LEVEL II
CARTAN
The Cartesian Access Method for Data Structures with n-dimensional Keys

Thesis by
Stephen Vaughn/Petersen
Major, United States Air Force

AFIT-79-108

In Partial Fulfillment of the Requirements for the degree of Doctor of Philosophy

California Institute of Technology
Pasadena, California

1979 (Submitted September 20, 1978)

This document has been approved for public release and sale; its distribution is unlimited.
**Title:** The Cartesian Access Method for Data Structures with n-dimensional Keys

**Author:** Maj Stephen Vaughan Peterson

**Performing Organization:** AFIT Student AT:

**Controlling Office:** AFIT/IR
WPAFB OH 45433

**Monitoring Agency:** UNCLASS

**DISTRIBUTION:**

- APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

**Security Classification:** UNCLASS

**Approved for Public Release:** 25 SEP 1980

**Key Words:** (Continue on reverse side if necessary and identify by block number)

**Abstract:** (Continue on reverse side if necessary and identify by block number)

**Attached:**
ACKNOWLEDGMENTS

I wish to thank Dr. Frederick B. Thompson for his guidance and support as my teacher and friend during my stay at Caltech. My graduate studies were supported by the Air Force Institute of Technology, for which I am very grateful. In addition, I am indebted to the Deputy Chief of Staff/Data Systems, Headquarters, Strategic Air Command, for an assignment conducive to the pursuit of my project. In particular, I extend my appreciation to Staff Sergeants Dennis D. Hewitt and Thomas C. Howard for their assistance in testing and production applications of the CARTAN programs. So many of my co-workers have provided support by listening to ideas, proofreading, etc., that I am unable to list them all by name, but I offer my thanks to all.

My heartfelt thanks go to Captain Gene C. Bloom for his support, encouragement and friendship; he managed to kick me out of ruts and bottom dead center innumerable times. Finally, I simply cannot do justice with any words here except to say, "Thanks!" to my lovely wife Sue, who has had to endure much while I completed this project. Without her love and support, this never would have been finished.

Opinions expressed in this paper are my own and are not to be considered an official expression by the Department of the Air Force. If any omissions or errors remain due to any lack of thoroughness or general laziness on my part, they are my own and I claim full responsibility for them.
ABSTRACT

The Cartesian Access Method (CARTAM) is a data structure and its attendant access program designed to provide rapid retrievals from a data file based upon multidimensional keys; for example, using earth surface points defined by latitude and longitude, retrieve all points within x nautical miles. This thesis describes that data structure and program in detail and provides the actual routines as implemented on the International Business Machine (IBM) System/370 series of computers. The search technique is analogous to the binary search for a linear sorted file and seems to run in $O(\log(N))$ time. An indication of the performance is the extraction, in less than 25 milliseconds CPU time on an IBM 370, Model 3033, of all points within a 10,000-foot circle from a geographic data base containing approximately 100,000 basic records.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgments</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iv</td>
</tr>
<tr>
<td>Illustrations</td>
<td>v</td>
</tr>
</tbody>
</table>

**Chapter**

<table>
<thead>
<tr>
<th></th>
<th>Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Background and Problem Definition</td>
<td>5</td>
</tr>
<tr>
<td>II</td>
<td>An Unusual Data Structure for the Real Line</td>
<td>9</td>
</tr>
<tr>
<td>III</td>
<td>Generalization to n-dimensional Space</td>
<td>25</td>
</tr>
<tr>
<td>IV</td>
<td>An Application Programmer’s View of CARTAN</td>
<td>41</td>
</tr>
<tr>
<td>V</td>
<td>Inside CARTAN for the Maintenance Programmer</td>
<td>63</td>
</tr>
<tr>
<td>VI</td>
<td>CARTAN in Use</td>
<td>87</td>
</tr>
<tr>
<td>VII</td>
<td>Assessments and Recommendations</td>
<td>102</td>
</tr>
</tbody>
</table>

List of References | 105 |
Appendix

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>CARTAM Source</td>
<td>106</td>
</tr>
<tr>
<td>B</td>
<td>Subroutine VECTOR</td>
<td>136</td>
</tr>
<tr>
<td>C</td>
<td>VECTOR Source</td>
<td>144</td>
</tr>
<tr>
<td>D</td>
<td>Copy Books for COBOL Programs Using CARTAM</td>
<td>163</td>
</tr>
<tr>
<td>E</td>
<td>Index Load Program Source</td>
<td>165</td>
</tr>
<tr>
<td>F</td>
<td>VSAM File Definition Example</td>
<td>172</td>
</tr>
<tr>
<td>G</td>
<td>Circle Search Program Source</td>
<td>173</td>
</tr>
<tr>
<td>H</td>
<td>Inclusion/Exclusion Area Search Program Source</td>
<td>181</td>
</tr>
<tr>
<td>I</td>
<td>FORTRAN Subroutine to Expand Longitude</td>
<td>191</td>
</tr>
</tbody>
</table>

ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Ring Structure Example</td>
<td>14</td>
</tr>
<tr>
<td>4-1a</td>
<td>Cartesian Square Subdivision</td>
<td>29</td>
</tr>
<tr>
<td>4-1b</td>
<td>Corresponding List Structure</td>
<td>29</td>
</tr>
<tr>
<td>4-2</td>
<td>Conditions for Intersection</td>
<td>32</td>
</tr>
<tr>
<td>4-3</td>
<td>Flow Chart of INTERSECTION_FUNCTION</td>
<td>36</td>
</tr>
<tr>
<td>5-1</td>
<td>Calling Sequence Requirements</td>
<td>43</td>
</tr>
<tr>
<td>5-2</td>
<td>Communication Block</td>
<td>48</td>
</tr>
<tr>
<td>6-1</td>
<td>DSECT of Communication Block</td>
<td>72</td>
</tr>
<tr>
<td>6-2</td>
<td>DSECT of PCBAREA</td>
<td>73</td>
</tr>
<tr>
<td>7-1</td>
<td>Circle Search Conditions</td>
<td>90</td>
</tr>
<tr>
<td>7-2</td>
<td>Performance Statistics</td>
<td>93</td>
</tr>
<tr>
<td>7-3a</td>
<td>Inclusion Area Search Example</td>
<td>95</td>
</tr>
<tr>
<td>7-3b</td>
<td>Exclusion Area Search Example</td>
<td>95</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

The age of information is upon us. Whether the computer has been developed to allow us to manipulate that information or to generate it is a moot question at this time; we do have large masses of data and must use the computer to manage them efficiently. The corporate data base has become an all-important entity in many, many cases, and the management and retrieval of information has become a far from trivial operation; witness the proliferation of data base management systems on the market today. I am not trying to address that massive subject; rather a small corner concerned with the efficient searching and retrieval of pertinent information to answer some rather specific questions.

It is extremely rare that a question is asked which requires access to an entire data base to develop the answer. In the vast majority of cases, we only need to examine certain rather small subsets of the available data. Many of these instances involve the determination of a key value or a range of key values which are then used to access the appropriate record(s) to answer the original query. So far
these keys have been single-dimensional values used to probe a linear sequential file of some particular organization. There have been many methods developed to solve these types of problems; Knuth devotes an entire volume to them [8]. However, if the information is keyed by multi-dimensional values, such as points in Cartesian space or locations on the surface of the earth, existing methods do not readily lend themselves to answer questions of proximity or nearness.

This paper presents a solution to the problem of efficient probes into multi-dimensional data using a method of quadrature to develop a data structure which has become very useful for questions such as: "Which resorts are within a day's drive of my home?"; "How many doctors and dentists are located in the state of Arizona?"; "What types of navigation aids are available for an airline route from San Francisco to Moscow?", etc. I shall develop this structure and the implementation of some computer programs which provide the answers to these and other similar questions.

The first of three main divisions of this thesis is a step-by-step development of the data structure and its algorithms. In order to establish an initial environment, Chapter II briefly describes some geographic data files in use at Headquarters, Strategic Air Command (SAC) and the methods that were used to query those files. After examination of the problem, the basic algorithms for our solution
these keys have been single-dimensional values used to probe a linear sequential file of some particular organization. There have been many methods developed to solve these types of problems; Knuth devotes an entire volume to them [8].

However, if the information is keyed by multi-dimensional values, such as points in Cartesian space or locations on the surface of the earth, existing methods do not readily lend themselves to answer questions of proximity or nearness.

This paper presents a solution to the problem of efficient probes into multi-dimensional data using a method of quadrature to develop a data structure which has become very useful for questions such as: "Which resorts are within a day's drive of my home?"; "How many doctors and dentists are located in the state of Arizona?"; "What types of navigation aids are available for an airline route from San Francisco to Moscow?", etc. I shall develop this structure and the implementation of some computer programs which provide the answers to these and other similar questions.

The first of three main divisions of this thesis is a step-by-step development of the data structure and its algorithm. In order to establish an initial environment, Chapter II briefly describes some geographic data files in use at Headquarters, Strategic Air Command (SAC) and the methods that were used to query those files. After examination of the problem, the basic algorithm for our solution
is developed in Chapter III. This development is in one dimension, specifically the real line, as illustration to allow comparison with existing file search strategies, in particular the binary search scheme. As such, the algorithm and structure will appear very cumbersome; the utility of the method becomes apparent in Chapter IV as the structure and algorithms are generalized to n dimensions.

The second section of this paper covers the technical aspects of the actual implementation. Chapter V is intended as a user's guide for the programmer/analyst who plans to use this n-dimensional programming technique to solve a specific problem. The implementation is as a subroutine, and this chapter describes the calling sequences and the results that are to be expected. Chapter VI goes into the internal workings of CARTAM and is maintenance information intended for the assembly level programmer who wishes to both install the system on his own hardware and/or maintain it while in use.

Once the reader is aware of the available operations, a series of examples is presented in the third section to demonstrate the use of the system. Chapter VII describes a few of the current application programs in day to day use at Headquarters SAC. These programs may prove to be useful to the reader in their own right, but the main purpose is to illustrate some methods and show how the data structure may
be used. I hope that they will serve as jumping-off places for solutions to existing problems that had been deemed either unsolvable or too costly to solve using previously known methods. Chapter VIII concludes with some thoughts and recommendations on possible future applications and improvements.

The appendices, with one exception, are listings of the programs that have been in use at SAC for the last year. Appendix B contains a detailed description of a distance-calculation function or metric used to compute geodetic distances on the surface of the earth. This metric is used throughout the examples in Chapter VII.
CHAPTER II

BACKGROUND AND PROBLEM ENVIRONMENT

The data structure and access techniques as described in this thesis were developed primarily at Headquarters, Strategic Air Command, Omaha, Nebraska, and specifically applied to geographic data files used by the Joint Strategic Target Planning Staff. These particular files are used as concrete examples and are not intended to imply that these are the only possible applications; the method may be applied to any multi-dimensional data file.

The first file that was examined consists of approximately 50,000 records describing points on the surface of the earth. Most of the information in each of these records is of no consequence to this discussion except for a unique 21 character key which can be used for retrieval of a desired complete record, and the latitude and longitude which specify the location of the item on the earth.

Queries against this file by location have been limited to small areas which allowed use of a limiting procedure based upon a range of latitude values. This procedure started with an external sort based on the concatenation of
latitude and longitude into a single key used for sort sequence. The resultant file was then read a record at a time, checking for inclusion inside a gross "box" defined by constant latitude and longitude, storing candidate prime keys in an internal table. Since the file is sorted with a major key of latitude, the read procedure is terminated when the input latitude is greater than the upper limit of the box. Note, however, that many records are read which will fail the gross longitude check.

After the table of candidate keys is built in main memory, a finer discrimination is made with an appropriate metric to arrive at the final set of accepted records. Some applications are summarizations that permit the packaging of several distinct queries into a single program. Since each candidate may then be examined for each criterion, a large number of the disk input operations are eliminated. However, this method is absolutely memory-bound and cannot afford a criterion resulting in a large candidate subset of the original file.

An attempt at clustering has been applied to this geographic data resulting in an "island" system. These islands have been defined such that each island is disjoint from all others with a minimum separation between any two adjacent islands. The island assignment procedure is simply a scan through the entire file as described above, looking for the
island that is less than the minimum distance away from the new point. Another way to consider the clustering is that an island is the collection of all those points that are within the maximum separation of another point. This does manage to cluster points in manageable groups in most cases, but occasionally islands grow to an unwieldy size. Those islands are then manually broken up by using a smaller separation distance.

Once the islands have been assigned, a non-trivial process, subsequent processing is usually done on an island basis. An application program is given an island to process, at which time all members of that island are read into main memory and the necessary fine discrimination is applied to that subset. This methodology is not too unmanageable as long as the number of members does not get too large; anything over approximately 500 records begins to degrade performance. The island approach also limits the fine discrimination to a distance criterion no greater than the minimum separation between islands. If the desired distance is greater than the minimum separation, the method breaks down completely since the search area may need more than one island.

A second major file concerns points used to describe country and coastal boundaries for mapping applications. This data set contains approximately 100,000 data points.
and is stored in a sequence suitable for display on an x-y plotting device. The mapping software is capable of discarding those points outside of the area being mapped, but the entire file must be read each time, which drives the computing times to rather large values. When maps are being prepared in a batch environment for hard-copy output to be produced on a flat-bed plotter, the high CPU time may be acceptable, but not in an interactive environment with maps to be displayed on a CRT device. The only known method of operation was to pre-build desired maps overnight, which restricted a user to those, and only those, maps. If, for any reason, the user changed his mind, new maps were not available until at least the next day.

As can be seen, in many instances we have been strictly memory-bound for area type queries after reading the entire source file. The attempt at clustering the data has improved this to some extent, but only if the distance criterion is not too great. Even so, programs have been required to define internal table space to allow for the maximum size of a cluster and discrimination within the cluster required a distance calculation from the point of interest to every member of that cluster. The data structure and techniques described in the remaining chapters have removed these restrictions entirely.
CHAPTER III

AN UNUSUAL DATA STRUCTURE
FOR THE REAL LINE

The problem of retrieval of information from a large file is usually solved by determining a unique key for each record, imposing an ordering operator (>) on the key field and subsequently storing the data in a linear fashion on secondary storage. Retrievals may then be accomplished by several efficient search strategies, e.g., binary search, hashing, etc. If the individual records are substantial in size, indexes are useful in reducing secondary storage access time, but the problem of searching the index has not changed.

An order is imposed upon the key values to increase the amount of available information. A linear sweep of such a file may be terminated when the key value becomes greater than the desired argument, where a random ordering would require examination of every key value in the file. This linear probing of a sorted file results in an average access of $N/2$ records, where $N$ is the total number of keys in the file of interest. A much faster technique is the so-called binary search, which probes the median record in a sorted
file and determines which half might contain the desired key, thus discarding the other half. Considering the remaining sub-file as a file itself, the median record of the sub-file is then probed. This algorithm terminates successfully when the desired key is found, or terminates unsuccessfully when adjacent keys in the file bracket the desired value. The binary search algorithm accesses an average of approximately log2(N) records and is said to run in log(N) time. These algorithms have an underlying assumption that the key values may be mapped one-to-one with a subset of the integers in a meaningful way which allows for the application of an ordering operator and subsequent sorting of the file.

However, if the file consists of geographic data, for example, with latitude and longitude for coordinates, the concept of ordering becomes nebulous at best. It is true that on a general purpose computer, the latitude and longitude may be defined in such a fashion as to each reside in a computer word of, say, 32 bits. These two computer words could be concatenated into a 64-bit key value, and the file could then be sorted accordingly. A problem arises when trying to decide which coordinate is to be considered as the major portion of the key. If latitude is chosen as the major key, then data points with identical latitude will be "close" together in the file, but data points with identical longitude may be "far" apart in the file structure.
Since points on the surface of the earth as denoted by latitude and longitude have their own problems in relation to a metric, let us suspend consideration of geographic points for now and concentrate on a Cartesian space, i.e., the cross product of the real line, in n dimensions. The simplest Cartesian space is the real line itself where n = 1. Thus, the following discussion will be limited to the one-dimensional case and may appear unnecessarily complicated at times, but remember that the eventual goal is the extension to n dimensions.

Let us examine a binary search strategy as applied to a linear, sorted file. In particular, consider a "uniform binary search" as described by Knuth [8, pg 413] using Shar's modification.

Given a table of records R1, R2, ..., Rn, whose key values are in increasing order K1 < K2 < ... < Kn, we can search for a specified argument K, using algorithm C:

\begin{verbatim}
C[Initialize]
Set i := 2^\lfloor k \rfloor where \( k = \lfloor \log_2(n) \rfloor \).
\end{verbatim}

(\( \lfloor \log_2(n) \rfloor \) is the floor of \( \log_2(n) \) or the greatest integer \( \leq \log_2(n) \); i.e., \( k = \lfloor \log_2(n) \rfloor \) is an integer such that \( k \leq \log_2(n) < k + 1 \).)
If $K = Ki$, algorithm terminates successfully.

If $K < Ki$, set $d := 2^{sk}$, go to C2.

If $K > Ki$ and $m = 2^{sk}$, algorithm terminates unsuccessfully,

but if $a > 2^{sk}$, reset $i := a + 1 - 2^{sk}$

where $j = \log_2(m - 2^{sk}) + 1$.

(note that $2^{sk} - 1 \leq a + 1 - 2^{sk} \leq 2^{sk}$)

set $d := 2^{sk}$, and go to C3.

C2[Decrease i]

If $d \leq 1$, algorithm terminates unsuccessfully;

else set $d := d/2$,

set $i := i - d$,

go to C4.

C3[Increase i]

If $d \leq 1$, algorithm terminates unsuccessfully;

else set $d := d/2$,

set $i := i + d$,

go to C4.

C4[Compare]

If $K < Ki$, go to C2.

If $K > Ki$, go to C3;

otherwise $K = Ki$ and

algorithm terminates successfully.
The choice of the underlying storage organization for our table of records is a crucial consideration. If the table is small enough to be contained entirely within the primary store of the computer, transformation of the index value i into a displacement into the table is a simple calculation. However, complete residence in primary store may be prohibitively restrictive, as a table of any appreciable size must be on secondary storage. In addition, the transformation of the index into a displacement into a multi-dimensioned table becomes complex. For these reasons, and others as will become apparent later, I have chosen to store structural information in an explicit binary tree, with modifications. Instead of the left and right links of the usual binary tree, I use the child and twin pointers of a ring structure or circular list. This ring structure as illustrated in figure 3-1* also includes the parentage information usually provided by an up-link without needing the additional pointer space in the record entry. A single bit in each record serves to indicate when a twin pointer is in fact an up-link. It is also convenient to include an

*The usual depiction of chains in linked lists in diagrams is from left to right. The usual representation of a negative number in a general purpose computer is with a bit set to "1". When a linked list chain is arranged in ascending order based on a bit string of arithmetic signs, we then have an inversion between a picture of a line segment and the corresponding list. I hope this will cause no problems to the reader.
explicit indication as to whether a particular record is the positive or negative child of its parent record. This indicator is a single bit in the one-dimensional case.

