' sequence before going into action. The input stream will
then be processed until the sequence '@e' is reached (or EOF).
The entire sequence @E ... @e will then be replaced with the result of the
evaluation and Ev will start scanning for @E again.

While reading the input stream the output format (LFORMAT) can be changed
with the command '@>LFORMAT string'. The format of the results inserted in
the out stream will default to decimal notation. To change the format for
the entire stream (or just the beginning of it) the LFORMAT keyword can be
used as normal.
If the output format is changed in the input stream with the @> command,
the output format will stay that way until the next @>.
```

```
NOTE: The syntax of the string after @> is exactly the same as in the
      LFORMAT string. See
                Usage
                 for mor info.

A short example illustrates this better than words:

A text file 'asc.txt' contains the folowing data:
```

```
ASCII Table:

Char   Dec   Hex   Bin

  A    @E'A@>@d@e    $@E'A@>@h2@e   @@E'A@>@b8@e
  B    @E'B@>@d@e    $@E'B@>@h2@e   @@E'B@>@b8@e
  C    @E'C@>@d@e    $@E'C@>@h2@e   @@E'C@>@b8@e
  D    @E'D@>@d@e    $@E'D@>@h2@e   @@E'D@>@b8@e
  E    @E'E@>@d@e    $@E'E@>@h2@e   @@E'E@>@b8@e
  F    @E'F@>@d@e    $@E'F@>@h2@e   @@E'F@>@b8@e
  G    @E'G@>@d@e    $@E'G@>@h2@e   @@E'G@>@b8@e
  H    @E'H@>@d@e    $@E'H@>@h2@e   @@E'H@>@b8@e
```

```
A call to Ev will now give the output:

> Ev FROM "asc.txt"

ASCII Table:

Char   Dec   Hex   Bin

  A    65    $41   %01000001
  B    66    $42   %01000010
  C    67    $43   %01000011
  D    68    $44   %01000100
  E    69    $45   %01000101
  F    70    $46   %01000110
  G    71    $47   %01000111
  H    72    $48   %01001000

The call 'Ev FROM stdin < asc.txt' will give the exact same result.

To redirect the output to a file the keyword TO can be used as normal.

NOTE: A call to Ev with a file containing no @E will simply copy the file
      to the output stream.


Using variables

When using multiple expression evaluation it might be useful to remember
certain values from one expression to another. Ev has this opportunity.

The syntax is: name:value}

The name can be any string (max 100 characters) and is case sensitive.
```

The value is any expression that Ev can evaluate and can include other variables or even a recursive call to the same one. All variables are initialized to 0 if no other value is given.

To change the value of a variable simply give it an expression in the value field. If the field is left empty, (real emty that is, not even a space is allowed), the current value will be returned.

Eg: will initialize the variable horse to 5 and return it.

      will return the value of horse (5).

    +1} will increase the value of horse and return the
                          result (6).

A variable is interpreted as an expression and must be treated that way. For instance the variable horse will after the example above be exactly the same as (6), with parenthesis.

All variable usage will result in output, even initializations. To prevent this the command @q can be used in the expression string.

Eg: @E @q @e

    This will, if fed as a stream to Ev, initialize the variable hosre
    to 7 quietly. No output will be fed to the outstream from this expression.

One @q command is enough per expression, it will effect the entire @E ... @e sequence. @q can not be placed within the variable value.

@q will ofcourse work fine even in expressions without variables, but it would be pretty poitless. The expression would be evaluated and the result thrown away. So, the @q is mainly for variable manipulation under the surface.

For a real hairy example
                click here

1.8 Variable example

Example of inputfile

_file: asc.txt_____

ASCII Table:

Char   Dec   Hex

```
  @E@>@c@e@E@>@d7@e@E@>   $@h2@e
  @E+1}@>@c@e@E@>@d7@e@E@>   $@h2@e
  @E+1}@>@c@e@E@>@d7@e@E@>   $@h2@e
  @E+1}@>@c@e@E@>@d7@e@E@>   $@h2@e
```

