bombs inside Petersburg. Brasilia has a hot but equable climate. Petersburg has floods, subsidence, sweltering summers, and merciless frost heave. Brasilia is run down in places and has its slums, but the level of decay in Petersburg is truly gothic. Brasilia is young, but at least it knows what it stands for. Petersburg has a full-blown identity crisis. In the past century it's been Petersburg, Petrograd, Leningrad, and then Petersburg over again. And now it's not even a political capital any more.

But Brasilia's just not very interesting. Petersburg has real magic. ■

***Bruce Sterling** is a writer, editor, and net activist. He was co-founder, with William Gibson, of the cyberpunk movement, and his face was on the cover of the first issue of Wired. He can be reached at "bruces@well.com".*

U&lc Online is the companion publication to International Typeface Corporation's quarterly magazine, U&lc (Upper and Lower Case: The International Journal of Graphic Design and Digital Media). You can visit their web site at "www.itcfonts.com".

The Chronology of Y2K Problems

*— by **Clement Kent***
Toronto, Ontario, Canada

THE CHRONOLOGY OF Y2K PROBLEMS is more varied than most people recognize. Many have of course already happened, like the woman in England who recently received notice of her insurance policy which would expire in 1900, or the various credit card processing systems which last year failed to accept cards expiring in "00".

By the time you read this, two much discussed failure dates will most likely have passed. Numerous Cobol systems are reputed to have used "9/9/99" as a null or error value in date fields. When these systems encounter September 9, 1999 in reality, they may suffer from a form of cognitive dissonance. While not precisely a Y2K problem, this is obviously closely related.

Similar, but not really a Y2K problem, is the date rollover in the Global Positioning System's date field in August 1999. When specified in the 1970's, space-hardened memory was expensive, and the system's designers chose to implement dates as week number since the beginning of 1980 plus a day number. The field chosen to hold the week number has already overflowed by the time you read this and it is now officially January or February 1980, according to the GPS satellite time. This won't effect the actual location function of GPS receivers, but it is an interesting quirk to compensate for on those large networks which use the GPS signal as a universal clock to synchronize remote net members.

At last,
perhaps sometime in 2001 or 2002,
most of the Y2K bugs
will have been found and fixed.
And then. . .

That takes care of August and September, 1999. October through December will most likely be enlivened by a little-discussed, by likely quite prevalent, sort of bug: the Y2K rollout bug. Already certain automated teller machines in my home town were unexpectedly down due to difficulties encountered during the rollout of the new, Y2K-resistant banking transaction software. The interruption of service was short, but indicative of things to come. Of course, the organizations rolling out Y2K releases now (1998 and early 1999) are the well prepared ones. The ones whose rollout is deferred until Q4 1999 will be disproportionately those for whom the Y2K project was a death march project. Even if all Y2K bugs have been eliminated from such software, the odds that rollout of large systems by a harassed death-march project crew will be flawless approach zero.

December's bills will of course have the "*Pay by Jan 05 00 to avoid interest penalties*" somewhere on the form. I am looking forward to the first Christmastime reports of bills threatening 1% per month interest penalties if the accounts are not settled by Jan 1900.

January will of course be the time of peak interest and awareness. It is likely, however, that many bugs will crop up after January 1 00, usually because the software isn't exercised until later (the weekly, monthly, quarterly, semiannual, or annual jobs).

To add to the gradually decreasing background noise of periodic jobs failing, we may expect another small peak on or around Feb 29 00. (See the article on *"Pope Gregory on Mars"* on page 16). In my APL experience, the ratio of century bugs found in Y2K remediation to leap year bugs found, over a large code base, was about 10 to 1. I suspect that the field failure rate will run more like 8 to 1, since people and their testing efforts tend to be a bit more focused on the century problem. The potential for serious failures is of course as large, per bug, from leap day bugs as from century bugs: even a single failure in a single routine can bring down a large system. There are several documented examples already of the impact of the much more stupid mistake of not recognizing ordinary leap years. Ulrich and Hayes (*The Year 2000 Software Crisis*) cite the example of a state lottery system which could not process any tickets on Feb 29, 1996, and lost all of the revenue for the day. Yourdon (*Time Bomb 2000*) gives the more catastrophic example of the New Zealand aluminum smelting facility which lost microcomputer control of its crucible heating systems at midnight of Feb 29, with resulting permanent damage in the millions of dollars. The frantic night shift only began to realize what might have gone wrong when they got a 1:00 a.m. distress call from a smelter in Australia (one time zone away) whose similar system had just gone down.

