The Icon Analyst

In-Depth Coverage of the Icon Programming Language

August 1995
Number 31

In this issue ...

Visual Interfaces

Editors' Note: Most of the material in this article and ones on similar subjects to follow is adapted from the forthcoming book on graphics programming in Icon.

In an earlier article [1], we described how a user can convey information to a program using mouse and keyboard events. Except for the simplest applications, it is more helpful to organize interaction between the program and the user by using interface tools such as buttons, menus, and sliders. Such interface tools provide a visual interface between the user and the program.

Using such tools, a wide range of functionality can be provided in ways that are convenient, familiar, and easily understood. For example, clicking on a button on the application window can be used to tell the application to perform some action, pulling down a menu can be used to select among operations, and dragging on a slider can be used to change a numerical value.

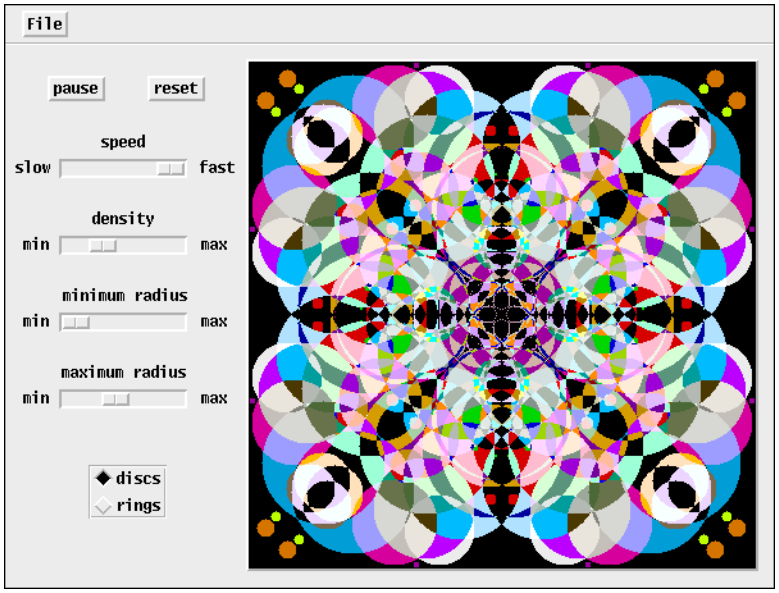
This article describes an application with a visual interface and then goes on to describe the interface tools that are available. Subsequent articles will explain how to build a visual interface and how it fits into a complete program.

An Example Application

The image below illustrates an application that displays a multicolored kaleidoscopic image.

The image is produced by drawing circles. The colors, sizes, and positions of the circles are chosen at random. Circles are drawn until the specified density (number of simultaneous circles) is reached, at which point the oldest circle is erased and a new one drawn. This continues until the user intervenes.

The pause button allows the user to suspend drawing, which is not resumed until the user presses this button again. The reset button clears the image and starts the drawing process from scratch. The sliders allow the user to control the speed of drawing, the density, and the minimum and maximum radius for circles. At the bottom, the user can choose between discs (solid circles) or rings (outlines). The File menu allows the user to take a snapshot of the image or quit the application.



A Kaleidoscope

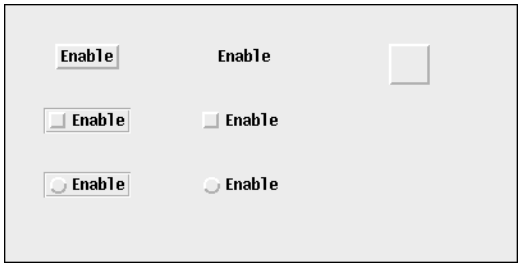
Interface Tools

Buttons

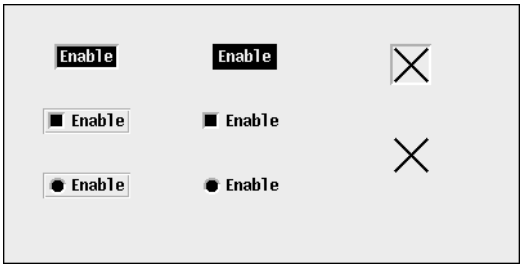
Buttons are among the most simple and commonly used interface tools. Pressing a mouse button when the mouse cursor is on a button amounts to "pushing" the button.

Buttons support two kinds of functionality. An ordinary button only has a momentary effect: It remains on only as long as it's held down, then reverts to its original state. A toggle button stays on when it is pushed, and it must be pushed a second time to turn it off. Both kinds of buttons are highlighted while on.

There are four basic button styles and outlines are optional. The button at the right is called an X-box button; unlike other buttons styles, it has no text associated with it.



The nature of highlighting depends on the style of the button. As you see here, there is an X-box button without an outline; it's only visible when it's highlighted.

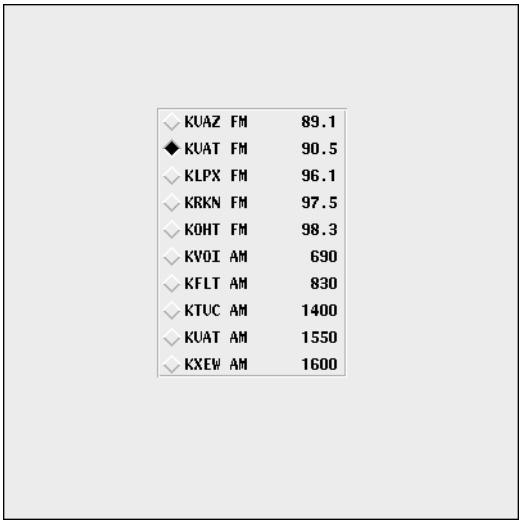


X-box buttons and buttons with squares or circles at the left give the impression that they can be set. Consequently, they are best used for toggles.

