REC/C for Complex Arithmetic

Harold V. McIntosh
Departamento de Aplicación de Microcomputadoras,
Instituto de Ciencias, Universidad Autónoma de Puebla,
Apartado postal 461, 72000 Puebla, Puebla, México.

January 20, 2001

Abstract

`REC/C` is one of a series of specialized `REC` programs; in this case one which acts like a hand held graphing calculator for complex arithmetic.

Contents

1 Introduction

Three ingredients are required to understand this program. First and foremost, familiarity with complex arithmetic is required. Given that the design is accomplished by using REC, an understanding of that language is evidently required, even though REC is mostly used for executing formulas on a pushdown stack using reversed Polish notation. Finally, the program is implemented in Objective C running under the NeXTSTEP operating system, to take advantage of its excellent visual interface and other attractive features.

1.1 the theory of complex numbers

Although the virtues of complex numbers as things which can give you the square root -1 are mentioned early on in the educational process, very little seems to be done with them thereafter. If it weren't for complex impedances in electrical circuits, they might not even get much use at the college level.

Later on, quantum mechanics uses complex wave functions and mathematicians get to playing with Galois theory. But of course, one must certainly aspire to an algebraically complete field. Still, for the general public, the Mandelbrot set may provide the first actual contact with the intricacies of complex analysis, and then with a vengeance when it turns out that fixed points attract critical points all over the place, that intricate patterns repeat themselves over and over again, and yet there is a system to it all.

There are expositions of complex numbers at all levels, starting with Nahin's historical survey *An Imaginary Tale* [8] and including their application to geometry in Hahn's *Complex Numbers and Geometry* [7]. A good engineering exposition of complex numbers and their uses can be found in Guillemin's *The Mathematics of Circuit Analysis* [10]. Ahlfors' *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable* [5] may well be the most authoritative presentations on an advanced level, whereas Knopp's volumes on *Theory of Functions* [6] are both comprehensive and quite classical.

1.2 the programming language REC

The programming language `REC` was derived from `LISP` over a period of time, and found to be useful for minicomputers such as Digital Equipment's PDP-8, or early microprocessors such as the IMSAI or Polymorphic 88, given its extreme conciseness. That, in fact, is the reason why it is still preserved as an interface to programs written in other languages running with other operating systems.

`REC` itself is strictly a control structure, using parentheses to group sequences of instructions which are supposed to be executed in order — in other words, programs. The instructions

themselves, which are subdivided into operators and predicates, are not part of the control structure, but are always defined separately to create specialized variants on REC.

Besides the use of parentheses for aggrupation, the two punctuation marks, colon and semicolon, are used for repetition and termination, respectively; they were somewhat borrowed from musical notation. Spaces, tabs, all the `C` "white space," are used for cosmetic purposes and must be ignored.

There is also a mechanism for defining subroutines; definitions and their associated symbols alternate between a balanced pair of curly brackets, terminating in a main program which remains nameless. Definitions so introduced are valid only within their braces, permitting symbols to be reused over and over again on different levels and in different places. A subroutine is actually executed by prefacing its name with an "at sign," as in `@x`.

The distinction between operators and predicates is somewhat artificial, made mostly for convenience; a predicate which is always `true` is an operator. The values `true` and `false` of a predicate govern the sequence of operations, `true` meaning that the text of the program continues to be read in order, consecutively from left to right.

But `false` makes use of the punctuation, implying a skip to the next colon, semicolon, (or right parenthesis, in their absence) at the same parenthesis level. A colon means repetition starting back at the left parenthesis, a semicolon gives a `true` termination, realized by going forward to the closing right parenthesis, Arriving at a right parenthesis without the benefit of punctuation gives a `false` termination, but more generally inverts the action of predicates enclosed in such an interval.

This convention allows negating a predicate: `(p)` behaves oppositely from `p`. The other boolean combinations of predicates are easily written; "p and q" translates into `(pq;)`, "p or q" into `(p;q;)`. Moreover, all programs are predicates, even the main program, thereby reporting their results to the next higher program level, the one in which they occur as a subroutine.

`REC/C` uses about forty operators, of which only the counter `!n!` and some comparisons are predicates. They are more easily remembered if they are broken down into specific classes, such as the arithmatical operations, or the evaluation of some predefined functions. The central operating element of `REC/C` is a pushdown list onto which complex constants will be gathered, later to be evaluated, consulted, or replaced by the results of diverse calculations. The flow of data in and around the stack is shown in figure 7.

binary arithmetic operations :

+ - add top two numbers, leave only their sum

- - subtract top two numbers, leave their difference

* - multiply top two numbers, leave their product

/ - divide top two numbers, leave their quotient

& - exchange top two items on stack leaving both

unary arithmetic operations on top of stack :

j - complex conjugate

n - negative

f - multiply by real factor, as in $xx.x$ f

library of common functions :

C - hyperbolic cosine
E - complex exponential
F - fractional linear mapping (z+1)/(z-1)
L - complex logarithm
r - complex square root
T - hyperbolic tangent

push constants onto the stack :

X - push 1
Y - push i
Z - push 0
u - push 0.1
v - push 0.1 i
x - push 0.01
y - push 0.01 i

graphical display from the top of the stack :

G - move to first point of a drawn line
g - continue line to additional points
Q - draw a colored square (one argument)
q - draw a small colored square (2 args)
s - draw a small stereo square (2 args)
z - draw a small circle (one argument)

save and restore constants, adjust top of stack :

R - Rn to recover nth constant by pushing it
S - Sn to save nth constant leaving stack intact
P - push by duplicating the top of the stack
p - pop the stack by discarding its top element

REC background :

!n! - (!n! ... :;) repeat ... n times
$xx.x$ - floating point number

set parameter values :

M - graphic multiplier: 10.0 M, default
m - square size: 1.25 m, default
c - compression threshold (nominal 0.15)

predicates :

A - Is angle a multiple of 90 degrees?
I - Is real part an integer?
i - Is real part a multiple of 0.1?
lx - supervise recursive depth

There are several documents available explaining REC [1], including a read.me file in the rec subproject of each of the REC files.

1.3 the NeXTSTEP operating system

One of the principal advantages of the NeXTSTEP operating system is its use of the PostScript language for all input and output. *All* includes displaying information on the monitor, reading and writing files from disk, the expected task of printing files on a printer, and even includes transmitting them by modem. Amongst the agreeable conveniences is not having to struggle with a screen dump to save information which was originally presented visually.

The appearance which NeXTSTEP presents to the user is a *File Viewer* which is a window filled with either icons or text, as the user prefers. Moving a cursor by mouse (or sometimes keyboard arrows) indicates a choice of file, each one of which responds in its own manner to clicking or dragging. If it is a subdirectory, additional details are shown; if it is a text file, a page will be opened for editing, while programs will simply be loaded and executed.

Besides the *TextView* window, almost always overlaid as soon as programs begin to execute, there is room around the margins for icons reminding the user of programs and text files which have been held in abeyance, for frequently used programs, and the like. Some space is reserved in the upper left hand corner for the operational menu of the program being executed.

In addition to the mere operating system, which loads and executes programs and interchanges information between different locations, there is an extensive collection of service programs, ranging from language compilers to spelling checkers and even a copy of the Bible. Two of the more important utilities are the *Program Manager* and the *Interface Builder*.

The *Program Manager* mostly checks dates on files, to be sure that the source code for a program is not more recent than the object code. To do so, it maintains lists of all kinds of associated material such as header files, icon images, sounds, and what not. The *Interface Builder* permits interactive programming, in the sense that windows can be created, stocked with assorted artifacts, and have them interconnected, all through the suitable interpretation of mouse movements. Much programming can be accomplished in those terms, at least during the stages of initial layout.

Going beyond simple tinkering, one needs to use the language Objective C which is a mixture of C++ and SmallTalk. The latter uses *Methods*, which are statements with the syntax `[destination operation xxx :argument1 xxx :argument2 ...]`, where `xxx` denotes arbitrary intercalcated text, usually taking the form of helpful comments or comprising pieces of the body of a simple declaration in which the arguments are embedded. The `destination` is an *Object*, which is really a fancy name for a program to be executed[1]. SmallTalk envisioned parallel programming, in which several independent programs could communicate with one another by exchanging messages coordinating their activities.

A familiarity with Objective C will be necessary to follow the program listings which are about to be described. In the process it cannot be avoided noticing that NeXTSTEP already includes a vast number of predefined methods destined to manage the user interface. They are all located in a directory, together with supporting information, which can be consulted online, or selectively printed out for further enlightenment.

[1] *Objects* are actually programs packaged together with data and data structures, independecizing them from one another.

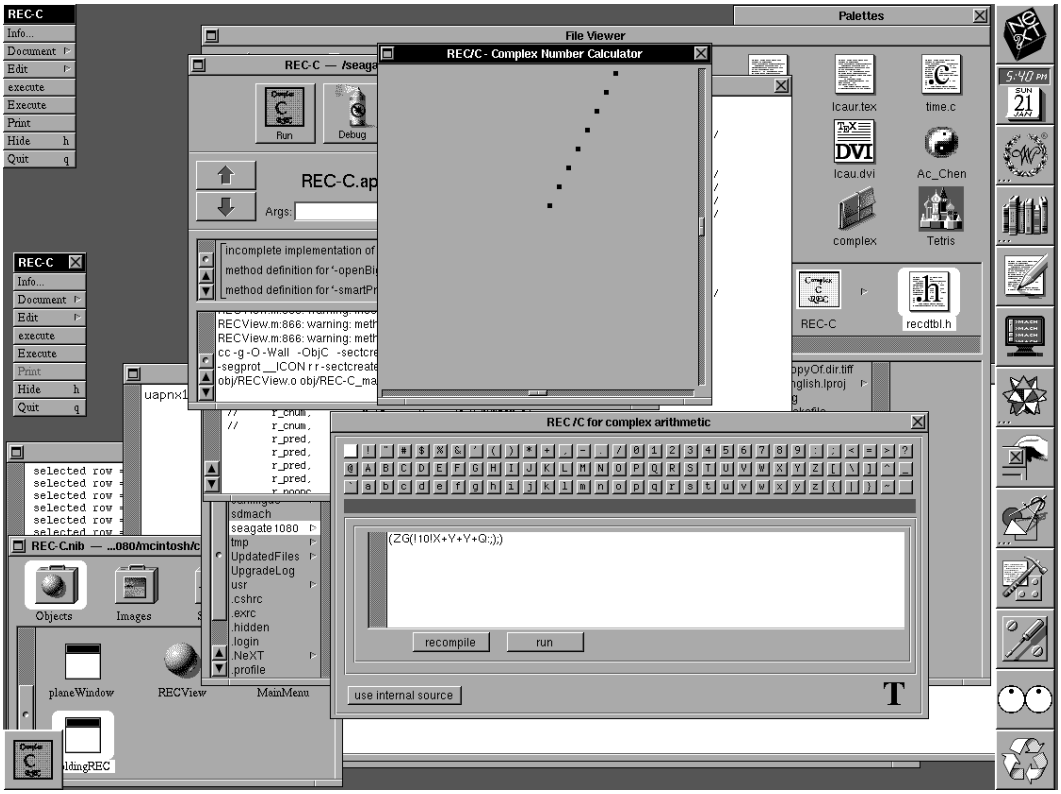


Figure 1: A REC/C program, running against a background consisting of the File Viewer and some other programs.

2 Program appearance

A user of `REC/C` will deal mostly with two windows, the main window and the auxiliary display window. There may be other panels and windows; for example the program menu makes its presence known every time the program is executed, and is so ubiquitous that it may not be even be counted amongst the program's windows. The main window of the Interface Builder, which customarily sits in the lower left hand corner of the screen as the interfaces are built, exhibits all the panels, windows, and accessible Objects, as shown for REC/C in Figure 2.