```
> Ev FROM asc.txt

ASCII Table:

Char   Dec   Hex

  A     65   $41
  B     66   $42
  C     67   $43
  D     68   $44
```

Not very exciting but it is an example. More powerful things can be done using scripts of some kind.

1.9 Ev commands

```
                Command Index


!          Expression operator
                Faculty
               ##         Expression value
                Octal value
               #x#        Expression value
                Hexadecimal value
               $#         Expression value
                Hexadecimal value
               %          Expression operator
                Modulo
               &          Expression operator
                and
               '#         Expression value
                ASCII value
               *          Expression operator
                Times
               **         Outstream command
                '*' character
               *a         Outstream command
                Alert
               *b         Outstream command
                Backspace
               *f         Outstream command
                Formfeed
               *n         Outstream command
                Newline
               *r         Outstream command
                Carriage return
               *t         Outstream command
                Horizontal tab
               *v         Outstream command
```

```
 Vertical tab
+         Expression operator
 Plus
-         Expression operator
 Minus
/         Expression operator
 Division
<<        Expression operator
 Left shift
>>        Expression operator
 Right shift
^         Expression operator
 Power
|         Expression operator
 or
~         Expression operator
 not
0#        Expression value
 Octal value
0x#       Expression value
 Hexadecimal value
@@        Outstream command
 '@' character
@>        Instream command
 New LFORMAT string
@b#       Outstream command
 Binary format
@c#       Outstream command
 Character format
@d#       Outstream command
 Decimal format
@E        Instream command
 Begin evaluation
@e        Instream command
 End evaluation
@h#       Outstream command
 Hexadecimal format
@n#       Outstream command
 Decimal format
@o#       Outstream command
 Octal format
@q        Instream command
 Quiet
@x#       Outstream command
 Hecadecimal format
@{Name}   Instream command
 Variable
abs()     Expression function
 Absolute value
acos()    Expression function
 Arccosine
and       Expression operator
 and
APPEND    Commandline keyword
 Append on existing file
asin()    Expression function
 Arcsine
```

```
atan()    Expression function
 Arctangent
b#        Expression value
 Binary value
cos()     Expression function
 Cosine
cosh()    Expression function
 Hyperbolic cosine
d#        Expression value
 Decimal value
e         Expression constant
 2.718281828459045
eqv       Expression operator
 Bitwise equvialence
FROM      Commandline keyword
 Input stream
l         Expression operator
 Left shift
LFORMAT   Commandline keyword
 Output format string
ln()      Expression function
 Natural logarithm
log()     Expression function
 Ten-logarithm
log2()    Expression function
 Two-logarithm
logx()    Expression function
 Any-logarithm
lsh       Expression operator
 Left shift
m         Expression operator
 Modulo
mod       Expression operator
 Modulo
not       Expression operator
 not
o#        Expression value
 Octal value
OP        Commandline keyword
 Operator in expression
or        Expression operator
 or
pi        Expression constant
 3.141592653589793
r         Expression operator
 Right shift
READFILES Commandline keyword
 Treat filenames as files
rnd       Expression value
 Random value
rsh       Expression operator
 Right shift
SCRIPT    Commandline keyword
 Script mode
sin()     Expression function
 Sine
sinh()    Expression function
```

```
 Hyperbolic sine
sqrt()    Expression function
 Square root
tan()     Expression function
 Tangent
tanh()    Expression function
 Hyperbolic tangent
TO        Commandline keyword
 Output stream
VALUE1    Commandline keyword
 First expression value
VALUE2    Commandline keyword
 Rest of expression
x         Expression operator
 xor
xor       Expression operator
 xor
```

1.10 Any bugs?