Radio Buttons

Radio buttons are collections of buttons in which only one button is on at any time. Pushing a button turns it on and highlights it, and turns off the previously selected button.

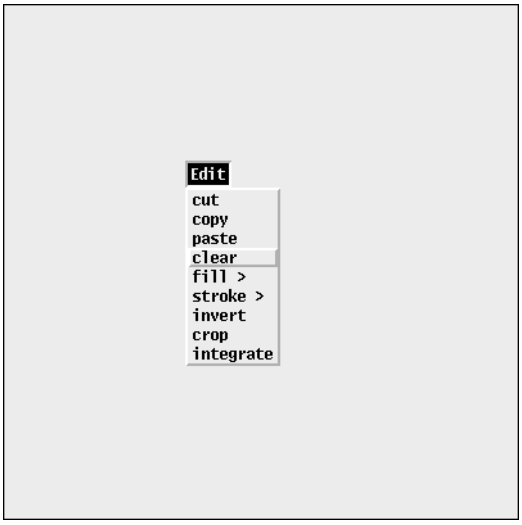
Only one style is provided for radio buttons:



The example here was chosen to emphasize the origin of the term "radio button". Radio buttons can, of course, have any labels such as the names of colors available in a particular application.

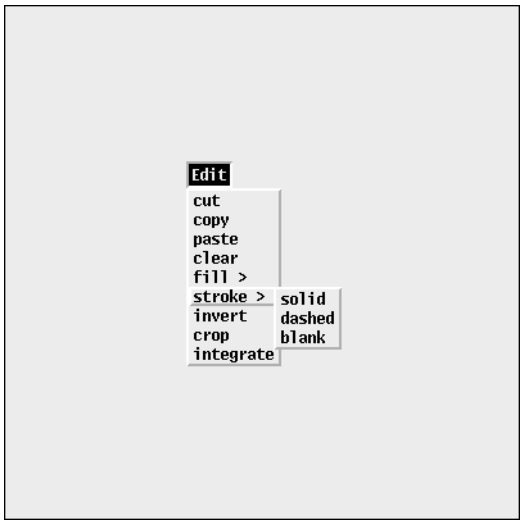
Menus

A menu is a button that conceals a list of items. When you push the menu button, the list of items is "pulled down" and the item under the mouse cursor is highlighted. As you drag over the items on the list, the item under the mouse cursor is highlighted:

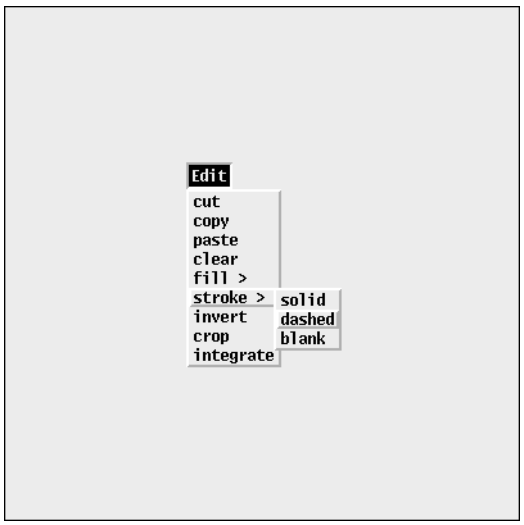


Releasing the mouse button with the mouse cursor positioned on an item selects that item. If you drag off the list and release the mouse button, the list disappears and no item is selected.

A menu item can itself be a menu. Such items are identified by an angle bracket at the right. If you select one of these items, its menu appears to the right:



You can then drag onto this submenu and select an item there:

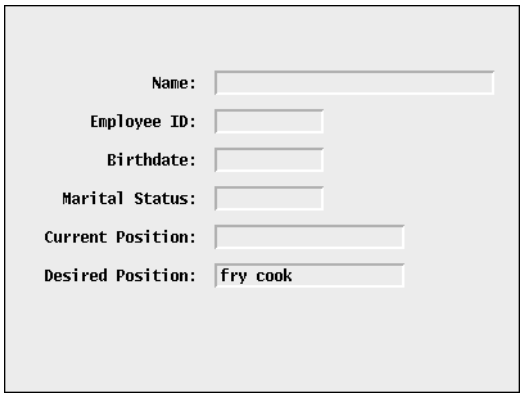


If you drag off a submenu and select another item from the main menu, the submenu disappears.

Submenus themselves can have submenus. There is no limit to this hierarchical structure, but more than two or three levels is confusing to most users. Some users do not like submenus at all.

Text-Entry Fields

Text-entry fields allow the user to enter textual information:



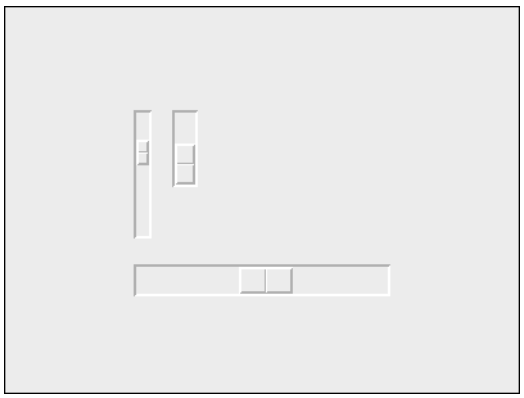
Each field has a label and space for the user to enter text. The maximum number of characters allowed can be specified; it determines the width of the field. A suggested value for a field can be given, as shown in the last text-entry field.

You can select a text-entry field by clicking on it, at which point an "I-beam" text cursor appears and you can enter or edit text. The I-beam cursor shows the current place in the field where typed text is inserted. This cursor can be positioned in the text by clicking with the mouse pointer at the desired location. Dragging over characters in the text field selects them for editing and highlights them (reversing the foreground and background colors). Characters that are typed then replace the selected ones. A backspace character deletes all the selected characters. If no character is selected, a backspace character deletes the character immediately to the left of the text cursor, if there is one.

