The Centinel Data Format: Reliably Communicating through Time and Place

Clarence Lehman[1], **Shelby Williams**[2], **and Adrienne Keen**[3]
[1]University of Minnesota, 123 Snyder Hall, 1474 Gortner Avenue, Saint Paul, MN 55108, USA
[2]University of Minnesota, 100 Ecology, 1987 Upper Buford Circle, Saint Paul, MN 55108, USA
[3]London School of Hygiene and Tropical Medicine, Keppel Street, London WC1E 7HT, UK

"A library book lasts as long as a house, for hundreds of years."
—Thomas Jefferson, 1821

Abstract—*A common experience among scientists and engineers is storing and sharing data, the capacity for which has advanced immensely since laboratory notebooks were only paper and ink. However, since that time, the sustainability of data has decreased. Even though our digital data should be safer and more secure than ever, a continuing cascade of obsolescence in computer media and software can actually make it less so. Here we outline an ensemble of free tools and techniques that we call "Centinel," designed to manage, communicate, and archive digital datasets. Rather than embedding error-correcting codes as part of the computer media, Centinel exposes them and places them with the data and metadata. Thus even printed copies of the data form reliable storage media that can last indefinitely without intervening attention. Centinel complements standard methods for data sustainability, such as data migration. Unified approaches, as we outline here, benefit reliability and longevity of data.*

Keywords: database, data archive, data longevity, data reliability, error correcting codes

1. Introduction

In 1815 began one of the largest scientific data collection projects ever launched [1]. Legions of surveyors walked regularly spaced transects along 2,500,000,000 meters of the Louisiana Territory, recording the biological species, geographic locations, and diameters of selected trees near periodic sample points—plus other information on soils, vegetation, and boundaries of wetlands. For almost a century the survey continued. Now, another century after the last data were recorded, the results form one of the most visible efforts ever, organizing the rural landscape into square sections along those transects. The results also form one of the best preserved and widely available datasets ever. Think of which present datasets, in your personal experience, are guaranteed to be extant and usable well into the 22nd century.

A large part of the reason the survey data survived was that it was recorded on paper and protected at many different governmental sites. In the meantime, technology changed immensely. Computers emerged and increased in capacity so relentlessly that the Library of Alexandria's ancient charge of organizing and cataloging all human knowledge began to draw within reach. Global access to digital data can make that knowledge available to all. Large-scale private enterprises are aiming at this goal, but individuals in academia and industry are established sources of knowledge and therefore have a special role in achieving this.

Here we are addressing that role—of scientists, engineers, and others who collect empirical data, share it, and want to preserve it for the future. In this report we explain how digital computer techniques of today combine naturally with paper methods of prior centuries to create a form of digital storage that can reliably persist into future centuries and improve electronic processing today.

