Orpie v1.3 User Manual

Paul J. Pelzl

July 17, 2004

"Because the equals key is for the weak."

1 Introduction

Orpie is a console-based RPN (reverse polish notation) desktop calculator. The interface is similar to that of modern Hewlett-Packard^{TM} calculators, but has been optimized for efficiency on a PC keyboard. The design is also influenced to some degree by the Mutt email client[1] and the Vim editor[2].

Orpie does not have graphing capability, nor does it offer much in the way of a programming interface; other applications such as GNU Octave[3] are already very effective for such tasks. Orpie focuses specifically on helping you to crunch numbers quickly.

Orpie is written in Objective Caml (aka Ocaml)[4], a high-performance functional programming language with a whole lot of nice features. I highly recommend it.

2 Installation

This section describes how to install Orpie by compiling from source.

Before installing Orpie, you should have installed the GNU Scientific Library (GSL)[5] version 1.4 or greater. You will also need a curses library (e.g. ncurses[6]), which is almost certainly already installed on your system. Finally, Ocaml 3.07 or higher is required to compile the sources. You will need the Nums library that is distributed with Ocaml; at least on Debian, Nums is available as separate packages `libnums-ocaml` and `libnums-ocaml-dev`.

I will assume you have received this program in the form of a source tarball, e.g. "`orpie-x.x.tar.gz`". You have undoubtedly extracted this archive already (e.g. using "`tar xvzf orpie-x.x.tar.gz`"). Enter the root of the Orpie installation directory, e.g. "`cd orpie-x.x`". You can compile the sources with the following sequence:

```
$ ./configure
$ make
```

[1] http://www.mutt.org
[2] http://vim.sf.net
[3] http://www.octave.org
[4] http://caml.inria.fr/
[5] http://sources.redhat.com/gsl/
[6] http://www.gnu.org/software/ncurses/ncurses.html

Finally, run "`make install`" (as root) to install the executables. "`configure`" accepts a number of parameters that you can learn about with "`./configure --help`". Perhaps the most common of these is the `--prefix` option, which lets you install to a non-standard directory[7].

3 Quick Start

This section describes how to use Orpie in its default configuration. After familiarizing yourself with the basic operations as outlined in this section, you may wish to consult Section 4 to see how Orpie can be configured to better fit your needs.

3.1 Overview

You can start the calculator by executing `orpie`. The interface has two panels. The left panel combines status information with context-sensitive help; the right panel represents the calculator's stack. (Note that the left panel will be hidden if Orpie is run in a terminal with less than 80 columns.)

In general, you perform calculations by first entering data on to the stack, then executing functions that operate on the stack data. As an example, you can hit `1<enter>2<enter>+` in order to add 1 and 2.

3.2 Entering Data

3.2.1 Entering Real Numbers

To enter a real number, just type the desired digits and hit enter. The space bar will begin entry of a scientific notation exponent. The '`n`' key is used for negation. Here are some examples:

Keypresses	Resulting Entry
`1.23<enter>`	`1.23`
`1.23<space>23n<enter>`	`1.23e-23`
`1.23n<space>23<enter>`	`-1.23e23`

3.2.2 Entering Complex Numbers

Orpie can represent complex numbers using either cartesian (rectangular) or polar coordinates. See Section 3.5 to see how to change the complex number display mode.

A complex number is entered by first pressing '`(`', then entering the real part, then pressing '`,`' followed by the imaginary part. Alternatively, you can press '`(`' followed by the magnitude, then '`<`' followed by the phase angle. The angle will be interpreted in degrees or radians, depending on the current setting of the angle mode (see Section 3.5). Examples:

Keypresses	Resulting Entry
`(1.23, 4.56<enter>`	`(1.23, 4.56)`
`(0.7072<45<enter>`	`(0.500065915655126, 0.50006591...`
`(1.23n,4.56<space>10<enter>`	`(-1.23, 45600000000)`

[7] The default installation prefix is `/usr/local`.

3.2.3 Entering Matrices

You can enter matrices by pressing '[]'. The elements of the matrix may then be entered as described in the previous sections, and should be separated using ',]'. To start a new row of the matrix, press '[]' again. On the stack, each row of the matrix is enclosed in a set of brackets; for example, the matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

would appear on the stack as [[1, 2][3, 4]].