Since the file is being stored as an explicit binary tree, note that additional records are being generated, and the concept of an "i-th" record for the algorithms becomes imprecise. Assume for the moment that the key values \((K_i)\) are integers uniformly distributed over the interval \(-X\) to \(+X\) where \(X = 2^{2^x}\) and \(x\) is the smallest integer greater than or equal to \(\log_2(\max(|K|))\), i.e.,

\[ x - 1 < \log_2(\max(|K|)) \leq x. \]

Then a root record with a key value of 0 and a delta of \(X\) defines the interval \(0\times X\) as a cover for all key values of interest, i.e., a line segment that contains all key values within it. Dividing the interval in half, the root segment now has a positive child and a negative child at the next level of detail. In the ring structure under consideration, the positive child is reached from the child pointer of the root record, while the negative child is reached by following the twin pointer of the positive child. The negative child record will have the parent indicator set showing that the twin pointer in that record points back to the parent, closing the ring. Carried to the logical conclusion, each record in the file defines a finite length line segment by specifying the center coordinate value and a delta or line length to either side of the center.
There are some important points to keep in mind about the line segments as defined by the file records. The children of a given record subdivide the line segment as defined by the parent record. In particular, if we consider a record as defining a set, which is exactly a line segment in the one-dimensional case, the set intersection of records connected by twin pointers is empty, while the union of those same records is identical to the parent record. These conditions of intersection and union also imply that the intervals defined by the records are only half-closed, specifically, closed at the left end and open at the right end. As an example, assume that we have a set of key values such that \(-15 \leq K_i \leq +15\). Then, \(x = 4\), and the first few generated binary tree records are:

<table>
<thead>
<tr>
<th>Record num</th>
<th>Key( (K_i) )</th>
<th>Delta</th>
<th>Twin ptr</th>
<th>Child ptr</th>
<th>Direc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>16</td>
<td>---</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>1*</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2*</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-4</td>
<td>4</td>
<td>7</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>7</td>
<td>-12</td>
<td>4</td>
<td>3*</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>2</td>
<td>9</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>2</td>
<td>4*</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>5*</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

The asterisks in the twin pointer column indicate the end of the ring, i.e., the parent pointer. Note that the delta value for each record defines the distance from the center.
to either end of the line segment, i.e., delta is one-half the length of the interval. Graphically this can be represented by:

If the key values are dense in the integers, i.e., the difference between consecutive keys is exactly one, then the length is halved each time we follow a child link or descend one level in the tree. Also, if we follow the twin link, unless marked as an up-link, we remain at the same level in the tree, but go to the complementary line segment. However, since key values are very rarely dense in the integers, strict adherence to the notion of equal deltas at the same level in the tree would result in extra nodes which have only one child instead of two. Therefore, we eliminate an extraneous node by replacing it in the ring with its only child. Notice that now delta values are not necessarily halved when following a child link, nor are they equal along a twin chain. Thus, it becomes useful to explicitly carry the delta value in the record entry.
The binary tree as stored on a secondary storage medium contains two basic types of records: terminal records corresponding to the original data points, and internal nodes or branch points of the tree which have been generated due to the structure definition. Each record, accessed through a pointer of value \( P \), consists of:

1) a key or coordinate value of the center of the interval \( K(P) \)
2) a delta value of one-half of the length of the interval \( D(P) \)
3) a child pointer \( \text{Child}(P) \)
4) a twin pointer \( \text{Twin}(P) \)
5) if the record is a terminal, additional data germane to the original data record
6) various flags, such as:
   a. node or terminal indicator
   b. end of twin chain in ring, and
   c. the sign of the difference between the record's coordinate and the coordinate of the parent of this record as a direction indicator \( Q(P) \)

It is obvious that construction of this explicit binary tree generates overhead with the node records. Since extraneous nodes have been eliminated, any record with a non-null child pointer has two children. To determine just how much overhead is generated, let \( t \) be the number of terminals.
present, and let \( x \) be the number of generated nodes. If \( t' \) and \( t'' \) are subsets of \( t \) such that \( t' = 2^{k'} \) and \( t'' = 2^{k''} \) for some integers \( k' \) and \( k'' \), then the number of nodes generated for the appropriate subtrees are \( x' \) and \( x'' \). Applying the summation of a geometric progression with a ratio of 2, and noting that any two subtrees may be connected with one additional node, we obtain:

\[
x' + x'' = (t' - 1) + (t'' - 1) + 1 = t' + t'' - 1.
\]

By induction, then, \( x = t - 1 \).

When storing the tree on a secondary storage medium, it is useful to have a master node, the root, at a location in the file that is always known. The only location that is always known is the first one; therefore, we add an additional node to the structure as the master root record, which makes the total number of generated nodes equal to the number of terminal records.
With the structure as just defined, the earlier search algorithm C is modified to give algorithm T to search for a given argument K:

**T1[Initialize]**
Set P := root.

**T2[Compare]**
Set D := K - K(P).
If D = 0 and D(P) = 0, terminate successfully.
[Record is a node if D(P) > 0.]
If D ≥ 0, go to T3;
else go to T4.

**T3[D positive]**
If D ≥ D(P), terminate unsuccessfully;
else set P := Child(P),
go to T2.

**T4[D negative]**
If D < -D(P), terminate unsuccessfully;
else set P := Twin(Child(P)),
go to T2.

When searching for a specific argument K, algorithm T may seem unnecessarily complicated. However, if the search is for all records with key values in the range K ± d, algorithms T may be extended in the following fashion with a stack, as algorithm T':
-21-

**R1[Initialize]**
Set P := root.

**R2[Compare]**
Set D := K - K(P).
If D ≥ 0, go to R3;
else go to R4.

**R3[D positive]**
If D ≥ (d + D(P)), go to R6;
else go to R5.

**R4[D negative]**
If D < -(d + D(P)), go to R6;
else go to R5.

**R5[Check overlap]**
If |D| ≤ (d - D(P)),
   present entire subtree as successful,
   go to R6;
else set P := Child(P),
   push Twin(P) to stack,
   go to R2.

**R6[Pop stack]**
If stack is empty, terminate;
else pop P := top of stack,
   go to R2.
Algorithm 2 allows extraction of information from the binary tree structure. However, before any extractions can be performed, the tree must be built. After initialization and definition of the file by writing a master node record, repeated insertions using algorithm 1 will build the file.

I'1[Initialize insert]
Set \( K := \) key value of record to be inserted.
Set \( P := \) root (pointer to master node).

I'2 Set \( D := K - K(P) \).
Set \( Q := \text{sign}(D) \).
If \( |D| < D(P) \), go to I'3.
If \( |D| > D(P) \), go to I'5.
otherwise \( (|D| = D(P)) \), so
if \( Q = + \), go to I'5 (open end of interval);
else go to I'3 (closed end of interval).

I'3[Inside]
Set \( P' := P \).
Set \( P := \text{Child}(P) \).

I'4[Walk ring]
If \( Q = Q(P) \), go to I'2.
If \( Q > Q(P) \), set \( P := \text{Twin}(P) \), \[ "+" < "-" \]
go to I'4;
else go to I'5.
I*5[Outside; record(I) to be inserted was inside the line segment defined by node(P) and was on the Q side of the center of that segment. The existing child on that same side, record(P), defines a line segment which does not include the new record(I). Replace record(P) in the ring with a new node(P'), and make the new record(I) and record(P) children of node(P').]

Set D(P') := D(P).
Set K(P') := K(P).
Set Q(I) := Q.

Repeat [Adjust Record(P')]

Set D(P') := D(P')/2;
If Q(I) = "+",
then set K(P') := K(P') + D(P'),
else set K(P') := K(P') - D(P');
Set Q(I) := sign(K(I) - K(P'));
Set Q(P) := sign(K(P) - K(P'));
until Q(I) ≠ Q(P).
I've adjusted pointers

If $Q(I) < Q(P)$ \[ Q^+ < Q^- \]
then
- set $Child(P^n) := I$
- set $Twin(I) := P$
- set $Twin(P) := P^n \text{ and mark as parent}$

else
- set $Child(P^n) := P$
- set $Twin(P) := I$
- set $Twin(I) := P^n \text{ and mark as parent}$

The structure and techniques just described are much too complicated for efficient application to data keyed from the real line. However, the real line is simply the degenerate case of the eventual goal, n-dimensional space, and is described in detail for ease of illustration. As will be seen in the next chapter, the n-dimensional case is obtained from this development with quite simple extensions.
CHAPTER IV

GENERALIZATION TO n-DIMENSIONAL SPACE

The last chapter discussed at some length a rather unusual data structure for information keyed by a single coordinate. In this chapter, I will present the extensions to the data structure and algorithms which provide for the n-dimensional case and give the rationale for the design.

One of the more obvious questions concerns the use of a ring structure rather than the usual binary tree linkage of elements. After all, each record carries two link pointers while the ring has only two elements. The two pointers could just as well have been left and right links, eliminating the requirement to walk over the positive record in order to access the negative record. However, in extending to a higher dimensionality, the number of pointers required to define the structure increases exponentially.

In particular, in n-dimensional space, a given ring may contain up to $2^n$ entries. The ring structure allows this expansion of the number of entries with no additional pointer requirements, while a separate pointer in the record for each possible child rapidly consumes an inordinate
amount of space. The ring structure also accommodates the absence of records very nicely, while individual pointers would have null values in many cases. Then there are additional physical limitations imposed by the computer hardware. As an example, consider the IBM 360/370 series of computers which use an address of 24 bits. If individual pointers were carried in a record, an application with 25 dimensions, for example, would require a record format with $2^{25}$ pointers. This technique obviously would require a record much greater in size than the entire available computer memory.

The overhead generated by the tree structure is a direct result of the node records that define the structure. This overhead has been minimized to an extent by elimination of extraneous nodes, i.e., those nodes which would have only a single child. I have shown that in the one-dimensional case the number of node records is equal to the number of terminal records. For the n-dimensional case, this number becomes an upper bound for the worst case situation where any given node has only two children. Most nodes in the n-dimensional case will have more than two children; in other words, a twin chain will normally be longer than two entries, but in no case will the length of the twin chain be greater than $2^nn$. 
The upper bound $U$ for the number of nodes in a file with $t$ terminal records is exactly equal to $t$. The lower bound $L$ is attained when every node has $r = 2^n$ children or the twin chain length is $r$. As was done for the one-dimensional case, $t$ could be broken down as a summation of integer powers of $r$, but since $r$ subtrees would have to be joined under a junction node to maintain optimality, and we are only interested in a lower bound, it is convenient to assume that $t$ is already an integer power of $r$. Using the sum of a geometric progression once again, now with a ratio of $r$ between successive terms, the lower bound is:

$$L = 1 + \frac{t - 1}{r - 1}.$$

For an example, assume $n = 2$ and $t = 65,536 = 4^8$. Then the upper bound $U = t = 65,536$ node records, while the lower bound $L = 21,846$ or roughly $0.3t$ node records. The approximate range of $0.3t$ to $1.0t$ therefore indicates the actual number of nodes. Actual experience with a geographic data file has resulted in a file structure with approximately $0.7t$ node records.

These considerations, then, dictate the use of a ring structure while the record content as given in the last chapter is extended for $n$ dimensions as:

1) a key or coordinate values for the center of a (hyper-)square $K_i(P)$

2) a delta value of one-half the length of a side $D(P)$
3) a child pointer \( \text{Child}(P) \)
4) a twin pointer \( \text{Twin}(P) \)
5) application dependent data for terminal records
6) various flags:
   a. node or terminal indicator
   b. end of twin chain indicator
   c. a quadrant indicator of \( n \) sign bits of the difference between each coordinate of the record and the corresponding coordinate of the parent record \( Q_i(P) \)

As an example of the list structure compared to an actual square from a Cartesian space, see figure 4-1. Figure 4-1a shows the example square, while figure 4-1b depicts the list as defined by the node and terminal records. The root node A defines the outer square which is then subdivided by the four children, B, C, D and E. The square defined by node E is then subdivided further by its children, F, G and Z while the children of B, C and D are not shown. Node G is then subdivided even further by H, x and y. Again, the children of F and H are not shown. The terminal record Z specifies the only data point in the “+” quadrant of E, while the “−” quadrant is empty as indicated by the absence of a corresponding record in the list. Terminal records x and y likewise specify the only data points in appropriate quadrants of G. Overall, the process
Cartesian Square Subdivision

Figure 4-1a

Corresponding List Structure

Figure 4-1b
of subdivision is continued until a quadrant of a given square contains a lone terminal record; a node record is never defined unless it would have at least two children.

The \( n+1 \)-tuple \((K_1(P), K_2(P), \ldots, K_n(P), D(P))\), where each coordinate \( K_i(P) \), in connection with \( D(P) \), defines a half-open interval as in the one-dimensional case, defines a square if \( n = 2 \), a cube if \( n = 3 \), and a hyper-cube if \( n > 3 \). Since a cube may be considered a hyper-square, and examples are presented in two dimensions much more facilely than in higher dimensions, I shall use the term square in the remainder of this paper to refer to the object defined by the \( n+1 \)-tuple. In a similar vein, I shall use the term rectangle when referring to the object defined by an ordered pair of \( n \)-tuples; the first \( n \)-tuple is a vector of coordinates defining the lower limits of the intervals or the lower left corner, while the second \( n \)-tuple is a vector of the upper limits of the intervals or upper right corner. Note that in the case of the rectangle, the intervals defining the sides are closed at each end.

The rectangle is used primarily in conjunction with an area search request, algorithms \( R' \), but is also useful in the insertion scheme, algorithms \( I' \), by allowing the rectangle to degenerate to a point. In both instances, the algorithms essentially ask the question, "Does a square as
stored in the file intersect with the search rectangle? If it does, is the square totally inside the rectangle or vice versa? Let's examine the area search first.

As will be seen when algorithm B* is extended to n dimensions, the question of intersection is as stated above. See figure 4-2 for some pictorial examples of possible situations with a search rectangle as defined by X. Squares A, B, C and D have non-empty intersections with X, but there is insufficient information to make a positive decision; the structure must be examined further at a finer level of detail. Square E has an empty intersection with rectangle X; therefore, we may discard the entire subtree by proceeding immediately along the twin chain. Square F is totally enclosed by X; thus, the entire subtree may be accepted as meeting the search criteria.

Returning to square D for a moment, there is additional information available, namely only one particular child of the square could possibly be of use to the search request. As will be seen, determination of the intersection involves arithmetic on the coordinates; construction of a Q type bit string is very simple. If such a bit string is constructed for each of the limit vectors, high and low, and the bit strings are then identical, the only child of interest will be exactly that child with the same bit string Q_i(P).
Conditions for Intersection

Figure 4-2
The search application uses an ordered pair of n-tuples or vectors to define the rectangle, while the insertion algorithm uses a single vector as input for the record to be inserted. If we let that single vector be used twice, i.e., as a definition of a degenerate rectangle, the same set intersection function may then be used in the insertion algorithms. It will turn out to be useful to allow insertion of terminal records with identical coordinates, although differing ancillary data, which can be done by inserting a node record with a zero-valued delta and then chaining terminal records as children of that node. If the set intersection function is able to indicate whether the degenerate rectangle is totally inside the square and vice versa, and if both conditions are true, then the identity intersection would be indicated. Note that as a result of the half-open character of the square definition intervals and the closed nature of the rectangle defining intervals, the identity intersection technically could never occur. However, since computer arithmetic is finite in nature, the identity intersection can occur, but only when the intersection is between a degenerate rectangle and a node with a zero delta or a terminal, i.e., a data point, which is exactly the condition that the insertion algorithm will need.
Since the set intersection function is very important to both the search and insertion algorithms, and will be an extremely high-use section of computer code, it is developed here in detail.

Let the search rectangle $X$ be defined by the ordered pair of $n$-tuples $((x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n))$ where $x_i < y_i$. The square $A$ from the file is defined by the $n+1$-tuple $(a_1, a_2, \ldots, a_n, d)$, where the delta value $d \geq 0$.

[In the following, the symbol $\&$ is for logical "and"; the symbol $|$ is used for logical "or".]

1. At least part of the rectangle is outside of the square if the intersection of $X$ and $\neg A$ is not empty. The intersection is not empty if there exists an $i$:

$$(a_i - d > x_i) \lor (y_i > a_i + d) \lor (a_i + d = y_i \land d \neq 0).$$

Rearranging terms,

$$(a_i - x_i > d) \lor (y_i - a_i > d) \lor (y_i - a_i = d \neq 0).$$

Since $d \geq 0$ by definition, the two terms containing $y_i$ may be combined, giving

$$(a_i - x_i > d) \lor (y_i - a_i \geq d > 0).$$
2. For the converse of condition 1, at least a portion of the square is outside of the rectangle if the intersection of $A$ and $\neg X$ is not empty, which is the case if there exists an $i$:

$$(xi > ai - d) \cap (ai + d > yi).$$

Rearranging terms,

$$(ai - xi < d) \cap (yi - ai < d).$$

3. The intersection of the rectangle $X$ with the square $A$ is empty if there exists an $i$:

$$(ai - d > yi) \cap (ai + d < xi) \cap (ai + d = xi \land d \neq 0).$$

Rearranging terms,

$$(ai - yi > d) \cap (xi - ai > d) \cap (xi - ai = d \neq 0).$$

As in condition 1, $d \geq 0$ allows the combination of the terms containing $xi$ giving

$$(ai - yi > d) \cap (xi - ai \geq d > 0).$$

Figure 4-3 shows a flow chart of INTERSECTION_FUNCTION after combining the three tests; the two Q bit strings are also set as appropriate.
Flow Chart of INTERSECTION_FUNCTION

Figure 4-3
Algorithm I* may now be extended to n dimensions to give us algorithm I:

11[Initialize insert]
   Set Ki := coordinate values of record to be inserted.
   Set P := root (pointer to master node).

12 Execute INTERSECTION_FUNCTION (record(P),Ki,Ki).
   If "Ki is inside record(P)", go to I3.
   If "Ki is outside record(P)", go to I5;
   otherwise an identity intersection, go to I5a.

13[Inside]
   Set P' := P.
   Set P := Child(P).

14[Walk ring]
   If Qi = Qi(P), go to I2.
   If Qi > Qi(P), set P := Twin(P), ["+" < "-" ]
      go to I4;
   else go to I5.

15a[Add a duplicate coordinate record]
   Set Qi := all "+".
   If record(P) is a node, go to I7;
   else set P' := P,
      go to I5.
I5[Outside; record(I) to be inserted was inside the square defined by node(P') and was in the Qi quadrant of that square. The existing child in that same quadrant, record(P), defines a square which does not include the new record(I). Replace record(P) in the ring with a new node(P"), and make the new record(I) and record(P) children of node(P").]

Set $D(P") := D(P')$.
Set $K_i(P") := K_i(P')$.
Set $Q_i(I) := Q_i$.
Repeat [Adjust Record(P")] 

Set $D(P") := D(P")/2$;
For $i = 1$ to $n$, do begin;

If $Q_i(I) = "+"$
then set $K_i(P") := K_i(P") + D(P")$,
else set $K_i(P") := K_i(P") - D(P")$;

Set $Q_i(I) := \text{sign}(K_i(I) - K_i(P"))$;
Set $Q_i(P) := \text{sign}(K_i(P) - K_i(P"))$;
end;

until $Q_i(I) \neq Q_i(P)$.
Adjust pointers

If $Q_l(I) < Q_l(P)$ then

set Child($P^m$) := $I$,
set Twin($I$) := $P$,
set Twin($P$) := $P^m$ and mark as parent;

else

set Child($P^m$) := $P$,
set Twin($P$) := $I$,
set Twin($I$) := $P^m$ and mark as parent.
Finally, we generalize algorithm R* to the n-dimensional case of algorithm R:

R1[Initialize]
Set P := root.
(Li is the low limit vector,
Hi is the high limit vector for rectangle X)

R2[Compare]
Execute INTERSECTION_FUNCTION (Ki(P), Li, Hi).
If "intersection of Ki(P) and X is empty",
go to R3.
If "Ki(P) is inside X", Present entire subtree
as successful,
go to R3;
else (overlap)
set P := Child(P),
push Twin(P) to stack,
go to R2.

R3[Pop stack]
If stack is empty, terminate;
else set P := top of stack, [pop]
go to R2.
CHAPTER V

AN APPLICATION PROGRAMMER'S VIEW
OF CARTAN

The structure that has been defined in the last two chapters is concerned only with a multi-dimensional key value. Depending on the specific application, the full gamut of additional information ranging from nothing, to a primary key into another file, to the entire data record could be carried in the structure. Since the proposed structure is applicable to many situations, it has proven useful to design a program that is concerned only with the structure, letting the particular application provide the necessary drivers specific to their own data and use thereof.