The main window, labelled "execute" in Figure 2, comes with two alternative forms, one of which was shown in Figure 4. The difference between them is that the browser panel can be exchanged for a text panel, into which programs can be read from disk storage, or composed on the spot. Traffic goes both ways, because the contents of the window can be transferred to the disk, either as a new file or as a correction to the existing file. The text handling variant is shown if Figure 3. The button at the bottom of the panel, displaying either the title "use external source," to load the text insert, or "load internal source" to load the browser insert,

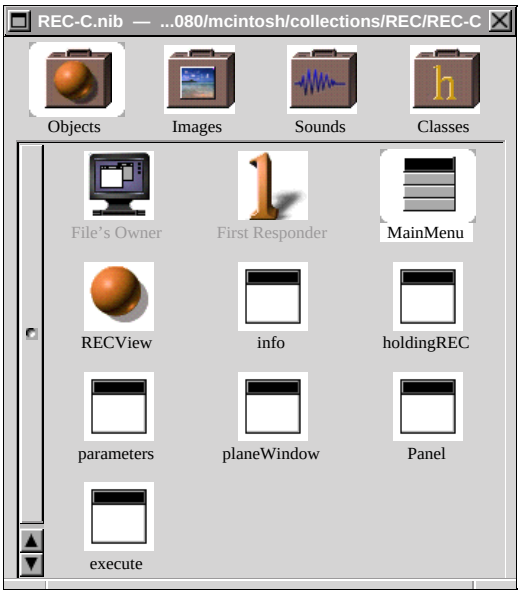


Figure 2: Main window for REC/C.nib, showing the ingredients of REC/C.

governs the transition between the two forms.

2.1 text windows

There are some standard procedures for incorporating text windows in a view of one's own, which are probably best obtained by copying them from a program which already has them, making appropriate changes in file names, storage arrays, and options. Lacking examples or desiring better information, the book of Garfinkel and Mahoney [3] can be consulted. The tricky aspect of the process is that files have to be opened on at least two levels, first in the MACH or C level, and then again on the NeXTSTEP level; correspondingly they must be closed in stages, running along in reverse order.

Any file manipulation should be connected to the item "Document" in the program's menu, which can be seen in Figure 6, so as to preserve the uniformity in appearance of NeXTSTEP programs.

All text fields are automatically connected to an editor. Besides performing the expected functions of inserting, deleting, and searching, they can share information with other programs through a buffer called the pasteboard. It communicates with editors running in other windows or even from different programs. Besides responding to the keyboard, editing can be accomplished by selecting options in the "Edit" item on the main menu, once having wiped the editing I-beam cursor over a portion of text.

If the "Services" item has been incorporated in a program menu (which has not been done in `REC/C`), it is possible to have still more elaborate communication between programs, passing selected text either as data or as program instructions.

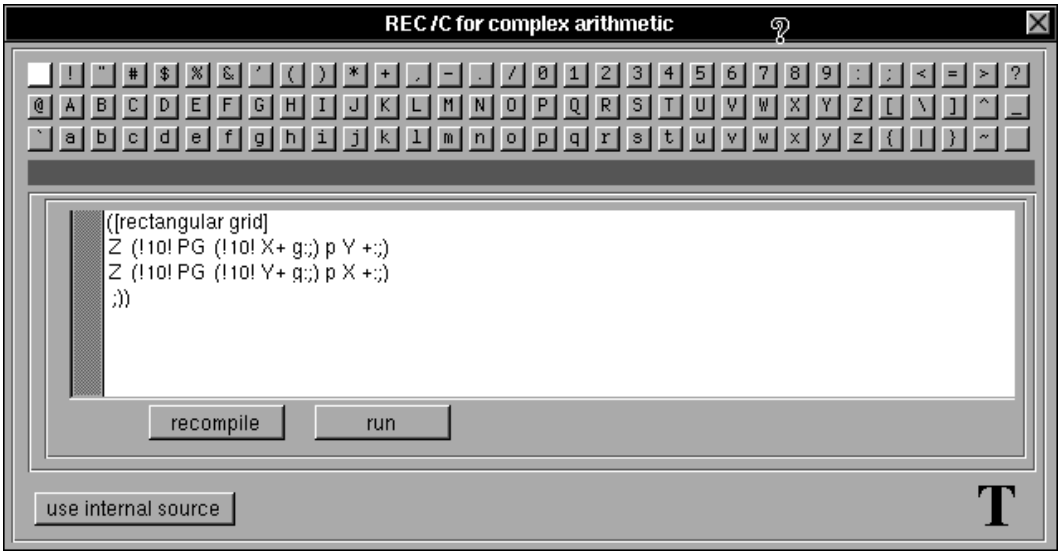


Figure 3: The REC/C main window, which contains the REC directory, a program definition area, and a viewer for the resulting construction.

Of course, there is no reason that a REC program cannot be placed in some file using the editor the regular way, and subsequently executed. The advantage in editing from the `REC` program itself is that the code can be tested immediately, and saved only after it is performing satisfactorily.

2.2 browsers

Browsers are another aspect of the operating system which can be incorporated in application programs at the programmer's desire; the procedure is much the same as for coupling text fields to files; namely to copy a browser which is already working, or to look it up in a textbook. The new browser program has to supply the text (or even icons) that the browser will display, which is accomplished by delegation. A delegate can be associated with an object by mouse dragging in the interface builder, or set up through programming.

A delegate is expected to provide methods which could have been incorporated in the delegating program, but for some reason, weren't. The principal reason is that there was no way of forseeing, when the master program was written, exactly what the methods were supposed to do. After all, each application will have its own items to display, and its own reactions to their selection; such information is best incorporated in the application program itself.

Looking around the REC/C main window, two other text fields can be seen. One, which is alwas present, is used for holding error messages and comments, such as displaying the definition of REC operators. Although there is no requirement within the language to do so, it is customary to use single letters (or ASCII characters) for the operators because of their

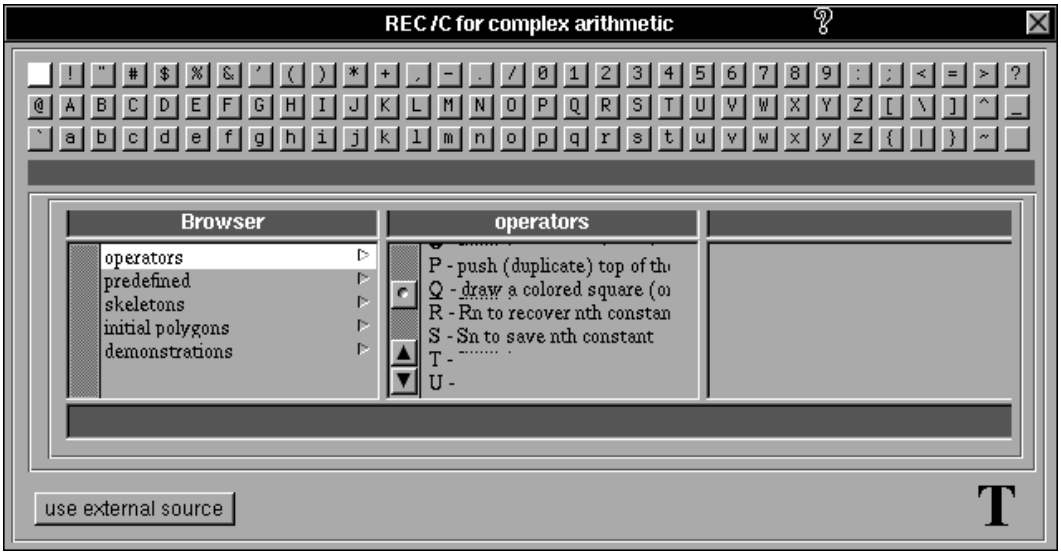


Figure 4: The REC/C main window, which contains the REC directory, a program definition area, and a viewer for the resulting construction.

convenience and ease of remembering. Of course, such a principle only works when there are enough letters to go around, which is one of the reason for tailoring individual REC programs instead of concocting a universal or general purpose REC.

The other text field accompanies the browser to hold REC programs after their selection by a double mouse click. They can be edited, and executed by a carriage return, but the field is not connected to the filing mechanism. Nevertheless, copying and pasting between the big text panel and the browser line is always possible, so the text field in the browser is not completely isolated.

The browser in the main window of REC/C is stocked with information of various types. The first item copies the header file which identifies the operators and predicates in REC/C, a variant of which has to be included in all REC programs. It supplements the button panel for REC definitions, because it is easier to scan a list of possibilities than to press the buttons one by one. On the other hand, for quick recollection or to identify an unfamiliar letter in a displayed program, the buttons are handier.

The remaining items carry either the coordinates of some frequently used basic polygons or instructions for developing particular strips. Since the counter only admits constants as its argument, some editing may be required to adapt a generic "`!n!`" to the given polygon.

In the first case, a double click on the selected polygon will set up the coordinates, in the second, the double click will move the REC program to the nearby text field for editing and execution.

2.3 the complex plane in its own window

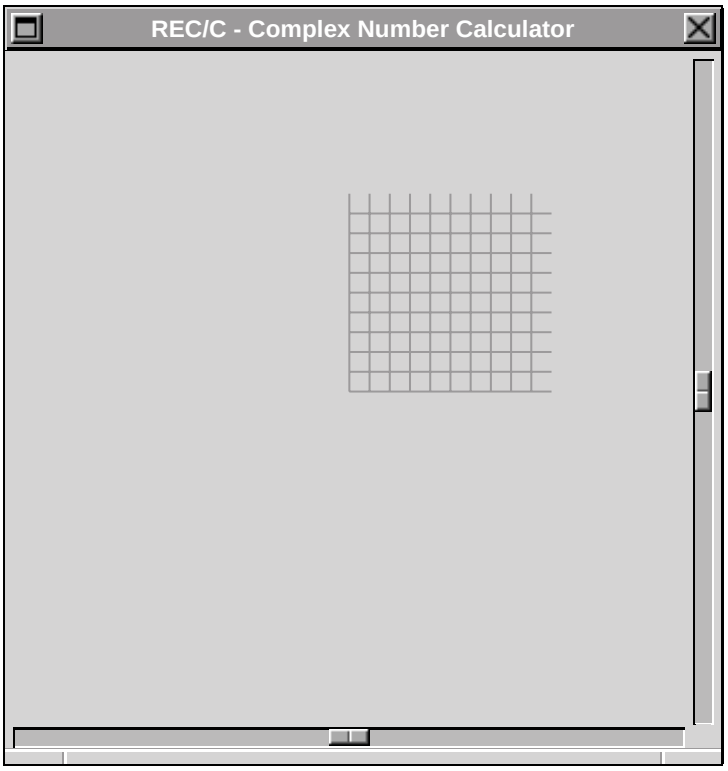


Figure 5: The auxiliary window can be enlarged or reduced by using the window sizer on its bottom margin. The two sliders position the image horizontally and vertically so that it can be kept on the page and centered.

Drawing pictures brings up a series of problems, not all of which are evident at first. Scale and centering are among the most obvious, easily compensated by including provisions for moving the figure and changing its size. Conflicts can still remain, because excessive compression may make the figure difficult to read, or because the final result has to be presented on a sheet of paper of given size. The next remedy is to divide a figure into panels, or use scrolling, at the price of increased programming complexity.

`REC/C` provides a separate window, distinct from the main, program editing, window, as shown in Figure 5. The image which it contains may be copied into the pasteboard by invoking the "Copy View" button on the editing menu, then transferred into Draw by pasting. The "Print" button on the main menu can also be used, supposing the drawing window is selected, to send the image directly to a printer. It is important to check the selection, because it is always the selected window which will be printed or copied.