Examples of matrix entry:

Keypresses	Resulting Entry
[1,2[3,4<enter>	[[1, 2][3, 4]]
[1.2<space>10,0[3n,5n<enter>	[[12000000000, 0][-3, -5]]
[(1,2,3,4[5,6,7,8<enter>	[[(1, 2), (3, 4)][(5, 6), (...

3.2.4 Entering Integer Constants

An exact integer may be entered by pressing '#' followed by the desired digits. The base of the integer will be assumed to be the same as the current calculator base mode (see Section 3.5 to see how to set this mode). Alternatively, the desired base may be specified by pressing space and appending one of {b, o, d, h}, to represent binary, octal, decimal, or hexadecimal, respectively. On the stack, the representation of the integer will be changed to match the current base mode. Examples:

Keypresses	Resulting Entry
#123456<enter>	# 123456 d
#ffff<space>h<enter>	# 65535 d
#10101n<space>b<enter>	# -21 d

Note that integer constants may have unlimited length, and the basic arithmetic operations (addition, subtraction, multiplication, division) will be performed using exact arithmetic when both arguments are integers.

3.2.5 Entering Variable Names

A variable name may be entered by pressing '@' followed by the desired variable name string. The string may contain alphanumeric characters, dashes, and underscores. Example:

Keypresses	Resulting Entry
@myvar	@ myvar

Orpie also supports autocompletion of variable names. The help panel displays a list of pre-existing variables that partially match the name currently being entered. You can press '<tab>' to iterate through the list of matching variables.

As a shortcut, keys <f1>-<f12> will enter the variables ("registers") @ r01 through @ r12.

3.2.6 Entering Data With an External Editor

Orpie can also parse input entered via an external editor. You may find this to be a convenient method for entering large matrices. Pressing 'E' will launch the external editor, and the various data types may be entered as illustrated by the examples below:

Data Type	Sample Input String
integer constant	#12345678_d, where the trailing letter is one of the base characters {b, o, d, h}
real number	-123.45e67
complex number	(1e10, 2) or (1 <90)
real matrix	[[1, 2][3.1, 4.5e10]]
complex matrix	[[(1, 0), 5][1e10, (2 <90)]]
variable	@myvar

Notice that the complex matrix input parser is quite flexible; real and complex matrix elements may be mixed, and cartesian and polar complex formats may be mixed as well.

Multiple stack entries may be specified in the same file, if they are separated by whitespace. For example, entering (1, 2) 1.5 into the editor will cause the complex value (1, 2) to be placed on the stack, followed by the real value 1.5.

The input parser will discard whitespace where possible, so feel free to add any form of whitespace between matrix rows, matrix elements, real and complex components, etc.

3.3 Performing Basic Function Operations

Once some data has been entered on the stack, you can apply operations to that data. For example, '+' will add the last two elements on the stack. By default, the following keys have been bound to such operations:

Keys	Operations
+	add last two stack elements
-	subtract element 1 from element 2
*	multiply last two stack elements
/	divide element 2 by element 1
^	raise element 2 to the power of element 1
n	negate last element
i	invert last element
s	square root function
a	absolute value function
e	exponential function
l	natural logarithm function
c	complex conjugate function
!	factorial function
%	element 2 mod element 1
S	store element 2 in (variable) element 1
;	evaluate variable to obtain contents

As a shortcut, function operators will automatically enter any data that you were in the process of entering. So instead of the sequence 2<enter>2<enter>+, you could type simply 2<enter>2+ and the second number would be entered before the addition operation is applied.

As an additional shortcut, any variable names used as function arguments will be evaluated before application of the function. In other words, it is not necessary to evaluate variables before performing arithmetic operations on them.

3.4 Performing Extended Function Operations

One could bind nearly all calculator operations to specific keypresses, but this would rapidly get confusing since the PC keyboard is not labeled as nicely as a calculator keyboard is. For this reason, Orpie includes an *extended command* syntax.

To activate an extended command, press `''` (quote key), followed by the first few letters/digits of the command, then hit enter. Orpie offers an autocompletion feature for extended commands, so you only need to type enough of the command to identify it uniquely. The matching extended commands will appear in the left panel of the display, to assist you in finding the appropriate command.

To avoid interface conflicts, extended commands may be entered only when the entry buffer (the bottom line of the screen) is empty.