The data structure has been named a Cartesian Index as a result of one of the earliest applications, a latitude/longitude index of a geographic installation file. This file consisted of records varying in length from 320 bytes to 4,600 bytes that were keyed by a 27-byte key for many purposes. The Cartesian file structure was built to provide rapid answers to area search questions, but once the installations were determined, additional information was usually required. Therefore, the ancillary datum carried in the
Cartesian file in the terminal records was the 21-byte primary key value to be used for access into the master file. The Cartesian file thus became a secondary index in two-dimensional space; hence the name Cartesian Index.

The name of the program used to probe the Cartesian Index derives from IBM terminology. IBM provides many different "access methods" to process their various file structures and the program I am describing herein is intended to provide a method of access to the Cartesian Index file; the name CARTesian Access Method (CARTAN) seemed appropriate. In order to make CARTAN readily available to an end user, it is written as a subroutine, allowing the user's specific driver programs to be written in any language supporting a CALL function, usually a high order language.

Communication between the calling program and CARTAN is through a set of calling arguments or parameters. Depending on the function being requested, CARTAN expects from one to six parameters as indicated by figure 5-1. (Function codes are described in detail later.) A 28-byte communication block is required for all requests and is used to pass control and status information between the driver program(s) and CARTAN. It is the only parameter required when logically connecting or logically disconnecting a file or when deleting a record. When inserting data, CARTAN needs a
CALL CARTAN ( )

<table>
<thead>
<tr>
<th>(generic) function</th>
<th>parm</th>
<th>COMM</th>
<th>USER</th>
<th>COORD</th>
<th>DATA</th>
<th>VECTR</th>
<th>DELTA</th>
<th>LOW</th>
<th>HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>[1]</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>[1]</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLSE</td>
<td>[1]</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISRT</td>
<td>[3]</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GR</td>
<td>[6]</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRXX</td>
<td>[4]</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHNG</td>
<td>[3]</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLET</td>
<td>[1]</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Calling Sequence Requirements

Figure 5-1

vector of coordinate values and the ancillary data defined by the user to be stored in the terminal record. For all retrieval requests, CARTAN returns a user-data field, a vector of coordinate values and a single delta value. The GR request is treated in a special manner in that it is used to initiate a rectangle or area search which requires the two additional limit vectors defining the search rectangle. A change request applies to the user data only, but CARTAN was designed to also ensure that the coordinates of the terminal record were not inadvertently changed by the driver program which is why the coordinate vector is a required
argument. On the other hand, deletion of a record, be it terminal or node, is an extreme change of coordinates and user data; there is no requirement to pass additional data to CARTAN beyond the communication block. In all cases, CARTAN looks for the required number of parameters and ignores any additional arguments that may be supplied. CARTAN will also allow, as an optional zero-th parameter, a parameter count argument indicating the number of parameters to be used. If present, this parameter count will be used, and the actual number of arguments will not be checked further. Note also that if the parameter count is present, the total number of parameters is from two to seven, as opposed to one to six.

Before any search queries can be answered, the Cartesian file must be defined and initially loaded. It is assumed that the data set has been allocated disk space; see appendix P. Definition of the file consists of telling CARTAN how many coordinates are to be stored in a record, i.e., the dimensionality of the file, and the type of arithmetic to be used, such as integer or floating point. It was intended that a Cartesian file should be loaded as a separate process, since certain efficiencies are gained thereby; thus, the use of the LOAD command to logically connect and define the file, followed by repeated use of the insert (ISRT) command to store data records. As this information is added to the Cartesian file, a new node
record is constructed if necessary to account for the structure and the new terminal record is added; the relative byte address of the new terminal is returned to the driver program for any use that is desired. The load process is terminated and the file is disconnected with the CLSE command.

Once the file has been defined and loaded, subsequent processing is initiated with OPEN to logically connect it and any desired processing may then be performed. This would normally be retrievals, but the maintenance functions of insert, delete and change are also permitted. The CLSE command logically disconnects the file as before.

This gives a very rough idea as to the various ways that CARTAN is called. Since the communication block is considerably more complicated than the remaining arguments, let me defer its description for a moment and describe the formats of the other parameters first.

The parameter count is always an optional argument in those languages that use the standard IBM method of indicating the end of a variable length parameter list, namely the high order bit of the last address set to one. The IBM supported languages COBOL and FORTRAN always flag the last address, while PL/I normally does not. An assembly language programmer has the option of setting the bit or not as he chooses. If not, the parameter count argument must be
supplied. The parameter count field, parameter 0, specifies the number of additional parameters in the list. As such, it must be a 32-bit fullword binary integer of the appropriate value.

The user-data area, parameter 2, is an input argument to CARTAN for insertions and changes, and an output argument for all retrievals. The user data is variable in length with two 16-bit halfword binary integer fields in the communication block controlling the actual length of the user data.

Since CARTAN allows most of the modes of arithmetic normally used on the IBM 360/370 computers, the last four parameters must take into account the length of individual coordinate values. For instance, if the arithmetic being used is halfword integer, the unit of size is two bytes, while double-precision floating-point arithmetic uses eight-byte values. Therefore, the delta value is a single unit long as determined by the mode of arithmetic while the coordinate vector and the low and high limit vectors are each n units long. The coordinate vector is an input field for insertions and changes, and an output field for all retrievals, as is the user-data area. The limit vectors are explicit input fields for a rectangle search initiation (GR) and must be distinct from the coordinate vector. They are not moved to an internal area by CARTAN; the location
pointers are retained and the vectors repeatedly reaccessed during subsequent retrievals within the rectangle. Thus, the limit-vector values should not be modified during those retrievals except for unusual circumstances as they may be implicit input fields for other retrieval requests.

The remaining parameter, the communication block, is diagrammed in figure 5-2 and is now described in detail below. Following the descriptions of the fields are the lists of valid function codes and status codes as returned by CARTAM.

**DDNAME**

The eight-byte logical name of the file to be processed is stored in DDNAME. Since CARTAM must retain much more than 28 bytes of bookkeeping information, e.g., file control blocks, buffers, stack, etc., the DDNAME also serves as a label for that additional main memory area.

**Function Code**

The four-byte function code carries the request code telling CARTAM which function is to be performed. For retrieval requests it is probably better to consider this code as a concatenation of up to four subfunction codes. Valid function codes are described below.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DDNAME</td>
</tr>
<tr>
<td>4</td>
<td>(8 Bytes)</td>
</tr>
<tr>
<td>8</td>
<td>Function Code</td>
</tr>
<tr>
<td>12</td>
<td>(4 Bytes)</td>
</tr>
<tr>
<td>12</td>
<td>Status Code</td>
</tr>
<tr>
<td>14</td>
<td>(2 Bytes)</td>
</tr>
<tr>
<td>16</td>
<td>Mode</td>
</tr>
<tr>
<td>18</td>
<td>Pad</td>
</tr>
<tr>
<td>16</td>
<td>Relative Byte Address (RBA)</td>
</tr>
<tr>
<td>20</td>
<td>(4 Bytes)</td>
</tr>
<tr>
<td>20</td>
<td>Number of Coordinates</td>
</tr>
<tr>
<td>22</td>
<td>Number of Buffers</td>
</tr>
<tr>
<td>20</td>
<td>Maximum User Area Length (NUAL)</td>
</tr>
<tr>
<td>24</td>
<td>True User Data Length (TUDL)</td>
</tr>
<tr>
<td>24</td>
<td>Number of Disk Reads</td>
</tr>
<tr>
<td>24</td>
<td>Number of Disk Writes</td>
</tr>
</tbody>
</table>

Communication Block (28 Bytes)

Figure 5-2
Status Code

The two-byte status code provides the indication as to the success or failure of the CARTAN request. A value of EBCDIC blanks is returned if CARTAN is able to perform the function as requested. Non-blank values signal unsuccessful completion for a variety of reasons which may or may not be actual error conditions. A complete list of status codes follows the function codes.

Node or Terminal Indicator (WORT)

CARTAN returns a character to the driver program in WORT on successful retrieval requests to allow differentiation between node and terminal records. The three possible values returned by CARTAN are:

1) N - a node was retrieved
2) T - a terminal record was retrieved
3) X - a terminal record was retrieved, but the area intended to receive the user data was too short to accommodate all ancillary data as stored on the file.
A relative byte address (RBA) is used internally by CARTAN to build the structure pointers. Whenever CARTAN successfully inserts or retrieves a record, the record RBA is also returned to the driver program for use if desired. A Get Direct retrieval function is provided to allow direct entry into the Cartesian Index file. Examples of the use of this value would be storage of the RBA in the master record of the primary file as a cross-reference, or temporary retention of the RBA for later retrieval of selected user data not initially needed. As a cross-reference example, consider obtaining a record from the primary file by some means other than coordinate search and then desiring to find all other records within a certain distance as defined by a metric on the coordinates. Use of the RBA to position directly to the corresponding terminal record in the Cartesian Index and then climbing the structure to the appropriate level may be much faster than working down the tree from the root.

The record RBA field is also used by CARTAN to return additional error information whenever a disk operation was unsuccessful. Refer to [3,4] for an explanation of those codes. Finally, when the file is closed, CARTAN returns the high used RBA as an indication as to the amount of space on the file that was actually used.
Maximum User Area Length (MUAL)

The halfword integer in the MUAL field specifies the length of the area that is being provided by the user for a retrieval request. This number is the maximum number of bytes that CARTAM will return, see MORT above, and is also the length to which the user-data area will always be padded with the pad character, see Pad below.

True User Data Length (TUDL)

The actual length in bytes of the character string in the user-data area is placed in the TUDL field. This value must be filled by the driver program on an insert request. For retrieval requests, CARTAM stores the actual number of data bytes, not counting pad characters, that have been placed in the user-data area of the driver program. This value will never be set by CARTAM to a value greater than that currently stored in the MUAL field.

Number Reads, Writes

Two halfword binary integer fields are incremented by CARTAM each time a physical disk read or write is performed. These fields are zeroed out during open processing. The fields are maintained and presented for information only.
The remaining field definitions have meaning only when CARTAM is requested to open the file: function code is LOAD for initial file load or OPEN. Other than the mode, these fields are alternate usages of the NORT and RBA fields.

Mode Indicator

CARTAM allows the user to specify the type of arithmetic to be used for the coordinates by supplying a value in the mode indicator if the function is LOAD; otherwise, CARTAM returns an appropriate value based on the particular file. No further reference is made to this field in subsequent calls. The four valid EBCDIC character values are:

1) H - for 16-bit halfword integer binary,
2) F - for 32-bit fullword integer binary,
3) E - for 32-bit single-precision floating point,
4) D - for 64-bit double-precision floating point.

Pad Character

In many cases, the user-supplied data being carried in the terminal records are variable-length character strings. On a retrieval request, the driver program specifies the length of the area that is being provided to receive this user data. When that area is too short, CARTAM so indicates with an "T" returned in NORT. However, when the area is longer than necessary, it will be padded out to the end with the character supplied in the pad field of the communication block.
Number of Coordinates

The dimensionality of the space being represented is determined by the number stored in this halfword field, and is the number of coordinates carried in a record of the file. The field is filled by the driver program if the function is LOAD and filled by CARTAN if the function is OPEN.

A somewhat arbitrary limit of 512 dimensions has been imposed, mainly because a limit must be established somewhere. Storage must be allocated for the bit strings generated by INTERSECTION_FUNCTION, and 64 bytes was chosen. A further limit is that the total length of a coordinate vector must be less than one-half the length of a physical record to allow storage of at least two logical records per physical record.

Number of Buffers

CARTAN obtains main memory from the operating system to use as buffers or page slots for disk input and output operations. The driver program may specify the maximum number of page slots that are to be acquired (≤ 32). CARTAN always tries to acquire at least four page slots.
Valid Function Codes

LOAD

LOAD indicates to CARTAN that the file is being defined and opened for the first time and that a series of insertions is forthcoming. The driver program must specify the mode of arithmetic and the number of coordinates to be stored. The data set referenced by the logical file name DDNAME may be an empty data set or one that had previously been used. However, any information present in the file will be destroyed.

If a file is opened for LOAD, the only valid commands are ISRT and CLSE. All others will be flagged as invalid and ignored.

OPEN

After a file has been defined, loaded, and closed again, subsequent processing is initiated with OPEN which logically connects the file to the program. All function codes are treated as valid, including ISRT which will extend the file. If the data set is empty, the open processing will fail.

On return from a successful open, CARTAN will have filled the mode and number of coordinates fields of the communication block. A file must be opened before any other function codes will be recognized.
CLSE

CLSE requests a wrap-up, including final write of any modified records to disk. Upon successful return, the record RBA field will contain the high used RBA as an indication as to actual space utilization of the file.

ISRT

A new record is inserted as a terminal record with the ISRT request. If necessary, a new node record is also built. The RBA of the new terminal record is returned for the driver program's use as desired.

GM

This is a request to Get Master node record; it would be used to start over at the root of the tree if performing a specialized search procedure.

GP

Climbing the structure to a higher level is accomplished by a Get Parent request. CARTAH retrieves the parent record of the last record retrieved.

GT

The next record at the same level in the tree is retrieved with a Get Twin request.

GC

The first record at the next level down in the tree is accessed through a Get Child request.
GD

If the driver program has the record RBA available, the corresponding record from the Cartesian file may be retrieved directly with Get Direct.

GN

The Get Next record in hierarchical sequence function is defined as: If the previous record accessed has a child, get that child; if it has no child, get the next twin; if there is no twin, i.e., the end of the twin chain was reached, get the twin of the parent of the previous record. Repeated requests using GN will walk through the entire file structure in this sequence.

GNT

The sequence described for GN is modified by not retrieving the child of the previous record. GNT would be used when it had been determined that a subtree is to be discarded.

The last seven function codes, GH through GNT, are provided as primitives for the unusual application that needs to follow a peculiar search strategy. They will each clear parentage if it had been set earlier. The first five of these codes may also set parentage by adding a "P" as the third character of the code, i.e., GNP, GPP, GTP, GCP, and
GDP. Parentage is set to limit a search to a particular subtree of the file structure and is primarily used with the next three function codes.

GNP

Unlike previous codes where a P in position three set parentage, Get Next in Parent uses a previously set parentage to retrieve records in a hierarchical sequence within a specified subtree. The GN function will walk though to the end of the file regardless of the staring point, while repeated use of GNP will traverse only the subtree as defined when parentage was set.

If parentage has been set by the GR function described below, CARTAM also performs a check using the INTERSECTION_FUNCTION to determine if the record intersects the search area. If the intersection is empty, the subtree consisting of the record and its children is automatically discarded and the twin record is immediately retrieved. If the record is a node and the intersection is limited to a single child of that node, that particular child is immediately retrieved, and it is noted that there will be no twin of that record to be retrieved later. In both cases, the check by INTERSECTION_FUNCTION is reapplied before returning the record to the driver program. If the intersection is neither empty nor a single child, the record is returned with the appropriate information fields filled.
Get Next in Parent, Twin, modifies the GNP sequence by skipping the child retrieval and discarding the subtree. This is done when the driver program applies a finer discrimination on a record than CARTAN can apply such as a true circle search as opposed to a rectangle search. The decision was made to only perform the simple rectangle search within CARTAN since specific applications could conceivably use any type of metric function for their discrimination purposes.

When the driver program makes the determination that it really knows that a node record is acceptable, or, in other words, it wants all of the subtree's terminal records without bothering to apply its discriminator, a Get Next in Parent, Leaves, series of requests will flush the subtree, presenting terminal records only. The term Leaves is used since the character T was used for Twin.
GR or GA

An area search is initiated with either of the equivalent Get Rectangle or Get Area requests. The INTERSECTION_FUNCTION will be used by CARTAN to check records during this GR and subsequent GNPx requests. The stack maintained by CARTAN is flushed and the search begins at the master or root record, setting parentage for GNPx.

GR L

If the rectangle search is the exact search required by the application, placing an "L" in position four will direct CARTAN to only return the terminals that are found inside the search rectangle on subsequent GNP or GNPL requests. After a GR L request, GNP and GNPL are equivalent.
CHNG

If a Cartesian file was loaded with a substantial amount of ancillary data in the terminal records, it is useful to be able to modify that information without having to reload the entire file. The CHange request tells CARTAM to replace the user data in the terminal record that had been retrieved on the previous call. CARTAM checks to see that the coordinates have not been inadvertently altered and that the new data string is not longer than the original string. If the new string is shorter, the terminal record's data area will be padded out to the original length with the pad character.

DLET

Any record in the Cartesian file may be DeLETed with the exception of the master root record. The structure pointers are adjusted to logically remove the record and a check is made to see if the ring now contains only one child. If so, the parent of the lone remaining child is replaced in its ring by that sole child. For integrity, CARTAM requires that the record be retrieved on the previous call. Note that either terminals or nodes may be deleted; deleting a node effectively deletes the entire subtree. Note also that CARTAM has no space reclamation capability -- deleting a record removes it from the structure, but the space is then unavailable for any future use until the file is reloaded!
Status codes as returned by CARTAM

NN (Two EBCDIC blanks) CARTAM successfully completed the requested function. New information has been updated as appropriate.

AD CARTAM did not recognize the function code; invalid code.

AI An error occurred while trying to open the file.
A numeric error code [3, pgs 58-60] from the operating system has also been placed in the RBA field of the communication block.

AJ A logical error was detected during a disk operation.
A numeric error code [3, pgs 67-69] from the operating system has also been placed in the RBA field of the communication block as for AI.

AN A mode error was detected: not H, F, E or D.

AO A physical error was detected during a disk operation.
A message was written to the program log and a numeric error code [3, pg 70] has been placed in the RBA field of the communication block as for AJ.

AX Too many coordinates were specified. The maximum is 512 or a total coordinate vector length less than one-half of the length of a physical record.
An error was detected on a change request. The change must be on a terminal that was retrieved on the previous call, the length of the user data must be the same or less, and the coordinates must not have been altered.

An error was detected in a delete request. The record to be deleted must have been retrieved on the previous call. The master root record cannot be deleted.

The requested record was not found. GE is typically returned during GNPlx processing.

There are no more records in the subtree being flushed by retrieving only terminals while using GNPl.

A duplicate record, coordinates and user data, was presented for insertion; the record was not inserted.

The user-supplied data to be stored with the terminal record is too long. The total length of user data, coordinates, and six bytes of structure data must be less than one-half the length of the physical record as stored on disk.

A short parameter list was presented to CARTAM, e.g., calling CARTAM with only the communication block and user data area, but not with the coordinate vector for an ISRT or CHNG.
CHAPTER VI

INSIDE CARTAM
FOR THE MAINTENANCE PROGRAMMER

The previous chapters have developed the basic algorithms and described the program I call CARTAM from a point of view intended for a prospective user of the system. This chapter deals with the fine detail required by a programmer assigned the task of reimplementing the system on different hardware or operating system or fixing CARTAM should it break.

The Cartesian Index file is a data structure maintained on a secondary storage medium, specifically a direct access disk or equivalent, which predicates usage of some sort of a disk address as the pointer value in the node and terminal records. The particular form of this disk address pointer depends upon the specific choice of the access methods as provided by IBM. Since we are concerned with random access to disk, there are actually only a few access methods available. The most primitive method of disk I/O provided by IBM is the execute channel program (EXCP) access method. However, this is rather too primitive as I have no desire to reinvent such things as physical error handling routines,
etc. The next alternative is the Basic Direct Access Method (BDAM) which would actually work quite well except that it does not handle variable length records with any great facility. If the records are defined as relatively large, then the internal blocking and deblocking could become somewhat messy, depending on the choice of notation for the record identification. As will be seen later, though, BDAM would have been quite acceptable.