2.4 the application menu

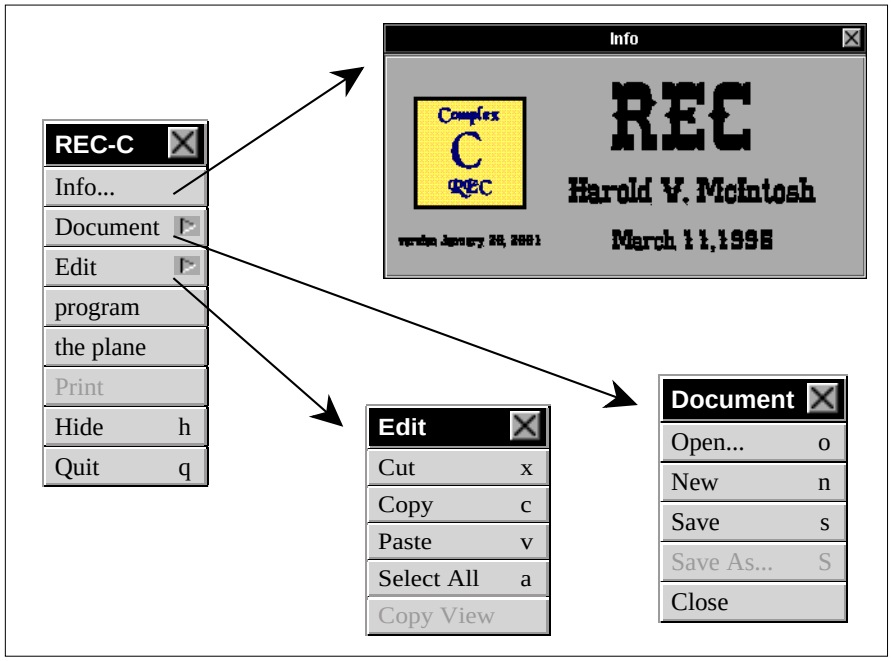


Figure 6: A program's menu will almost always begin with the items Info, Document, and Edit, and end with Hide and Quit. Print would be included as appropriate, usually near the end. Other items vary with the individual program.

Program execution in NeXTSTEP involves double clicking on an item in the FileViewer followed by use of the loaded program's main menu, by custom and tradition situated at the upper left hand corner of the monitor screen. It follows that designing a program would involve setting up a menu for the program, stocking it with commands, and writing (or drawing) the code to implement them. Figure 6 shows the menu for `REC/C` , with the submenus "Document" and "Edit" exposed for further inspection. The "Info" category is the place to insert the author's name and affiliation, possible copyright notices, and to the degree that seems appropriate or that good programming practice demands, descriptive information concerning the program and how to use it.

3 The rectbl.h header and the REC code

Besides other headers, such as `rec.h` which serves the `REC` compiler, and those germane to a particular application (such as the present `RECView.h`), one more should be provided which is devoted exclusively to defining the `REC` operators and predicates which are going to be used. Of course, the operators and predicates themselves must be defined, more likely in `C` than not; keeping them all together in one place facilitates program maintainance.

3.1 REC header

The header file seems excessively long when only a handful of definitions are actually stated, but in keeping with the tradition of using single ASCII characters as symbols, conserving the full list helps browsers and inspectors work uniformly.

Moreover, insertions or deletions to the table are confined to the locations already alloted for them. Such systematics often reduce error and increase programming convenience. In fact, for some of the more common variants, such as floating point numbers, comments in the form of quoted strings, or frequently used operators, including both forms in the table with the unused line neutralized by making it into a comment can reduce confusion and save programming effort.

This header file is also the place to declare all the REC functions which it introduces — at least the compiling and executing functions — if not all their satellites. Those can probably be accounted for in the REC implementation file with less overall effort.

```
/* rectbl.h -- G. Cisneros, 4.91; rev. 2.10.91 */
/* Mandatory declarations */

# include "rec.h"

int r_lpar();           /* compilation of a left parenthesis            */
int r_rpar();           /* compilation of a right parenthesis           */
int r_colon();          /* compilation of a colon                       */
int r_semicol();        /* compilation of a semicolon                   */
int r_code();           /* compilation of a simple operator             */
int r_oper1();          /* compilation of an operator with an ASCII arg */
int r_pred();           /* compilation of a simple predicate            */
int r_pred1();          /* compilation of a predicate with an ASCII arg */
int r_noopc();          /* compilation of no-op                         */
int r_comment();        /* compilation of a (bracket enclosed) comment  */
int r_ubrack();         /* compilation of an unbalanced right bracket   */
int r_lbrace();         /* compilation of a (brace enclosed) program    */
int r_ubrace();         /* compilation of an unbalanced right brace     */

int r_call();           /* execution of predicate @x (call subr. x)     */
int r_quit();           /* execution of operator _ (exit to O.S.)       */
int r_xbrace();         /* execution of brace-enclosed program          */

/* Optional: Counter predicate */

 int r_ctrc();          /* compilation of a counter */
 int r_ctrx();          /* execution of a counter   */

/* Optional declarations, needed only in versions using quotes and
   numbers (see cnum.c and cquo.c); r_ld, r_ldch and r_xstr must
   be provided by user */

// int r_dquote(); /* compile doubly quoted string into symbol table */
// int r_squote(); /* compile singly quoted char into the program array */
// int r_cmin();   /* compile '-' as simple operator, call r_cnum if followed
```

```
//                    by digit or period */
// int r_cnum();   /* compile a number (WORD, LONG or REAL) into sym. table */

// int r_ld();         /* execution of compiled numbers */
// int r_ldch();        /* execution of apostrophe (single quote op.) */
// int r_xstr();        /* execution of quoted strings */

int r_cdblp(), r_ldblp();

/* Declarations of user-provided execution subroutines */

void roplb(), roplc(), ropue(), roplf(), roplg();
void roplh(), roplj(), roplm(), ropln(), roplo();
void roplp(), roplq(), ropls(), roplr(), ropls();
void roplu(), roplv(), roplx(), roply(), roplz();

void ropub(), ropuc(), ropuf(), ropug(), ropuh(), ropuk();
void ropul(), ropum(), ropun(), ropuo();
void ropup(), ropuq(), ropur(), ropus(), roput(), ropuu();
void ropux(), ropuy(), ropuz();

int  rprua(), rprla(), rprll(), rprui(), rprli();

void ropqm(), roppl(), ropmi(), ropti(), ropdi(), ropam();
int  rprns(), rprps();

int  rprze(), rpron(), rprtw(), rprth();

/* Table of compiling/executing function definitions,
   a triple [compile, execute, comment] for each printing ASCII character */

struct fptbl dtbl[] = {
    r_noopc,  FALSE,    " space (separator)",                    /* [blank] */
    r_ctrc,   r_ctrx,   " !n! - count to n   ",                        /* ! */
//  r_dquote, r_xstr,   "",                                 /* " quoted string */
    r_noopc,  FALSE,    "",                                 /* " quoted string */
    r_pred,   rprns,    " # - random predicate, probability 1/2  ",    /* # */
    r_cdblp,  r_ldblp,  " $xxx.x$ - floating point number ",           /* $ */
    r_pred,   rprps,    " % - random predicate, probability 1/10 ",    /* % */
    r_code,   ropam,    " & - exchange top two items on stack",        /* & */
//  r_squote, r_ldch,   " quoted character",                           /* ' */
    r_noopc,  FALSE,    "",                                  /* '  quoted char */
    r_lpar,   FALSE,    " ( - start of expression",                    /* ( */
    r_rpar,   FALSE,    " ) - end of expression",                      /* ) */
    r_code,   ropti,    " * - multiply top of list, leave product ",   /* * */
    r_code,   roppl,    " + - add top of list, leave sum",             /* + */
    r_noopc,  FALSE,    " , ",                                 /* , separator */
//  r_cmin,   FALSE,    " - ",                                         /* - */
    r_code,   ropmi,    " - - subtract top of list, leave difference",  /* - */
//  r_cnum,    r_ld,    " , ",                                 /* . f.p. number */
    r_noopc,  FALSE,    " . ",                                         /* . */
```

```
    r_code,   ropdi,    " /  - divide top of list, leave quotient",     /* / */
//  r_cnum,     r_ld,    " 0 ",    /* 0 number */
//  r_cnum,     r_ld,    " 1 ",    /* 1 number */
//  r_cnum,     r_ld,    " 2 ",    /* 2 number */
//  r_cnum,     r_ld,    " 3 ",    /* 3 number */
//  r_cnum,     r_ld,    " 4 ",    /* 4 number */
//  r_cnum,     r_ld,    " 5 ",    /* 5 number */
//  r_cnum,     r_ld,    " 6 ",    /* 6 number */
//  r_cnum,     r_ld,    " 7 ",    /* 7 number */
//  r_cnum,     r_ld,    " 8 ",    /* 8 number */
//  r_cnum,     r_ld,    " 9 ",    /* 9 number */
    r_pred,    rprze,    " 0 - switch ",    /* 0 */
    r_pred,    rpron,    " 1 - switch ",    /* 1 */
    r_pred,    rprtw,    " 2 - switch ",    /* 2 */
    r_pred,    rprth,    " 3 - switch ",    /* 3 */
    r_noopc,   FALSE,    " 4 ",    /* 4 number */
    r_noopc,   FALSE,    " 5 ",    /* 5 number */
    r_noopc,   FALSE,    " 6 ",    /* 6 number */
    r_noopc,   FALSE,    " 7 ",    /* 7 number */
    r_noopc,   FALSE,    " 8 ",    /* 8 number */
    r_noopc,   FALSE,    " 9 ",    /* 9 number */
    r_colon,   FALSE,    " : - repeat from opening lparen ",  /* : iteration */
    r_semicol, FALSE,    " ; - T exit at closing rparen ",    /* ; true exit */
    r_noopc,   FALSE,    " < ",                                /* < */
    r_noopc,   FALSE,    " = ",                                /* = */
    r_noopc,   FALSE,    " > ",                                /* > */
    r_code,    ropqm,    " ? - print register values",         /* ? */
    r_pred1,   r_call,   " @x - call subroutine x",            /* @ */
    r_pred,    rprua,    " A - Is angle a multiple of 90 degrees?",     /* A */
    r_code,    ropub,    " B - ",                              /* B */
    r_code,    ropuc,    " C - hyperbolic cosine ",            /* C */
    r_noopc,   FALSE,    " D ",                                /* D */
    r_code,    ropue,    " E - complex exponential",           /* E */
    r_code,    ropuf,    " F - fractional linear mapping (z+1)/(z-1) ", /* F */
    r_code,    ropug,    " G - move to first point of a drawn line",    /* G */
    r_code,    ropuh,    " H - ",                              /* H */
    r_pred,    rprui,    " I - Is real part an integer?",      /* I */
    r_noopc,   FALSE,    " J - ",                              /* J */
    r_code,    ropuk,    " K -        ",                       /* K */
    r_code,    ropul,    " L - complex logarithm ",            /* L */
    r_code,    ropum,    " M - graphic multiplier: $10.0$ M, default",  /* M */
    r_code,    ropun,    " N - ",                              /* N */
    r_code,    ropuo,    " O - ",                              /* O */
    r_code,    ropup,    " P - push (duplicate) top of the stack",      /* P */
    r_oper1,   ropuq,    " Qk - draw a colored square (one argument)",  /* Q */
    r_oper1,   ropur,    " R - Rn to recover nth constant ",   /* R */
    r_oper1,   ropus,    " S - Sn to save nth constant ",      /* S */
    r_code,    roput,    " T - hyperbolic tangent",            /* T */
    r_code,    ropuu,    " U - ",                              /* U */
    r_noopc,   FALSE,    " V ",                                /* V */
    r_noopc,   FALSE,    " W ",                                /* W */
```

```
    r_code,    ropux,    " X - push 1 ",                                /* X */
    r_code,    ropuy,    " Y - push i ",                                /* Y */
    r_code,    ropuz,    " Z - push 0 ",                                /* Z */
    r_comment, FALSE,    " [begin comment]",                            /* [ */
    r_noopc,   FALSE,    "",                                            /* \ */
    r_ubrack,  FALSE,    " [end comment]   ",                           /* ] */
    r_noopc,   FALSE,    " ^ - ",                                       /* ^ */
    r_code,    r_quit,   " _ - abandon program",                        /* _ */
    r_noopc,   FALSE,    " ` - ",                                       /* ` */
    r_pred,    rprla,    " a - Is angle a multiple of 15 degrees?",     /* a */
    r_noopc,   FALSE,    " b ",                                         /* b */
    r_code,    roplc,    " c - compression threshold (nominal 0.15) ",  /* c */
    r_noopc,   FALSE,    " d - ",                                       /* d */
    r_noopc,   FALSE,    " e - ",                                       /* e */
    r_code,    roplf,    " f - multiply by real factor: $xx.x$ f ",     /* f */
    r_code,    roplg,    " g - continue line to additional points",     /* g */
    r_code,    roplh,    " h -    ",                                    /* h */
    r_pred,    rprli,    " i - Is real part a multiple of 0.1?",        /* I */
    r_code,    roplj,    " j - complex conjugate",                      /* j */
    r_noopc,   FALSE,    " k - ",                                       /* k */
    r_pred1,   rprll,    " lx - pushdown management; (lk deep; ok;)",   /* l */
    r_code,    roplm,    " m - square size: $1.25$ m, default",         /* m */
    r_code,    ropln,    " n - negative",                               /* n */
    r_code,    roplo,    " o - ",                                       /* o */
    r_code,    roplp,    " p - pop the stack = discard top element ",   /* p */
    r_oper1,   roplq,    " qk - draw a small colored square (2 args)",  /* q */
    r_code,    roplr,    " r - complex square root",                    /* r */
    r_code,    ropls,    " s - draw a small stereo square",             /* s */
    r_noopc,   FALSE,    " t - ",                                       /* t */
    r_code,    roplu,    " u - push 0.1",                               /* u */
    r_code,    roplv,    " v - push 0.1 i ",                            /* v */
    r_noopc,   FALSE,    " w - ",                                       /* w */
    r_code,    roplx,    " x - push 0.01",                              /* x */
    r_code,    roply,    " y - push 0.01 i ",                           /* y */
    r_code,    roplz,    " z - draw a small circle",                    /* z */
    r_lbrace,  r_xbrace, " { - start of program",                       /* { */
    r_noopc,   FALSE,    " | - ",                                       /* | */
    r_ubrace,  FALSE,    " } - end of program",                         /* } */
    r_noopc,   FALSE,    " ~ - "                                        /* ~ */
};
```