Sliders

A slider specifies a numerical range visually. Numerical values, which can be integers or real numbers, are associated with the end points of a slider. A "thumb" marks the current position in the range. Dragging the thumb of a slider changes the value in the range covered by the slider.

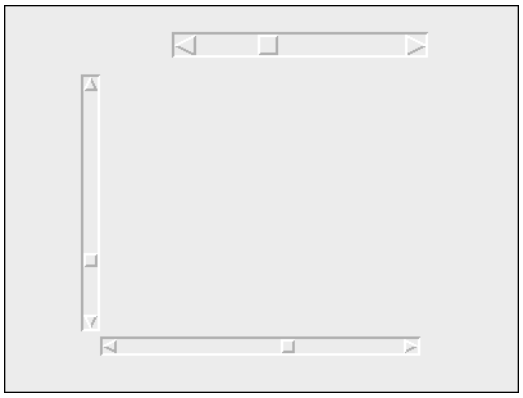
Sliders may be vertical or horizontal:



Sliders may have different sizes as shown in this figure.

Scroll Bars

Scroll bars are very similar to sliders, although they have a different visual representation:

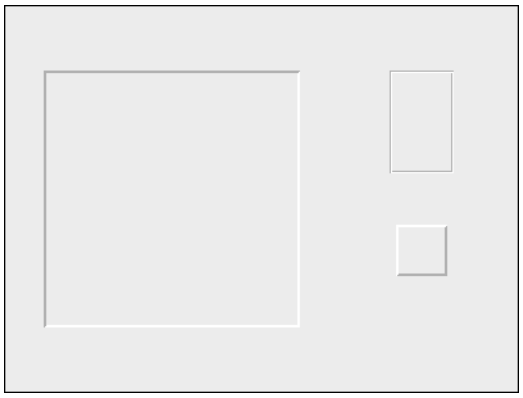


Dragging the thumb of a scroll bar has the same effect as dragging the thumb of a slider. In addition, clicking on an arrow at the end of a scroll bar moves the button incrementally in the direction indicated.

Sliders usually are used for setting values, while scroll bars typically are used to select a portion of a larger image for display in a smaller area.

Regions

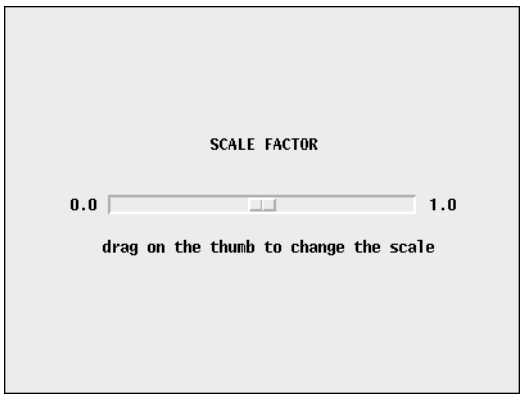
A region is a rectangular area that serves to accept events within its boundary. This figure shows three regions:



There are four region styles: sunken, grooved, raised, and invisible (which we can't show).

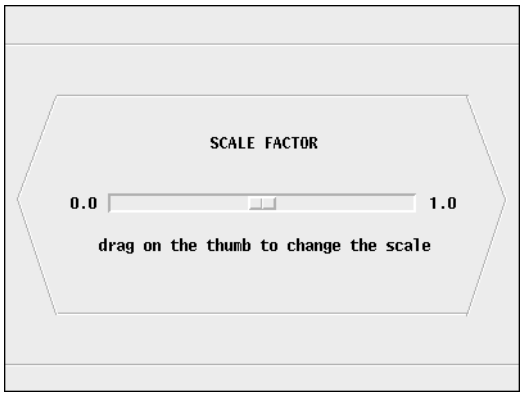
Labels

A label consists of text. Labels can be used to identify tools, indicate values, and so forth:



Lines

Lines can be used to visually delineate areas of an interface:



Although lines are only "decoration", they nonetheless can be very helpful in making an interface visually understandable.

Next Time

In the next article on visual interfaces, we'll show the connection between interface tools and a program that uses them. After that we'll go on to describe the process of building a program with a visual interface.

Reference:

1. "Handling Events in X-Icon", Icon Analyst 19, pp. 4-5.

Back Issues

Back issues of The Icon Analyst are available for $5 each. This price includes shipping in the United States, Canada, and Mexico. Add $2 per order for airmail postage to other countries.

The Versum Problem – Continued

In the last issue of the Analyst, we introduced versum sequences — adding a number and its reversal, continuing the process with the result.

The interest in such sequences comes from the fact that palindromic numbers typically occur frequently early on. This phenomenon is not found in successive additions without reversal – palindromes are found much less frequently without reversal than with it. The "problem" is that for a few starting numbers (called *seeds*), no palindrome appears even after millions of steps.

In the previous article on this subject, we posed some questions for which programs could be used to find answers or at least produce suggestive evidence. In this article, we'll present some results of our explorations and say a little more about what is known about the problem.

For investigating versum sequences, it sometimes is handy to have a file that contains many terms in a sequence. Such a file then can be processed in a variety of ways without having to regenerate the sequence. (The size of such files can be a problem, however — the first 20,000 terms in the versum sequence for 196 amount to more than 82MB.)

Producing Versum Sequences

Computing versum sequences takes time. Granted, Icon is far from the fastest language for such work, but it does have the necessary facilities built in: large-integer arithmetic, automatic conversion between integers and strings, and string reversal. And, of course, exploratory programming is faster and easier in a high-level language than in a lower-level one.