The implementation of CARTAN as described here uses IBM's Virtual Storage Access Method (VSAM) [3,4] for physical access to the disk file structure. VSAM was primarily intended as a high performance replacement for the Indexed Sequential Access Method (ISAM), but does provide support for three basic types of direct access file organizations which can be used for almost any application. Since VSAM is used for basic system support in later versions of large operating systems as supplied by IBM, e.g., OS/VS2 Multiple Virtual Storage (MVS), and it isolates a program from device dependencies better than other methods, it seemed to be a good choice.

The direct counterpart to ISAM as provided by VSAM is a key sequential data set (KSDS) which is used to store data indexed by a unique primary one-dimensional key. However, the whole intent of this paper concerns multi-dimensional keys, so we have no appropriate key to suggest use of a KSDS.
VSAM also provides a counterpart to the BDAM file organization known as a relative record data set (RRDS). Unfortunately, an RRDS requires fixed length records which are referenced by "relative record numbers", and the concerns of a BDAM data set are applicable here as well.

The third structure supported by VSAM is an entry sequenced data set (ESDS) as a counterpart to the usual sequential file organization. However, VSAM does allow random access to any position in the file by means of a four-byte relative byte address (RBA), which turned out to be ideal for my purposes. An ESDS may be viewed as a unique virtual address space defined by a four-byte address ranging from 0 to 4,294,967,295. Early in the development process, it was intended to store node and terminal records as distinct records maintained by VSAM. However, as the development proceeded and more of the performance options as provided by VSAM were incorporated, it became desirable to perform blocking and deblocking within CARTAM rather than VSAM. This became a very simple masking operation as VSAM stores information on secondary storage in units of control intervals (CI) which may be almost any size from 512 bytes to 32,768 bytes, but are physically stored as multiples of a physical record which may be 512, 1024, 2048 or 4096 bytes in length. One of the performance options used by CARTAM results in the seemingly reasonable restriction of limiting the CI size to that of a physical record or a
maximum of 4,096 bytes. Each CI requires a minimum of seven bytes of control information, which leaves the remainder available for CARTAN's use. Thus, the largest record that may be stored by CARTAN is 4,089 bytes, but a further limit is rather arbitrarily imposed to limit a logical record to no more than half of a physical record in order to store at least two information records in one block. Keeping all of this in mind, CARTAN uses a VSAM ESDS as a logical memory of four billion bytes, storing the Cartesian Index file as a linked list with four-byte RBA pointer values.

An inability to extend a data set's space on disk is due to one of the performance options as used by CARTAN which prevents immediate usage of an empty or newly defined VSAM data set. Preformatting the data set with zero-filled records the first time an empty data set is opened solves the initial problem, and once preformatted, all records in the file may be retrieved on a random basis by relative byte address. However, when the original space allocation is exhausted, the data set will not automatically overflow into secondary extents when records are being inserted. If space is exhausted, there is no choice but to reallocate the file with more space and rebuild. As an indication of the actual utilization of the file space, the high used RBA is returned to the driver program when the file is closed.
Reflection at this point makes it obvious that the relative record organization of VSAM or even the Basic Direct Access Method may indeed be used. Careful selection of the physical record size to a proper power of two will allow CARTAN to operate with those file organizations with a minimum of change to the code.

The Cartesian file is built with two basic types of records, nodes and terminals. As mentioned earlier, these records consist of:

1) coordinate value(s),
2) a delta value,
3) a child pointer,
4) a twin pointer,
5) user data if a terminal, and
6) various flags.

If we examine some of these items, we find that first of all, a terminal record always has a null child pointer since terminal records are, by definition, those records with no children. The terminal record also corresponds to an original data point which has a delta value equal to zero, at least in terms of the file structure. The utility of a node or terminal flag now becomes apparent. A single bit serves to indicate the presence of a child pointer and a delta value or the mutually exclusive user data with, of course, its length.
The delta value as carried in the record also deserves some attention. While studying the algorithms, it becomes apparent that delta should probably be an integer power of two. In particular, consider a specific application on the computer using integer arithmetic. If one starts with the smallest non-zero delta value and proceeds through the tree structure towards the root, the delta is obviously such an integral power of two. Equally obviously, traversing the tree in the direction away from the root requires integer powers of two in order to prevent "gaps" due to a truncated division. If we now examine the usual internal representation of our delta value, we find that, for integer arithmetic, delta is stored as a fullword or halfword with only a single bit set to one somewhere in the (half)word. A natural method of storing this number in less space is to use a logarithmic representation, specifically log to the base of two. The normal internal representation of a floating point value is normalized hexadecimal with an exponent and mantissa. For an integer power of two, this mantissa is given by a single hexadecimal digit that is always in the leftmost position in the mantissa; only the 12 high order bits of a floating point delta are ever other than zero. Thus, we can store our delta value in the node record in only 12 bits, leaving the other 4 bits of a halfword available for some flags. Since a delta value is defined to be a non-negative number, I use the sign bit of
the representation to indicate whether delta is stored as a truncated floating point number or as a logarithm. There is an apparent ambiguity for a representation of zero, since it obviously cannot be stored as a logarithm. However, a "true zero" as used by IBM for both integer and floating point arithmetic is stored as all binary zeroes, so it works out very nicely.

The Cartesian Index file records are now constructed as follows. The length of the user data stored in a terminal record is variable, but since a terminal has a defined delta of zero, we may carry the length of the user data in the space otherwise occupied by delta. The list pointers, of course, are each four bytes long, while coordinate values may be two, four or eight bytes long, depending on the mode of arithmetic being used. Finally, after packing everything together into a record, we have:

```
| DLP | TWIN | COORDS... | Q | CHILD | UserData...
```