3.2 REC/C data flow

Figure 7 shows the data flow in REC/C, which is not very extensive. There is essentially one array, a pushdown list of ample dimension, which will hold some complex numbers. There data can be introduced, following which partial results of computations can be stored, and eventually used as coordinates for the graphing operators. These operators display points or lines in the Plane Window in various forms.

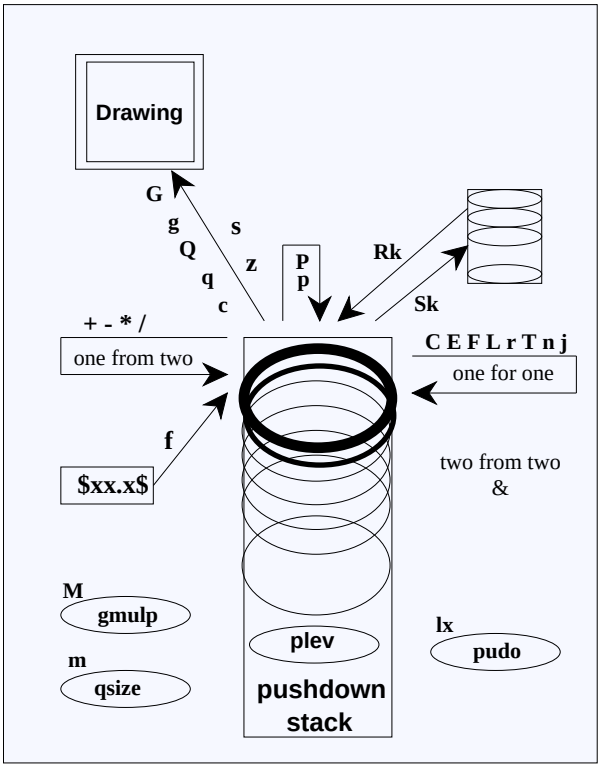


Figure 7: Data movements caused by the `REC/C` operators.

3.3 `REC/C` operators and predicates

The following transcript gives the actual `REC/C` operators and predicates, and can be consulted as the final authority on how they actually work.

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*             definition of REC operators and predicates                  */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

int rec(z) char *z; {if (rec_c(z,rprg,PLEN,dtbl)==1) rec_x(rprg);}

int rprze() {return(options[0]==TRUE);}
int rpron() {return(options[1]==TRUE);}
int rprtw() {return(options[2]==TRUE);}
int rprth() {return(options[3]==TRUE);}

int  rprla() {double ex, wy;                          /* complex logarithm */
if (plev>=HMAX) return;
ex = thestack[plev][0];
wy = thestack[plev][1];
```

```
thestack[plev][0] = 0.5*log(ex*ex+wy*wy+0.01);
thestack[plev][1] = atan2(wy,ex);
}

void roplc() {cgrad = r_dblpar;}                    /* square multiplier */

void roplf() {
thestack[plev][0] *= r_dblpar;
thestack[plev][1] *= r_dblpar;
}

void roplg() {double ex, wy;
if (plev>=HMAX) return;
ex = gmulp*thestack[plev][0];
wy = gmulp*thestack[plev][1];
PSlineto(ex+exoff,wy+wyoff);
}

void roplh() {}                                      /* halve step      */

int  rprli() {double ex, wy, rr, ss;                   /* 1/10 integer? */
ex = thestack[plev][0];
wy = thestack[plev][1];
rr = gmulp*sqrt(ex*ex+wy*wy);
ss = (double)((int)rr);
return (rr>ss-0.01 && rr<ss+0.01);
}

void roplj() {thestack[plev][1] = -thestack[plev][1];} /* complex conjugate */

int  rprll() {int c;                                 /* pushdown accounting */
c = *r_pc++;
if (c >= '0' && c <= '9')
    {if (c-'0' > pudo) return FALSE; else return TRUE;}
else if (c == '+') {pudo++;   return TRUE;}
else if (c == '-') {pudo--;   return TRUE;}
else if (c == 'z') {pudo = 0; return TRUE;}
else return TRUE;
}

void roplm() {qsize = r_dblpar;}                      /* square multiplier */

void ropln() {                                                /* negate */
thestack[plev][0] = -thestack[plev][0];
thestack[plev][1] = -thestack[plev][1];
}

void roplo() {}

void roplp() {if (plev<1) return; plev--;}            /* pop pushdown list */
```

```
void roplq() {                                  /* show a colored rectangle */
  double ex, wy, eu, ve, rr, th;
  int c;
if (plev < 1) return;
eu = thestack[plev][0];
ve = thestack[plev][1];
ex = thestack[plev-1][0];
wy = thestack[plev-1][1];
PSstroke();
c = *r_pc++;
switch (c) {
  case 'l':                          /* logarithmic absolute value */
    rr = 0.5*log(eu*eu + ve*ve + 1.0);
    th = atan2(ve,eu);
    PSsetcmykcolor(cos(th),cos(th+2.1),cos(th+4.2),0.1*rr);
    break;
  case 'v':                           /* colored by absolute value */
    th = 6.28*tanh(cgrad*(eu*eu + ve*ve));
    rr = 0.1*atan2(ve,eu);
    PSsetcmykcolor(0.67*cos(th+0.5),cos(th+1.8),cos(th+4.3),0.45*rr);
            break;
  case 'a':                             /* colored by phase angle */
            rr = tanh(0.5*(eu*eu + ve*ve));
            th = atan2(ve,eu);
    PSsetcmykcolor(0.67*cos(th+0.2),cos(th+1.8),cos(th+4.3),0.45*rr);
            break;
  case 'r': PSsetrgbcolor(1.0,0.0,0.0); break;
  case 'g': PSsetrgbcolor(0.0,1.0,0.0); break;
  case 'b': PSsetrgbcolor(0.0,0.0,1.0); break;
  case 'k': PSsetgray(NX_BLACK); break;
  case 'c': PSsetcmykcolor(1.0,0.0,0.0,0.0); break;
  case 'm': PSsetcmykcolor(0.0,1.0,0.0,0.0); break;
  case 'y': PSsetcmykcolor(0.0,0.0,1.0,0.0); break;
  default:  PSsetgray(NX_BLACK); break;}
PSrectfill(gmulp*ex+exoff,gmulp*wy+wyoff,qsize,qsize);
}

void roplr() {double ex, wy, eu, ve, rr;            /* complex square root*/
ex = thestack[plev][0];
wy = thestack[plev][1];
rr = sqrt(ex*ex+wy*wy);
eu = 0.5*(rr + ex);
ve = 0.5*(rr - ex);
thestack[plev][0] = sqrt(eu);
thestack[plev][1] = (wy>=0.0)? sqrt(ve): -sqrt(ve);
}

void ropls() {double ex, wy, eu, ve, rr, th;  /* stereo colored squarelet */
if (plev < 1) return;
eu = thestack[plev][0];
ve = thestack[plev][1];
```

```
ex = gmulp*thestack[plev-1][0];
wy = gmulp*thestack[plev-1][1];
rr = 0.5*log(eu*eu + ve*ve + 1.0);
th = atan2(ve,eu);
PSsetcmykcolor(cos(th),cos(th+2.1),cos(th+4.2),0.1*rr);
PSrectfill(ex+exoff-2.0*rr-95.0,wy+wyoff,0.90,0.90);
PSrectfill(ex+exoff+2.0*rr+95.0,wy+wyoff,0.90,0.90);
}

void roplu() {                                              /* push 0.1 */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.1;
thestack[plev][1]=0.0;
}

void roplv() {                                             /* push 0.1 i */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.0;
thestack[plev][1]=0.1;
}

void roplx() {                                               /* push 0.01 */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.025;
thestack[plev][1]=0.00;
}

void roply() {                                             /* push 0.01 i */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.00;
thestack[plev][1]=0.025;
}

void roplz() {double eks, wye;                             /* draw circle */
PSstroke();
eks = gmulp*thestack[plev][0];
wye = gmulp*thestack[plev][1];
PSsetgray(NX_BLACK);
PSmoveto(eks+4.0,wye);
PSarc(eks, wye, 4.0, 0.0, 360.0);
PSmoveto(eks,wye);
PSstroke();
}

// -------------------------------

int  rprua() {double ex, wy, th;    /* is angle a multiple of 90 degrees?*/
```

```
ex = thestack[plev][0];
wy = thestack[plev][1];
th = atan2(wy,ex);
}

void ropub() {}

void ropuc() {double ex, wy;                          /* hyperbolic cosine */
if (plev>=HMAX) return;
ex = 0.5*thestack[plev][0];
wy = 0.5*thestack[plev][1];
thestack[plev][0] = cosh(ex)*cos(wy);
thestack[plev][1] = sinh(ex)*sin(wy);
}

void ropue() {double ex, wy, rr;                   /* complex exponential */
ex = thestack[plev][0];
wy = thestack[plev][1];
rr = exp(ex);
thestack[plev][0] = rr*cos(wy);
thestack[plev][1] = rr*sin(wy);
}

void ropuf() {double ex, wy, dd;                          /* (z+1)/(z-1) */
ex = thestack[plev][0];
wy = thestack[plev][1];
dd = ex*ex + wy*wy - 2.0*ex + 1.001;
thestack[plev][0] = (ex*ex+wy*wy-1.0)/dd;
thestack[plev][1] = -(2.0*wy)/dd;
}

void ropug() {double ex, wy;           /* set up initial point of a line */
if (plev>=HMAX) return;
ex = gmulp*thestack[plev][0];
wy = gmulp*thestack[plev][1];
PSmoveto(ex+exoff,wy+wyoff);
}

void ropuh() {}                                          /* double step */

int  rprui() {double ex, wy, rr, ss;                        /* integer? */
ex = thestack[plev][0];
wy = thestack[plev][1];
rr = sqrt(ex*ex+wy*wy);
ss = (double)((int)rr);
return (rr>ss-0.01 && rr<ss+0.01);
}

void ropuk() {}

void ropul() {double ex, wy;                           /* complex logarithm */
```