```
Bug reports
```

Well, if I knew about the bugs I would fix them, not release the program.
I am not Micro$oft.

But I'm only human and the code is big so the program probably contains
several strange lifeforms...
...Please let me know!

Expressions resulting in huge output strings will eventually start writing
in unallocated memory. This will only happen if a single number in the
expression is larger than 1000 digits or if the LFORMAT string is used in
a destructive maner. I.e. LFORMAT="@b9999999999" will do bad things.
I do not consider this a bug since the allocated memory has to have a fix
size. No matter how big chunk of memory I allocate, it will always be
posible to outrange it. The output string can at the moment be 4000 bytes
long. For multiple expression evaluation this means 4000 bytes per
expression. (Text between expressions do not occupy any memory.)
I belive that's enough for normal usage, if you need a bigger output
string feel free to send me an eMail and I will increase the buffer.

```
How to contact me.
Disclaimer
```

If this program gets strange input (non mathematical things), it might
behave strange, (who wouldn't). But strange input will probably give
parse error, and it is most unlikely that it crashes your computer.
Please try!

If you do manage to destroy anything with this software I will not

feel responsible in any way.

It is possible to overwrite important files with the TO keyword. It does
not warn if the filename given already exists!

1.11 Hey! I wrote this!

This program was written by:

Jesper Wilhelmsson
jive@algonet.se

Please let me know what you think about it!

On my web page you can find tons of source codes in several diferent
programming languages. Test programs, Games, compilers and an operating system
among other things.

There are also a lot of help texts for programmers.
All in swedish though...

http://welcome.to/syntaxerror

1.12 Copy, copy, copy!!!

This program is FreeWare!

This means that you can spread it freely in what forms you may please.

The one restriction is that no one is alowed to make any money out of it!
It is strictly forbidden to pay any money to anyone in order to get a copy.
(Including the author, I don't want any money - I just want eMail!)

1.13 How has this evolved?

History

v1.0 (2000.01.13)
First public release.

v1.01 (2000.01.18)
Fixed bug in argument parser. Accidentally used <= instead of <, oops.
Not a public release.

v1.1 (2000.01.25)

```
       Several new functions: asin(), acos(), atan(), sinh(), cosh(), tanh(),
                              ln(), log()
       Increased the precition of pi and e to fifteen decimals.
       Output can now show up to fourteen decimals.
       Added ESC-codes: *a, *b, *f, *r, *v (see
                Usage
                 for info).
       Fixed problem with % and * characters.
       New Power function will give an exact answer up to 10^22. C's standard
       pow() only gives an exact ansver up to 10^9.
       Ev can now read input from a file or stdin with multiple expression
       evaluation, (see
                Advanced usage
                 for info).
       Variables added.
       Now using standard Amiga argument parser, hopefully this will give
       better compilance with Eval.
       Various optimations made executable smaller and faster!

v1.2  (2000.02.04)
       Added keywords APPEND, READFILES and SCRIPT.
       Replaced all %~with @ in commands and codes. The use of % created some
       unpleasant behaviour in script mode due to inconsequence in LFORMAT
       strings between different applications. This means that the codes for
       different bases is no longer compilant with Eval.

v1.3  (2000.04.10)
       Locale support, English is built in and Swedish locale file is included.
       New functions: log2(), logx() (sorry, I forgot about them..)
       Various optimizations.
```

1.14 Other great products

```
(: Other stuff made by me :)

W R I G G L E  v1.5

It is almost the standard worm-game except for
the controls.  You control  the  worm with the
mouse, which can be tricky at first, but after
a few rounds you'll  se the advantages  with a
'non square  worm'.  Move  freely  around  the
board  and pick up boxes.  There is, ofcourse,
no danger involved  in crossing your own tail.
Have you  ever  seen  a worm  that  dies  just
because he stumbles on his own tail?

Can be found at Aminet:  game/misc/Wriggle.lha



Char Counter

Finds the n:th character in a file and display
```

it together with its suroundings. Nice when a compiler or something tells you there is an error at character 4711 in a file.

Can be found at my homepage, including source:
http://welcome.to/syntaxerror

S n o O o w ! v1.1

Configurable Snoowy window on your Worbench You can change the size and possition of the window, the number of snowflakes and the wind.

Can be found at Aminet: game/gag/SnoOow.lha
Source code is to be found at my homepage.