DLP is the delta/length and flags field, two bytes long. Expanding it out to the bit level:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>45</td>
</tr>
</tbody>
</table>
```

If bit 15 = "1", then "end of set" or record is the last record on the twin chain, i.e., TWIN actually points at the parent record, closing the ring.
If bit 14 = '1', then this record is a node, and bits 0-11 are the representation for delta.

If bit 0 = '1', then bits 2-7 are the log2(delta) and the antilog is obtained by shifting a value of 1 to the left this many positions.

Otherwise, bits 0-11 are to be moved to a work area and extended with zeroes to arrive at a representation suitable for arithmetic.

If bit 14 = '0', then this record is a terminal and bits 0-11 represent a scaled binary integer value depicting the length of the user data string stored behind Q.

Bits 12 and 13 are unused.

The TWIN pointer is a four-byte field and is present in all records. Actual interpretation is modified by bit 15 in the DLF field.

The COORDS field contains the coordinate vector for the record and is a*n bytes long where a = 2, 4 or 8 depending on the node of arithmetic.

Q is the quadrant indicator to label children of a parent node and is a bit string that carries the sign of the difference between coordinates of the record and the corresponding coordinates of the parent record. The length
of this field is \( q \) bytes where \( q = (n + 7)/8 \) using truncated integer division. The twin chain is also maintained in sorted order using the \( Q \) field as an ascending sort-key.

The four-byte CHILD pointer appears only in node records and points to the first of two or more records at the next lower level in the structure. The coordinates and delta of the node record define a square that completely covers all of its children. The records at the next lower level define a disjoint set of squares whose union is less than or equal to the parent square.

Finally, the user-data field is a variable length field carried in terminal records only. The actual length of this area is determined by the 12 high-order bits of DLP.

The primary argument in the CARTAM calling sequence is the communication block, which is where CARTAM receives all request instructions and returns status and other information. Figure 6-1 shows the assembly dummy control section (DSECT) definition. As the DSECT is the assembly program's view of the communication block described in the last chapter, most of the entries should be self-explanatory.
In order for CARTAN to operate, it needs a fair amount of additional main memory for control blocks, buffers and bookkeeping information. CARTAN must also be prepared to operate on more than one file at a time for the driver applications. Therefore, CARTAN obtains additional main memory for each file that is opened. The character string passed in as a DDNAME is used as a label to identify that block of memory as it pertains to any particular file. These blocks are linked on a bi-directional list and the proper file control area as defined in figure 6-2 is
FCBAREA  DSECT
  USING  * R12

FCBLABEL  DS  CL8  LABEL IS FILE DDNAME
PREVFCB  DS  A  BACKWARD AND
NEXTFCB  DS  A  FORWARD LINKS

IFGACB  DSECT=NO  GENERATED ACB
IFGRPL  DSECT=NO  GENERATED RPL
DS  OD
LWACBAR  EQU  IFGRPL-IFGACB
LNRPAR  EQU  *-IFGRPL

CISIZE  DS  P  CONTROL INTERVAL SIZE
AVSPAC  DS  P  AVAILABLE SPACE
ENDRBA  DS  P  ENDING RBA
LRECL  DS  P  LOGICAL RECORD SIZE = CISIZE-7

MVNODCS  DS  A (NODEAREA) FOR MVCL INST
DS  P (FLLNOD)
RCDADD  DS  A
DS  P (CHLDUD@)

CURRBA  DS  P  RBA OF RCD W/ CORE ADDR IN RCDADD
BUFB@  DS  A  LOCATION AND
@SUBPOOL  DS  OX
LNGBUP  DS  P  LENGTH OF PAGING AREA
PRIORT  DS  A  TOP OF LRU RING

DElWK  DS  D  EXPANDED DELTA FROM RETRIEVED RCD
PRNTDEL  DS  D  EXPANDED DELTA FOR NODEAREA

SPLTMSKS  DS  OXL6  MASKS TO SEPARATE RBA'S INTO
CIMSK  DS  P  CONTROL INTERVAL RBA
DSPHSK  DS  H  AND DISPLACEMENT
DS  H  UNUSED

LODEARGS  DS  OXL6  SEPARATED RBA TO BE LOADED
LODECI  DS  P
LODEDSP  DS  H
DS  H  UNUSED

DSECT of FCBAREA

Figure 6-2 (Part 1 of 3)
DIREC@ DS (MAX#BPES)XL(LDIRECTORY) PAGING DIRECTORY

MISCFLGS DS XL3 MISCL FLAGS
ISRTONLY EQU B*10000000* FILE OPENED FOR LOAD
FILEEXTND EQU B*01000000* FILE HAS BEEN EXTENDED
FSTISRT EQU B*00000001* FIRST INSERTION HAS NOT BEEN DONE
SENDPAD DS C PAD FOR USER DATA AREA

XTRAPRM DS A

SETFREGS DS XL4*80* R3 EX MASK FOR BIT STRING
 DS P*0* R4 COORDINATE VECTOR INDEX
 DS A(QSTRL) R5 BIT STRING ADDRESS
 DS P R6 INDEX INCREMENT
 DS P R7 INDEX LIMIT VALUE

SETPADDS DS A R8 A(SETSM.0)

GREX@ DS A R9 LOW SEARCH COORDINATES

GREXH DS A R10 HIGH SEARCH COORDINATES

GRFLAG EQU B*10000000* IF SET, DOING "GR" SEARCH
TRMONLY EQU B*01000000* IF SET, WANTS TERMINALS ONLY

TPPPRT DS H POINT IN STACK OF TEMP PARENT
STPPRT DS H POINT IN STACK OF PARENT
STKTOP DS H TOP OF STACK

DS X*0* ZEROES TO CLEAR BIT STRINGS

SETFLGS DS X SET INTERSECTION FUNCTION FLAGS

SNGLCHLD EQU B*10000000* INTERSECTION IS ONE CHILD ONLY

EMLTSET EQU B*00000010* INTERSECTION IS EMPTY

EMLTNX EQU B*00000010* SOME OF "SQUARE" OUTSIDE

XMTNINE EQU B*00000001* SOME OF SEARCH OUTSIDE

QSTRL DS XL64 BIT STRINGS
QSTRH DS XL64 OF DIFFERENCE SIGNS
QSTRO DS XL64

DS D UNUSED
DS D PERMANENT PIECE OF STACK

STACK DS 128D

MAXSTKL EQU **STACK

DSE of PCBAREA

Figure 6-2 (Part 2 of 3)
located each time CARTAN is entered. If a file control area cannot be located and the function code is other than OPEN, LOAD or CLSE, a status code of 'AD' is returned indicating an invalid function code. If an area is located and the function code is OPEN or LOAD, a status code of 'AD' is again returned.
FCBARA defines an area of main memory that is acquired on a page boundary, i.e., an even multiple of 4096. This is the main work area for CARTAM for the particular file being processed.

FCBLABEL is the file name from the communication block and is used as the identifying label for the work area.

PREVFPCB and NEXTFPCB are forward and backward links for the work area(s) and are anchored inside CARTAM directly. Since the register save area is also inside CARTAM, CARTAM is not re-entrant, but is serially re-usable.

IFGACB and IFGRPL are IBM supplied definitions of the access control block and request parameter list for the VSAM access method. CISIZE through LRECL receive information about the file for later use. ENDRBA indicates whether the data set already has information or if it must be preformatted; if so, AVSPAC is used to find out how long the data set is.

The four words beginning at MVNODCS are set up to load the control registers for an MVCL or CLCL instruction, each of which requires two addresses and two lengths. The fourth register also carries a pad character as the high order byte.

CURR RBA is used to retain the RBA of the most recently accessed terminal or node record. It is primarily used for checking on a delete or change request.
BUFRA, SUBPOOL and LNBUFF refer to the additional main memory obtained for input/output buffers or the paging area. PRIORT points at the top of the priority ring that is maintained for the paging directory (DIREC3) in a least recently used (LRU) manner.

DELUK is the work area for an expanded delta so that it may be used in arithmetic statements. It is filled in the LODE routine every time a new record is accessed. PRNTDEL is the corresponding expanded delta value for the record being constructed in NODEAREA.

SPLTHSKS is composed of CIMS K and DSPMSK which are used to split an RBA pointer into an RBA address of the control interval and a displacement. DSPMSK = CISIZE - 1 because CISIZE is an integer power of two as defined by VSAM. Then, CIMS K is simply the one’s complement of DSPMSK.

The masks are used as logical “and” masks against LODECI and LODEDSP which compose LODEARGS. The paging directory is then searched for LODECI; if not there, the oldest slot is picked to read in the proper control interval. The translation is completed by adding LODEDSP to the page frame address to arrive at the main memory address of the data record being referenced.

MISCF LGS are miscellaneous flags; use is obvious.
XTRAFRM is an extension of the paging directory. IBM provides a PGRLSE macro to specify release of a virtual memory area. This macro is used in the input/output routine as an attempt to gain efficiency by releasing a virtual page just prior to a read operation so that the operating system will not bring that page in from paging store simply to write over it with a new record from disk. The parameters for PGRLSE are the low address and the high address plus one of the area to be released; these addresses are exactly the page frame addresses as stored in the paging directory for the page slot being released along with the address of the next slot. XTRAFRM provides that "next slot" frame address for the last paging directory entry.

SETFREGS through GRIBS are preset values for the general purpose registers R3 through R10 used in the set intersection function. R3 contains a one bit mask to set a position in the Q bit string as addressed by R5. R4 is the index into the various coordinate vectors and is incremented by the value stored in R6 in a BILE instruction. R7 contains the limit for R4, i.e., (R7) = n*(R6) - 1. R8 has the address of the entry point into the appropriate arithmetic dependent code while R9 and R10 point at the lower and upper limit vectors. The set function also assumes that R1 points at the current node or terminal record being examined. SETFLGS carries the results of the set intersection function while QSTRH and QSTRL have been set according to the arith-
metic differences during the course of the calculations. QSTRO is used only during insertions to adjust the coordinates of the new node record being built as a parent.

TnpPrnt holds the location in the stack that is to be considered a temporary parent for the purpose of presenting, without further checking, all terminal records in a subtree that has been accepted. STKPRNT holds the location in the stack that is to be considered the parent level for Get Next within Parent processing while STKTOP always points at the top of the stack.

STACK is a 128 entry stack used to remember the parent backtrack chain along with the next twin entry. The parent backtrack trail is retained primarily for insertions to climb the parent chain in hopes that consecutive insertions were relatively "close" to each other, thus reducing chain chasing as much as possible. The twin pointers are retained for GNP processing to negate the requirement for input of a parent record solely to retrieve the twin pointer when accessing the parent's twin. Each entry in the stack is two words: the left word carries the parent backtrack trail, the right word carries the next twin. Upon exit from CARTAM, the top entry of the stack has zero in the left position; the right word has the child pointer of the record being returned to the driver program, which is zero if the record is a terminal. The second entry down in the stack has the
RBA of the record being returned as the left side value which will be the parent as the stack grows. The right side of this stack entry is the twin pointer from the returned record unless the record is marked as the end of a twin chain, in which case, zero is stored. This entry is always the next twin for GNP. As the stack is popped, either because the child value at the top was zero or the subtree is being bypassed, the twin value is picked up from the right side and stored in the left side. The twin and child pointers of that new record are then stored as before. Obviously, if the twin pointer was zero, the stack is simply popped one more level.

FILECNTL is a 32 byte area of control information to be stored on the file at RBA = 0. This information is derived from data provided when the function code was LOAD and then stored in the file. When the function code is OPEN, these 32 bytes are retrieved from the file and stored here. Only 16 bytes are used at this time.

NIUSDRBA contains the number of bytes used by CARTAN for insertions. It is the actual RBA of the next available byte in the VSAM file and is obtained and updated whenever a new record is inserted. If it has changed since the file was opened, the control information is rewritten to the file.

FLMODE holds the EBCDIC character defining the mode of arithmetic: H, F, E or D.
FL#COOR is a halfword integer value specifying the number of coordinates (n) in a coordinate vector.

FLCV contains the actual length of a coordinate vector in bytes. \((FLCV) = (FL#COOR) \times 2, 4 \text{ or } 8\) as appropriate.

DELTA through COORDS\# are symbolic equates defining the internal record structure. QSTR\# would be an equate to the beginning of the Q bit string in the record, but, due to the variable length of a coordinate vector, is stored as a value equal to COORDS\# plus the length of a coordinate vector.

QSTR\#M1 holds the length of the Q bit string less one. The IBM execute instruction requires this value for proper operation. \((QSTR\#M1) = ((FL#COOR) - 1)/8\) using integer division.

CHLDUD\# has the displacement to the child pointer for a node which is also the displacement to the user data for a terminal record. \((CHLDUD\#) = (QSTR\#) + (QSTR\#M1) + 1\)

FLLNOD holds the total length of a node for this file. The value stored in FLLNOD is 4 more than that in CHLDUD\#. In order to be able to store at least two logical records per physical record or control interval, the total length must be less than an arbitrary 2000 bytes or one-half the physical record length, whichever is smaller.
MODEAREA is work space to remember the contents of a possible parent record for insertions. That information is then modified while constructing the actual record that is to be entered into the file. RPLMSG is an overlay of the last 132 bytes and is used only by VSAM to return an error message. If such an error had occurred, any temporary record would be useless anyway.

Appendix A contains the entire assembly listing of the CARTEF routine. Within the routine are several logical units that are described here.

The LODE section of code is a closed subroutine to convert an RBA to a main memory address. The RBA is split into a control interval RBA plus a displacement into that CI. If the CI is already in memory, it is logically moved to the top of the LRU ring, the displacement is added to the proper frame address in R1, the delta is expanded, the twin pointer from the record is inserted in B2, and control is returned to the point of call. If the CI was not in main memory already, the oldest slot is determined from the end of the end of the LRU ring and the CI in that slot is written to disk if it had been modified. The new CI is then read into the frame and treated as above. The logic of this section of code was modeled after the paging scheme as described in REL Paging Services [9].
The overall logic of CARTAM is actually quite simple. On entry, a search is made for the proper PCBAREA, building a new one if necessary, the function code is examined, and control is transferred to the appropriate section. Most retrievals eventually go through the RTNVALS section which moves the coordinate vector to the driver program's area along with the user data if the record was a terminal. The area receiving the user data is padded out with the pad character in any case. The expanded delta value is also placed in the proper location and the MUST indicator is set.

A Get Master record is a request for the master node and would be issued if the driver program wished to restart an unusual search strategy. The stack pointers are reset to put the master RBA in the master (-1) position of the stack which is then adjusted with twin and child pointers as usual.

The RBA for a Get Direct request probably will not be found in the stack, but the stack is checked just to make sure. Note that a GD request will probably flush the stack which must be considered in Get Parent and Get Next requests.

The Get Twin and Get Child requests are simple pops of the stack. If a zero value is picked up after the pop, an indication of no record found is returned: STATUS = GE. The Get Parent is slightly more complicated due to the possibility of GD requests flushing the stack. If the stack
is exhausted during the pop operation, the twin chain must be followed to find the next parent record. All of the requests so far described may set parentage, in which case the location in the stack of the record being returned is stored in STKPRT as a parent marker.

The Get Next and Get Next in Parent operate in a similar fashion except that GNPx will terminate at the parentage as stored in STKPRT while GN will continue through the twin chains even after the stack is exhausted. GNPx processing is also slightly more complicated because the INTERSECTION_FUNCTION is used if the search had been initiated by a GR request. If the INTERSECTION_FUNCTION determines that only one child of a node is useful, that child is retrieved immediately and the next twin entry in the stack for that record is cleared, indicating no further records along that chain. If the record is a node and the fourth position of the function code is an "L", a branch is taken to the top of this section of code to immediately retrieve the next record.

The insertion algorithm attempts to take advantage of resident records and any actual proximity of consecutive inputs by popping the stack, using the parent backtrack trail. The stack is repeatedly popped until a node record is found which defines a square that actually contains the point I which is to be inserted. INTERSECTION_FUNCTION is