2. What Centinel is and is not

The general topic that Centinel addresses has been long discussed (e.g., [2] [3] [4] [5] [6] [7] [8] [9]) and a complete solution is not yet available. Centinel combines the words "century" and "sentinel," guarding data for extended periods. One goal for Centinel is to ensure that the digital data it encodes will be accessible in a century or more, without the need for care and intermediate steps by humans. A second goal is to protect data over a shorter term, from the time of initial creation to the time of final processing. Centinel works by (1) keeping all metadata with the data, (2) protecting data with line-by-line error correcting codes, (3) providing a format easily readable by humans as well as computers and scanners, (4) supporting a reliable digital format that works on any media, including paper and verbal communications, to protect data from unintentional alteration, and (5) supplying an extensible, self-defining format with accompanying tools that help computer programmers know that the data entering their programs are correct. Centinel is an approach to data management, but also a set of basic computer utilities for writing, reading, editing, separating, joining, ordering, and aligning data. It avoids structures that are error prone

```
6674844762232577  Keyword SpAbbr: Abbreviations for species names. Abbreviations contain the
0629561874138616    first three letters of the genus name followed by the first three letters
0211050455008008    of the species name. The full species names are recorded with their
5515307245627135    abbreviations in table "species codes" at the end of the chapter.
5915322805104717  Keyword Date: Date species was collected. Format year-month-day.
1453182442695072  Keyword CollID: Unique code assigned to species sample collected.
1382423906566782  Keyword Cover: Estimated canopy cover, in percent. Dashes indicate missing
5953391885352618    data. (See "methods" at the end of the chapter.)
0748783303437946  Keyword HtMax: Maximum height, in meters. Dashes indicate missing data.
0229302812296440    (See "methods" at the end of the chapter.)
0602554115737437  Keyword HtMin: Minimum height, in meters. Dashes indicate missing data.
0229302812296440    (See "methods" at the end of the chapter.)
0000000000000000
1976160343505769  :Site :Code     :SpAbbr  :Date       :CollID   :Cover :HtMax :HtMin
4554847814214755  :1600 :P1600D04 :Abibal  :1989-08-21 :AMB00555 :    - :    5 :    5
2645745581124348  :1600 :P1600D01 :Abibal  :1989-08-21 :AMB00604 :    2 :    1 :    1
1076375677295808  :1600 :R1600EA  :Abibal  :1989-08-24 :AMB00666 :    3 :    1 :    1
2000445884315808  :1600 :R1600EA  :Abibal  :1989-08-24 :AMB00668 :    5 :    6 :    6
0582355170295008  :1600 :R1600EA  :Abibal  :1991-08-05 :AMB01719 :    2 :    2 :    2
1485325476235008  :1600 :R1600EA  :Abibal  :1991-08-05 :AMB01722 :    4 :    6 :    6
4100414960104041  :1600 :R1600EA  :Acerub  :1991-08-05 :AMB01503 :    2 :    2 :    2
5773084583093978  :1600 :P1600B01 :Agrsca  :1989-08-25 :AMB00456 :    3 :    2 :    2
4766066289426272  :1600 :P1600D01 :Amerot  :1991-06-17 :AMB01439 :    2 :    2 :    2
```

Figure 1. *Excerpt of a sample Centinel data file from a large ecological database, with metadata above and error correcting codes called "centinels" at left. Here colons separate columns rather than vertical bars. In the Centinel structure, error detection and correction stays with the data rather than with the computer medium.*

and supports good data management practices, for example as outlined in [10] and [11].

Centinel is not intended to substitute for large-scale interactive databases undergoing continual manipulation, such as in PostgreSQL, MySQL, or Access. It is, however, a good format for long and medium-term retention of such databases, as Centinel format can be readily exported from them through simple utility programs, and conversely, imported through conventional means or by scanning. Nor is Centinel intended as a complete solution to the problem of storing all data at national and international scales (e.g. [12] [13]), but rather as a solution for individual research and development groups to help maintain their data.

The Centinel format shown in Figure 1 supports the movement of data through place and time. A dataset documented sufficiently with complete descriptions as its metadata, and protected with error correcting "centinels," can be transmitted to another researcher in a distant place without separate documentation and time spent explaining the data, or equivalently it can be transmitted forward to another researcher in the distant future. In other words, it can be archived. Instead of error detecting and correcting codes being applied to the storage media, as is the common method today, codes in Centinel are applied to the data themselves, and stay with the data through all media changes. That simple but unusual characteristic fills a gap in existing data methods and provides confidence in the data across distant places and times. Multiple printed copies of the data can be stored throughout the world and scanned with optical character recognition in the remote future. The centinels, checked automatically against the scanned results, are the essential link to data reliability.

As in some other databases, Centinel has multiple equivalent formats, which we call "singular," "columnar," and "mixed." Long lines of data in singular format can extend onto new lines, indented as in Figure 1. Here is a simpler file in singular format:

```
Class:   1
ID:      123
Age:     21
Region:  SSA

Class:   1
ID:      47
Age:     7
Region:  UK

Class:   2
ID:      723
Age:     70
Region:  US
```

Below are the same data in columnar format:

```
 | Class   | ID    | Age   | Region
 | 1       | 123   | 21    | SSA
 | 1       | 477   |  7    | UK
 | 2       | 723   | 70    | US
```

And below is mixed format:

```
Class:  1
 | ID     | Age    | Region
 | 123    | 21     | SSA
 | 477    |  7     | UK

Class:  2
 | 723    | 70     | US
```

These formats are interchangeable. The choice is a matter of space, readability, and ease of processing. All software written to handle Centinel data should process the three formats equally.