```
ex = thestack[plev][0];
wy = thestack[plev][1];
thestack[plev][0] = 0.5*log(ex*ex+wy*wy+0.01);
thestack[plev][1] = atan2(wy,ex);
}

void ropum() {gmulp = r_dblpar;}                    /* set graphing scale */

void ropun() {}                                    /* reflect in next edge */

void ropuo() {}

void ropup() {                                       /* push top of stack */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=thestack[plev-1][0];
thestack[plev][1]=thestack[plev-1][1];
}

void ropuq() {double ex, wy; int c;            /* show a colored rectangle */
c = *r_pc++;
switch (c) {
  case 'r': PSsetrgbcolor(1.0,0.0,0.0); break;
  case 'g': PSsetrgbcolor(0.0,1.0,0.0); break;
  case 'b': PSsetrgbcolor(0.0,0.0,1.0); break;
  case 'k': PSsetgray(NX_BLACK); break;
  case 'c': PSsetcmykcolor(1.0,0.0,0.0,0.0); break;
  case 'm': PSsetcmykcolor(0.0,1.0,0.0,0.0); break;
  case 'y': PSsetcmykcolor(0.0,0.0,1.0,0.0); break;
  default:  PSsetgray(NX_BLACK); break;}
ex = thestack[plev][0];
wy = thestack[plev][1];
PSrectfill(gmulp*ex+exoff,gmulp*wy+wyoff,qsize,qsize);
}

void ropur() {int n;                                /* recover constant # n */
if (plev>=HMAX) return;
plev++;
n = *r_pc++-'0';
if (n<0 || n>10) return;
thestack[plev][0] = constant[n][0];
thestack[plev][1] = constant[n][1];
}

void ropus() {int n;                                   /* save constant # n */
n = *r_pc++-'0';
if (n<0 || n>10) return;
constant[n][0] = thestack[plev][0];
constant[n][1] = thestack[plev][1];
}
```

```
void roput() {}

void ropuu() {}

void ropux() {      /* push 1 */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=1.0;
thestack[plev][1]=0.0;
}

void ropuy() {      /* push i */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.0;
thestack[plev][1]=1.0;
}

void ropuz() {      /* push 0 */
if (plev>=HMAX) return;
plev++;
thestack[plev][0]=0.0;
thestack[plev][1]=0.0;
}

/* ------------ */

void ropqm() {}

void roppl() {
if (plev < 1) return;
thestack[plev-1][0] = thestack[plev-1][0] + thestack[plev][0];
thestack[plev-1][1] = thestack[plev-1][1] + thestack[plev][1];
plev--;
}

void ropmi() {
if (plev < 1) return;
thestack[plev-1][0] = thestack[plev-1][0] - thestack[plev][0];
thestack[plev-1][1] = thestack[plev-1][1] - thestack[plev][1];
plev--;
}

void ropti() {double re, im;
if (plev < 1) return;
re = thestack[plev-1][0]*thestack[plev][0] -
thestack[plev-1][1]*thestack[plev][1];
im = thestack[plev-1][0]*thestack[plev][1] +
thestack[plev-1][1]*thestack[plev][0];
thestack[plev-1][0] = re;
thestack[plev-1][1] = im;
```

```
plev--;
}

void ropdi() {double ex, wy, rr, re, im;
if (plev < 1) return;
ex = thestack[plev][0];
wy = thestack[plev][1];
rr = ex*ex+wy*wy;
if (rr < 0.00001) rr = 0.0001;
re = thestack[plev-1][0]*thestack[plev][0] +
thestack[plev-1][1]*thestack[plev][1];
im = - thestack[plev-1][0]*thestack[plev][1] +
thestack[plev-1][1]*thestack[plev][0];
thestack[plev-1][0] = re/rr;
thestack[plev-1][1] = im/rr;
plev--;
}

int rprns() {return (int)(random()&01);}                /* random 1/2 pred */

int rprps() {return random()%10==1;}                   /* random 1/10 pred */

void ropam() {double xx;                             /* exchange top of stack */
if (plev>=HMAX) return;
if (plev < 1) return;
xx = thestack[plev-1][0];
thestack[plev-1][0]=thestack[plev][0];
thestack[plev][0]=xx;
xx = thestack[plev-1][1];
thestack[plev-1][1]=thestack[plev][1];
thestack[plev][1]=xx;
}
```

4 RECView.h header listing

Although the headers for a methods file can contain all the things that ordinary headers do, they are mostly given over to listing objects and method prototypes for their associated file. If the Interface Builder has been used without the benefit of a header file, there is an option called "unparsing" which will create one. This is usually done only once when a program is first created. Conversely, if the text of the header file has been changed, the `.nib` file can be updated by using "parse."

It is worth noting that there is no convenient way to supply several arguments to a method which is going to be connected graphically in the Interface Builder. But methods are not supposed to look at each other's data anyway; they should only transmit the data or pointers to the data in response to a request. To know who made a request, and thus where to send the reply, a method can volunteer its own name which the called program can recognize as an argument called `:sender`, at which time it knows how to ask for further services or data.

That is why :sender is the only argument of most of the methods in the prototype list.

```
/*
 * RECView.h -- Interface file for the RECView class
 *
 * You may freely copy, distribute, and reuse the code in this example.
 * NeXT disclaims any warranty of any kind, expressed or implied, as to
 * its fitness for any particular use, or even that it has a use at all.
 *
 */

#import <appkit/appkit.h>

@interface RECView:View {

        id          tiDat;

        id          browser;

        id          exsli;
        id          wysli;
        id          planeView;
        id          planeWindow;
        id          infoPanel;
        id          parPanel;

        id          comment;

        id          execWin;
        id          execLine;
        id          RECValue;

        id          keyWindow;
        id          menuCopy;
        id          menuPrint;

        char        *filename;
        char        **filenames;

        id          windows;
        id          externalRECBox;
        id          internalRECBox;
        id          theBigBox;
        id          theLilBox;
        id          xtrnWin;
}

- ascii:sender;
- recMatrix:sender;

- rexsli:sender;
```

```
- rwysli:sender;

- run:sender;
- recompile:sender;
- gho:sender;
- setOption:sender;
- windowWillClose:sender;
- loadRECPanel:sender;
- loadInternal:sender;
- loadExternal:sender;
- loadLowerPaneltwo:thisRuleBox;
- nhul:sender;
- browserDoubleClick:sender;

- open:sender;
- setFilename:(const char *)aFilename;
- save:sender;
- saveAs:sender;

- copyREC:sender;
- copySomeView:theView;
- printREC:sender;

- initFrame:(const NXRect *)frameRect;
- drawSelf:(const NXRect *)rects :(int)rectCount;
- sizeTo:(NXCoord)width :(NXCoord)height;
- mouseDown:(NXEvent *)theEvent;

@end

/* end RECView.h */
```

5 RECView.m program listing

REC/C is short enough that a single method file, RECView.m and its associated header, RECView.h, are sufficient to contain the program. Nevertheless the use of such artifacts as the browser and text windows, in addition to the main Window and View controls, and the C implementation program for the REC operators, divide the program into several natural parts, each of which can be described separately.

5.1 declarations

The declarations shown below contain several extraneous elements, and are only included for consistency with the rest of the program, in case the definition of terms used elsewhere is needed.

```
/*
 * RECView.m -- Implementation file for the RECView class
```

```
 *
 * You may freely copy, distribute, and reuse the code in this example.
 * NeXT disclaims any warranty of any kind, expressed or implied, as to
 * its fitness for any particular use, or that it even has a use.
 *
 */

#import "rec.h"
#import "recdtbl.h"
#import "RECView.h"
#import <NXCType.h>
#import <math.h>
#import <stdlib.h>
#import <string.h>

# define CLEN  600 /* buffer length for rec source        */
# define PLEN 2000 /* buffer length for rec object        */
# define HMAX   50 /* maximum depth pushdown stack        */
# define CMAX   10 /* number of saved constants           */
# define  NMU    5 /* number of items in first column     */
# define   NY    8 /* number of demonstration programs    */
# define  NIP   12 /* number of initial polygons          */
# define  NSK    5 /* number of prototype skeletons       */
# define PI 3.14159

//int  rec(char *z);

char   cstr[CLEN] = { " (()) " };       /* console rec program     */
Inst   rprg[PLEN];                      /* rec program array       */
int    phil;                            /* length of cstr    */
char   idef[1][30] = { " (()) " };  //(z(!100!0r:;)1r2r(0r:;);) a" };
char   recrule[NY][CLEN] = {
" ([pentagonal] X5 P CN CN CN CN C p ;)",
" ([bipentag] X5 P CN CN CN Cn Cn Cn Cn C p ;)",
" ([two sectors] B C R2 C R1 C N C ;)"
};
char   skeletons[NSK][CLEN] = {
" ([the polygon] B C ;)"
" ([top polygon] O C ;)",
" ([sector] O (!n! C N :;) ;)",
" ([binary] O (!n-1! C N :;) (!n-1! C n :;);)",
" ([second] O (!n! C N :;) ;)"
};

int    isErrm=FALSE;
int    drawBack=YES;
int    exSource=YES;
int    options[4]={FALSE,FALSE,FALSE,FALSE};
char   errm[50];
double cgrad=0.15;    /* threshold in tanh compression                    */
double gmulp=10.0;    /* scale factor turning complex numbers into pixels */
```

```
double qsize=1.50;    /* size of small pixel squares on screen           */

double exoff=0.0, wyoff=0.0;
extern double r_dblpar;
int    plev=0;                                          /* pushdown depth  */
double constant[CMAX][2];                               /* saved constants */
double thestack[HMAX][2];                               /* operating stack */

@implementation RECView

/* ---------------  A P P   I N I T I A L I Z A T I O N   ----------------- */

- appWillInit:sender {printf("appWillInit = REC/C\n"); return self;}

- appDidInit:sender {
 char c[3], fecha[30], tiempo[30], *df;
 int i;
 struct tm  *timeptr;
 time_t  secsnow;
 printf("appDidInit = REC/C for Complex Arithmetic\n");

   time(&secsnow);
   timeptr = localtime(&secsnow);
   sprintf(fecha,"%d-%d-19%02d\n",
((timeptr->tm_mon) + 1),
timeptr->tm_mday,
        timeptr->tm_year);
   [tiDat setStringValue: fecha];

   sprintf(tiempo,"%02d:%02d:%02d\n",
timeptr->tm_hour,
timeptr->tm_min,
        timeptr->tm_sec);
//   [titime setStringValue: tiempo];

///* initialize r_vst with select standard procedures */
//   rec_c(idef[0],xprg[0],XLEN,dtbl);
//   r_vst['a']=xprg[0];

  [browser getTitleFromPreviousColumn: YES];
  [browser setDoubleAction: @selector(browserDoubleClick:)];
  [browser setPath: "/demonstrations"];

  [self loadExternal: self];

  return self;
}

- appWillTerminate:sender {printf("goodbye\n"); return self;}

/* ---------------------------- G E N E R A L ---------------------------- */
```

```
/* = = = = = = =  P a r a m e t e r  A d j u s t m e n t  = = = = = = = */

- rexsli:sender {int i; /* set x-offset */
exoff=120.0*[sender doubleValue];
  [planeView display];
return self;
}

- rwysli:sender { /* set y-offset */
wyoff=120.0*[sender doubleValue];
  [planeView display];
return self;
}

/* = = = = = = = = = = =  P r o g r a m  E d i t o r  = = = = = = = = = = */

- recMatrix:sender { printf("selected row = %3d\n ",[sender selectedRow]);
  [comment setStringValue:
dtbl[ [sender selectedRow] * 32 + [sender selectedCol] ].r_cmnt];
return self;
}

- ascii:sender {
  dtbl[ [sender selectedRow] * 32 + [sender selectedCol] ].xfun();
return self;
}