invoked in each instance with the I coordinate vector used as both the low and high limit vectors. When a good parent has been found, CARTAM turns around and descends the tree structure. Since a node P was found that contains I, it is known in which direction I lies in relation to the center of P because INTERSECTION_FUNCTION sets QSTRH and QSTRL in the FCBAREA. Thus, CARTAM walks the child/twin chain looking for the child with a matching Q string. If no record is found with a matching Q string, I is inserted as a terminal record in the proper position in the chain.

If a record C was found with a matching Q string, INTERSECTION_FUNCTION is invoked again to determine if I is inside C. If truly inside, CARTAM treats record C as the P node and loops back to continue with the descent. If the intersection was empty, a new node must be constructed to replace C in the chain we have been following. This new node becomes the parent of C and the new terminal I and the coordinate values of the new node are adjusted to ensure that C and I have differing Q strings in relation to their new parent.

If the intersection of C and I was an identity intersection, the coordinates of I matched the coordinates of C and C is either a terminal or a node with a zero-valued delta. If C is itself a terminal, it is replaced in its chain with a new node with a delta defined as zero and both
C and I are chained as children of that new node. If C was a node with zero delta, I is simply added as another child. In this case, all children, including C and I, have identical Q strings, indicating an all positive direction.

Change and delete requests require that the record be retrieved on the immediately preceding call to CARTAM. A change allows only the user data to be modified and it must not be extended. To ensure that a change request is not incorrectly used to change coordinates, CARTAM requires the coordinate vector which must still agree with the record in the file. If the coordinates still match, and the record is indeed a terminal, the user data is moved from the driver program's area into the file record, replacing the user defined data in entirety.

Only terminal records may be changed, but both terminal and node records may be deleted. A record is logically deleted by adjusting the pointers to skip over it. Space is not reclaimed! After the pointers have been adjusted, the length of the chain is examined to ensure that the chain is at least two members long. If the chain has only one member, the parent of the chain is replaced in its ring by the sole remaining child.
CHAPTER VII

CARTAM IN USE

The preceding discussion gave some general search algorithms with no particular rationale behind them. Let us look at some specific applications that have been implemented at Headquarters, Strategic Air Command. Our computer environment is an IBM System 370, Model 3033, using OS/VS2, Multiple Virtual Storage (MVS) as the operating system. Secondary storage consists of IBM 3330 Model 1 and Model 11 disks and IBM 3350 disks. In all of my examples, the data are points on the surface of the earth defined by latitude (lat) and longitude (lng).

The first file is stored on 18 cylinders of a 3330 disk volume and contains roughly 100,000 terminal records as data points, each carrying an average of 15 bytes of user-defined information. The latitude and longitude in this file are stored as arc seconds in signed binary integers with the convention of north and east positive. The driver program to load this file executes in approximately 55 seconds of central processor (CPU) time and 15 minutes elapsed time in our normal batch production multi-programming environment.
The metric function used to calculate distance on the earth is an implementation of a great elliptic evaluation which provides geodetic distance in meters; see appendix B for a discussion of VECTOR. Since this metric function tends to be expensive in computation, an estimator value has been devised which provides an estimated radius in meters of a circle guaranteed to completely enclose the square defined by a node or terminal record's coordinates. The value of this estimator E is:

\[ E = 45.0 > 43.645 = \sqrt{2} \times (1852 \text{ meters/60 arc secs}) \]

(1852 meters per nautical mile;
1 nautical mile per arc minute;
1 arc minute per 60 arc seconds)

It might seem that a better estimate of the radius for a circumscribing circle could be obtained by using VECTOR to measure the distance from the center of the square to the lower left corner for example. Unfortunately, some of the nodes near the root of the tree carry latitude values in the range of 145°. With VECTOR calculating geodetic distance, a much smaller number than expected is the result. Since search strategies will not be attempting any accurate determination of the inclusion of an area inside a node-defined square, rather the reverse, the upper bound approach with E was chosen.
Probably the simplest application of CARTAM is to search for those data points within an arbitrary circle. As a first approximation to the desired circle with center coordinates \((\text{lat}_0, \text{lng}_0)\), define a search rectangle to enclose the final desired circle. The delta latitude value is the appropriate number of arc seconds equivalent to the circle radius \(D_0\), while the delta longitude is that same number of arc seconds divided by the cosine of the latitude to allow for convergence at the poles. Therefore, the limit vectors are:

\[
\begin{align*}
\text{lvec} &= (\text{lat}_1, \text{lng}_1) \quad \text{and} \quad \text{hvec} = (\text{lat}_h, \text{lng}_h) \quad \text{where} \\
\text{lat}_1 &= \text{lat}_0 - D_0, \quad \text{lng}_1 = \text{lng}_0 - \left(\frac{D_0}{\cos(\text{lat}_0)}\right), \\
\text{lat}_h &= \text{lat}_0 + D_0, \quad \text{lng}_h = \text{lng}_0 + \left(\frac{D_0}{\cos(\text{lat}_0)}\right).
\end{align*}
\]

See figure 7-1 for the conditions that will be tested by algorithm CS below. Within the diagram:

- \text{line } \text{AX} = \text{DELTA(A)} \times E
- \text{line } \text{BY} = \text{DELTA(B)} \times E
- \text{line } \text{CZ} = \text{search radius} = D_0
- \text{line } \text{CA} = \text{VECTOR distance from C to A}
- \text{line } \text{CB} = \text{VECTOR distance from C to B}

\text{square A is inside search circle because}

\[
\begin{align*}
\text{CA} &< \text{CZ} - \text{AX} \\
\text{AX} &< \text{CZ} - \text{CA} \\
\text{AX} &< -(\text{CA} - \text{CZ})
\end{align*}
\]
THE CARTESIAN ACCESS METHOD FOR DATA STRUCTURES WITH N---ETC(U)

1979
S V PETERSEN

UNCLASSIFIED
AFIT-79-2250
Circle Search Conditions

Figure 7-1
square B is outside search circle because
CZ < CB - BY
BY < CB - CZ

Moving "GR" to the function code initially, we have:

Repeat

CALL CARTAM(COMM_BLOK, USER_DATA,
COORDS, DELTA,
lvec, hvec);

if STATUS_CODE = SPACES, then begin;
   Set AX := E * DELTA;
   Set CA := VECTOR(lat0,lng0,lat1,lng1);
   if AX ≤ CZ - CA, then begin;
      /* square A for example */
      Set FUNC := 'GNPL';
      repeat
         if TERMINAL, then
            Present terminal records
            as successful;
            CALL CARTAM(COMM_BLOK, USER_DATA,
                       COORDS, DELTA);
         until STATUS_CODE ≠ SPACES;
      Set FUNC := 'GNP';
      if STATUS_CODE = "GR", then
         Set STATUS_CODE := SPACES;
   end;
else

if \( 1 \times < CA - Cz \), then

Set FUNC := 'GNPT';

/* discard subtree (square B) */

else

Set FUNC := 'GNP';

/* to examine next level down */

end;

until STATUS_CODE ≠ SPACES;

This algorithm asks CARTAM for successive nodes and terminals inside an initial search rectangle. As a record is returned by CARTAM, it is checked to see:

1) if it is entirely within the final circle, then all terminals of the subtree are presented as found;
2) if it is entirely outside the final circle, the subtree is discarded;
3) if neither condition is met, the tree structure is descended one more level to examine the children.

The process is continued until no more nodes or terminals remain in the search rectangle to be examined. See appendix 6 for a COBOL program written for this task.

This particular driver program with the highly original name of ONETENE (variant of ONETIME) has been used extensively as a test vehicle during the development of CARTAM. It was written to display the results of a primitive circle
### Performance Statistics

<table>
<thead>
<tr>
<th>Number of search points</th>
<th>1</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8 page slots</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU seconds</strong> for run</td>
<td>.19</td>
<td>1.38</td>
<td>2.60</td>
<td>5.01</td>
<td>7.47</td>
<td>9.89</td>
</tr>
<tr>
<td><strong>CPU seconds/search point</strong></td>
<td>.19</td>
<td>.0243</td>
<td>.0243</td>
<td>.0242</td>
<td>.0243</td>
<td>.0243</td>
</tr>
<tr>
<td><strong>Number of reads/search point minimum</strong></td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>22</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td>22</td>
<td>24.04</td>
<td>24.09</td>
<td>24.01</td>
<td>24.02</td>
<td>24.30</td>
</tr>
<tr>
<td><strong>maximum</strong></td>
<td>22</td>
<td>32</td>
<td>34</td>
<td>34</td>
<td>41</td>
<td>51</td>
</tr>
<tr>
<td><strong>16 page slots</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU seconds</strong> for run</td>
<td>.19</td>
<td>1.29</td>
<td>2.41</td>
<td>4.55</td>
<td>6.98</td>
<td>9.78</td>
</tr>
<tr>
<td><strong>CPU seconds/search point</strong></td>
<td>.19</td>
<td>.0224</td>
<td>.0224</td>
<td>.0219</td>
<td>.0227</td>
<td>.0240</td>
</tr>
<tr>
<td><strong>Number of reads/search point minimum</strong></td>
<td>21</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>21</td>
<td>23/24</td>
<td>20/23</td>
<td>20</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td>21</td>
<td>22.28</td>
<td>22.23</td>
<td>22.14</td>
<td>22.19</td>
<td>22.43</td>
</tr>
<tr>
<td><strong>maximum</strong></td>
<td>21</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td><strong>32 page slots</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU seconds</strong> for run</td>
<td>.20</td>
<td>0.95</td>
<td>1.69</td>
<td>3.17</td>
<td>4.83</td>
<td>6.55</td>
</tr>
<tr>
<td><strong>CPU seconds/search point</strong></td>
<td>.20</td>
<td>.0155</td>
<td>.0151</td>
<td>.0149</td>
<td>.0155</td>
<td>.0159</td>
</tr>
<tr>
<td><strong>Number of reads/search point minimum</strong></td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>21</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>11/12</td>
<td>12</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td>21</td>
<td>11.74</td>
<td>11.15</td>
<td>10.69</td>
<td>10.77</td>
<td>10.68</td>
</tr>
<tr>
<td><strong>maximum</strong></td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 7-2
search as applied against the installation index file. Input is the Cartesian Index file which is to be searched, and a file of control cards, each of which contains the latitude and longitude of the center of a search circle. Test runs have usually been made with a 10,000 foot radius for the search. The overall logic consists in reading a control card, searching the Cartesian file for all data points within 10,000 feet and printing the accepted records. This procedure is then repeated for each card in the input file. Figure 7-2 presents a table of selected statistics as an indication of performance. The table is cumulative in nature; the different lengths of runs are from termination at specified numbers of control cards. For example, the statistics for 300 points were obtained by extending the 200 point run by 100 more points. The entries for number of reads are the numbers of physical disk accesses that were made for each control card read during the run.

An obvious extension to the circle search is a search for those installations inside the area defined by the mathematical union of k circles as shown in figure 7-3a. We modify algorithm CS by defining the search rectangle to include all circles and checking distances to the center of each circle instead of just the one; initially setting a flag to indicate "outside-all-circles", a loop is executed on the metric. Once again moving "GR" to the function code initially, we now have:
Inclusion Area Search Example
Figure 7-3a

Exclusion Area Search Example
Figure 7-3b
Set ACCEPT_SQUARE := "inside-a-circle";
Set REJECT_SQUARE := "outside-all-circles";
Repeat
CALL CARTAN (CORN_BLOK, USER_DATA,
    COORDS, DELTA,
    lvec, lvec);
if STATUS_CODE = SPACES, then begin:
    Set AX := E * DELTA;
    Set flag := "outside-all-circles";
    for i = 1 to n, do begin:
        Set CA := VECTOR(lati,langi,lat1,lng1);
        if AX <= CA, then
            Set flag := "inside-a-circle"
        else
            if AX > CA - CZ, then
                Set flag := "overlap-a-circle"
            end;
    end;
    if flag = ACCEPT_SQUARE, then begin:
        Set FUNC := "GZIP";
        repeat
            if TERMINAL, then
                Present terminal records
                as successful;
        CALL CARTAN (CORN_BLOK, USER_DATA,
            COORDS, DELTA);
        until STATUS_CODE # SPACES;
-97-

Set FUNC := 'GNP';
if STATUS_CODE = 'GN', then
    Set STATUS_CODE := SPACES;
    end;
else
    if flag = REJECT_SQUARE, then
        Set FUNC := 'GNPT';
        /* discard subtree */
    else
        Set FUNC := 'GNP';
        /* to examine next level down */
        end;
    end;
until STATUS_CODE ≠ SPACES;

The converse exclusion search strategy as shown in figure 7-3b is identical except that "inside-a-circle" is now the discard criterion, while "outside-all-circles" becomes the present successful terminals. Note that the distance check loop may be terminated immediately if the flag ever becomes "inside-a-circle". If the loop terminates with the flag still set at the initial value, the subtree is to be discarded. A rather neat programming dodge is to use CARMAH's function-code as the flag for the various conditions. Appendix B contains the COBOL program which performs this sort of search.
Algorithm CS may also be readily extended to provide a band search, at least in Cartesian space with a Euclidian metric \(d = \sqrt{x^2 + y^2}\). A band search is defined as the retrieval of all records within a given distance of a straight line passing through an appropriately defined "GR" search rectangle. As an example in two dimensions and assuming the appropriate units, the equation of the line is given by: \(Ax + By + C = 0\). Normalizing this equation by dividing by the \(\sqrt{A^2 + B^2}\) results in a metric function where the distance is determined by: \(d = ax + by + c\). The estimator \(E\) for a square defined by a file record is then given by: \(E = |a| + |b|\), which, when multiplied by the delta of the file record, gives the distance from the center of the square to a line parallel to the search line and that also passes through an appropriate corner of the square. Therefore, by replacing the two lines of algorithm CS as read:

\[
\text{Set AX := } E \cdot \text{DELTA;}
\]
\[
\text{Set CA := VECTOR(lat0,lng0,lat1,lng1);} \\
\text{with:}
\]
\[
\text{Set AX := } (|a| + |b|) \cdot \text{DELTA;}
\]
\[
\text{Set CA := } |a^X1 + b^Y1 + c|;
\]
we now have a band search for Cartesian space with a Euclidian metric.
Since CARTAH leaves the limit vectors available to the driver program at all times, a somewhat more extensive modification of algorithm CS suggests itself for a nearest neighbor search, by continually reducing the size of the search circle. As the search circle can be legitimately reduced only when a terminal record is examined, initialize the function code to 'GR L' to retrieve terminals only. Then the following algorithm will find the closest terminal record within an initial distance \( CZ \):

\[
\begin{align*}
\text{lat1} & := \text{lat0} - CZ; & \text{lng1} & := \text{lng0} - CZ/\cos(\text{lat0}); \\
\text{lath} & := \text{lat0} + CZ; & \text{lngh} & := \text{lng0} + CZ/\cos(\text{lat0});
\end{align*}
\]

CALL CARTAN(CORN_BLOK, USER_DATA,
COORDS, DELTA, lvec, hvec);

Set function code := 'GNPL';
while STATUS_CODE = blanks do begin;
Set \( CA \) := VECTOR(lat0,lng0,aplat,alng);
if \( CA < CZ \) then begin;
Set \( CZ \) := \( CA \);
\( \text{lat1} := \text{lat0} - CZ; \text{lng1} := \text{lng0} - CZ/\cos(\text{lat0}); \)
\( \text{lath} := \text{lat0} + CZ; \text{lngh} := \text{lng0} + CZ/\cos(\text{lat0}); \)
Save terminal information;
end;
CALL CARTAN(CORN_BLOK, USER_DATA,
COORDS, DELTA);
end;
When this algorithm terminates, the last terminal record saved will be the terminal closest to the initial search coordinates. Conceptually, terminals in the upper right quadrant ("++" direction) are successively examined, reducing the size of the search circle (probably) each time, until the closest terminal in that quadrant is found. Then examination of the remaining quadrants proceeds very quickly.

One final example has to do with a plotting application, in particular the presentation of maps with various levels of detail upon a graphical display device. If a particular area of the world were to be presented every time maps were requested, it would be a simple matter to construct a sub-image for display and call it up from secondary storage as required. However, if the areas to be mapped are defined by limits specified at run-time along with user-determined levels of detail, the number of pre-built maps becomes prohibitive due to the geometric explosion of combinations. The obvious solution is to build the maps upon request.

Our second example file is built in the Cartesian Index format for this purpose, containing as data the set of plottable points defining coastal and country boundaries. There are approximately 100,000 points in this file also, but this time our latitudes and longitudes are single precision floating point numbers expressed as arc radians. The terminal user-defined information contains a sequence