Printed copies of data with error-correcting centinels need not be limited to small data sets. For example, the genome of the fruit fly (*Drosophila melanogaster*), represented with one base-64 symbol for each of its 47 million codons, would require approximately 6000 pages—not absolutely prohibitive to print for an important, expensive dataset. By comparison, the King James Bible is 4.3 million characters, about one-tenth of this genome, and more than one copy of that work has been printed.

3. How Centinel works

Centinel protects data when they are complete and ready to be archived. But it can also be used when the data are first entered, to guard against accidental modifications of datasets undergoing incremental change.

To explain how Centinel works, we must consider what it means for data to be digital. Two properties are essential. First, the data must be represented by "symbols" that have only a finite number of states. Second, the shapes of any two distinct symbols must be separated by a sufficient gap, so that a symbol for one datum does not, except very rarely, degrade into a different symbol for a different datum. Symbols can take various forms—binary 0 and 1 encoded electronically in computer memories are one example of digital data. The Arabic numerals 0–9 printed on paper are another. With these ideas in mind, Figure 2 shows analog versus digital representations of a function, $y = f(x)$.

An analog form on paper could take the form of a graph, Figure 2A. The value on the vertical axis varies smoothly, and can be read to reasonable accuracy with a ruler and a careful eye. However, each time the graph is copied, its accuracy diminishes. The curve becomes successively blurred, the right side may get slightly skewed with respect to the left, and so forth. In contrast, the entire curve in digital form is defined by coefficients, Figure 2B. When this digital version is copied by re-typesetting, it will not degrade, for the individual symbols will be recognized for what they are and reproduced intact. A new font may even change 'x' to 'x', but the meaning of the symbol will remain.

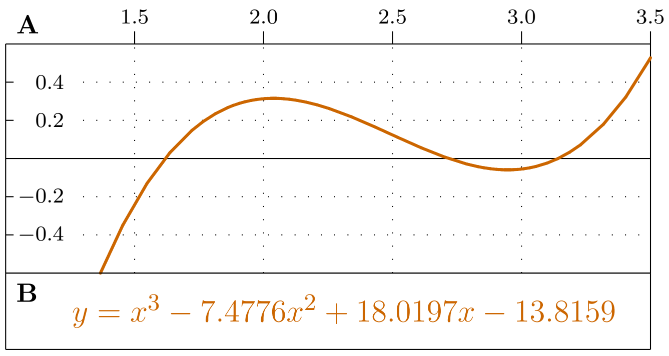

Figure 2. *Non-electronic analog and digital data for the same curve. Printed copies of the digital data (B) will not degrade over time as will the analog version (A) of the same data.*

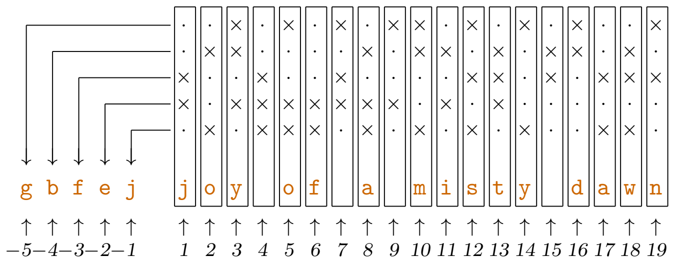

Figure 3. *Error-correcting "centinels" (left) for a 19-character message (right). Each centinel covers a distinct combination of columns, such that any unmatched centinels identify which column is in error and how to correct it. (See code in the appendix for details.)*

Thus digital data are not at all restricted to electronic media, but paper can carry digital data as well, and has done so for millennia. Moreover, some of the most common digital information read by computers today is recorded directly on paper, plastic, metal, and other substrates. The ubiquitous bar code is a case in point, though bar codes are not human-readable as Centinel-protected data are.