Computing versum sequences is an excellent example of the tension between speed and ease of programming. Try writing such a program in C or assembly language. You might want to do this if you need a very large number of terms in a versum sequence, but as we'll explain later, that's not needed or even useful for many purposes. Incidentally, there are utilities and applications that can be used to obtain versum sequences quickly. But, in any event, the Analyst is devoted to Icon and even such a computationally intensive problem gives insights into programming techniques in Icon that have applicability to many problems.

As we discussed in the previous article, conversions between large integers and strings, which are fundamental to the computation of versum sequences, are expensive, and it's important to avoid unnecessary ones.

Here's the loop that we used to write versum sequences to a file:

```
repeat {
   i +:= reverse(i)
   write(i := string(i))
   }
```

For each step, the loop performs only the two conversions necessary for the computation itself.

While the loop above is compact, it's handy to have a way to stop the program other than by killing it and to, in general, have more control over the computation.

Here's the program we used:

```
link options

procedure main(args)
   local start, output, input, i, opts, limit, name, max, count

   opts := options(args, "s+t+m+f:")
   start := (0 < \opts["s"]) | 196
   limit := \opts["t"] | –1
   max := opts["m"]
   name := \opts["f"] | (start || ".vsq")

   if input := open(name) then {
      count := 0
      while i := read(input) do {
         count +:= 1
         if count > limit then
            stop("*** number of existing terms exceeds limit")
         }
      close(input)
      }

   /i := start       # in case file doesn't exist or is empty

   if not integer(i) then stop("*** invalid data")

   output := open(name, "a") |
      stop("*** cannot open file")

   limit –:= \count

   until (limit –:= 1) = –1 do {
      i +:= reverse(i)
      if i > \max then break
      write(output, i := string(i))
      }

end
```

The –s option provides the seed. The –t option allows the maximum number of terms in the sequence to be specified and the –m option allows the sequence to be limited to a maximum value. Finally, the –f option allows the file name to be specified. The program also allows for extending an existing sequence by picking up the computation with the last value.

The limit on the number of terms and the magnitude complicate the loop, and you might think they would slow down the computation. They don't make a measurable difference, because the conversions between large integers and strings dominate the computation.

It's worth noting that different seeds may produce the same versum sequence. For example, the reversal of a seed, provided the seed does not end with the digit 0, has the same versum sequence as the seed. There are many more equivalences; in fact, there are only 207 different sequences in the seeds from 1 to 999. The equivalence of seeds is an important topic that we'll discuss later.

Assuming that you have a collection of files containing versum sequences, you can examine them in various ways using programs. (You certainly wouldn't want to print a file with many thousands of terms for bedtime reading.)

Palindromes in Versum Sequences

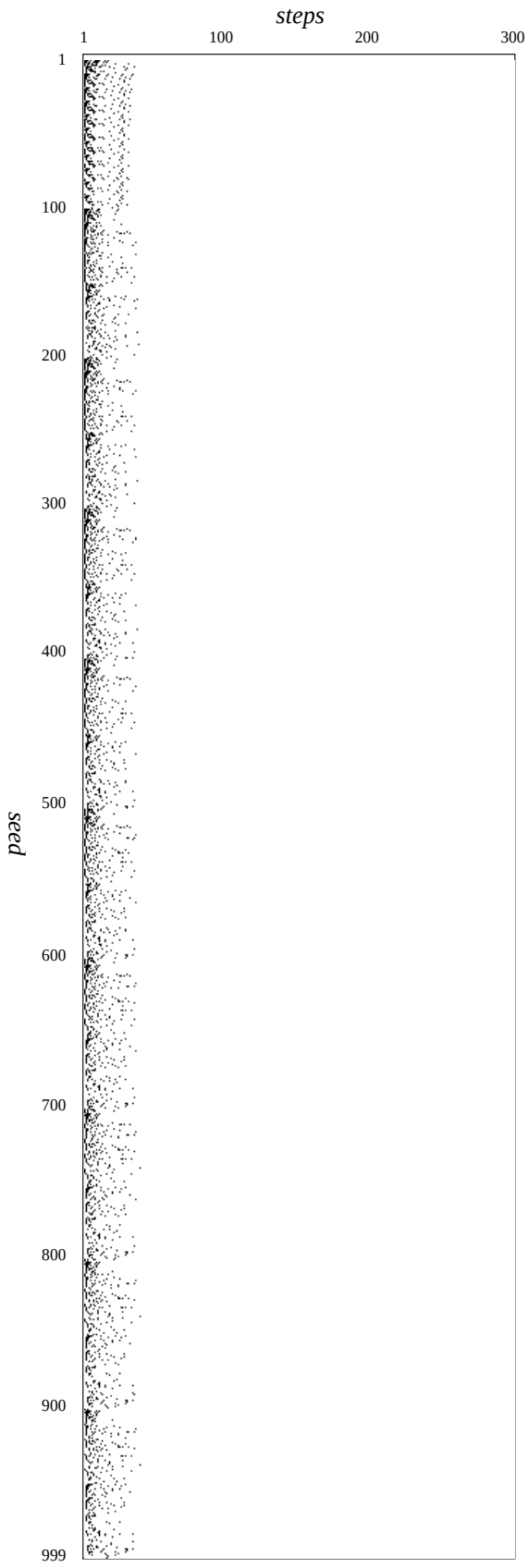
Palindromes are the reason for interest in versum sequences. Before going on, it's worth looking at the pattern of palindromes in versum sequences for consecutive small seeds. See the plot in the next column and the close-up view on the next page.

Two aspects of these plots are noteworthy: the distinct patterns of palindromes and the lack of any palindromes after only a few terms. If you think you see a palindrome in the white space at the right, it's a printing or paper flaw or a flyspeck — there aren't any and carrying out thousands more steps on several versum sequences that have many palindromes early on turns up no more.