/* = = = = = = = = =  P r o g r a m  E x e c u t o r  = = = = = = = = = */

- setOption:sender {int i;
  i=[sender selectedCol];
  options[i]=[[sender selectedCell] state];
return self;
}

- run:sender {[planeView display]; return self;}

- recompile:sender {
  [execWin getSubstring: cstr start: 0 length: [execWin textLength]];
  [window setDocEdited:YES];
return self;
}

- browserDoubleClick:sender {int i, j, k;
   if (strcmp([browser titleOfColumn: [browser selectedColumn]],
"initial polygons") == 0) {}
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
"demonstrations") == 0) {
    [RECValue setStringValue: " "];
    [execLine setStringValue: [sender stringValue]];
```

```
   strcpy(cstr,[sender stringValue]);
}
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
"skeletons") == 0) {
    [RECValue setStringValue: " "];
    [execLine setStringValue: [sender stringValue]];
}
   [planeView display];
return self;
  }

- gho:sender {
   strcpy(cstr,[sender stringValue]);
//printf("went and ghoed: %s\n",[sender stringValue]);
//printf("went and ghoed: %s\n",cstr);
   [planeView display];
   return self;
  }

/* - - - - - - - - -   F I L E   A N D   D I R E C T O R Y   - - - - - - - - - */

- setFilename:(const char *)aFilename {
if (filename) free(filename);
filename = malloc(strlen(aFilename)+1);
strcpy(filename, aFilename);
[window setTitleAsFilename:aFilename];
return self;
}

- open:sender {
  char *types[2] = {"rec",0};
  int  i, fd;
  NXStream *theStream;

  [[OpenPanel new] allowMultipleFiles: NO];
  [[OpenPanel new] chooseDirectories: NO];

  if ([ [OpenPanel new] runModalForTypes:types]) {
    [self setFilename: [[OpenPanel new] filename]];
    fd = open(filename, O_RDONLY, 06666);
    theStream = NXOpenFile(fd, NX_READONLY);
//      NXScanf(theStream,"%s",cstr);
      i=0;
      while (NXScanf(theStream,"%c",&cstr[i])!=EOF) i++;
      phil=strlen(cstr);
    NXClose(theStream);
    close(fd);
    [window setTitleAsFilename: filename];
    [window setDocEdited:NO];
  [execWin setSel: 0: [execWin textLength]];
  [execWin replaceSel: cstr];
```

```
  [execWin scrollSelToVisible];
  [execWin display];
    }
return self;
}

- saveAs:sender {
  id panel;
  const char *dir;
  char *file;

/* prompt user for the file name and save to that file */

  if (filename==0) {
    /* no filename; set up defaults */
    dir  = NXHomeDirectory();
    file = (char *)[window title];
    } else {
    file=rindex(filename,'/');
    if (file) {
      dir   = filename;
      *file = 0;
      file++;
      } else {
      dir  = filename;
      file = (char *)[window title];
      }
    }
  panel = [SavePanel new];
  [panel setRequiredFileType: ""];
  if ([panel runModalForDirectory: dir file: file]) {
    [self setFilename: [panel filename]];
    return [self save: sender];
    }
    return nil;  /* didn't save */
  }

- save:sender {
  int      fd;
  NXStream *theStream;

  if (filename==0) return [self saveAs: sender];
  [window setTitle: "Saving ..."];

  fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 06666);
  if (fd < 0) {
    NXRunAlertPanel(0, "Cannot save file: %s",0,0,0,strerror(errno));
    return self;
    }
  theStream = NXOpenFile(fd, NX_WRITEONLY);
  [execWin writeText: theStream];
```

```
  NXClose(theStream);
  close(fd);
  [window setTitleAsFilename: filename];
  [window setDocEdited:NO];
  return self;
}

- close:sender {const char *fname;
int         q;
  if ([window isDocEdited])
//    fname=filename ? filename : [sender title];
//    if (rindex(fname,'/')) fname = rindex(fname,'/') + 1;
    q = NXRunAlertPanel(
      "Save",
      "Save changes to %s?",
      "Save",
      "Don't Save",
      "Cancel",
      filename);
    if (q==1) {if (![self save:nil]) return nil;}
    if(q==-1) return nil;
//  [sender setDelegate: nil];
//  [proc free];
//  [self free];
  return self;
}

- windowWillClose:sender {
[self close: self];
printf("windowWillClose\n");
return self;
}

/*******   REC program management ********/

- loadRECPanel:sender {
if ([sender state]) [self loadInternal: sender];
else                [self loadExternal: sender];}

- loadInternal:sender {
  printf("Internal\n ");
  [self loadLowerPaneltwo: internalRECBox ];
  return self;
  }

- loadExternal:sender {
  printf("External\n ");
  [self loadLowerPaneltwo: externalRECBox ];
  return self;
  }
```

```
- loadLowerPaneltwo:thisRuleBox {NXRect bxRect, loRect;
  [theBigBox getFrame: &bxRect];
  [thisRuleBox getFrame: &loRect];
  [theLilBox setContentView: [thisRuleBox contentView]];
  loRect.origin.x=(bxRect.size.width-loRect.size.width)/2.0;
  loRect.origin.y=(bxRect.size.height-loRect.size.height)/2.0;
  [theLilBox setFrame: &loRect];
  [theBigBox display];
  [theLilBox display];
return self;
}

- textDidChange:textObject {
  return  [windows  setDocEdited: YES];
}
```

5.2 copying and printing Views

There is a mechanism for printing selected views, which is more selective than the printing options which can be connected by mouse dragging in the Interface Builder. Likewise Views can be copied to a pasteboard, where they are available for use by other programs, such as Draw. The problem is to know which Window, or which View to draw, although in principle a separate button for every one of them could be placed on the program's menu.

By making RECView a delegate of a Window which could be printed, it has an opportunity to respond to the announcement that the Window has become the key Window — in other words, the focus of attention of keyboard strokes and mouse movements and has been displayed in front of everything else. The response takes the form of connecting the one single copy button and the single print button to the View which is to be copied or printed.

To avoid the uncertainty of knowing what to print when some other window has taken priority, the "key resigned" can be used to deactivate the menu buttons. Of course, the substituting window can delegate RECView the responsibility of printing *it*, but otherwise the buttons in the menu remain disconnected.

```
/* - - - - - - - -  c o p y  a n d  p r i n t  v i e w s  - - - - - - */

- printREC:sender {[keyWindow smartPrintPSCode: sender]; return self;}

- windowDidBecomeKey:sender {
keyWindow=sender;
[menuCopy setTarget: self];
[menuCopy setAction: @selector(copyREC:)];
[menuCopy setEnabled: YES];
[menuPrint setTarget: self];
[menuPrint setAction: @selector(printREC:)];
[menuPrint setEnabled: YES];
return self;
}
```

```
- windowDidResignKey:sender {
[menuCopy setEnabled: NO];
[menuPrint setEnabled: NO];
return self;
}

- copyREC:sender {
    drawBack=NO;

    if (keyWindow==planeWindow)
[self copySomeView: planeView];
    if (keyWindow==infoPanel)
[self copySomeView: [infoPanel contentView]];
    if (keyWindow==execWin)
[self copySomeView: [execWin contentView]];
    if (keyWindow==parPanel)
[self copySomeView: [parPanel contentView]];

    drawBack=YES;
return self;
}

- copySomeView:theView {
NXRect zbounds;
id     pBoard;
//  drawView=viewnum;
//  *draw=YES;
  pBoard = [Pasteboard new];
  [pBoard declareTypes: &NXPostScriptPboardType num: 1 owner: self];
  [theView getBounds: &zbounds];
  [theView writePSCodeInside: &zbounds to: pBoard];
//  *draw=NO;
return self;
}
```

5.3 browser delegate methods

The program for the NeXTSTEP browser goes to considerable effort to create sliding panels, embody them in an attractive framework, fill them with information, and respond to the activities of the mouse. It still needs to be provided with information to display, and to be told how to respond. The program using the browser has to take charge, which it does by having been designated as a delegate. There are some methods which the browser supervisor expects to find, such as fillMatrix::: described below.

```
/*---------------- B R O W S E R  D E L E G A T E ------------------*/

- (int)browser:sender fillMatrix: matrix inColumn: (int)col {int  i, n;
    n=0;
    printf("browser\n");
```

```
switch(col) {
  case 0:
    for(i=0; i<NMU; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setLeaf : NO ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
      }
    [[matrix cellAt: 0: 0]  setStringValue: "operators"];
    [[matrix cellAt: 1: 0]  setStringValue: "predefined"];
    [[matrix cellAt: 2: 0]  setStringValue: "skeletons"];
    [[matrix cellAt: 3: 0]  setStringValue: "initial polygons"];
    [[matrix cellAt: 4: 0]  setStringValue: "demonstrations"];
    n=NMU;
    break;
  case 1:
    if(strcmp([[sender selectedCell] stringValue],"demonstrations") == 0){
      for(i=0; i<NY; i++) {
        [matrix addRow];
        [[matrix cellAt: i: 0]  setStringValue: recrule[i]];
        [[matrix cellAt: i: 0]  setLeaf : YES ];
        [[matrix cellAt: i: 0]  setLoaded : YES ];
        [[matrix cellAt: i: 0]  setEnabled : YES ];
        }
      n=NY;}

    if(strcmp([[sender selectedCell] stringValue],"predefined") == 0){
      for(i=0; i<1; i++) {
        [matrix addRow];
        [[matrix cellAt: i: 0]  setStringValue: idef[i]];
        [[matrix cellAt: i: 0]  setLeaf : YES ];
        [[matrix cellAt: i: 0]  setLoaded : YES ];
        [[matrix cellAt: i: 0]  setEnabled : YES ];
        }
      n=1;}

    if(strcmp([[sender selectedCell] stringValue],"operators") == 0){
      for(i=0; i<95; i++) {
        [matrix addRow];
        [[matrix cellAt: i: 0]  setStringValue: dtbl[i].r_cmnt];
        [[matrix cellAt: i: 0]  setLeaf : YES ];
        [[matrix cellAt: i: 0]  setLoaded : YES ];
        [[matrix cellAt: i: 0]  setEnabled : YES ];
        }
      n=95;}
    if(strcmp([[sender selectedCell] stringValue],"initial polygons") == 0){
       for(i=0; i<NIP; i++) {
        [matrix addRow];
        [[matrix cellAt: i: 0]  setStringValue: inpollies[i].title];
        [[matrix cellAt: i: 0]  setLeaf : YES ];
```

```
            [[matrix cellAt: i: 0]  setLoaded : YES ];
            [[matrix cellAt: i: 0]  setEnabled : YES ];
           }
          n=NIP;}
        if(strcmp([[sender selectedCell] stringValue],"skeletons") == 0){
           for(i=0; i<NIP; i++) {
            [matrix addRow];
            [[matrix cellAt: i: 0]  setStringValue: skeletons[i]];
            [[matrix cellAt: i: 0]  setLeaf : YES ];
            [[matrix cellAt: i: 0]  setLoaded : YES ];
            [[matrix cellAt: i: 0]  setEnabled : YES ];
           }
          n=NSK;}

            break;
        default: n=0; break; }
      return n;
    }

/* to distract the single click in a browser double click */
- nhul: sender {return self;}

- browserDoubleClick:sender {int i, j, k;
   if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "initial polygons") == 0) {
        k=[[sender matrixInColumn: [sender selectedColumn]] selectedRow];
        pgon=inpollies[k].order;
        edge = 0;
        for (i=0; i<pgon; i++) for (j=0; j<3; j++)
                pointbase[i][j] = inpollies[k].points[i][j];
           [execLine setStringValue: "(CP;)"];
          strcpy(cstr,"(CP;)");}
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "demonstrations") == 0) {
           [value setStringValue: " "];
           [execLine setStringValue: [sender stringValue]];
          strcpy(cstr,[sender stringValue]);
        }
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "skeletons") == 0) {
           [value setStringValue: " "];
           [execLine setStringValue: [sender stringValue]];
        }
   [lilView display];
return self;
  }
```

5.4 View methods

There are several methods which the programmer must supply to create or modify the content of a View. Which ones are needed and their complexity depends on the intricacy of the planned view. The most essential methods are:

- - initFrame: sets up the overall characteristics of the View, such as its background color, the location and orientation of its coordinate system, and similar details.
- - drawSelf:: draws the View each time it is invoked. It is therefore responsible for the entire content of the View, which may depend on parameters which vary between successive invocations.
- - mouseDown: has an argument describing the interruption, which can be used as the basis for decisions about how to respond to the interruption.
- - sizeTo:: would be used to respond to a change in the size of the View; for example to ensure centering the origin of coordinates.

Beyond these essentials there are other possibilities, such as assigning a delegate.