The following functions are available as extended commands:

Extended Commands	Functions
`inv`	inverse function
`pow`	raise element 2 to the power of element 1
`sq`	square last element
`sqrt`	square root function
`abs`	absolute value function
`exp`	exponential function
`ln`	natural logarithm function
`10^`	base 10 exponential function
`log10`	base 10 logarithm function
`conj`	complex conjugate function
`sin`	sine function
`cos`	cosine function
`tan`	tangent function
`sinh`	hyperbolic sine function
`cosh`	hyperbolic cosine function
`tanh`	hyperbolic tangent function
`asin`	arcsine function
`acos`	arccosine function
`atan`	arctangent function
`asinh`	inverse hyperbolic sine function
`acosh`	inverse hyperbolic cosine function
`atanh`	inverse hyperbolic tangent function
`re`	real part of complex number
`im`	imaginary part of complex number
`gamma`	Euler gamma function
`lngamma`	natural log of Euler gamma function
`erf`	error function
`erfc`	complementary error function
`fact`	factorial function
`gcd`	greatest common divisor function
`lcm`	least common multiple function
`binom`	binomial coefficient function
`perm`	permutation function

Extended Commands (con't)	Functions
`trans`	matrix transpose
`solvelin`	solve a linear system of the form Ax = b
`mod`	element 2 mod element 1
`floor`	floor function
`ceil`	ceiling function
`toint`	convert a real number to an integer type
`toreal`	convert an integer type to a real number
`add`	add last two elements
`sub`	subtract element 1 from element 2
`mult`	multiply last two elements
`div`	divide element 2 by element 1
`neg`	negate last element
`store`	store element 2 in (variable) element 1
`eval`	evaluate variable to obtain contents
`purge`	delete a variable
`total`	sum the columns of a real matrix
`mean`	compute the sample means of the columns of a real matrix
`sumsq`	sum the squares of the columns of a real matrix
`var`	compute the unbiased sample variances of the columns of a real matrix
`varbias`	compute the biased (population) sample variances of the columns of a real matrix
`stdev`	compute the unbiased sample standard deviations of the columns of a real matrix
`stdevbias`	compute the biased (pop.) sample standard deviations of the columns of a matrix
`min`	find the minima of the columns of a real matrix
`max`	find the maxima of the columns of a real matrix
`utpn`	compute the upper tail probability of a normal distribution
`rand`	generate a random number between 0 and 1 (uniformly distributed)

3.5 Performing Basic Command Operations

In addition to the function operations listed in Section 3.3, a number of basic calculator commands have been bound to single keypresses:

Keys	Operations
\	drop last element
|	clear all stack elements
<pagedown>	swap last two elements
<enter>	duplicate last element (when entry buffer is empty)
u	undo last operation
r	toggle angle mode between degrees and radians
p	toggle complex display mode between rectangular and polar
b	cycle base display mode between binary, octal, decimal, hex
v	view last stack element in a fullscreen editor
E	create a new stack element using an external editor
P	enter π on the stack
C-L	refresh the display
<up>	begin stack browsing mode
Q	quit Orpie

3.6 Performing Extended Command Operations

In addition to the function operations listed in Section 3.4, there are a large number of calculator commands that have been implemented using the extended command syntax:

Extended Commands	Calculator Operation
drop	drop last element
clear	clear all stack elements
swap	swap last two elements
dup	duplicate last element
undo	undo last operation
rad	set angle mode to radians
deg	set angle mode to degrees
rect	set complex display mode to rectangular
polar	set complex display mode to polar
bin	set base display mode to binary
oct	set base display mode to octal
dec	set base display mode to decimal
hex	set base display mode to hexidecimal
view	view last stack element in a fullscreen editor
edit	create a new stack element using an external editor
pi	enter π on the stack
refresh	refresh the display
about	display a nifty “About Orpie” screen
quit	quit Orpie

3.7 Browsing the Stack

Orpie offers a *stack browsing mode* to assist in viewing and manipulating stack data. Press `<up>` to enter stack browsing mode; this should highlight the last stack element. You can use the up and down arrow keys to select different stack elements. The following keys are useful in stack browsing mode:

Keys	Operations
`q`	quit stack browsing mode
`<left>`	scroll selected entry to the left
`<right>`	scroll selected entry to the right
`r`	cyclically "roll" stack elements downward, below the selected element (inclusive)
`R`	cyclically "roll" stack elements upward, below the selected element (inclusive)
`v`	view the currently selected element in a fullscreen editor
`E`	edit the currently selected element with an external editor
`<enter>`	duplicate the currently selected element

The left and right scrolling option may prove useful for viewing very lengthy stack entries, such as large matrices. The edit option provides a convenient way to correct data after it has been entered on the stack.