Nevertheless, Feb 29 and March 1 are easier target dates than Feb 1, March 31 or Apr 1 or July 1. I suspect rare longer-period jobs (monthly, quarterly, etc.) will show one of the highest ratios of in-field downtime to pre-remediation bugs, simply because most organizations have had to focus testing effort around Jan 1 and Feb 29. Often putting expensive test systems through the many other conceivable failure dates simply could not be cost justified.

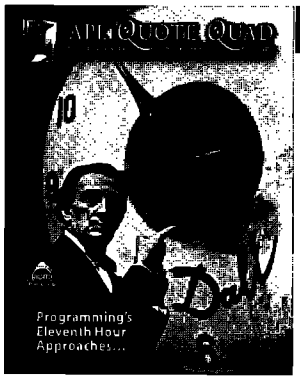
At last, perhaps sometime in 2001 or 2002, most of the Y2K bugs will have been found and fixed. At this point we can begin thinking about the 2011 bugs, the 2038 bugs, and so forth. It seems absurd to think that the versions of Microsoft Excel which incorrectly represent dates past 2011 will still be in use twelve years from now, but it seemed absurd to us two or three *decades* ago that any of the mainframe systems of the time would see the millennium.

Most absurd of all is to think that the new, modern, replacement languages and operating systems of the present have similar structural date bugs built into them. Yet quite a few flavors of C, Unix, etc., have dates represented as the number of seconds since an arbitrary date in the 1970's, stored in a four-byte integer. Most of these overflow some time around 2038. I hope to live long enough to laugh at the hapless multitudes trying to find and fix all the toasters with faulty C code in them before 2038.

Last, and perhaps least believable, will be the failure in the second half of the 21st Century due to our Y2K bug fixes. Many Y2K bug fixers, myself included, have been guilty of using a fixed windowing or pivot date to solve century bugs. That is, we kept the two digit year, but added code that said "*yyyy*← 1900 + *yy* + 100 × *yy*<50". This treats "50" as "1950" but "49" as "2049". Various pivot dates have been used, and one can only hope devoutly that half a century is long enough that all of this "fixed" legacy code will be on the junk heap before the window expires. ■

***Clement Kent** has been a happy user of APL since being exposed to it in high school in 1969. He consults in the areas of software engineering and project management in the financial services sector, as well as the application of software to computer integrated manufacturing for semiconductor companies. For the last three years his primary involvement with APL has been in leading Y2K projects. He is currently on the Executive Committee of the Toronto APL Special Interest Group, and organized the November 1998 symposium "Challenges in Medical and Biotechnology Software" which highlighted some commercial APL products in this field. He may be reached at "clementk@acm.org".*

Illustration by Jan Hicklew



COVER STORY

Y2K and APL—An Overview

*—by **Clement Kent***
Godel Computer Solutions Ltd.
clementk@acm.org

I HAVE WORKED ON Y2K PROJECTS IN APL since July of 1997, which makes me a sort-of-expert. My goal in this article is to help those of you still in the throes of Y2K work, and to give the rest of you—those who have completed your Y2K work—a basis for evaluating your own projects. Of course, if you haven't yet started your Y2K project, you're doomed, and you may as well stop reading and take up dental floss farming in Montana.

My Y2K experience has been with large applications on mainframe APLs, although I have experimented with some vendors' PC APLs to understand the issues there. Whether you have a large or small application, on mainframe, mini, or PC, you still need to understand some Y2K issues common to all these environments.

Is Y2K a problem for APL?

In a word, *yes*. Why?

The real answer is, *human nature*. Although I have worked on Y2K for a subjectively very, very long time now, and am quite sensitized to its issues, when I make notes in my personal diary I still write "meeting Jan 27 99" and the like. Writing the '19' is just too tedious.

You will read many articles that explain in excruciating detail how expensive memory or disk was in the 60's or 70's or 80's (note we don't say "nineteen eighties," do we?). But programmers, even quite good ones, continue to create code which leaves out the century, even in the 90's.

In one customer's Y2K project we looked at a number of large applications, none more than ten years old. Memory or disk wasn't a real issue for these programmers any more. Yet, almost all of these applications had some problems when placed on a test machine which was advanced to 01/01/2000, and several of the most critical ones flat-out died.

A second, darker side of human nature provides an equally important reason for doing your APL Y2K project. This is the human need for a scapegoat when things go wrong. In some countries, especially the United States, lawyers and the media facilitate this need and earn big bucks in the process. What they earn, you or your customer pays for. So even if you have strong reason to believe your application never ever, honest to Bob, used any non-compliant code, you'd better prove it prior to the big '00'.