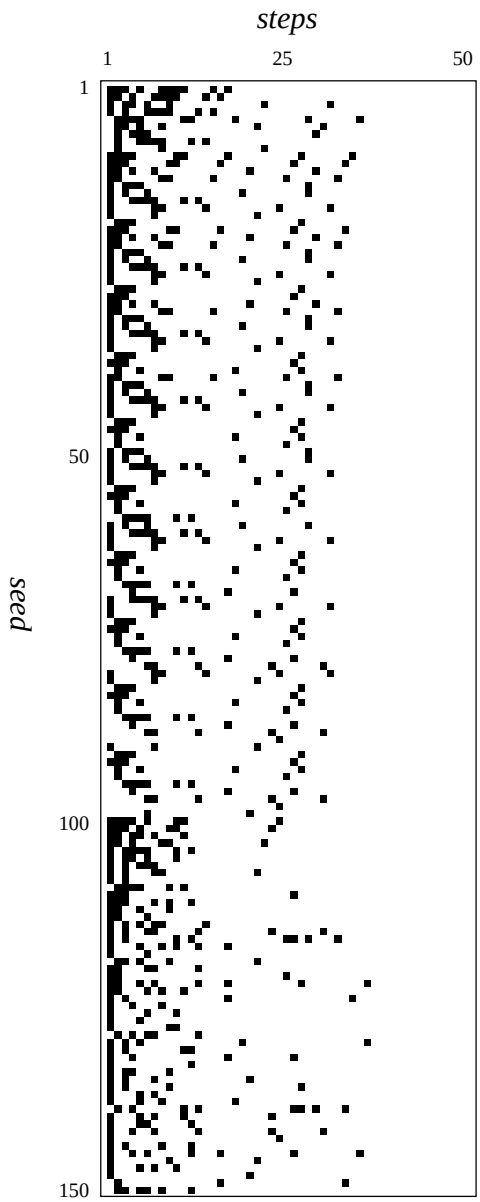
Here are some pieces of trivia about the palindromes for seeds 1 through 999:

The seed with the most palindromes is 1. It has 10, 8 of which occur in succession starting with step 1.

The largest palindrome is 69567677677676596, which occurs in the sequence for 166, among others.



Palindromes in Versum Sequences for Seeds from 1 to 999



Close-up View for Seeds from 1 to 150

The palindrome that is farthest out is 1685872332785861 in the 39th term in the sequence for 739, among others.

The longest number of steps to the first palindrome is 24 in the sequence for 89, among others.

Of course, the absence of observed palindromes far out in versum sequences is no proof that they don't exist. In the next section, we'll present an argument as to why there probably are no palindromes farther out.

Here's a brute-force way to produce the plot on page 6:

```
link wopen

procedure main()
   local i, j, k

   # For a general program, limits should be given
   # on the command line.

   WOpen("canvas=hidden", "size=300,999") |
      stop("*** cannot open window")

   every i := 1 to 999 do {
      k := i
      every j := 0 to 299 do {    # 300 steps, but 0-origin
         k +:= reverse(k)
         if k == reverse(k) then DrawPoint(j, i – 1)
            }
         }

   WriteImage("palimage.gif")

end
```

As we mentioned earlier, it's more economical to precompute versum sequences. That's true for palindromes also; if you have a versum sequence you can get a list of palindromes easily:

```
procedure main()
   local count, line

   count := 0

   # standard input to standard output

   while line := read() do {
      count +:= 1
      if line === reverse(line) then write(count, " ", line)
      }

end
```

If the palindromes are in files named *n*.pal, where *n* is the seed, then a plot can be produced in this way:

```
link wopen

procedure main()
   local i, x, input, line

   WOpen("canvas=hidden", "size=300,999") |
      stop("*** cannot open window")

   every i := 1 to 999 do {
      input := open(i || ".pal") |
         stop("*** cannot open file for seed ", i)
      while line := read(input) do {
         line ? {
            x := tab(many(&digits)) – 1
            }
         DrawPoint(x, i – 1)
```

```
    }
  close(input)
  }

 WriteImage("palimage.gif")

end
```

We can do better than this, but we need to lay some groundwork first, which we'll defer to another article.

The Versum Sequence for 196

196 has been the primary focus of study for the versum problem because that is the smallest seed for which no palindrome has been found.

Early work on this sequence was done by hand and, of course, didn't get very far. This is responsible for the fact that it was long believed that a palindrome eventually would appear in every versum sequence. Now, using computers, the sequence for 196 has been carried out for millions of steps without finding a palindrome. It would serve little purpose to put more resources into extending previous attempts. But we did carry out the computation to more that 20,000 terms so that we would have a substantial amount of data for study. The 20,000th term is shown on page 10 — we couldn't resist showing a "specimen".

If we believe that 196 never produces a palindrome, how might we approach a proof? One method would be to show that there is a repeating pattern that precludes a palindrome. This is, in fact, the way it was proved that there are palindrome-free versum sequences in bases 2^n [1, 2]. An obvious place to start is to look at the first and last digits of successive terms — these match up in successive steps and might give a clue as to why palindromes don't occur.

The first few last digits are shown at the right of this column (it's easier to see patterns in a vertical array than in a horizontal one). As you can see, there are evident regularities — such as the sequence of repeating 7s and 8s. However, it's also clear that if there is a fixed repeat, it's large.

You won't get very far trying to find a repeat from looking at the digits. The obvious approach to finding repeats, at least for programmers, is to write a program. An easier approach is to view the digits graphically. That is what the image at the right is — resembling a beserker barcode.

In this image, the spaces between horizontal lines correspond to the leftmost digits in the versum sequence for 196. We've reduced it so we could show a significant number of digits — nearly 2,000.

There certainly are obvious patterns, but close examination reveals there's no fixed-length repeat in what's shown. Of course, the repeat might be larger or might only show up far out into the sequence.