4 Advanced Configuration

Orpie reads a run-configuration textfile (generally `/etc/orpierc` or `/usr/local/etc/orpierc`) to determine key and command bindings. You can create a personalized configuration file in `$HOME/.orpierc`, and select bindings that match your usage patterns. The recommended procedure is to copy the `orpierc` file provided with Orpie, and edit it to your needs.

4.1 `orpierc` Syntax

You may notice that the `orpierc` syntax is similar to the syntax used in the configuration file for the Mutt email client (muttrc).

Within the `orpierc` file, strings should be enclosed in double quotes (`"`). A double quote character inside a string may be represented by `\"` . The backslash character must be represented by doubling it (`\\`).

4.1.1 Setting Configuration Variables

Syntax: `set` *`variable=value_string`*

Several configuration variables can be set using this syntax; check Section 4.2 to see a list. The variables are unquoted, but the values should be quoted strings.

4.1.2 Creating Key Bindings

Syntax: `bind` *`key_identifier operation`*

This command will bind a keypress to execute a calculator operation. The various operations, which should not be enclosed in quotes, may be found in Section 4.3. Key identifiers may be specified by strings that

represent a single keypress, for example `"m"` (quotes included). The key may be prefixed with `"\\C"` or `"\\M"` to represent Control or Meta (Alt) modifiers, respectively; note that the backslash must be doubled. A number of special keys lack single-character representations, so the following strings may be used to represent them:

- `"<esc>"`
- `"<tab>"`
- `"<enter>"`
- `"<return>"`
- `"<insert>"`
- `"<home>"`
- `"<end>"`
- `"<pageup>"`
- `"<pagedown>"`
- `"<space>"`
- `"<left>"`
- `"<right>"`
- `"<up>"`
- `"<down>"`
- `"<f1>"` to `"<f12>"`

Due to differences between various terminal emulators, this key identifier syntax may not be adequate to describe every keypress. As a workaround, Orpie will also accept key identifiers in octal notation. As an example, you could use `\024` (do *not* enclose it in quotes) to represent Ctrl-T.

Orpie includes a secondary executable, `orpie-curses-keys`, that prints out the key identifiers associated with keypresses. You may find it useful when customizing `orpierc`.

Multiple keys may be bound to the same operation, if desired.

4.1.3 Creating Extended Command Abbreviations

Syntax: `abbrev` *`command_abbreviation operation`*

You can use this syntax to set the extended command abbreviations used within Orpie to represent the various operations. A list of available operations may be found in Section 4.3. The command abbreviations should be quoted strings, for example `"sin"` or `"log"`.

Orpie performs autocompletion on these command abbreviations, allowing you to type usually just a few letters in order to select the desired command. The order of the autocompletion matches will be the same as the order in which the abbreviations are registered by the rcfile–so you may wish to place the more commonly used command abbreviations earlier in the list.

Multiple command abbreviations may be bound to the same operation, if desired.

4.1.4 Creating Macros

Syntax: `macro` *`key_identifier macro_string`*

You can use this syntax to cause a single keypress (the *key_identifier*) to be interpreted as the series of keypresses listed in *macro_string*. The syntax for defining a keypress is the same as that defined in Section 4.1.2. The macro string should be a list of whitespace-separated keypresses, e.g. `"2 <return> 2 +"` (including quotes).

This macro syntax provides a way to create small programs; by way of example, the default orpierc file includes macros for the base 2 logarithm and the binary entropy function (bound to `L` and `H`, respectively), as well as "register" variable shortcuts (`<f1>` to `<f12>`).

Macros may call other macros recursively. However, take care that a macro does not call *itself* recursively; Orpie will not trap the infinite loop.

Note that extended commands may be accessed within macros. For example, `macro "A" "' a b o u t <return>"` would bind `A` to display the "about Orpie" screen.