number for its relative position along the plotted line as well as a coastal/country boundary indicator. Once the application program determines the map limits from the user for the session, CARTAM is requested to retrieve those points within the rectangle defined by those limits. Using the user-defined data stored with the terminal records, these points may then be sorted internally, plotted and displayed on the screen.

Using CARTAM to retrieve map points for construction of background maps has resulted in a drastic reduction in map preparation time. This is aptly illustrated by a comment in an internal document, STAMPS Graphics Utilities User's Manual, 1 February 1977. "Since creation of an image of a map background requires a considerable amount of time (up to five minutes CPU) it would be impractical and inefficient to build these backgrounds on-line. ... the time required to build the maps would prohibit using them on the system." While the "five minutes" refers to CPU time for an IBM System 360, Model 85, and current experience has been on a System 370, Model 3033, the same map backgrounds are now being built in roughly five seconds elapsed time. The performance has improved to the extent that pre-built maps are no longer used; in fact, as the application user desires to examine a smaller area, the map limits are recomputed and the map backgrounds are completely redone each time.
The past few chapters have described the use of the CARTAM routine and the associated Cartesian Index File with some examples of actual applications. These examples have been limited to two dimensions, specifically latitude and longitude on the surface of the earth, but there has been no intention to imply that CARTAM is limited to two dimensions. Nor is it necessary that the coordinate values carry the same units, such as arc measure in the case of latitude and longitude. A better separation would be obtained if each of the coordinates are scaled such that the ranges of values are approximately the same, but, again, there is no hard and fast requirement imposed by CARTAM. As an example, the installation file that was described earlier can very easily be defined with three coordinates instead of two by adding a coordinate carrying a numeric representation of a category, for instance. Effectively, this would separate the installations into categorical layers which may prove extremely useful in some cases. Since CARTAM does not apply any specific metric function to the records, the number and type
of coordinates is totally at the discretion of the user who may then apply whatever metric function is deemed appropriate for discrimination.

A final thought has to do with possible optimizations of the Cartesian file for large read-only applications. The file as built by repeated insertions tends to have pointer chains spread randomly over the file, which increases the number of physical retrievals from secondary storage. One possibility would be to recopy the Cartesian file once it had been completely loaded. The initially-loaded file would be read in the Get Next hierarchical sequence and copied in that order onto the final file. This would allow any searches using the "GNP" philosophy to proceed in a monotonic manner through the final Cartesian file. The other alternative might be to recopy the initial file in such a way as to group as many nodes of the same level on the same physical record (control interval) as possible, building a many-way tree a la Knuth [8, pg 471]. The usefulness of this may be open to conjecture if the majority of the searches are small circle searches, since this type of search proceeds down a single path of the tree for several levels.
The CARTAN routine has proven itself as a very useful, generalized method to construct a multi-dimensionally-keyed file and provide extremely rapid access to desired records therein. The programs have been implemented in demonstrated efficient code and have proved themselves in a variety of complex applications. With the help of this document, additional applications of these techniques should be very straightforward with implementation in a minimum of time.
LIST OF REFERENCES


5. ____*, *OS/VS2 Access Method Services*. 2nd ed. GC26-3841-1, 1976

6. ____*, *OS/VS2 Supervisor Services and Macro Instructions*. 1st ed. GC28-0756-0, 1976


APPENDIX A

CARTAH SOURCE

CARTAH TITLE * PROGRAM TO HANDLE N-DIMENSIONAL INDEX * MACRO DEFINITIONS*

MACRO
REQUATE 5W
LCLA 5I,5J,5K
LCLC 5C
5C SETC "R"
5J SETA 6
5K SETA 2
AIF (T5W EQ '0') .A
5C SETC "5W"
.A AIF (5C EQ "F") .GO
5K SETA 1
5J SETA 15
.GO ANOP
5C.5I EQU 5I
5I SETA 5I+5K
AIF (5I LE 5J) .GO
.END

MACRO
5LBL 2R 6R
5LBL SR 6R,6R
.END

MACRO
5LBL LPAGE 6PG
5LBL DS 0H
AIF (T6PG EQ '0') .SKLD
AIF (6PG(1,1) NE '(*)') .LD1
AIF (6PG(1) EQ '(R1)') .SKLD
LR R1,6PG(1)
AGO .SKLD
.LD1 L R1,6PG
.SKLD BAL R14,LPAGE
.END
MACRO

6LBL MPAGE 6PG
6LBL DS ON
AIF (T'6PG eq 'O') .SKLD
AIF ('6PG'('1', 1) NE '*) .LD1
AIF ('6PG' eq '(R1)' ) .SKLD
LR R1,6PG('1')
AGO .SKLD
.LD1 L R1,6PG
.SKLD BAL R14,KEPAGE
END

MACRO
SETFUNC 6H
LCLC 6A,6C,6L
USING SET6H.0H,88
AIF ('6H' NE 'F') .H1
6C
SETC 'L'
SET6H.0H HVC 0(4, R5), DELWK SUBJECT OF EXECUTE IN RTHVALS
AGO .H5
.H1 ANOP
6A
SETC '6H'
AIF ('6H' NE 'H') .H2
SET6H.0H HVC 0(2, R5), DELWK+2 SUBJECT OF EXECUTE IN RTHVALS
AGO .H5
.H2 ANOP
6L
SETC '6H'
6C
SETC '6H'
AIF ('6H' NE 'E') .H3
SET6H.0H HVC 0(4, R5), DELWK SUBJECT OF EXECUTE IN RTHVALS
AGO .HED
.H3 AIF ('6H' NE 'D') .H4
SET6H.0H HVC 0(4, R5), DELWK SUBJECT OF EXECUTE IN RTHVALS
AGO .HED
.H4 BNOTE 8, 'BAD TYPE CODE'
AGO .ND
.H5 ANOP
.SET6H.00 L R0, PRNDDEL
SRA R0, 1
HALVE DELTA
AIF ('6H' NE 'F') .HALL
BMP SET6H.8
AGO .HALFP
.HED ANOP
.SET6H.00 LSH 0, PRNDDEL
ESH 0, 0
HALVE DELTA
.LT6H.R 0, 0
.HALL BZ SHUDHR
.HALFP ANOP
.SET6H.01 ST6L 0, PRNDDEL
SET6H.02 LSL 0, PRNDDEL ADD OR
EZ R3, DELSIGN TH QSTRO-QSTRL (R5), 0
BNO ++6
-108-

LNL.R 0,0   SUBTRACT DELTA BASED ON BIT STRING
A5A 0,COORDS3(R4,R1)
ST6A 0,COORDS3(R4,R1)

SET6H.0 LSA 0,COORDS3(R4,R1) COORDINATE IN FILE   EI
S6A 0.0(R4,R10) COORDINATE FROM SEARCH|ISRT   IX
BP SET6H.2
BR SET6H.1
C&C 0,DELWK
BL SET6H.3
B SET6H.4

SET6H.1 LP6L.R 0,0   (EI - IX) < 0
C&C 0,DELWK
BL SET6H.3
OI SETFLGS,INOTINE PART OF SEARCH OUTSIDE "SQUARE"
B SET6H.4

SET6H.2 EX R3,NEGHI   QSTBL-QSTRL(R5),0
C&C 0,DELWK   (EI - IX) > 0
BHR SET6H.3
OI SETFLGS,EMPTYSET INTERSECTION IS EMPTY

SET6H.3 OI SETFLGS,INOTINE PART OF "SQUARE" OUTSIDE

SET6H.4 LSA 0.0(R4,R9) LOW SIDE SEARCH COORDINATE  XL
S6A 0,COORDS3(R4,R1) FILE COORDINATE  E1
BP SET6H.6
BZ SET6H.5
EX R3,NEGLO   QSTBL-QSTRL(R5),0
LP6L.R 0,0   (XL - EI) < 0

SET6H.5 C&C 0,DELWK
BL SET6H.7
BHR R14
OI SETFLGS,INOTINE PART OF SEARCH OUTSIDE
B R14

SET6H.6 C&C 0,DELWK
BL SET6H.7
OI SETFLGS,EMPTYSET INTERSECTION IS EMPTY
BR R14

SET6H.7 AIF ("8H" NE "P") .ND
BZ SHDNVR
L RO,PRINTDEL FULL WORD INTEGER INFINITE DELTA
SRL R0,1
B SET6H.01

SET6TRYR EQU SET6H.00-SET6H.01 OFFSET FOR EI IN RTNVALS
SET6TRY1 EQU SET6H.00-SET6H.01 OUTER LOOP OFFSET IN F4A
SET6TRY2 EQU SET6H.02-SET6H.01 INNER LOOP OFFSET IN F4A

SET6TRY3 EQU SET6H.0-SET6H.01 LOOP OFFSET IN INTRSECT

ND DROP R8
* PUNCH A LINK EDITOR CONTROL CARD TO FORCE PAGE ALIGNMENT

PUNCH * PAGE CARTAH

TITLE ' PROGRAM TO HANDLE N-DIMENSIONAL INDEX'

CARTAH CSBCT
USING *,R15
B  PASTID
DC  AL1(L*ID)
ID  DC  C*CARTAH..SYSDATE..SYSSTIME*
PRINT NOGEN

PASTID STH  R14,R12,12(R13)
LE  R14,R13
STD  F0,SAVEPFR0
STD  F2,SAVEPFR2
CHOP  0,4
BAL  R13,PASTCONS
DROP  R15
USING *,R13
DC  16F'0'   SAVE AREA

PARHADDR DC  A(0)
PARENCT EQU  PARHADDR,1

SAVEPFR0 DC  D'0'
SAVEPFR2 DC  D'0'
SETPSAVE DS  1F'
ORG  SETPSAVE
XTMDSAVE DS  P
LODESAVE DS  7F
ORG

MASTERPG DC  A(L*FILECNTL)   RBA OF MASTER PAGE

REQUATE
REQUATE P

MAXBFRS EQU  32  MAXIMUM NUMBER OF BUFFERS
MINBFRS EQU  4  MINIMUM NUMBER OF BUFFERS

SHUTDOWN ABEND 97,DUMP,STEP
STKOVFLO ABEND 24,DUMP,STEP
TITLE: PROGRAM TO HANDLE N-DIMENSIONAL INDEX WORK AREA DEFINITIONS

CORNBLOK DSECT USING *,R11

CBBDSNAME DS CL8 DDNAME OF FILE
CBFUCN DS 0CL4 FUNCTION CODE
CBFUCN1 DS C
CBFUCN2 DS C
CBFUCN3 DS C
CBFUCN4 DS C
CBSTATUS DS CL2 RETURN STATUS
CBMDE DS C MODE OF ARITHMETIC
CBMORT DS X MODE/TERMINAL INDICATOR
CBRD DS F RBA OF RECORD RETRIEVED/INSERTED
CBRAUDL DS H MAX LENGTH OF USER DATA AREA
CBRUDL DS H TRUE LENGTH OF USER DATA
CBGETS DS H COUNTER FOR VSAM "GETS"
CBPUTS DS H COUNTER FOR VSAM "PUTS"

* REDEFINITION IN EFFECT WHEN FUNC = "LOAD"|"OPEN"

ORG CBMORT
CBPAD DS C USER DATA AREA PAD CHARACTER
CBBYS DS E # COORDINATES
CBBUFRS DS E # PAGING BUFFERS TO BE USED

ORG

DIRECTORY EQU 0,16
RBA EQU 0,4 RBA OF PAGE IN FRAME
FRM EQU 4,4 FRAME CORE ADDRESS
FLGS EQU 6,1
CWLADDR EQU 8,4 CORE ADDRESS OF VSAM CONTROL INFO
FWD EQU 12,4 FWD LINK ON LRU RING
PCBAREA DSECT USING *,R12
FCBLABEL DS CL8 LABEL IS FILE DDNAME
PREVPCB DS A BACKWARD AND
NEXTPCB DS A FORWARD LINKS
IPGACB DSECT=NO GENERATED ACB
IPGRPL DSECT=NO GENERATED RPL
DS OD
LWACBAR EQU IPGRPL-IPGACB
LWPLAN EQU *-IPGRPL
CISIZE DS P CONTROL INTERVAL SIZE
AVSPAC DS P AVAILABLE SPACE
ENDRBA DS P ENDING RBA
LRECL DS P LOGICAL RECORD SIZE = CISIZE-7
MNODCS DS A NODEAREA FOR HVCL INST
DS F (PFLMOD)
RCDADD DS A (CHLDUGA)
CURRBA DS P RBA OF RCD W/ CORE ADDR IN RCDADD
BUFRAY DS A LOCATION AND
#SUBPOOL DS OX
LNSGP DS F LENGTH OF PAGING AREA
PRIORT DS A TOP OF LRU RING
DELWK DS D EXPANDED DELTA FROM RETRIEVED RCD
PRNTDEL DS D EXPANDED DELTA FOR NODEAREA
SPLTNSKS DS OXL6 MASKS TO SEPARATE RBA'S INTO
CLNSK DS P CONTROL INTERVAL RBA
DSPHMSK DS H AND DISPLACEMENT
LODEARGS DS OXL6 SEPARATED RBA TO BE LOADED
LODCl DS F
LODEDSP DS H UNUSED
DIREGDS DS (MAX@BFRS)XL(L'DIRECTORY) PAGING DIRECTORY
BISCFILGS DS XL3 BISCL FLAGS
ISETOULY EQU B'10000000' FILE OPENED FOR LOAD
FILEITND EQU B'01000000' FILE HAS BEEN EXTENDED
FIRSTISHT EQU B'0000000001' FIRST INSERTION HAS NOT BEEN DONE
SENDPAD DS C PAD FOR USER DATA AREA
ITRAPRE DS A
SETFREGS DS XL4'80' R3 2X MASK FOR BIT STRING
DS F'0' R4 COORDINATE VECTOR INDEX
DS A(QSTNL) R5 BIT STRING ADDRESS
-112-

<table>
<thead>
<tr>
<th>DS F</th>
<th>R6 INDEX INCREMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS F</td>
<td>R7 INDEX LIMIT VALUE</td>
</tr>
<tr>
<td>SETPADDR DS A</td>
<td>R8 A(SETPM.0)</td>
</tr>
<tr>
<td>GRISE DS A</td>
<td>R9 LOW SEARCH COORDINATES</td>
</tr>
<tr>
<td>GREST DS A</td>
<td>R10 HIGH SEARCH COORDINATES</td>
</tr>
<tr>
<td>GRPLAG EQU B'100000000*' IF SET, DOING &quot;GR&quot; SEARCH</td>
<td></td>
</tr>
<tr>
<td>THRO EQU B'000000000*' IF SET, WANTS TERMINALS ONLY</td>
<td></td>
</tr>
<tr>
<td>TIPPERNT DS H</td>
<td>POINT IN STACK OF TEMP PARENT</td>
</tr>
<tr>
<td>SKPPRINT DS H</td>
<td>POINT IN STACK OF PARENT</td>
</tr>
<tr>
<td>STKTOP DS H</td>
<td>TOP OF STACK</td>
</tr>
<tr>
<td>DS I'0*'</td>
<td>ZEROS TO CLEAR BIT STRINGS</td>
</tr>
<tr>
<td>SETPLGS DS X</td>
<td>SET INTERSECTION FUNCTION FLAGS</td>
</tr>
<tr>
<td>SNGLCCHILD EQU B'100000000*' INTERSECTION IS ONE CHILD ONLY</td>
<td></td>
</tr>
<tr>
<td>EMPTYSET EQU B'000000100*' INTERSECTION IS EMPTY</td>
<td></td>
</tr>
<tr>
<td>EMOTWIN EQU B'000000010*' SOME OF &quot;SQUARE&quot; OUTSIDE</td>
<td></td>
</tr>
<tr>
<td>EMOTINE EQU B'000000001*' SOME OF SEARCH OUTSIDE</td>
<td></td>
</tr>
<tr>
<td>QSTRL DS XL64</td>
<td>BIT STRINGS</td>
</tr>
<tr>
<td>QSTRH DS XL64</td>
<td>OF DIFFERENCE SIGNS</td>
</tr>
<tr>
<td>QSTRO DS XL64</td>
<td></td>
</tr>
<tr>
<td>DS D</td>
<td>UNUSED</td>
</tr>
<tr>
<td>DS D</td>
<td>PERMANENT PIECE OF STACK</td>
</tr>
<tr>
<td>STACK DS 128D</td>
<td></td>
</tr>
<tr>
<td>FILECMTL DS XL32</td>
<td>FILE CONTROL INFORMATION</td>
</tr>
<tr>
<td>ORG FILECMTL</td>
<td></td>
</tr>
<tr>
<td>HUSDRAA DS F</td>
<td>CURRENT HIGH USED RBA (ISRT USES)</td>
</tr>
<tr>
<td>FLNOD DS C</td>
<td>H</td>
</tr>
<tr>
<td>DS C</td>
<td>UNUSED</td>
</tr>
<tr>
<td>FLNCOOR DS H</td>
<td>8 COORDINATES</td>
</tr>
<tr>
<td>FLNCOOR FS H</td>
<td>(FLNCOORerase*(FLNCOOR)erase*(FLNCOOR))</td>
</tr>
<tr>
<td>DELTA R</td>
<td>0,2</td>
</tr>
<tr>
<td>RCDELGS EQU 1,1</td>
<td>4 BITS</td>
</tr>
<tr>
<td>PARENT EQU B'00001*' END OF TWIN CHAIN</td>
<td></td>
</tr>
<tr>
<td>MORDCD EQU B'00010*' RECORD IS A NODE</td>
<td></td>
</tr>
<tr>
<td>TWIN EQU DELTA+L*DELTA+4 TWIN POINTER</td>
<td></td>
</tr>
<tr>
<td>COORDS EQU TWIN+L*TWINS START OF COORDINATE VECTOR</td>
<td></td>
</tr>
<tr>
<td><em>QSTRL EQU COORDS+</em>(FLNCV)</td>
<td></td>
</tr>
<tr>
<td>QSTRLH DS N</td>
<td>L STRING LENGTH MINUS 1</td>
</tr>
<tr>
<td>CHILDPTR DS N</td>
<td>CHILD PTR</td>
</tr>
<tr>
<td>FLMOD DS N</td>
<td>TOTAL LENGTH OF A NODE RECORD</td>
</tr>
<tr>
<td>* = L<em>DELTA+L</em>TWINS+<em>(FLNCV)erase</em>(QSTRLH+1)+L*CHILDPTN &lt;= 2000</td>
<td></td>
</tr>
</tbody>
</table>

* SO FAR 16 bytes are left |

| ORG |
| NODEAREA DS XL2000 | NODE CONSTRUCTION WORKSPACE |
| FCBLNG EQU E-FCBLABEL HOPEFULLY < 4096 |
| ORG E-132 |
| RPLMSG DS CL132*RPL MESSAGE AREA* |
TITLE " PROGRAM TO HANDLE N-DIMENSIONAL INDEX * INITIAL ENTRY"

CARTAH CSECT
PASTCONS ST R13,8(R14) LINK SAVE AREAS
ST R14,4(R13)
ST R1,PARMADDR SAVE PARAMETER LIST ADDRESS
L R11,0(R1)
CLI 0(R11),0 OPTIONAL PARM COUNT PRESENT?
BNE PASTPC
L R15,0(R11) PARAMETER COUNT
LA R1,4(R1)
ST R1,PARMADDR STEP PAST COUNT
L R11,0(R1) ADDRESS OF COMBLOK
B PSTPC

PASTPC LA R15,1 COUNT PARAMETERS
LA R0,5 NEED AT MOST 6

CNTPC TM 0(R1),B'10000000'
BO STPCT
LA R1,4(R1)
LA R15,1(R1)
BCT R0,CNTPC

STPCT STC R15,PARMCNT
BVC CBSTATUS,=C' * INITIAL GOOD RETURN STATUS
L R9,=A(NOPCB)
USING NOPCB,R9
LA R12,NULLABEL

FINDFCB LR R8,R12
L R12,NEITFCB LOOK FOR PROPER PCB
CLC CBDDNAME,PCBLABEL
BH FINDFCB
BLR R9 NOT ON CHAIN; GO MAKE A NEW ONE
CLC CBFUNC,=C'CLSE' IS ON CHAIN; R12 IS NOW BASE
BE CLSE
B CHKFUNC
DROP R9

LTORG

NULLABEL DC 2F'0' HEAD AND
DC A (0)
DC A(ENDLABEL)

ENDLABEL DC 2F'-1' TAIL FOR PCB CHAIN
DC A(NULLABEL)
DC A(0)
-114-

TITLE * PROGRAM TO HANDLE N-DIMENSIONAL INDEX * CONVERT AN RBA TO A CORE ADDRESS*

MKPAGE  RVI  LOD5+1,X'F0'  MARK A CI AS MODIFIED
        B  LODE

LDPAGE  RVI  LOD5+1,X'00'  LOAD ONLY; WILL NOT BE CHANGED

LODE    STH  R14.R4,LODSAVE
        ST  R1,COREBBA
        ST  R1,LODECI  RBA OF CI +
        STH  R1,LODEDSP  DISPLACEMENT
        NC  LODEARGS, SPLITSKS
        BZ  LERADITO  ZERO RBA IS AN ERROR
        LA  R4, PRIORT-FWD  START AT TOP OF PRIORITY LIST

LOD1    L    R0,FWD(R4)
        LTR  R0,RO
        BZ  LOD2  CI WAST NOT IN CORE
        LR  R3.R4
        LR  R4.RO
        CLC  LODEC1(3), RBA(R4)
        BNE  LOD1

LOD5    OV  FLGS(R4), ** MARK IF NECESSARY
        MVC  FWD(L*FWD,R3),FWD(R4)  RESET LBU LIST
        MVC  FWD(L*FWD,R4),PRIORT
        ST  R4, PRIORT
        L  R1,PRIOR(R4)  GET CORE ADDRESS
        AH  R1, LODEDSP
        ST  R1, RCDADD
        ZR  R2
        TH  RCDFLGS(R1), NORDCD
        DNO  LOD8  TERMINAL HAS NO DELTA STORED
        TH  DELTA @(R1), B'10000000'
        DO  LOD7  STORED AS LOG2
        L  R2, DELTA @(R1)
        N  R2, X'FFFF0000'  CLEAR GARBAGE
        B  LOD8

LOD7    IC  R15, DELTA @(R1)  TAKE ANTILOG2
        LA  R2, 1
        SLL  R2, 0(R15)

LOD8    ST  R2, DELWK  STORE EXPANDED DELTA
        LN  R14.R0, LODSAVE
        LH  R3.R4, LODSAVE+20
        L  R2, TWINS(R1)  EXIT WITH TWIN PTR IN R2
        TH  RCDFLGS(R1), PARENT
        BNEOR  R14
        ZR  R2  ZERO R2 FOR END OF TWIN CHAIN
        BR  R14
LOD2
LA R2, IPGEPRL
MODCB RPL=(R2), AREA=(*, FRM(R4)), ABG=(S, RBA(R4))
TH FLGS(R4), X 'FO', IS IT MARKED?
BZ LOD4
WI FLGS(R4), X 'OF', CLEAR MARK FLAG
LA R14, 1
AH R14, CB#PUTS
STH R14, CB#PUTS
PUT RPL=(R2) WRITE OUT MODIFIED CI

LOD4
MVC RBA(L'RBA, R4), LODECI RBA OF CI TO READ
LA R14, 1
AH R14, CB#GETS
STH R14, CB#GETS
L R0, FRM(R4) TRY TO TELL MVS NOT TO-bother
L R1, FRM+L'DIRECTRY(R4) PAGING IN AREA
PGRLSE LA=(0), HA=(1)
GET RPL=(R2)
B LOD5

XTLST
XLST LERAD=(LERADXT, A), SYNAD=(SYNADXT, A)

LERADXT0
LA R0, 16 LOGICAL ERROR EXIT
ST R0, CBRBA
B LERADXT1

LERADXT SHOWCB RPL=(1), AREA=(S, CBRBA), LENGTH=4, FIELDS=PDBK
LERADXT1
MVC CBSTATUS, =C'AJ'
B RTN

SYNADXT
MVC RPLMSG+10 (2), WTO MSG+2 PHYSICAL ERROR EXIT
LH R15, RPLMSG+4
STH R15, RPLMSG+8
LA R15, RPLMSG+4(R15)
MVC 0 (4, R15), WTO MSG+8
WTO MF=(E, RPLMSG+8) DISPLAY ERROR MESSAGE ON JES
MVC CBSTATUS, =C'AO'
B RTN

WTO MSG WTO '1234', ROUTCDE=(11), DESC=(6), MF=L

LTORG
TITLE ' PROGRAM TO HANDLE N-DIMENSIONAL INDEX * PERFORM REQUESTED RETRIEVE FUNCTION  

CHKFUNC

| LH | R7, PLLCV | LENGTH OF COORD VECTOR |
| LH | R8, QSTBLN1 | LENGTH OF Q BIT STRING - 1 |
| CLC | CBFUNC, = 'C' ISRT' |
| BE | ISRT |
| TM | MISCPLGS, ISRTONLY |
| BO | NOTG |
| L | R1, RCDADD |
| ZR | R15 |

SHOULD BE A "G" REQUEST

CLI | CBFUNC1, 'C' G' |
| BH | NOTG |
| BL | CHKDLCH |

CLI | CBFUNC2, 'C' A' |
| BH | NOTG |
| CLI | PARMCNT, 4 |
| BL | SHORTLIST |
| IC | R15, CBFUNC2 |
| IC | R15, CHDTBL (R15) |
| B | NOTG (R15) |

CHDTBLX

DC | 64X'00' |

CHDTBL

EQU | CHDTBLX-C'A'M+1 |
ORG | CHDTBL+C'A'M C'ABCD' |
DC | AL1 (GR-NOTG, 0, GC-NOTG, GD-NOTG) |
ORG | CHDTBL+C'A'M C'MNOPQR' |
DC | AL1 (GR-NOTG, GN-NOTG, 0, GP-NOTG, 0, GR-NOTG) |
ORG | CHDTBL+C'A'M |
DC | AL1 (GT-NOTG) |
ORG |

SHORTLIST MVC | CBSTATUS, = 'C'SL' TOO FEW ARGUMENTS |
B | RTN |

NORCD MVC | CBSTATUS, = 'C'GE' |
B | RTN |

POPIT ZR | R0 |

POP STACK FOR MOST "G" REQUESTS

| LH | R14, STKTOP |
| AH | R14, = AL2 (-L'STACK) |
| BR | R15 |
| STH | R14, STKTOP |
| L | R0, STACK+4 (R14) |
| BR | R15 |

CHKDLCH CLC | CBFUNC, = 'C' CHNG' |
| BE | CHNG |
| CLC | CBFUNC, = 'C' DLET' |
| BE | DLET |

NOTG MVC | CBSTATUS, = 'C'AD' INVALID CODE |
B | RTN |
GP
BAL R15,POPI
BM NORC
BAL R15,POPI
BM NORC
BAL R15,POPI
BM GPNS
L R0,STACK(R14)
B GETIT

GPNS
L R0,THIN(R1) RAN OUT OF STACK ENTRIES
LTR R0,R0
BZ NORC
TH RCDPLGS(R1),PARENT
BO GETIT
HERE IT IS