```
/* = = = = = = = = = = = = = =   R E C V I E W   = = = = = = = = = = = = = */

- initFrame:(const NXRect *)frameRect
/*
 * Initializes the new RECView object.  First, an initFrame: message is sent
 * to super to initialize RECView as a View.  Next, the RECView sets its own
 * state -- that it is opaque and that the origin of its coordinate system
 * lies in the center of its area.
 */
{
[super initFrame:frameRect];
[self setOpaque:YES];
[self translate:floor(frame.size.width/2) :floor(frame.size.height/2)];
return self;
}

- drawSelf:(const NXRect *)rects :(int)rectCount
/*
 * Draws the RECView's background and axes.  If there are any points,
 * these are drawn too.
 */
{
if (rects == NULL) return self;

// PSsetgray(NX_WHITE);
PSsetgray(NX_LTGRAY);
if (drawBack) NXRectFill(&rects[0]);
PSsetgray(NX_DKGRAY);
// PSsetgray(NX_BLACK);

    plev=0;
```

```
    isErrm=FALSE;
    [RECValue setStringValue: "*"];
    if (rec(cstr)) [RECValue setStringValue: "T"];
              else [RECValue setStringValue: "F"];
    if (isErrm) {[comment setStringValue: errm]; isErrm=FALSE;}
    PSstroke();

return self;
}

- sizeTo:(NXCoord)width :(NXCoord)height
/*
 * Ensures that whenever the RECView is resized, the origin of its
 * coordinate system is repositioned to the center of its area.
 */
{
[super sizeTo:width :height];
[self setDrawOrigin:-floor(width/2) : -floor(height/2)];

return self;
}

@end
```

6 Examples of calculations with complex arithmetic

It is worth running through some simple exercises in order to become familiar with the REC complex number graphing calculator. In fact, the only visible results from the graphing calculator are those shown in the graph rectangle, so the first order of business is to relate the dimensions of this rectangle to the size of the complex numbers, which can be done by placing colored dots at specified locations.

Rarely will just placing complex numbers on the graph plane be of interest. It is more likely that functions will be evaluated, which implies a whole series of argument points and their corresponding function values. But graphing that combination would imply four dimensions, two each for the argument and for the value, and there are only two dimensions available to work with.

One common method of representing a complex function is to choose arguments along the arc of some curve, maybe even a segment of a line parallel to either the real or the imaginary axis. The function values will sit along some other curve, which can be left as it is, or marked somehow with the values of the argument — say, tick marks at regular argument intervals. Carried further, whole families of arcs, such as coordinate patches, could be mapped into new families, the nature of the function to be deduced from the distortion induced by this mapping. Such factors as the spacing between coordinate lines, or the change of size and orientation of coordinate rectangles would be used to interpret the function.

An alternative is to draw a contour map, in which the arcs drawn represent curves of constant absolute value or of constant phase, or both. Of course, the contours could identify constant real and imaginary parts, but modular contouring seems to be more informative.

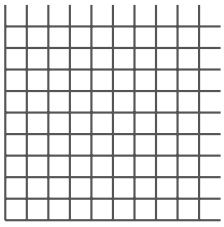
Working with absolute value alone, a three-dimensional surface can be imagined as being

drawn in perspective. By engraving lines of constant phase upon it, the full function can be displayed. At one time it was fashionable to place a collection of such plaster models in an exhibit case to show that mathematicians actually worked with real objects. Some idea of their beauty can be gleaned from an examination of Jahnke and Emde's classical *Tables of Functions with Formulae and Curves* [11] with its numerous line drawings of complex functions,

Yet another embellishment is to use stereoviews, which can be either of the three-dimensional absolute value surface seen from the side, or the contour lines seen from directly above. Both are special cases of projections taken from arbitrary angles, and there is no reason that two or more views cannot be combined into the same picture. For example, a perspective view seen to be floating over a contour map sitting just below it.

6.1 representation by line images

Figure 8 shows a `REC/C` program to map out a coordinate grid in the first quadrant by drawing lines of length 10 (as defined by the two counters) with integer coordinates. The pair of operators `G` and `g` correspond to PostScript functions `moveto` and `lineto` according to whether they establish an initial point or subsequent points for drawing line segments.



```
([rectangular grid]
Z (!10! PG (!10! X+ g:;) p Y +:;)
Z (!10! PG (!10! Y+ g:;) p X +:;)
;)
```

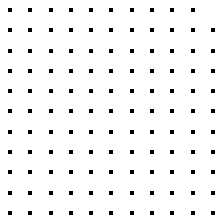
Figure 8: Coordinate grids can reveal locations in the complex plane.

Clearly each coordinate set has an inner loop which draws a line of a certain length by repeating unit segments. The reason that one long line isn't drawn at once is that no provision for a variable length line was included amongst the `REC` operators, which in turn is a question of programming style. As it is, there are operators generating real or imaginary numbers at three scales, and they alone are used to make increments or decrements.

Each outer loop displaces the line it draws until it gets a grid. Evidently the counter in the outer loop should have been `!11!` to close the grid, since there is always one more boundary than there are intervals.

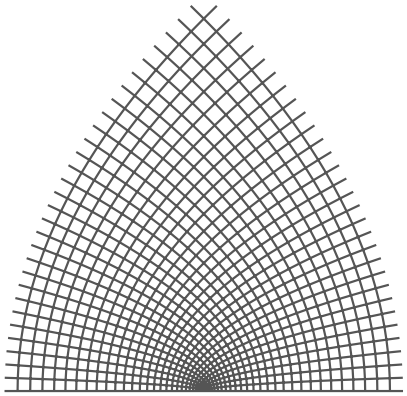
Instead of drawing line segments to establish a coordinate system, Figure 9 shows how to place dots at integer coordinate positions. This example can be studied in more detail by including the operator which defines the dot size and varying their size. Moiré effects can arise when the resolution of the dots requested disagrees with the texture of the printer which renders the final image.

Once the drawing of coordinate grids has been understood, the mappings induced by a variety of common functions can be explored. Figure 10 explores the consequences of the mapping $w = z^2$, which turns the coordinate grids into two families of orthogonal parabolas, opening in opposite directions. By mapping from a single quadrant, the image fills only half a plane, which already implies that the mapping will have a two-valued counterimage.



```
([rectangular grid]
Z (!10! P Qk (!10! X+ Qk :;) p Y +:;)
Z (!10! P Qk (!10! Y+ Qk :;) p X +:;)
;)
```

Figure 9: Coordinate grids in the complex plane can be implied by dots placed at integer coordinates.



```
([parabolic grid]
Z (!30! P PP*Gp (!30! u+ PP* g p :;) p v +:;) p
Z (!30! P PP*Gp (!30! v+ PP* g p :;) p u +:;) p
;)
```

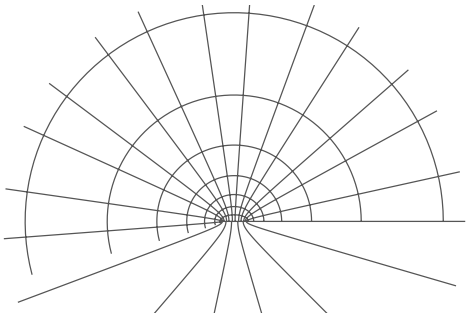
Figure 10: The parabolic grid.

The image lines are more dense in this example than in the previous one because the middle scale increment has been used, which in turn was partially done to get smoother curves. But since squares increase rapidly, only three times (the square root of ten) as many increments were used, to keep the final figure down to the same size as before.

Another mapping, whose contours are ellipses and hyperbolas rather than parabolas, is the hyperbolic cosine, illustrated in Figure 11. Except for implied inclusion of an exponential factor, the hyperbolic cosine is nothing more than $w = (z + 1/z)$ and the mapping involved is just as algebraic as the squaring which makes parabolas.

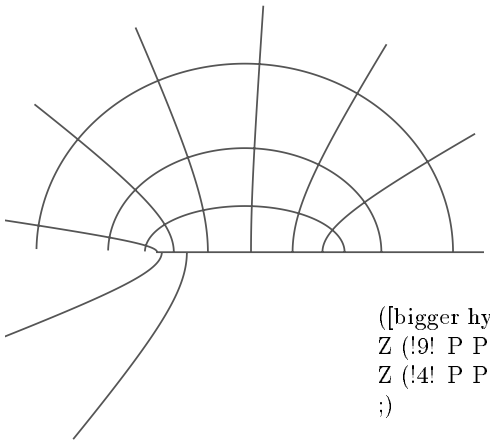
A lesson to be learned from Figure 11 is the convenience of proportioning the dimensions of the graphical representation so that 1 is never too near the center nor too near to the edge. The reason is not very complicated; first because 1 sits at the dividing line between growth and shrinkage of products of successively more numbers, and then because once there are even three or four factors, the scale of everything has completely changed. So 1 had better sit at a good vantage point.

Compare Figure 12 where the increments are now much finer, more of them are used, and the multiplier of 50 has been used to give a better relationship between dimensions within the complex plane and the size of the pixels in the screen image. The unit 1 is now quite a bit further out from the center than it was before, still leaving room for other small integers.



```
([hyperbolic cosine grid]
Z (!20! P PC G p (!72! u+ PC g p :;) p Y +:;) p
Z ( !8! P PC G p (!68! v+ PC g p :;) p X +:;) p
;)
```

Figure 11: With respect to the hyperbolic cosine, coordinate axes in the z plane map into confocal hyperbolas and ellipses, respectively, in the w plane. The foci sit at the points ± 1.



```
([bigger hyperbolic cosine grid] 50 M
Z (!9! P PC G p (!140! x+ PC g p :;) p Y +:;) p
Z (!4! P PC G p (!250! y+ PC g p :;) p X +:;) p
;)
```

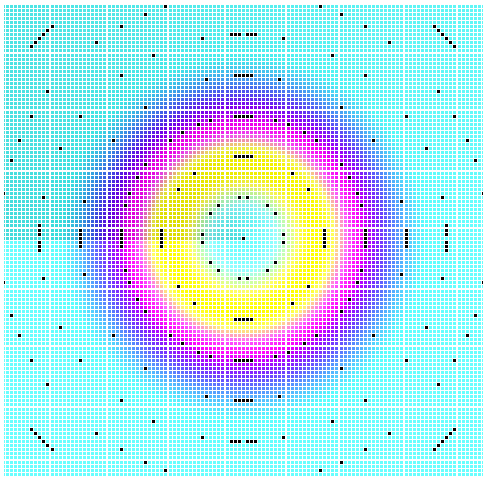
Figure 12: The hyperbolic cosine map at a larger scale.

6.2 representation by coloration

Color can be used to enhance drawings, but unless it is used with some discretion, the results can be totally confusing. One reason for this is that sharp boundaries are necessary for distinguishing objects, resulting in an amorphous appearance when viewing slowly graduated color schemes. Camouflaging makes good use of this circumstance. Some surface texture, such as inscribed contour lines, will greatly augment any use of color which may be present.

Nevertheless, scanning a region with the intention of placing colored pixels throught its interior can be performed easily, even when deducing contour lines would be quite a bit more complicated. One way to get a representation of complex numbers through a color code would be to use the color wheel which is thoroughly familiar in the television industry for the phase angle of the number. The grey scale would be entrusted to the modulus, running from white at zero to black at infinity, say.

Figure 13 shows an attempt at combining the two approaches, to put black pixels at integer moduli while color coding the rest of the plane. It is not entirely successful, mainly due to the interaction of pixel size with the criterion for integervaluedness.