4.2 Configuration Variables

The following configuration variables may be set as described in Section 4.1.1:

- `datadir`
  This variable should be set to the full path of the Orpie data directory, which will contain the calculator state save file, temporary buffers, etc. The default directory is `"~/.orpie/"`.

- `editor`
  This variable may be set to the fullscreen editor of your choice. The default value is `"vi"`. It is recommended that you choose an editor that offers horizontal scrolling in place of word wrapping, so that the columns of large matrices can be properly aligned. (The Vim editor could be used in this fashion by setting `editor` to `"vim -c 'set nowrap'"`.)

- `hide_help`
  Set this variable to `"true"` to hide the left help/status panel, or leave it on the default of `"false"` to display the help panel.

- `conserve_memory`
  Set this variable to `"true"` to minimize memory usage, or leave it on the default of `"false"` to improve rendering performance. (By default, Orpie caches multiple string representations of all stack elements. Very large integers in particular require significant computation for string representation, so caching these strings can make display updates much faster.)

4.3 Calculator Operations

Every calculator operation can be made available to the interface using the syntax described in Sections 4.1.2 and 4.1.3. The following is a list of every available operation.

4.3.1 Functions

The following operations are functions–that is, they will consume at least one argument from the stack. Orpie will generally abort the computation and provide an informative error message if a function cannot be successfully applied (for example, if you try to compute the transpose of something that is not a matrix).

For the integer constant data type, basic arithmetic operations will yield an exact integer constant result. Division of two integer constants will yield the quotient of the division. The more complicated functions will generally promote the integer constant to a real number, and as such the arithmetic will no longer be exact.

- `function_10_x`
  Raise 10 to the power of the last stack element (inverse of function_log10).
- `function_abs`
  Compute the absolute value of the last stack element.
- `function_acos`
  Compute the inverse cosine of the last stack element. For real numbers, The result will be provided either in degrees or radians, depending on the angle mode of the calculator.
- `function_acosh`
  Compute the inverse hyperbolic cosine of the last stack element.
- `function_add`
  Add last two stack elements.
- `function_arg`
  Compute the argument (phase angle of complex number) of the last stack element. The value will be provided in either degrees or radians, depending on the current angle mode of the calculator.
- `function_asin`
  Compute the inverse sine of the last stack element. For real numbers, The result will be provided either in degrees or radians, depending on the angle mode of the calculator.
- `function_asinh`
  Compute the inverse hyperbolic sine of the last stack element.
- `function_atan`
  Compute the inverse tangent of the last stack element. For real numbers, The result will be provided either in degrees or radians, depending on the angle mode of the calculator.
- `function_atanh`
  Compute the inverse hyperbolic tangent of the last stack element.
- `function_binomial_coeff`
  Compute the binomial coefficient (“n choose k”) formed by the last two stack elements. If these arguments are real, the coefficient is computed using a fast approximation to the log of the gamma function, and therefore the result is subject to rounding errors. For integer constant arguments, the coefficient is computed using exact arithmetic; this has the potential to be a slow operation.

- `function_ceiling`
  Compute the ceiling of the last stack element.
- `function_cos`
  Compute the cosine of the last stack element. If the argument is real, it will be assumed to be either degrees or radians, depending on the angle mode of the calculator.
- `function_cosh`
  Compute the hyperbolic cosine of the last stack element.
- `function_conj`
  Compute the complex conjugate of the last stack element.
- `function_div`
  Divide element 2 by element 1.
- `function_erf`
  Compute the error function of the last stack element.
- `function_erfc`
  Compute the complementary error function of the last stack element.
- `function_eval`
  Obtain the contents of the variable in the last stack position.
- `function_exp`
  Evaluate the exponential function of the last stack element.
- `function_factorial`
  Compute the factorial of the last stack element. For a real argument, this is computed using a fast approximation to the gamma function, and therefore the result may be subject to rounding errors (or overflow). For an integer constant argument, the factorial is computed using exact arithmetic; this has the potential to be a slow operation.
- `function_floor`
  Compute the floor of the last stack element.
- `function_gamma`
  Compute the Euler gamma function of the last stack element.
- `function_gcd`
  Compute the greatest common divisor of the last two stack elements. This operation may be applied only to integer type data.
- `function_im`
  Compute the imaginary part of the last stack element.
- `function_inv`
  Compute the multiplicative inverse of the last stack element.