A significant separation between symbols in appearance or physical state keeps unavoidable small degradation in information from changing the message, because one symbol does not easily degrade into another. However, separation of symbols is not enough. For highest reliability, error correcting codes must be applied to the digital data to prevent rare alterations of one symbol into another from changing the message, except with negligibly small probability.

Centinel uses a "Hamming code" for arbitrary symbols, a generalization of the original code [14] for binary digits. Such codes we call "centinels," and they appear at the left of each line, at the end of each printed page, and at the end of each file. They can correct any single-symbol error in a line and detect any two-symbol errors. In addition, with high probability they detect multiple-symbol errors, including errors in the centinels themselves.

Each symbol is assigned a small integer and the integers for a given subset of columns are summed. The sum, modulo the number of symbols, is translated back to a symbol, as in columns -1 to -5 of Figure 3. This is repeated for carefully chosen subsets of columns which allow errors to be located and corrected. Then the results are translated to decimal form, as in Figure 1, to mask the actual random combinations of symbols, which by happenstance can spell out any word.

Complete details are in the Centinel algorithms (appendix). These details are part of the metadata and should be included with archived data.

4. Comparison with other approaches

A standard approach to data archiving is a rigorous effort of continually transferring data from old media and old software to new, before the old media and software become completely obsolete—keeping the data "alive" so to speak.

That is called "migration" [12]. It is a practical, well-tested method, though it can be labor intensive and susceptible to catastrophic failure.

Successful migration requires a central discipline maintained over long periods. Any lapse in the chain of migration will result in the complete loss of data. Successful migration will be practical for large, well funded data sets. However, for many small data sets, discipline and funding can easily lapse over long periods of time.

Timing is key, as migration must take place while (1) machines that can read the media still exist, (2) programs encoding the information are still operational, and (3) the media and the information stored on it have not deteriorated.

It follows that the best chance of success in data preservation will be for (1) media that require no advanced or specialized machinery to read them, (2) formats that require no complex computer programs to process them, or at worst require the simplest programs that can be described completely in a few pages of text, as in the Centinel algorithm (appendix), and (3) media and encoding methods that will themselves last a century or more. Centinel allows data preservation with a single migration.

A second method is called "encapsulation." Fully successful migration to new media will be worthless if the software that accesses the data ceases to exist. For example, an organization producing software may go out of existence and no other organization may support the old format. This has happened repeatedly in the history of computing. Encapsulation aims to include with the data all software that accesses the data, in a form that can be translated to future machinery. That is, of course, easiest when the corresponding software is as limited as possible.

Two other methods proposed for data archiving are "emulation" and "technology-preservation." In emulation, the complete hardware and software architectures to retrieve the data are migrated forward with the data and "emulated" on the future system. That practice was widespread and successful among mainframe computers in the 1960s, where one generation of computers would emulate the hardware of the generation before. But as computers become increasingly complex in their architecture and operating software, it becomes difficult to make this practical into the indefinite future.

In technology-preservation, the actual hardware and software is preserved, museum-style, along with the data for future access. This is problematic, however, for today's computers are built for the moment, not built to last, and may not even boot up properly after a decade of disuse.

Therefore, emulation and technology-preservation are not related to Centinel, but migration and encapsulation are. Centinel implements encapsulation in the simplest form—under 100 lines of code (appendix)—and with a single migration, creates digital documents that last as long as possible—up to a century or more.

5. Suggestions

In conclusion, we offer the following: (1) To keep electronic data safe, prepare early for archiving. (2) Archive data in the simplest formats possible. (3) Document data to the highest standards. (4) Associate documentation directly with the data it describes, ideally in the same file. (5) Keep multiple copies in separate locations. (6) Regularly convert working files from proprietary databases to archival format. (7) Keep printed copies of critical data, with Centinel-like guard symbols and documentation for future recovery.

For full details and utility programs supporting this project, see www.cbs.umn.edu/centinel.