We have a program that purports to find fixed-length repeats in sequences and it finds nothing in the first 20,000 digits starting at the beginning and then progressively farther out. We say "purports" because the program is a little strange and we don't have a lot of confidence in its correctness.

Incidentally, other digit positions show no fixed-length patterns either.

Of course, there might be a pattern or even an evolving one that is not fixed in length. But all the evidence we have is discouraging.

The program that produced the image at the right is quite simple and worth showing for that:

```
$define Width    100
$define Height  2000

procedure main()
  local y, w
  WOpen("canvas=hidden", "size=" ||
    Width || "," || (Height)) |
       stop("*** cannot open window")

  DrawLine(0, 0, Width, 0)

  y := 0
  while w := read()[–1] do {     # get last digit
    if y + w > Height then break
    y +:= w
    DrawLine(0, y, Width, y)
    y +:= 1
    }
  WriteImage("196ld.gif")

end
```

Next, we'll consider something more mundane.

Digit Frequencies

In the article on the versum problem in the last issue of the Analyst, we mentioned digit frequencies as a possible starting point for investigating versum sequences. Such information might provide insight, and it relates to an argument against the occurrence of palindromes that we'll describe later. The results we have to offer are largely negative, but we're including them because there are interesting matters related to programming.

There are many ways to compute the frequency of characters in a file. Because of the amount of data to process, it's well worth picking a good method. And comparisons of performance of various methods should be made before investing a lot of computational time — intuition is notoriously bad for such things.

Looking at the numbers in versum sequences, we noticed frequent occurrences of fairly long runs of the same digit (another subject for further investigation). Consequently we thought that using string scanning and counting runs in a single step might be faster than a more straightforward method. Here's the first procedure we tried:

```
procedure digitcnt(file)
   local result, c, i

   result := list(10, 0)

   every line := !file do {
      line ? {
         while c := move(1) do {
            i := (*tab(many(c)) + 1) | 1
            result[c + 1] +:= i
            }
         }
      }

   return result

end
```

We decided to return a list, since that seemed to be the most convenient form to use. We could have combined the last two lines in the while loop but that wouldn't have made a significant difference in performance.

Incidentally,

```
every line := !file do …
```

produces the same results as the more familiar

```
while line := read(file) do …
```

We used the latter originally but changed it for presentation here to compare with the code in an alternative procedure.

If we'd just stopped there, we would have wasted a lot of computational time later. Instead, we tried writing the most basic but compact method we could think of:

```
procedure digitcnt(file)
   local result

   result := list(10, 0)

   every result[!!file + 1] +:= 1

   return result

end
```

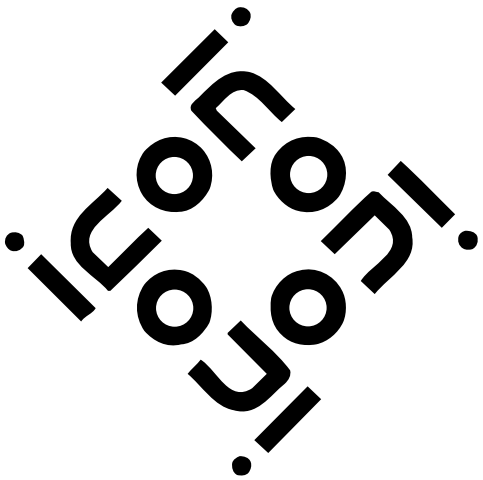
It was worth the effort; the second procedure is more than twice as fast as the first for versum sequences. (The results might be quite different for other kinds of data.) The reason for the poorer performance of the first method is not just the use of string scanning. It's mostly because of the extra steps in the computation.

Having chosen a method for counting digits, we went on to examine the digits in the first 20,000 terms in the versum sequence for 196. We expected interesting results because of a remark about this sequence made by a person who spent some time

Icon on the Web

Information about Icon is available on the World Wide Web at

http://www.cs.arizona.edu/icon/www/



on the 196 problem: “A small sample of actual numbers was overabundant in 1s, 5s, 7s, and 8s …” [3]. Indeed, looking at the sequence, this seems to be true. An actual count performed on 20,000 terms shows something quite different:

0	8515005	10.380%
1	8067614	9.835%
2	8171275	9.961%
3	8040019	9.801%
4	8218323	10.018%
5	8220543	10.021%
6	8038674	9.799%
7	8173337	9.964%
8	8068533	9.836%
9	8514972	10.380%
total:	82028295	
digit average:	4.500	

The digit averages for individual terms in the sequence vary somewhat — as they must — but they never stray far from a flat distribution.

The digit frequency for the versum sequence of 169 — just a reordering of the digits — is quite similar to the one above. Unlike 196, 169 produces three palindromes early on:

1130
➸ 1441
➸ 2882
5764
10439
103840
152141
➸ 293392
…

It would seem that 196 just “got off on the wrong foot”. Actually, there’s something to that notion. We’ll explain in the next section.