- `function_lcm`
  Compute the least common multiple of the last two stack elements. This operation may be applied only to integer type data.
- `function_ln`
  Compute the natural logarithm of the last stack element.
- `function_lngamma`
  Compute the natural logarithm of the Euler gamma function of the last stack element.
- `function_log10`
  Compute the base-10 logarithm of the last stack element.
- `function_maximum`
  Find the maximum values of each of the columns of a real NxM matrix, returning a 1xM matrix as a result.
- `function_minimum`
  Find the minimum values of each of the columns of a real NxM matrix, returning a 1xM matrix as a result.
- `function_mean`
  Compute the sample means of each of the columns of a real NxM matrix, returning a 1xM matrix as a result.
- `function_mod`
  Compute element 2 mod element 1. This operation can be applied only to integer type data.
- `function_mult`
  Multiply last two stack elements.
- `function_neg`
  Negate last stack element.
- `function_permutation`
  Compute the permutation coefficient determined by the last two stack elements 'n' and 'k': the number of ways of obtaining an ordered subset of k elements from a set of n elements. If these arguments are real, the coefficient is computed using a fast approximation to the log of the gamma function, and therefore the result is subject to rounding errors. For integer constant arguments, the coefficient is computed using exact arithmetic; this has the potential to be a slow operation.
- `function_pow`
  Raise element 2 to the power of element 1.
- `function_purge`
  Delete the variable in the last stack position.
- `function_rand`
  Generate a random real-valued number between 0 (inclusive) and 1 (exclusive). The deviates are uniformly distributed.

- `function_re`
  Compute the real part of the last stack element.
- `function_sin`
  Compute the sine of the last stack element. If the argument is real, it will be assumed to be either degrees or radians, depending on the angle mode of the calculator.
- `function_sinh`
  Compute the hyperbolic sine of the last stack element.
- `function_solve_linear`
  Solve a linear system of the form Ax = b, where A and b are the last two elements on the stack. A must be a square matrix and b must be a matrix with one column. This function does not compute inv(A), but obtains the solution by a more efficient LU decomposition method. This function is recommended over explicitly computing the inverse, especially when solving linear systems with relatively large dimension or with poorly conditioned matrices.
- `function_sq`
  Square the last stack element.
- `function_sqrt`
  Compute the square root of the last stack element.
- `function_stdev_unbiased`
  Compute the unbiased sample standard deviation of each of the columns of a real NxM matrix, returning a 1xM matrix as a result. (Compare to HP48's `sdev` function.)
- `function_stdev_biased`
  Compute the biased (population) sample standard deviation of each of the columns of a real NxM matrix, returning a 1xM matrix as a result. (Compare to HP48's `psdev` function.)
- `function_store`
  Store element 2 in (variable) element 1.
- `function_sub`
  Subtract element 1 from element 2.
- `function_sumsq`
  Sum the squares of each of the columns of a real NxM matrix, returning a 1xM matrix as a result.
- `function_tan`
  Compute the tangent of the last stack element. If the argument is real, it will be assumed to be either degrees or radians, depending on the angle mode of the calculator.
- `function_tanh`
  Compute the hyperbolic tangent of the last stack element.
- `function_to_int`
  Convert a real number to an integer type.

- `function_to_real`
  Convert an integer type to a real number.
- `function_total`
  Sum each of the columns of a real NxM matrix, returning a 1xM matrix as a result.
- `function_transpose`
  Compute the matrix transpose of the last stack element.
- `function_utpn`
  Compute the upper tail probability of a normal distribution.
  $utpn(m, v, x) = \int_x^\infty \frac{1}{\sqrt{2\pi v}} \exp\left(-\frac{(m-y)^2}{2v}\right)\, dy$
- `function_var_unbiased`
  Compute the unbiased sample variance of each of the columns of a real NxM matrix, returning a 1xM matrix as a result. (Compare to HP48's `var` function.)
- `function_var_biased`
  Compute the biased (population) sample variance of each of the columns of a real NxM matrix, returning a 1xM matrix as a result. (Compare to HP48's `pvar` function.)

4.3.2 Commands

The following operations are referred to as commands; they differ from functions because they do not take an argument. Many calculator interface settings are implemented as commands.