6. Acknowledgements

We thank Eville Gorham, Jan Janssens, Todd Lehman, Eric Lind, Richard McGehee, David Tilman, Richard Barnes, and all others who lent help and encouragement during this ongoing project. This project was supported in part by a National Science Foundation LTER grant to David Tilman and by a University of Minnesota database grant to Eville Gorham.

References

[1] L. A. Schulte and D. J. Mladenoff, "The original US public land survey records. their use and limitations in reconstructing presettlement vegetation," *Journal of Forestry*, vol. 99, pp. 5–10, 2001.

[2] J. Rothenberg, "Ensuring the longevity of digital documents," *Scientific American*, vol. 272, pp. 42–47, 1995.

[3] A. Waugh, R. Wilkinson, B. Hills, and J. Dell'oro, "Preserving digital information forever," *Proceedings of the Fifth ACM Conference on Digital Libraries*, pp. 175–184, 2000.

[4] D. Butler, "The future of electronic scientific literature," *Nature*, vol. 413, pp. 1–3, 2001.

[5] C. Tristam, "Data extinction," *Technology Review*, vol. 105, pp. 37–42, 2002.

[6] K.-H. Lee, O. Slattery, T. Lu, R. McCrary, and Victor, "The state of the art and practice in digital preservation," *Journal of Research of the National Institute of Standards and Technology*, vol. 107, pp. 93–106, 2002.

[7] S. Ong, "Worm storage is not enough," *IBM Systems Journal*, vol. 46, pp. 363–369, 2007.

[8] U. Duerig, "High density multi-level recording for archival data preservation," *Applied Physics Letters*, vol. 99, p. 023110, 2011.

[9] J. Marberg, "Towards SIRF: Self-contained information retention format," *Proceedings of the Annual International Systems and Storage Conference, Haifa, Israel*, 2011.

[10] E. T. Borer, E. W. Seabloom, M. B. Jones, and M. Schildhauer, "Some simple guidelines for effective data management," *Bulletin of the Ecological Society of America*, vol. 90, pp. 205–214, 2009.

[11] M. C. Whitlock, "Data archiving in ecology and evolution: Best practices," *Trends in Ecology and Evolution*, vol. 26, pp. 61–65, 2011.

[12] S. Rabinovici-Cohen, M. E. Factor, D. Naor, L. Ramati, P. Reshef, S. Ronen, J. Satran, and D. L. Giaretta, "Preservation datastores: New storage paradigm for preservation environments," *IBM Journal of Research and Development*, vol. 52, pp. 389–399, 2008.

[13] H. Heslop, S. Davis, and A. Wilson, "An approach to the preservation of digital records," *National Archives of Australia, Link at http:// www. naa. gov. au/ recordkeeping/ er/ digital_ preservation/ summary. html or http:// www. naa. gov. au*, 2000.

[14] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 26, pp. 147–160, 1950.

[15] B. Kernighan and D. Ritchie, "The C programming language," *PrenticeHall, Englewood Cliffs, NJ*, 1978.

7. Appendix: The Centinel algorithm

The complete algorithm that encapsulates Centinel files is given here in a subset K&R C [15]. The material below, together with Kernighan and Ritchie's book, should allow the algorithm to be transcribed into future programming languages and the data to be extracted from Centinel files as long as the printed form is extant.

The algorithm adds an error-correcting code to each line of a text-based file, another to each page, and a third to the entire file. Each output line begins with a decimal error correcting code guarding that line, and also guarding the error correcting code itself, then the text of the line. In printed form another decimal code guards the entire page and a third guards the entire file.

In computing the error correcting code, leading and trailing white space is skipped, multiple blanks count as a single blank, and end-of-line codes are not counted. The code at the beginning of the line is not counted either. The assignment between symbols and numbers is specified in array *s* below, where 'a' is number 1, 'b' is number 2, 'A' is number 27, and so forth. Any similar assignment could be substituted.

In the algorithms below, flow control and reserved words are bolded, variables and function names are italicized, and certain operations such as '<=', '>=', '!=', and '==' are displayed in a mathematical form as '$\leq$', '$\geq$', '$\neq$', and '$\equiv$', respectively.