9679965784387555443279730005474692024737713580900809774183613739888914573995108557731848439025133906360689346289445190574120630207548596723903681330420362601753311308169319832416516329260057905200687925330264021388622163771595399738001310754358170989759100222870783230334283045189548956313958789995185600269507118976572683597452049180407326226672012682269155802393083227920422615986176937064916287684453975512510087825262790033769833596486960710622306806806954673196197425688199282531840443267117917077516285768482322553106590059974535677335645189998467660433856930753561177968012639857967395209757843681895023948424981221127979462941390214063066049604759953515181426047362066005200170241443100204339131988343044503800259404222000958807269209331604860081418943121102264338866702766831071588530949980921985976851793920229123587503040769280204622266734167703627123801595382774261514573630267023337493146255700070509185172895219970887446290525260919289871844202204044607000122408305000126767585153572442143241043152358587394851851031727310860717914643240873489092652779312269315167344537685819868325006738634045693406622880812750483577531139899719518441903767069772890965374599450618306831815796530344039100929265431687171327464202726153805340721294440851766310470359059986081971313108712807799932435977769948220008934336237208031046675705745072854487321033845572628457240346013996511322050117310853747843766020895835414510474688543798625384557938795684814283449337726604646570551623936600995455027867310806886202710987929776406635225113797908835591966889084870625918464845231439428320761997483737514391884608620482589564007860039148999406439491434337228077594814272051793395840502767615087746820862682792169252907629149186440040914913626203290526716579487131936597374733805343777944227781846174451408287670015464487797721962063940952994629995839929194756365606693495897406568322160050208780456038119401000501812021737545591495113762737594207576851843997482312005887251390425161406818515991918911734768367775699837248089793401724697904019602543133777966557135856786709800482461213956799741538504968912705266599165959959540433772059100833425500804766422698024320799357535793669765336205400251239118795887912444767338287016517502938689558987129360286661067310749334646486625034554034746461748428050041869362163886231661452201718876989184956360656598754412605332545605911867731235398820097220525204983467592131757137528038958356570877632262356669241918400646425554572702495447567608931202757898188902964348224718634825938533191472231074512092292955722654273158648767305688491472039115529000837216097171591938675403080614612614210129069912017597622749044273319115758899103009447247080631675464411849119963797894044401043033912880130462223515620617486609542650604737617533885020124510011244490913532554926834807183366852773067831141657518369664822786089669944768878127177758001707512648087258640424722874077124092313765299735602430864524440389471963920406243016630187379155179047602995090230360390825714020797991633504661603330879952528860271106524013743987446044137280299164909171133362783070048265374515916197075618148734642481060416449304755478750946237262953402877773806300917996269394912834469743128091335498981950021638777955377959331545185238419989577875920703819951675553084826499832665549331570970147599915168914804077132907750224826020485614100883992568336288974083556575206326530285786069134802411131569046414328336865605204589506875358975344801197245919353067206591044000678354156308502472573043354222315678153235752444609464102850203123752961131234448578085584028081334059698538755277279551558739289155334837582222675006366938326775890851273495675122211283531852320596809131170711475579751383930871337761895529658538463304371992255641095978459342855470829113959760576595373911998133516767647755930449265279775577776168855417363521089507463079339260366377174974073817551113590853219242537600804079130370271454468879158172075128195715547917495003558808989767896796444687424255513559315437101796434090219797802320106479744623107994639295954504476599365322911988717230399727574771501576019134871474847378983950231493808747385641695309195850051883767170040327068892291234640066635044605829463998102264479747001332086879130904337081006455129553165415248754546085888779890808756299495710736617591821461370861977765444172074021970471799682425291146798631115581738036958077366395293407927460680902645381345897157777576797167195393065774577515232998020473495585969858211929064667144053880591135660120013412274834856935499205773427793037316798547500718102201759513226714538200222168648448116798868751483066459958611239563834455089293785615497187735577357969504440817204955808648455311410593574202031482914659063442675313509756122235533393743733049026623547770003401066016713528206317921975535699534696949954015065785447145454086623011320953086079757212473359167574538047997273286528949799042757411952842206770933167139741997041999684096907514395556328005628381355466268918406920579774890024733471645233959773559677461299502898846330918104469654381205928626997190136182787771024469251733738068874557392955505160085136448731826480702510604573651851069297263331180908462002972732430644888457309424711063168814260088033305066515335099798120407519093964031981699195849981550973871135520253604018469273993043424469025305537002576223190530779388226425135962780736225617109768770621987876349977979598128377864716452511288593871585644707095376284661352191053430011045200215893347158364071552549066846270166153112640419882293293490144503888073689119481243646751361896428538004010087675110232624508582267957091200698220024262263169813045757401951817906018381009255118212641948875938578468612635516218659193920143810331741023357296374467074337344602988927997572020309056668336042073744655146469938201438665641523467671757429599297357317581302056752805014359236909189035312377582191553622351720444899555705266057098978771819234425524168846017396724994093385715374654044453942668646475339470127700567919540218798569758402958047197818436773542297776078219222519946026325769752985357530079243208061246684971154154380009591873330460058595619956635972108600495383267585507587966030226329499791857865854174477878723125626059194088073181934088718328379966876538684281109181005167185052615315193141789600202284799433826758571140672726830060519455464703021720411009402173056498879204006221485669380675052876956636574019399285900353992600404592790287977734755100658829041533825392877124408762435084364738056382417740855186250814027263095091399346820410268091430603072852680286487715077582940486042971592715184867717237244410483470400094039996970036589518502681549709351572748488915811293483413155846382061606849088965919553891979731151164550566798396791162026887179136697215446880664042516507554740651883284537242938649784085448361699625688548391631454749911967723974835791470015923311459941955394364592528554734002368555816048571746773913971362362443090002283907777953314000771721691131317917958996194317391455825793548312794450726251829156383217079702556392899093154302569850714860282704399646455910818787077620914482591700993113577537404821897822660439754044573770952386882858584533485161395220397725629108348794225642970606910381623916705959279585325134004324213428635157585761209940489323200069545139130243926809182005362508175478796991259837157190518000754263239573331976202648531615257727359610822162739775253666122651197296813920678522202112839716858058022908994893579607113756829666993246131102233881428996840612390297260994800032140495200940435025478023084440200123523207199260065926373171417162535895750584066047030308414035408862221118952484032148828633975791150475975993521987877116624802976284339568649998814574337855344800599856023561132957586816256717087117733330482353720027864346927912675497795096404216228058684706327978329998163428790004226579244487772619470639781580615125020713479304197661952286220166622732713972050243795486374678911795061006681698997750413550836080551371542923477069311002067879081763457102209748982606068351226993120452122629875101619750062824515524138914061903102457195273915133176398426706844702037020385992533993534096063608340431933858038755791699375310889926415490476918999086317627529197563500047981255555683398558986 8