- `command_about`
  Display a nifty "about Orpie" credits screen.
- `command_begin_browsing`
  Enter stack browsing mode.
- `command_begin_extended`
  Begin entry of an extended command abbreviation.
- `command_begin_variable`
  Begin entry of a variable name.
- `command_bin`
  Set the base of integer constant representation to 2 (binary).
- `command_clear`
  Clear all elements from the stack.
- `command_cycle_base`
  Cycle the base of integer constant representation between 2, 8, 10, and 16 (bin, oct, dec, and hex).
- `command_dec`
  Set the base of integer constant representation to 10 (decimal).

- `command_deg`
  Set the angle mode to degrees.
- `command_drop`
  Drop the last element off the stack.
- `command_dup`
  Duplicate the last stack element.
- `command_enter_pi`
  Enter π on the stack.
- `command_hex`
  Set the base of integer constant representation to 16 (hexadecimal).
- `command_oct`
  Set the base of integer constant representation to 8 (octal).
- `command_polar`
  Set the complex display mode to polar.
- `command_rad`
  Set the angle mode to radians.
- `command_rect`
  Set the complex display mode to rectangular (cartesian).
- `command_refresh`
  Refresh the display.
- `command_swap`
  Swap stack elements 1 and 2.
- `command_quit`
  Quit Orpie.
- `command_toggle_angle_mode`
  Toggle the angle mode between degrees and radians.
- `command_toggle_complex_mode`
  Toggle the complex display mode between rectangular and polar.
- `command_undo`
  Undo the last calculator operation.
- `command_view`
  View the last stack element in an external fullscreen editor.
- `command_edit_input`
  Create a new stack element using an external editor.

4.3.3 Edit Operations

The following operations are related to editing during data entry. These commands cannot be made available as extended commands, since extended commands are not accessible while entering data. These operations should be made available as single keypresses using the `bind` keyword.

- `edit_angle`
  Begin entering the phase angle of a complex number. (Orpie will assume the angle is in either degrees or radians, depending on the current angle mode.)
- `edit_backspace`
  Delete the last character entered.
- `edit_begin_integer`
  Begin entering an integer constant.
- `edit_complex`
  Begin entering a complex number.
- `edit_enter`
  Enter the data that is currently being edited.
- `edit_matrix`
  Begin entering a matrix, or begin entering the next row of a matrix.
- `edit_minus`
  Enter a minus sign in input.
- `edit_scientific_notation_base`
  Begin entering the scientific notation exponent of a real number, or the base of an integer constant.
- `edit_separator`
  Begin editing the next element of a complex number or matrix. (This will insert a comma between elements.)

4.3.4 Browsing Operations

The following list of operations is available only in stack browsing mode. As extended commands are unavailable while browsing the stack, these operations should be bound to single keypresses using the `bind` keyword.

- `browse_echo`
  Echo the currently selected element to stack level 1.
- `browse_end`
  Exit stack browsing mode.
- `browse_drop`
  Drop the currently selected stack element.

- `browse_dropn`
  Drop all stack elements below the current selection (inclusive).
- `browse_keep`
  Drop all stack elements *except* the current selection. (This is complementary to `browse_drop`.
- `browse_keepn`
  Drop all stack elements above the current selection (non-inclusive). (This is complementary to `browse_dropn`.
- `browse_next_line`
  Move the selection cursor down one line.
- `browse_prev_line`
  Move the selection cursor up one line.
- `browse_rolldown`
  Cyclically "roll" stack elements downward, below the selected element (inclusive).
- `browse_rollup`
  Cyclically "roll" stack elements upward, below the selected element (inclusive) .
- `browse_scroll_left`
  Scroll the selected element to the left (for viewing very large entries such as matrices).
- `browse_scroll_right`
  Scroll the selected element to the right.
- `browse_view`
  View the currently selected stack element in a fullscreen editor.
- `browse_edit`
  Edit the currently selected stack element using an external editor.

4.3.5 Extended Entry Operations

The following list of operations is available only while entering an extended function or command. These operations must be bound to single keypresses using the `bind` keyword.

- `extended_backspace`
  Delete a character from the extended entry string.
- `extended_enter`
  Execute the selected extended function or command.
- `extended_exit`
  Cancel extended entry.

4.3.6 Variable Entry Operations

The following list of operations is available only while entering a variable name. As extended commands are unavailable while entering variables, these operations should be bound to single keypresses using the `bind` keyword.