DATA STRUCTURES

```
#define C       256                             1. Maximum character code plus 1.
#define L       120                             2. Maximum data length, excluding guard symbols.
#define G       8                               3. Number of guard symbols.
#define COL     9                               4. Number of symbols columns displayed on the page.
#define PAGEL   50                              5. Number of lines per page.
#define IDENT   127                             6. Identity symbol.

char s[] =                                      7. Character set available for present application.
  "_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
  " .,;:!?+-*/\\=\"'()[]{}<>^&%|";

int nchar;                                      8. Maximum number of characters in present application.
char seq[C];                                    9. Sequence number for each symbol in the set.
char f[C][C];                                  10. Modulo sum and difference tables.
char ptn[L][G + 1];                            11. Pattern of guard symbols for each position.
int pagef = PAGEL;                             12. Number of lines on first page.
int pages = PAGEL;                             13. Number of lines per subsequent page.

int ipage = 0;                                 14. Page index.
int ifile = 0;                                 15. File index.
char in[L + 1];                                16. Input line.

char line[L + 1], page[L + 1], file[L + 1];    17. Current line, page, and file.
char guard[G + 1];                             18. Guard symbols, individual characters.
```

END OF PAGE

Upon entry to the algorithm, (1) *page* contains a list of symbols representing the current page. **(2)** *ipage* indexes the next entry for the page. **(3)** *a* is set if a blank line should follow the code, indicating end of page. (This is not used on the last page of the file, because the code for the entire file follows immediately.) **At exit, (1)** Guard symbols for the page are displayed. **(2)** *guard* is destroyed. **(3)** *ipage* is set to zero.

```
seqpage(a) int a;
{
   if (ipage ≡ 0) return;
   page[ipage] = 0; ecc(guard, page);
   seqn(guard, "; ", ""); if (a) printf("\n");
   ipage = 0; }
```

MAIN PROGRAM

```
$main(argc,\ argv)$ int $argc$; char $*argv[\,]$;
{ char $c$; int $i,\ j,\ k$;
```

```
    if $(argc > 1)$
    { $pagef = atoi(argv[1])$;
      if $(pagef < 2\ ||\ pagef > 100)\ pagef = PAGEL$;
      $pages = pagef$; }
```

1. If an entry parameter has been supplied, take it to be the page length.

```
    if $(argc > 2)$
    { $pagef = atoi(argv[2])$;
      if $(pagef < 2\ ||\ pagef > 100)\ pagef = PAGEL$; }
```

```
    $s[0] = IDENT$;
    for $(i = 0;\ s[i];\ i = i + 1)\ seq[s[i]] = i$;
    $nchar = i$;
```

2. Determine the number of symbols in the set while developing a list of sequence numbers.

```
    for $(i = 0;\ i < C;\ i = i + 1)$
    for $(j = 0;\ j < C;\ j = j + 1)$
      $f[i][j] = IDENT$;
```

3. Clear the modulo addition table.

```
    for $(i = 0;\ s[i];\ i = i + 1)$
    for $(j = 0;\ s[j];\ j = j + 1)$
    { $k = i + j$; if $(k \geq nchar)\ k = k - nchar$;
      $f[s[i]][s[j]] = s[k]$; }
```

4. Construct tables mapping all symbol pairs to corresponding sums.

```
    for $(i = 3;\ i \leq 7;\ i = i + 2)\ colgen(i,\ G - 1)$;
    $ipage = 0;\ ifile = 0$;
```

5. Generate odd guard patterns.

```
    while $(fgets(in,\ L,\ stdin))$
    { $i = strlen(in)$;
      if $(in[i - 1] \equiv '\backslash n')\ in[i - 1] = 0$;
```

6. Compute the error-correcting code for the line.

```
      $line[0] = '-'$;
      for $(i = j = 0;\ in[i];\ i{+}{+})$
      { $c = in[i]$; if $(seq[c] \equiv 0)\ c = '\ '$;
        if $(line[j] \equiv '\ '\ \&\&\ c \equiv '\ ')$ continue;
        $line[{+}{+}j] = c$; }
      $line[{+}{+}j] = 0$;
```