```
([evaluate function] P (I qk; qv; ) p;) f
($20.0$M
Z (!60! P @f (!60! u + @f :;) p v + :;) p
Z (!60! P @f (!60! u + @f :;) p v - :;) p
Z (!60! P @f (!60! u - @f :;) p v + :;) p
Z (!60! P @f (!60! u - @f :;) p v - :;) p
;)
```

Figure 13: Color coding of the plane according to the absolute value of each point. The black squares represent an attempt to distinguish contours at discrete heights.

It is probably much more understandable if the rainbow sequence is used to color code the modulus, ignoring the phase, or perhaps assigning it a slight grey scale. Of course, almost any system will highlight zeroes and poles, either because of color extremes or because they are enveloped by multiples of 2π radians with the attending color display. The problem lies in conveying a realistic appreciation for all the intervening values.

Because several coloring schemes are plausible, the operator for coloring little squares, `qk`, has been given a parameter, whose `ASCII` character values are assigned as follows:

l - shading by logarithm of modulus, color by wheel

v - color by hyperbolic tangent of modulus, shading by angle

a - angle by color wheel, modulus by shading

r - red

g - green

b - blue

c - cyan

m - magenta

y - yellow

k - black

o - orange

v - violet

Whatever the case, there is room to insert other alternatives in the list of parameter symbols, or to change the ones shown.

Figure 14 shows two common functions of a complex variable, the hyperbolic cosine and the hyperbolic tangent. Of coures, they become the ordinary cosine and tangent after rotation by 90° and show the periodicity of these functions along the imaginary axis.

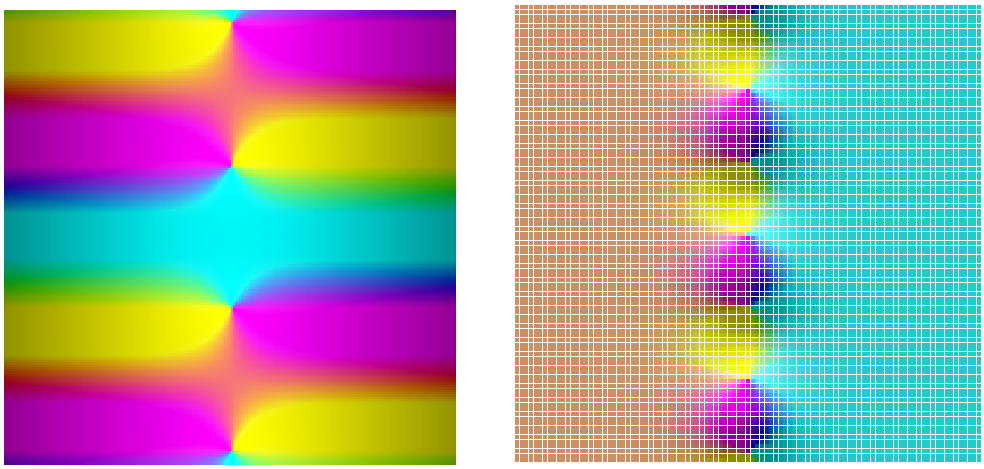


Figure 14: Color coded representation of the hyperbolic cosine (left) and the hyperbolic tangent (right).

6.3 representation by stereoviews

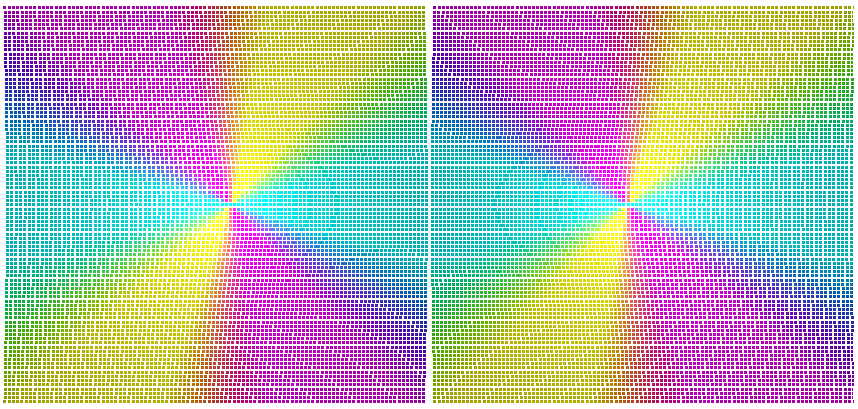


Figure 15: Stereo view of the function $w = z^2$.

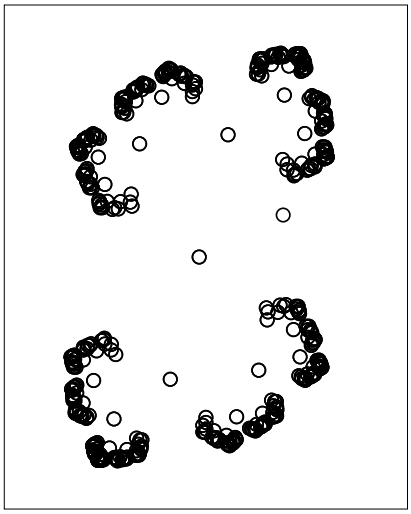
Figure 15 shows the square function in a stereopair, with some vestiges which are more likely to be a moiré depending on pixel size than anything which was planned. The effect is like looking down the inside of a conical paper cup, but the bottom should be more rounded.

The following `REC/C` code generates the pair.

```
{     ( [define function]  PP* s  p;) f
  (Z  (!100! P @f  (!100! u + @f  :;) p v + :;) p
   Z  (!100! P @f  (!100! u + @f  :;) p v - :;) p
   Z  (!100! P @f  (!100! u - @f  :;) p v + :;) p
   Z  (!100! P @f  (!100! u - @f  :;) p v - :;) p
   ;)   }
```

6.4 recursive root trees

When working with the Mandelbrot set and related topics such as Julia curves, it is convenient to be able to follow out their recursive development. One way to arrive at the Julia curves is to calculate the images of the fixed points of a function, such as the quadratic $w = z^2 + c$ which underlies the Mandelbrot set. Since two square roots always arise in each counterimage and each of their counterimages should be calculated in turn, the whole structure resembles a gigantic binary tree.



```
{
( l9; l+ R0 - r z P n z @f p f l- ; ) f
($100$ M $0.5$ X f $0.25$ Y f + S0 z lz
f ;)
}
```

Figure 16: A root tree, which will either define the boundary of a basin of attraction or make a Fatou dust.

To process it, one half is always saved while the other is run down to completion, following which the first half is given the same treatment. Such saving can be done on the pushdown list where complex arithmetic is being performed. In fact, the use of a pushdown list was dictated by foreseeing applications such as this one. Nevertheless the pushdown list is finite, only a part of the infinite calculation will ever be performed, so each recursive chain should be stopped when some assigned depth has been reached — such as the extent of the pushdown list.

Additionally, performance of the recursion accounts for only part of the activity on the pushdown list, making it convenient to reckon the recursion depth independently, to one side. With luck, or planning, the recursive depth can be kept low enough to avoid pushdown overflow.

With some forty of fifty letters having been assigned to complex operations, there are not many left over, and certainly not nice mnemonic ones. But there are ways to keep within the philosophy of single letters; say by using double letters, which is a category which the `REC` compiler recognizes. To keep track of the depth of recursions, the heretofore unused `l` has been chosen, with a single-letter parameter, which is assigned as follows.

z - Initialize the depth to zero

`+` - Increment the depth

`-` - Decrement the depth

`k` {k = 0, 1, ..., 9} - `True` if the depth exceeds `k`, `False` otherwise.

From this specification, `lx` is seen to be a predicate, but it is always true except when testing for depth, at which time it is used in the form `(lk too deep; go on;)`. Most likely, "`too deep`" would be null.

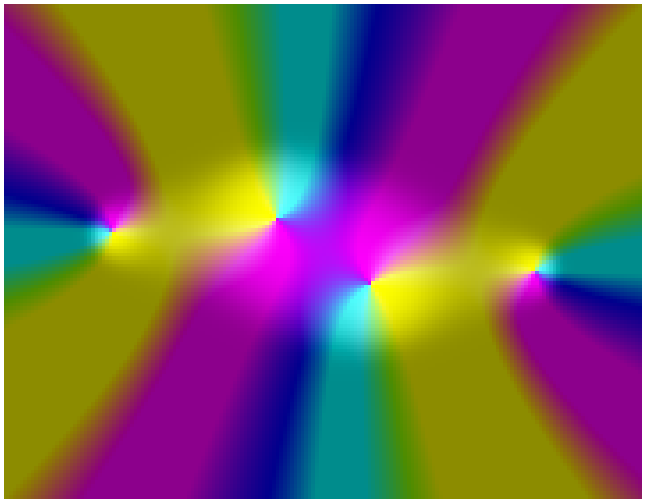


Figure 17: Color coded representation of an initial stage of the iteration towards a Julia set.

As another illustration of colorization, which can profitably be used in conjunction with a root tree, Figure 17 shows the result of a couple of iterations of the Julia set for the parameter -1. The light spots are zeroes, the rest is left to the imagination; apparently the rainbow is assigned to the phase of the function rather than its modulus.

However, to examine all the possibilities of complex function behavior is more the province of an article devoted to complex analysis than to the description of a program for performing the calculations. Even so, `REC/C` can be expected to evolve with further usage, as routes to better orgsnization become evident, or additional facilities are added to accomodate demands for additional calculations.

7 Acknowledgements and disclaimer

The NeXT computer on and for which this program has been developed is an acquisition from a CONACYT grant, which must have been one of its more productive investments, given that the computer has been in daily use for many hours since its acquisition.

The program `REC/C` described in the document can hardly be expected to be error free. Moreover, if it is successful, it will be subject to user demands for improvements, corrections, and assorted changes. Not all of them will necessarily be reflected back in this manual, which is intended more to explain the program than to give a verbatim documentation.

References

[1] Harold V. McIntosh and Gerardo Cisneros, "The programming languages REC and Convert," *ACM SIGPLAN Notices* **25** 81-94 (July 1990).

[2] Gerardo Cisneros, "Configurable REC," *ACM SIGPLAN Notices* **29** May 1995 (11 pages)

[3] Simson L. Garfinkel and Michael K. Mahoney, *NeXTSTEP^{TM} Programming*, Springer Verlag, New York, 1953. ISBN 0-387-97884-4

[4] Adobe Systems, Incorporated, *PostScript Language Reference Manual*, Second Edition, Addison-Wesley Publishing Company, Inc., Reading, Massachussetts, 1990. ISBN 0-201-18127-4.

[5] Lars V. Ahlfors, *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable*, McGraw-Hill Book Company 1990 (ISBN 0070006571).

[6] Konrad Knopp (translated by Frederick Bagemihl), *Theory of Functions*, Parts I and II, Dover Publications, Inc., New York, 1996.

[7] Liang-shin Hahn, *Complex Numbers and Geometry*, Mathematical Association of America, 1994 (ISBN 0-88385-510-0).

[8] Paul J. Nahin, *An Imaginary Tale: The Story of the Square Root of Minus One*, Princeton University Press, Princeton, 1998 (ISBN: 0691027951).

[9] Brian E. Blank, "Book Review: An Imaginary Tale: The Story of the Square Root of Minus One," *Notices of the American Mathematical Society* **46** 1233-1236 (1999).

[10] E. A. Guillemin, *The Mathematics of Circuit Analysis*, The M.I.T. Press, Cambridge, 1949.

[11] Eugene Jahnke and Fritz Emde, *Tables of Functions with Formulae and Curves*, Dover Publications, New York, 1945.

February 12, 2001