- `variable_backspace`
  Delete a character from the variable name.
- `variable_cancel`
  Cancel entry of the variable name.
- `variable_complete`
  Autocomplete the variable name.
- `variable_enter`
  Enter the variable name on the stack.

4.3.7 Integer Entry Operations

The following operation is available only while entering an integer; it can be made accessible by binding it to a single keypress using the `bind` keyword.

- `integer_cancel`
  Cancel entry of an integer.

5 Licensing

Orpie is Free Software; it has been made available under version 2 of the GNU General Public License (GPL). You should have received a copy of the GPL along with this program, in the file "COPYING".

6 Credits

Orpie includes portions of the ocamlgsl[8] bindings supplied by Olivier Andrieu, as well as the curses bindings from the Ocaml Text Mode Kit[9] written by Nicolas George. I would like to thank these authors for helping to make Orpie possible.

7 (Not So) Frequently Asked Questions

1. **Whatever happened to `rpc`?**

   rpc, of course, would be the predecessor to Orpie. It was written in C++.

   Over the years I have grown increasingly disenchanted with C++, and as a result I lost interest in maintaining rpc. When I settled on Ocaml as a replacement language, I began working on Orpie as a way to improve my Ocaml abilities.

[8] http://oandrieu.nerim.net/ocaml/gsl/
[9] http://www.nongnu.org/ocaml-tmk/

2. **What's wrong with C++? And what's this Ocaml thing?**

   <rant>
   C++ has had so many misfeatures bolted on that no one can possibly hold the entire syntax in his head at one time. The syntax is not only prohibitively vast, but also horribly ugly–especially when dealing with templates. The ability to use pointers presents many opportunities to shoot oneself in the foot. Garbage collection and exceptions exist as afterthoughts. In short, there are few compelling reasons to consider C++ for any project that does not require low-level hardware access.
   </rant>

   I found Ocaml after searching for a Better Language that offers good performance. Ocaml is a functional programming language with a pretty syntax, clean design, and all sorts of nice features that one really should expect from a modern programming language. It supports functional, imperative, and object-oriented programming paradigms, so it is well-suited to a broad class of programming tasks. The type-checking compiler is exceedingly strict, and I find that this improves my overall productivity by catching a lot of bugs prior to runtime. Ocaml also interfaces with C quite easily, and there are bindings available for many commonly used libraries.

3. **Does Orpie include any enhancements over the old `rpc`?**

   The biggest usability enhancement would be the rcfile for keybindings. The context-sensitive help panel is new, as is user-defined variable support. Integer data may be entered and manipulated with arbitrary precision (thanks to the Nums library that ships with Ocaml). The error messages provided by Orpie are, on average, more informative than those provided by rpc; this is a direct consequence of Ocaml's excellent exception handling support.

   I also believe that Orpie has fewer bugs in the input handler/parser, but this remains to be seen.

4. **When I use {+, -, *, /} on the numeric keypad under gnome-terminal, Orpie reads some letters instead of the expected symbols. Can I do anything about this?**

   I was able to fix the problem by executing

   ```
   $ export TERM=linux
   ```

   before launching Orpie. gnome-terminal sets `$TERM` to `xterm`, but apparently does not behave like a real xterm with respect to the numeric keypad.

5. **Orpie crashes with a segmentation fault on startup. What can I do?**

   Try deleting the calculator state file (probably `~/.orpie/calc_state` , unless you changed the `data_dir` configuration variable). Orpie makes use of Ocaml's marshalling feature, which is very convenient but doesn't respond well to corrupt data files.

8 Contact info

Orpie author: Paul Pelzl <pelzlpj@eecs.umich.edu>
Orpie website: http://www.eecs.umich.edu/~pelzlpj/orpie

Feel free to contact me if you have bugs, feature requests, patches, etc. I would also welcome volunteers interested in packaging Orpie for various platforms.

Orpie is developed with the aid of the excellent GNU Arch RCS[10]. Interested developers are advised to track Orpie development via my public archive:

```
pelzlpj@eecs.umich.edu--2004 \
        http://www-personal.engin.umich.edu/~pelzlpj/tla/2004
```

Do you feel compelled to compensate me for writing Orpie? As a *poor, starving* graduate student, I will gratefully accept donations. Please see `http://www.eecs.umich.edu/~pelzlpj/orpie/donate.html` for more information.

[10] http://www.gnu.org/software/gnu-arch/