7. Compress multiple blanks from the input line.

```
      $ecc(guard,\ line + 1);\ seqn(guard,\ "",\ in)$;
```

8. Compute the ECC guard symbols.

```
      $page[ipage{+}{+}] = guard[G - 1]$;
      if $(ipage \geq pagef)\ seqpage(1),\ pagef = pages$;
```

9. If this is the end of the page, prepare a code for the entire

```
      $file[ifile{+}{+}] = guard[G - 1]$;
      if $(ifile \geq L)\ ifile = ifile - 1$; }
```

10. If this is the end of the page, prepare a code for the entire

```
    $seqpage(0);\ file[ifile] = 0$;
    $ecc(guard,\ file);\ seqn(guard,\ ".",\ "")$;
    $ifile = 0$; }
```

11. At the end of the file, prepare a code for the entire file.

COMPUTE CENTINELS

Upon entry to the algorithm, (1) gs points to an area of length $G + 1$ to receive the results. **(2)** $line$ points to the line. **(3)** G defines the number of guard digits to be computed. **(4)** ptn defines which line positions contribute to which guard digits. **(5)** f contains the modulo-addition table for all symbols. **At exit,** gs contains the guard symbols for the line.

```
$ecc(gs,\ line)$ char $*gs,\ *line$;
{ int $i,\ j$;
```

```
    for $(i = 0;\ i < G;\ i = i + 1)\ gs[i] = IDENT$;
```

1. Clear all the guard symbols.

```
    for $(i = 0;\ i < G;\ i = i + 1)$
    for $(j = 0;\ line[j];\ j = j + 1)$
      if $(ptn[j][i] \equiv 'X')$
        $gs[i] = f[gs[i]][line[j]]$;
    $gs[G] = 0$; }
```

2. Generate each guard symbols.
3. following the table that shows which line positions contribute to which guard symbols.

CONVERT CENTINELS TO INTEGERS

Upon entry to the algorithm, (1) *gs* contains the guard symbols. **(2)** *sep* contains a separator character. **(3)** *sym* contains the string of symbols. **At exit,** *gn* contains the corresponding integer sequence numbers.

```
seqn(gs, sep, sym) char *gs, *sep, *sym;
{ int i;
  for ($i = 0$; $i < G$; $i = i + 1$)
    printf("%02d", seq[gs[i]]);
  printf("%s%s\n", sep, sym); }
```

1. Display the sequence numbers for the guard symbols.
2. Display the full line.

GENERATE PERMUTATIONS

Upon entry to the algorithm, (1) *n* defines the number of guard symbols to be marked. **(2)** *k* defines the position for the initial mark. **(3)** *l* defines the column number on the line, starting with 0. **(4)** *ptn* contains an area to receive the permutations. **(5)** *w* contains a work area for generating the permutations. **At exit, (1)** All permutations have been generated. **(2)** *l* is advanced by the number of combinations generated. **(3)** $ptn[0..l]$ contains the permutations generated thus far. **(4)** *w* contains the most recent permutation generated.

```
colgen(n, k) int n, k;
{ static char w[$G + 1$] = ""; static int $l = 0$; int i;
  if ($w[0] \equiv 0$)
    for ($i = 0$; $i < G$; $i = i + 1$) w[i] =' −';
  if ($n > 0$) for ($i = k$; $i \geq n - 1$; $i = i - 1$)
  { w[i] =' X';
    colgen($n - 1$, $i - 1$);
    w[i] =' −'; }
  else if ($l < L$)
  { for ($i = 0$; $i < G$; $i = i + 1$)
      ptn[l][i] = w[i];
    $l = l + 1$; }}
```

1. On the first call, establish a null pattern in the array.
2. Mark the guard symbol for each possible position and generate all permutations within that position.
3. If there are no deeper permutations, save the current permutation and advance the column number.