20,000th Value in the Versum Sequence for 196

Before we go on, here's a sanity check — a count of the digits in the first 20,000 terms of a plain addition sequence for 196 (no reversals):

0	4878175	10.007%
1	4876763	10.004%
2	4878850	10.009%
3	4872249	9.995%
4	4877423	10.006%
5	4872388	9.995%
6	4875460	10.002%
7	4872088	9.995%
8	4876302	10.003%
9	4864261	9.979%
total:	48743959	
digit average:	4.498	

Notice that without reversal, the length of terms in the sequence is considerably shorter than with reversal. There's a significance to this also.

Why Digits Count

The reason the digits in a number are important is that the reversal sum of a *n*-digit number is an *n*-digit palindrome if and only if there are no carries in the addition. (If there's a carry that produces an *n*+1-digit number, the result can be a palindrome, as in 605 + 506 = 1111.) Consequently the probability of a reversal sum producing a palindrome depends in a major way on the average of the digits — if it's high, the chance of a carry is large. When there are many digits, the chances of getting a palindrome are substantially reduced. The probability of getting a palindrome also depends on how the digits are distributed; they must "pair up" in the right way.

Assuming each different digit occurs about the same number of times as any other and that the digits are randomly distributed, the probability of a *n*-digit number *ever* leading to a palindrome has been estimated to be

$$10.222 \times 0.55^{(n/2)}$$

For $n = 400$ (a modest value in the land of versum sequences), this works out to be 1.2×10^{-51}, a mighty small number [3].

We have some reservations about the argument that leads to the formula above, since it omits the derivation of some intermediate results that are not obvious to us and has some evident errors, such as overlooking the fact that a palindrome may result even if there are no carries, as illustrated earlier. However, the general concept seems to be correct. Note, however, that it depends on the assumption of the random distribution of equally probable digits — which certainly is not true for many numbers in versum sequences.

Nonetheless, if you look at large numbers in versum sequences, it's easy to see that it's unlikely there will be no carries as the result of a reversal sum, and that the probability of carries in *n*-digit

The Icon Analyst

Madge T. Griswold and Ralph E. Griswold
Editors

The Icon Analyst is published six times a year. A one-year subscription is $25 in the United States, Canada, and Mexico and $35 elsewhere. To subscribe, contact

Icon Project
Department of Computer Science
Gould-Simpson Building
The University of Arizona
Tucson, Arizona 85721
U.S.A.

voice: (520) 621-6613

fax: (520) 621-4246

Electronic mail may be sent to:

icon-project@cs.arizona.edu

or

...uunet!arizona!icon-project

THE UNIVERSITY OF
ARIZONA ®
TUCSON ARIZONA

and

Bright Forest Publishers
Tucson Arizona

numbers producing n+1-digit palindromes is very small. Contemplate the number shown on page 10.

If the probability of a palindrome appearing in a versum sequence drops off quickly as the number of digits increases (about once every 2.3 steps for the sequence of 196), it's easy to see why palindromes might only occur in the early parts of versum sequences. "Getting off on the wrong foot" appears to have "doomed" the sequence for 196.

Next Time

We have a few more topics to discuss before leaving versum sequences. One is how many n-digit seeds produce identical versum sequences and how to identify them. Another interesting topic is the merger of versum sequences that start out differently but come together at some point. Last on our agenda is the question of what numbers occur in versum sequences. Not all occur, certainly; 1 and 3 are simple examples. (Recall that the seed is not part of the versum sequences it produces; otherwise the issue would be vacuous.)

These topics raise some interesting programming issues, which we'll discuss along the results we have.

References

1. "Palindromes by Addition in Base Two", Brother Alfred Brousseau, *Mathematics Magazine*, Vol. 42, November 1969, pp. 254-256.

2. "On Palindromes', Heiko Harborth, *Mathematics Magazine*, Vol. 46, March 1973, pp. 96-99.

3. Letter from E. Alan Phillips to Allan J. Gottlieb, September 20, 1979.

A Word of Thanks

Gregg Townsend reads every issue of the Analyst before it goes to press. He does an excellent job and with amazing quickness. Over the years he's caught many errors and saved us much potential embarrassment. What's more, he often makes valuable suggestions for general improvements.

We want to take this opportunity to express our sincere thanks to Gregg for all his help; help that benefits not just us but all our readers.

Random Numbers Revisited

Carl Sturtivant sent us a long, detailed, and interesting letter about Icon's random number generator and linear congruence generators in general. He was motivated by the puzzle we posed in the Issue 29 of the Analyst [1], but he covered a number of other topics.

He first noted that we gave the wrong value of the modulus in an earlier article [2]; it should have been 2^{31}, not 2^{31}–1. Our mistake came from having used logical AND ing rather than remaindering but changing to remaindering for exposition without correcting the number.

He then went on to comment about the properties of Icon's random number generator and some of the defects in it.

He concluded with a proof that the kind of regularity posed in our puzzle was to be expected and, in fact, is a property of all linear congruence generators with moduli that are powers of 2.

His letter contains a lot of mathematics, but his discussion is very lucid and easily understood. If you'd like a copy of his letter, let us know and we'll send you one, free of charge.

References

1. "Curiosity or Problem?", Icon Analyst 29, p. 6.

2. "Random Number Generators", Icon Analyst 28, pp. 4-6.

What's Coming Up

We didn't have room in this issue of the Analyst for another article on dynamic analysis. We'll give that priority for the next issue.

We also plan to have another article in the series on visual interfaces and a concluding article on versum sequences.