LPAGE (R0)
B GPNS

GT
BAL R15,POPI
BM NORC
BAL R15,POPI
BM NORC
LTR R0,R0
BZ NORC
B GETIT

GC
BAL R15,POPI
BM NORC
LTR R0,R0
BZ NORC
B GETIT

GR
B GRCODE
AREA SEARCH INITIALIZATION

GW
CLI CBFPNC3,C*P* GET NEXT
BE GNPCODE
(WITHIN PARENT)
BAL R15,POPI
BM GN001
TH RCDPLGS(R1),MODRC R STACK WAS EMPTY;
BWO GNT
LH R15,CHLDUO
L R0,0(R15,R1)

GW001
LTR R0,R0
BZ GWT
CLI CBFPNC3,C'T* IS SUBTREE TO BE SKIPPED?
BNE GETIT

GNT
BAL R15,POPI
BM GNTNS
LTR R0,R0
BZ GNT
B GETIT

GNTNS
L R0,STACK STACK WAS EMPTY;
LTR R0,R0
BZ NORC
LPAGE (R0)
L R0,THIN(R1)
TH RCDPLGS(R1),PARENT
BO GNTNS1
B GETIT
GET MASTER PAGE
GET DIRECT
CHECK STACK TO SEE IF IT IS THERE
START STACK WITH THIS RECORD
PUSH LATEST RECORD
INDICATE A NODE FOR STARTERS
LOAD PAD CHARACTER
RETURN RBA TO CALLER
GET COORDINATE VECTOR
LOAD PADING GROUP
RETURN (14, 12), T, RC=0
-119-

PUSHCH
ZR  R0  ZERO TO LEFT SIDE
ZR  R2
TH  BCDPLGS(R1),MODRCD CHILD (IF ANY) TO RIGHT
BNO PUSHTW
LE  R2,CHILDUD
L   R2,0(R2,R1)
PUSHTW
LE  R14,STKTOP IF PUSHING TWIN, CURRENT RBA
CH  R14,-AL2(HAISTKL-L*STACK) IN LEFT SIDE
BH  STKOFLO BECOMES PARENT
ST  R0,STACK(R14) FOR ALL ABOVE IT
ST  R2,STACK+4(R14) IN STACK
LA  R14,L*STACK(R14)
STH R14,STKTOP
BR  R15

POPITP
ZR  R2  POP STACK FOR GNOME PROCESSING
LE  R14,STKTOP
CH  R14,TMPRNT MARKED AS TEMP PARENT?
BH  GNPGM YES
CH  R14,STKPRNT MARKED AS PARENT?
BH  NORCD YES
AH  R14,-AL2(-L*STACK)
BH  NORCD STACK IS EMPTY
STH R14,STKTOP
L   R2,STACK+4(R14)
BR  R15

GNPGM
IC  TMPPRNT,TMPRNT FINISHED SUBTREE
TH  GRXH,TRHONLY
BNO NORCD
NI  GRXH,IX*FF*-TRHONLY
NVC CBSTATUS,=CMG
B  RTH

GRCODE
CLI PARMCT,6 AREA SEARCH SETUP
BL  SRLIST
L   R15,PARMAADDR
NVC GRXL,(L'GRXL+L'GRXH),16(R15) ADDRS OF LIMIT
NVI GRXH,GRFLAG VECTORS
XC  TMPPRNT(L'TMPRNT*L'STADDR),TMPRNT
CLI CBFUNC4,C*L
BNE +8
OI  GRXH,TRHONLY
NVI SETFLGS,0
NVC STKTOP,-AL2(-L*STACK)
LPAGE MASTERPG START WITH MASTER PAGE
B  GNPG
GUPCODE

GUPCODE VI SETFLGS,0
BAL R15,POPITP
CLI CBFUNC4,C'T'
BNE GUP0
TH GEXH3,TRONLY
BO GUP2
STH R14,THPRTN
O1 GEXH3,TRONLY
B GUP2

GUP0 CLI CBFUNC4,C'T'
BNE GUP2

GUP1 VI SETFLGS,0
GUPOCO BAL R15,POPITP
GUP2 LTR R0,R2
BZ GUP1
LPAGZ (R0)
TM SETFLGS,SNGLCHLD LOOKING FOR A SINGLE CHILD?
BNO GUP4

GUP4 BAL R15,PUSHTW
VI SETFLGS,0
TH GEXH3,GRFLAG GR PROCESSING?
BNO GUP5

GUP5 BAL R15,PUSCH
TH RCDFLGS (R1),MODRCD
BNO RTVALS RETURN ALL TERMINALS
TH SETFLGS,SNGLCHLD IF ONLY ONE CHILD OF
BO GWPOCO INTEREST, GET IT IMMEDIATELY
TH GEXH3,TRONLY
BO GUP1 CALLER WANTS TERMINAL ONLY
B RTVALS

LAST CBD READ IS TO BE MARKED TO RETRIEVE ALL TERMINALS OF
SUBTREE

IS CHILD SUBTREE TO BE DISCARDED?

LOOKING FOR A SINGLE CHILD?

NOT YET

FOUND IT; NEED NO MORE
PROGRAM TO HANDLE N-DIMENSIONAL INDEX INSERT FUNCTION

CLQRL CLC 0(0,R14),QSTRL
NEGLQ OI QSTRL-QSTRL(R5),0
NEGLH OI QSTRH-QSTRL(R5),0
DELSIGN TH QSTRH-QSTRL(R5),0

ISRT CLI PARMCNT,3
BL SHRTLIST
L B15,PARHADDR
L R6,4(R15) ADDRESS OF USER DATA
L B4,8(R15) ADDRESS OF COORDINATE VECTOR
LA R5,0(R4)
STH R4,R5,GXILH
TH CBTRUUDL,B*1000000
BO ISRT07 UD TOO LONG
LH B15,CBTRUUDL
AH R15,ENDUDL
SLL R15.1 TOTAL LENGTH MUST BE LESS THAN
C R15,LRECL HALF OF THE LRECL
BNH ISRT08

ISRT07 MVC CBSTATUS,=C'IU' USER DATA TOO LONG
B RTH

ISRT08 TH MISCFGS,FRSTISRT
BNO ISRT09
KI MISCFGS,X*FF'-FRSTISRT
LPAGE NASTERPG FIRST INSERTION ON A LOAD
BAL R15,CALCQSTR
NIP 0
B F6NEWTH

ISRT09 BAL R15,POPIT TOP OF STACK IS PROBABLY ZEROS
ISRT10 BAL R15,POPIT
BNH ISRT12
ZR R14
STH R14,STKTOP

ISRT12 L B9,STACK-L'STACK(R14) CLIMB PARENT DIRECTION
LPAGE (B9) UNTIL NODE COMPLETELY COVERS
BAL R15,INTERSECT NEW COORDS
B ISRT10 +0
TH SETFLGS,ENOTIX +4
BNO ISRT10
B B2
-122-

B2

LH R2,R5,AVNODCS
HVCL R2,R4
HVC PENTDEL,DELWK AS PROBABLE PARENT
LB R10,STKTOP
BAL R15,PUSCH
LTR R9,R2
B2 SHUDVR

C3

LPAGE (R9) LOOK FOR CHILD IN SAME DIRECTION
LA R14,COORDS@ (R7,R1) AS NEW COORDINATES
EX R8,CLQRL CLC 0 (0,R14),QSTRL
BH F6NEWTRN MISSED IT
BE QE
ST R9,STACK(R10) NOT YET
ST R2,STACK+4 (R10) (PUSH TWIN)
LTR R9,R2
BNZ C3
B F6NEWTRN NOT ON CHAIN INSERT TERMINAL

QE

LA R14,COORDS+NODEAREA (R7)
EX R8,HVQL
BAL R15,CALCQSTR ARE NEW COORDS INSIDE RECORD?
NOP 0
TH SETFLGS,EMPSET+EMOTIX
BZ XENATCH MATCHING POINT COORDS
BO F40 NO; EMPTY INTERSECTION
ST R9,STACK(R10) YES; TOTALLY INSIDE
ST R2,STACK+4 (R10)
B Q

CALCQSTR LA R14,QCALC CALC A FULL Q BIT STRING
B INTRO

INTRSECT LA R14,INRTST EXIT IMMEDIATELY IF NO INTERSECTION
INTRO STH R3,R10,SETPSAVE
LB R3,R10,SETPREGS
HVC SETFLGS (L*SETFLGS+L*QSTRL+L*QSTRH+L*QSTRO),SETFLGS-1
B SENTRY3(R8)
INRTST TH SETFLGS,EMPSET
BO INTREXIT EXIT TO +0 IF EMPTY
QCALC SBA R3,1
BNZ INTRLOOP
LA R3,B'00000000' NEXT BYTE ON Q STRING
LA R5,1(R5)
INTRLOOP BILE R4,R6,SENTRY3 (R8)
LA R15,4 (R15) EXIT TO +4 IF FULL LOOP WAS RUN
INTREXIT LB R3,R10,SETPSAVE
BR R15
-123-

F40
STM B1,B10,SETFSAVE
STC TWINA+NODEAREA,TWINA(R1)
LA R14,COORDS@+NODEAREA(R7)
BX R8,0VQLE
LA R1,COORDS(R1)
ST R1,GRXLW
LA R1,NODEAREA NODEAREA HOLDS NEW NODE INFO
LH R6,R10,SETFREGS+12

F4A
STC QSTR0,QSTRL
STC SETFLGS(L,SETFLGS+L*QSTRL+L*QSTRH),SETFLGS-1
LH R3,R5,SETFREGS
BAL R14,SETTRY1(R8) ADJUST COORDS IN NODEAREA
SRA R3,1 AND CALCULATE Q'S
BNZ F4B
LA R3,B'10000000'
LA R5,1(R5)

F4B
BXLE R4,R6,SETTRY2(R8)
CLC QSTRL,QSTRH
BE F4A STILL SAME Q, ADJUST AGAIN
ST R10,GRXLW RESET GRXLW
CLI SETTRY1+L*SETF00(R8),X'8A' "SRA"_OPCODE?
BNZ F4D
L R14,PRMTDEL
LH R15,X'7F00' CALC LOG2(DELTA)

F4C
LA R15,X'100' (R15)
SRA R14,1
BNZ F4C
STM R15,PRMTDEL

F4D
STC DELTA(2,R1),PRMTDEL
LH R1,R10,SETFSAVE QSTRL IS FOR LAST RECORD READ
B F5NEWMOD QSTRH IS FOR NEW TERMINAL

XENATCH
TH RCDFLGS(R1),NODRCD COORDS MATCH W/ DELTA = 0
BO XENATCH0
LH R2,R5,HRVNOCS RECORD IS A TERMINAL;
MVCL R2,R4 NEED A PARENT NODE W/ DELTA
XC DELTA@+NODEAREA,DELTA@+NODEAREA OF ZERO
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6HEWTRM</td>
<td>( R_{C, P} ) + NODEAREA, NODEA</td>
</tr>
<tr>
<td>LH</td>
<td>( R_{1}, P_{L} ) NODE LENGTH OF A NODE</td>
</tr>
<tr>
<td>BAL</td>
<td>( R_{14}, I_{TND} ) SLOT</td>
</tr>
<tr>
<td>CLC</td>
<td>QSTR, QSTRH</td>
</tr>
<tr>
<td>BE</td>
<td>P6HEWTRM</td>
</tr>
<tr>
<td>BE</td>
<td>P6CHTRM</td>
</tr>
<tr>
<td>L</td>
<td>( R_{15}, S_{T} + 4 ) (R10)</td>
</tr>
<tr>
<td>ST</td>
<td>( R_{15}, S_{T} ) (R10)</td>
</tr>
<tr>
<td>B</td>
<td>P6HEWTRM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXHATCH0</td>
<td>ST</td>
</tr>
<tr>
<td>ST</td>
<td>R2, S_{T} + 4 (R10) CHILDREN</td>
</tr>
<tr>
<td>LH</td>
<td>( R_{10}, S_{T} ) TOP</td>
</tr>
<tr>
<td>BAL</td>
<td>( R_{15}, P_{S} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6CHTRM</td>
<td>L</td>
</tr>
<tr>
<td>P6CHLP</td>
<td>LPAGE</td>
</tr>
<tr>
<td>LH</td>
<td>( R_{15}, C_{B} R_{U} U D L )</td>
</tr>
<tr>
<td>LR</td>
<td>( R_{14}, R_{6} )</td>
</tr>
<tr>
<td>LR</td>
<td>( R_{5}, R_{15} )</td>
</tr>
<tr>
<td>LH</td>
<td>( R_{4}, C_{H} L D U D @ )</td>
</tr>
<tr>
<td>AR</td>
<td>( R_{4}, R_{1} )</td>
</tr>
<tr>
<td>CLCL</td>
<td>( R_{4}, R_{14} )</td>
</tr>
<tr>
<td>BE</td>
<td>IISTAT</td>
</tr>
<tr>
<td>ST</td>
<td>( R_{0}, S_{T} ) (R10)</td>
</tr>
<tr>
<td>LTR</td>
<td>( R_{0}, R_{2} )</td>
</tr>
<tr>
<td>BWZ</td>
<td>P6CHLP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6HEWTRM</td>
<td>LH</td>
</tr>
<tr>
<td>AR</td>
<td>( R_{1}, C_{H} L D U D @ )</td>
</tr>
<tr>
<td>HVI</td>
<td>RCDPLGS+NODEAREA, 0</td>
</tr>
<tr>
<td>HVO</td>
<td>DELTA+NODEAREA, CBTUUDL USER DATA AREA LENGTH</td>
</tr>
<tr>
<td>LA</td>
<td>( R_{4}, C O O R D S + N O D E A R E A )</td>
</tr>
<tr>
<td>LB</td>
<td>( R_{5}, R_{7} )</td>
</tr>
<tr>
<td>L</td>
<td>( R_{2}, G R X R @ )</td>
</tr>
<tr>
<td>LR</td>
<td>( R_{3}, R_{5} )</td>
</tr>
<tr>
<td>MVCL</td>
<td>( R_{4}, R_{2} )</td>
</tr>
<tr>
<td>EX</td>
<td>( R_{8}, H V Q R H )</td>
</tr>
<tr>
<td>BAL</td>
<td>( R_{14}, I_{T N D} S L O T )</td>
</tr>
<tr>
<td>LH</td>
<td>( R_{5}, C B T R U U D L )</td>
</tr>
<tr>
<td>LB</td>
<td>( R_{7}, R_{5} )</td>
</tr>
<tr>
<td>MVCL</td>
<td>( R_{4}, R_{6} )</td>
</tr>
<tr>
<td>B</td>
<td>RTRBA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISTAT</td>
<td>MVCL</td>
</tr>
<tr>
<td>B</td>
<td>RTRBA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVRQL</td>
<td>MVCL</td>
</tr>
<tr>
<td>HVRQL</td>
<td>MVCL</td>
</tr>
<tr>
<td>HVRQN</td>
<td>MVCL</td>
</tr>
</tbody>
</table>
-125-

XTHDSLOT ST  R14,XTHDSAVE
OI  MISCLG'S,FILEXTH
L  R4,HIUSDRBA  NEXT AVAILABLE RBA
LR  R5,DSPHSK
NR  R5,R4
AR  R5,R1
C  R5,LRECL  ROOM IN CI?
BNH  XTHDO  YES
LR  R5,R1  NO,
V  R4,CIMSK  STEP TO NEXT CI
AL  R4,CISIZE

XTHDO
AR  R1,R4
ST  R1,HIUSDRBA  NEW AVAILABLE RBA
LR  R10,STRTOP  IF DOING ISRT, STACK
CH  R10,-AL2(L'STACK) SHOULDN'T HAVE < 1 ENTRY
BL  SHUDNVR
L  R1,STACK-L'STACK(R10)
ST  R4,STACK-L'STACK(R10) NEW RECORD GOES TO LEFT
LTR  R1,R1  SIDE
BZ  XTHD1
HPAGE (R1)  INSERT NEW RECORD ON TWIN CHAIN
HV  TWINA+NODEAREA,TWINB(R1)
ST  R4,TWINB(R1)
TH  RCDRLGS(R1),PARENT
BNO  XTHD2
HI  RCDRLGS(R1),I'PP'-PARENT RCD JUST LINKED TO
OI  RCDRLGS+NODEAREA,PARENT WAS END OF TWIN CHAIN
B  XTHD2
ITND1
HPAGE STACK-2*STACK(R10) INSERT NEW RECORD AS
LH B14,CHLDUD@ FIRST CHILD OF PARENT
L B2,0(R14,R1)
ST B4,0(R14,R1)
LA B14,NODEAREA
ST B2,TWIN@ (R14)

ITND2
TH BCDPLGS*NODEAREA,NODRC
BNO ITND3
LH B14,CHLDUD@
ST B2,NODEAREA (R14)
HPAGE (R2)
HVC TWIN@*NODEAREA,TWIN@ (R1)
ST B4,TWIN@ (R1)
TH BCDPLGS (R1),PARENT
BNO +6
OI BCDPLGS*NODEAREA,PARENT
OI BCDPLGS (R1),PARENT
LA B14,COORDS@ (R7,R1)
EX B8,HVQRL HVC 0 (0,R14),QSTRL

ITND3
ST B2,STACK-L*STACK+4 (R10)
LA B1,NODEAREA
BAL R15,PUSHCH
HPAGE (R8) LOAD AND MARK NEW CY
L B15,LBEC
L B10,PRIOR
L B10,PRH (R14)
AR B10,R15 POINT AT AND THEN
HVI 0 (R14),0 ADJUST VSAM CONTROL INFORMATION
STH B5,1 (R14)
STH B5,3 (R14)
SR B15,R5
STH B15,5 (R14)
LN B2,R5,HVNODCS
TH BCDPLGS*NODEAREA,NODRC
BNO +6
LE B5,R3 FULL LENGTH IF NODE
HVCL B4,R2
L B14,ITNDSAVE
BR B14
TITLE ' PROGRAM TO HANDLE N-DIMENSIONAL INDEX *  
CHANGE/DELETE FUNCTIONS' 

CHNG 
CLI  PARMCNT,3 
BL  SHRTLIST 
CLC  CBRBA,CURRRBA  MUST HAVE JUST BEEN RETRIEVED 
BNE  CHNGX 
TM  RCDPLG5(R1),MODRCD 
BO  CHNGX  CAN'T CHANGE DATA ON A NODE 
L  R9,PARMADDR 
L  R6,8(R9) 
LR  R3,R7 
LA  R2,COORDS(R1) 
CLCL  R2,R6  ENSURE COORDINATES WEREN'T CHANGED 
BNE  CHNGX 
LH  R5,DELTA(R1) 
SRL  R5,4 
L  R6,4(R9) 
LR  R7,CBTRUUDL 
CLR  R7,R5  CHECK LENGTH 
BH  CHNGX 
MPAGE  CBRBA 
LH  R4,CHLDUD& 
AR  R4,R1 
ISC  R7,B*1000*,SENDPAD 
NVCL  R4,R6  REPLACE USER DATA FIELD 
B  RTN 

CHNGX  
NVC  CBSTATUS,=""  
B  RTN 

DLETX  
NVC  CBSTATUS,=""  
B  RTN 

DLET  
L  R6,CBRBA 
CL  R6,MASTERPG  CAN'T DELETE MASTER RECORD 
BNE  DLETX 
CL  R6,CURRRBA  MUST HAVE BEEN JUST RETRIEVED 
BNE  DLETX 
LH  R9,CHLDUD& 
NVCL  R9,CBRBA,CBRBA 
RVC  RCDPLG5+NODEAREA,RCDPLG5(R1)  SAVE FLAG 
L  R3,TWIN(R1)  AND TWIN POINTER 
LH  R10,STKTOP 
SH  R10,=AL2(3*L*STACK) 
BNE  DLETO1 

DLETO1  
ZI  R10  PARENT NOT IN STACK 
L  R0,TWIN(R1)  WALK TWIN CHAIN TO FIND IT 
TM  RCDPLG5(R1),PARENT 
BO  DLETO2  FOUND IT 
LPAGE  (R0) 
B  DLETO1
DLET02
ST R0, STACK (R10)

DLET03
ST R2, STACK+4 (R10) (ENSURE PARENT'S TWIN IN STACK)
LA R14, COORDS (R7, R1) LOOK FOR PREDECESSOR
EI R8, RVQLR
MVQ QSTRH (TWIN+L*TWIN), 0 (R1) SAVE Q, TWIN PTR.
CL R6, P (R9, R1)
BNE DLETWIN

DLETCHILD
ST R3, 0 (R9, R1) SUCCESSOR IS NOW FIRST CHILD
LPAGE (R3)
LTR R2, R2
BE LOWETWIN

ST R0, STACK+4 (R10) DELETED RECORD WAS
ZR R0 FIRST OF ONLY TWO CHILDREN, LEAVE
ST R0, STACK+L*STACK (R10) STACK W/ SUCCESSOR AS
LA R15, 2*L*STACK (R10) FIRST (UNRETRIVED) CHILD
STH R15, STKTOP OF PARENT OF "X" B RTN

DLETWIN
ST R0, 0 (R9, R1) PARENT NOT IMMEDIATE PREDECESSOR
ZR R0 REMEMBER FIRST CHILD
BE DLET1

LPAGE (R0) WALK TWIN CHAIN
CLR R2, R6
BE DLETT2
LTR R0, R2
BNE DLETWIN ABEND 95, DUMP, STEP

DLETT2
ST R0, STACK+L*STACK (R10) SAVE IN LEFT SIDE OF
MPAGE (R0) STACK
ST R3, TWIN (R1)
TH RCDPLGS+MODEAREA, PARENT WAS "X" ON END OF
BUO DLETT3 CHAIN?
BUO RCDPLGS (R1), PARENT
ZR R3
CLR R4, R0 IS PREDECESSOR FIRST CHILD?
BE NOCHILD YES
BE DLETT3
ZR R0 PREDECESSOR IN PLACE OF "X", BUT SHOW
ST R0, STACK+2*L*STACK (R10) NO CHILD AS CHILD OF
ST R0, STACK+2*L*STACK+4 (R10) PRED (X) HAS BEEN
LA R15, 3*L*STACK (R10) PRESENTED EARLIER.
STH R15, STKTOP B RTN
* RECORD DELETED WAS ONE OF ONLY TWO
ON CHAIN
PREDECESSOR IS PARENT

LONETWIN NPAGE (R3) (RCDFLGS (R7), X*PF*-PARENT REPLACE
LONECHLD WI RCDFLGS (R7), X*PF*-PARENT REPLACE
MVC MVC W (L+W, R1), WMV+QSTRH TWIN POINTER
HI RCDFLGS+QSTRH, PARENT
OC RCDFLGS (L+RCDFLGS, R1), RCDFLGS+QSTRH ITS FLAG
LA R14, COORDS (R7, R1) AND Q STRING
EX R8, MVC MVC O (O, R1), QSTRING
L R5, STACK (R10) RBA OF PARENT TO BE REPLACED
RH R10, =AL2 (L*STACK) STORE
BM LONE03

ZB R10
L R0, TWINS (R1)
TM RCDFLGS (R1), PARENT
PC LONE02
LPAGE (R0)
B LONE01

LONE02
ST R0, STACK (R10)

LONE03
L R0, STACK (R10)
LPAGE (R0)
ST R2, STACK (R10) ENSURE PARENT'S TWIN IN STACK
CL R5, 0 (R9, R1)
BE LONE01 REPLACED PARENT FIRST ON CHAIN
L R0, 0 (R9, R1)
LA R9, TWINS

LONE05
LPAGE (R0) REPLACED PARENT IS ALONG TWIN CHAIN
CLB R5, R2
BE LONE10
LTR R0, R2
Bnz LONE05
B DLETNVR

LONE10
ST R4, STACK (R10) STORE PREDECESSOR IN
LTR R4, R4
Bnz LONE11
ST R3, STACK+4 (R10) PRED (X) IS A PARENT
LA R15, 2*STACK (R10) SUCCESSOR IS NON-NULL
LB R4, R3
B LONE12

LONE11
ST R3, STACK+2*STACK (R10) PRED (X) IS NON-NULL
ST R3, STACK+2*STACK+4 (R10) SUCC IS NULL
LA R15, 3*STACK (R10)

LONE12
STH R15, STACK (R10)
NPAGE (R0)
ST R4, 0 (R9, R1) STORE AS CHILD OR TWIN
B RTN
TITLE ' PROGRAM TO HANDLE N-DIMENSIONAL INDEX * MODE DEPENDENT "SET" FUNCTIONS'

LTORG

PUSH PRINT
PRINT GEN

SETFUNC F

SETFUNC H

SETFUNC E

SETFUNC D

POP PRINT

TITLE ' PROGRAM TO HANDLE N-DIMENSIONAL INDEX *
INITIALIZATION SECTION'

USING NOFCB, 29

NOFCB

CLC CBFUNC, "C"CLSE" DID NOT FIND
BE RTN

CLC CBFUNC, "C"OPEN"
BE NEWFCB

CLC CBFUNC, "C"LOAD"
BNE NOTG INVALID FUNCTION CODE

LH R2, CB#IS

CH R2, =AL2(8*QSTRL)

CHK NODE

VFC CBSTATUS, "C"AI"
B RTN
-131-

CHKMODE CLI CBNODE,C*D*  
BL MODEERR ERROR  
CLI CBNODE,C*H*  
BH MODEERR ERROR  
CLI CBNODE,C*G*  
BNE NEWFCB  
MODEERR MVC CBSTATUS,=C*AM*  
B RTN  

NEWFCB LB R7,SPFCHNG+2  
GETMAIN BU,LV=(R7),BHNDY=PAGE,SP=SUBPOOL0  
LR R6,R1  
LA R14,CBDDNAME  
LA R15,L*CBDDNAME  
MVCL R6,R14  
ST R1,NEXTFCB-FCBAREA (R8)  
ST R1,PREVF CB  
ST R12,NEXTFCB-FCBAREA (R1)  
LR R12,R1  
ST R8,PREVF CB  
GENCB BLK=ACB,DDNAME=(*,CBDDNAME),EILST=XTLST,  
LENGTH=LWACB,WAREA=(S,IPGACB),GEN AN ACB*  
MAREA=(S,RPLMSG),MLEN=L*RPLMSG, FOR FILE*  
MACRP=(CNV,DIR,ICI,IN,OUT,UBF)*  
CLC CBFUNC,=C*OPEN*  
BE OPENINIT  
MVI MISCLFLGS,ISRONLY+FRSTISBT  
MVC FLMODE,CBNODE  
STH R2,FL#COOR  
ZR R3  
IC R3,CBNODE  
SLL R3,3  
MODE CHARACTER * 8  
LB R4,MODETB-8*C*D*+6 (R3) INFINITE DELTA/FLAGS  
STH R4,DELTA@+NODEAREA FOR MASTER RECORD  
LB R4,MODETB-8*C*D*+4 (R3) LENGTH OF COORDINATE  
HB R4,FL#COOR  
STH R4,FLLCV  
LENGTH OF COORDINATE VECTOR  
BCTR R2,0  
FLOOR((#X+7)/8) - 1  
SBL R2,3  
= FLOOR((#X-1)/8)  
STH R2,QSTRLH1  
LENGTH OF Q BIT STRING MINUS 1  
LA R5,L*DELTA@+L*TWIN@+1 (R4,R2)  
STH R5,CHLDU&D  
DISPLACEMENT TO CHILD|USER DATA  
LA R5,4 (R5)  
CH R5,=AL2 (L*NODEAREA)  
BNE STLNODE  
AXERR MVC CBSTATUS,=C*AX*  
B CLSE3
STLMOD  STH  R5,PLLMOD  FINAL NODE LENGTH
LA  R5,L*FILECNTL(R5)
ST  R5,HIUSDRBA
XC  XTINDSAVE,XTINDSAVE
LA  R8,CARTINIT
BAL  R10,OPNINIT
CLC  HIUSDRBA,LRECL
BH  AXERR  LRECL TOO SMALL
LM  R4,R6,CISIZE
LTR  R6,R6
BNZ  CLSINIT
BCTR  R5,0  EMPTY DATA SET; PREFORMAT CI'S.
L  R2,PRIORT
HDCB  RPL=PRPL,AREALEN=(*,CISIZE),
*  RECLEN=(*,LRECL),AREA=(*,PRM(R2))

INITLOOP  PUT  RPL=PRPL
BXLE  R6,R4,INITLOOP

CLSINIT  CLOSE  CARTINIT  NOW DOWN TO WORK WITH REAL ACB
LA  R8,IPGACB
BAL  R6,MODOPN
L  R3,MASTERPG
HPAGE  (R3)  INITIALIZE MASTER PAGE
LR  R4,R7
SR  R4,R3
L  R5,LRECL
LA  R14,FILECNTL
L  R15,HIUSDRBA
MVCL  R4,R14
B  FININIT

MODETBL  DC  A(SETDOM),H*08*,XL2*7F83* D
DC  A(SETDOM),H*04*,XL2*7F83* E
DC  A(SETDOM),H*04*,XL2*9F03* F
DC  2F*0*  G
DC  A(SETDOM),H*02*,XL2*8F03* H
OPEN AN EXISTING FILE

OPENINIT LA R8,IPGACB
BAL R10,OPENINIT
L R3,MASTERPG
LPAGE (R3)
LR R4,R1
SR R4,R3
MVC FILECTL,0(R4) BRING IN FILE CONTROL INFO
MVC CBMODE,FLMODE RETURN MODE
MVC CB#XS,FL#COORD 5 # COORDS

FININIT MVC SENDPAD,CBPAD SAVE USER AREA PAD CHARACTER
ST R3,STACK-L*STACK MASTER PAGE RBA IN PERM STK
MVC STACK-L*STACK+4(L*TWING),TWING(R1)
BAL R15,PUSHCB
ZR R15
IC R15,FLMODE
SLL R15,3
LA R3,B'10000000' PRESET REGS FOR "SET" FUNCTION
ZR R4
INDEX
LA R5,QSTRL A(Q STRING)
LH R6,NODETBL-8*C*D'+4(R15) INDEX STEP
LH R7,FLLCV
BCTR R7,0 INDEX LIMIT
L R8,NODETBL-8*C*D'(R15) A(MODE SPECIFIC CODE)
STN R3,R8,SETPREGS
LA R2,NODEAREA A(NODEAREA)
LH R3,FLNOD L*NODE
L R4,ECODADD A(CURRENT RECORD)
LH R5,CHLDUDD A*NODE W/O CHLD PTR OR USER DATA
STH R2,R5,NVNODCS PRESET VALUES FOR NVCL INSTRS
B RTN

MODOPN MODCB ACB=(R8),DDNAME=(*,CBDDNAME)
OPEN ((R8))
LTR R15,R15
BZE R6
SHOWCB ACB=(R8),AREA=(S,CBBBA),LENGTH=4,FIELDS=ERROR
MVC CBSTATUS,-C*'AI'
B CLOSE3
OPMilit BAL R6,MODOPN
SHOWCB ACB=(R8),AREA=(S,CISIZE),LENGTH=12,
FIELDS=(CINT,AVSPAC,ENDRBA)
L R6,CISIZE
BCTR R6,0
STH R6,DSPMSK RBA DISPLACEMENT MASK
L R14,ENDLABEL
XR R14,R6
ST R14,CIMSK 1'S COMPLEMENT OF DSPMSK
SH R6.=H'6E'
ST R6,LRECL
LR R0,CB0BUFES LOAD # BUFFER PAGES BEING REQ.
XC CB#GETS(L,CB0GETS,L,CB0GETS+L,CB0PUTS),CB0GETS
CH R0,=*+10
BHR **8
LA R0,MAX#BUFES
MH R0,CISIZE+2
ST R0,PRNTDEL+4 MAXIMUM AMOUNT OF CORE REQ.
LA R0,MIN#BUFES
MH R0,CISIZE+2
ST R0,PRNTDEL MINIMUM AMOUNT OF CORE REQ.
LA R5,PRNTDEL
LA R3,BUFES
GETMAIN WU,LA=(R5),A=(R3),BNDRT=PAGE,SP=SUBPOOL$  

L R1,BUFES
L R14,CISIZE
L R15,LNBUP
MVIV $SUBPOOL, SUBPOOL$
SR R15,R14
AR R15,R1
LA R3,DIRECTR
ST R3,PRIORT
L R0,ENDLABEL LOAD A MINUS 1

SETFRM LR R4,R3 INITIALIZE PAGING DIRECTORY
LA R2,0(R6,R1) (R1) * (LRECL)
LA R3,L*DIRECTR(R4)
STH R0,R3,RBA(R4)
BLE R1,R14,SETFRM
XG FWD(4,R4),FWD(R4) CLEAR LAST LINK
ST R1,FRB(R3) STORE IN XTRFRM FOR PGRLESE
GENCB BLK=RPL,ACB=(S,IFGCAB), GENERATE AN RPL
* LENGTH=LNBPLAR,WAREA=(S,IFGRL),
* MSGAREA=(S,RPLMSG),MSGLEN=L*RPLMSG,
* AREALEN=(*,CISIZE),
* OPTCD=(CVN,DIR,SYM,NUP)
BR R10
CLSE MVC CBRBA,HIUSDRBA
TM MISCFILGS,FILEXTD
BNO CLSE0
BPAGE MASTERPG
S R1,MASTERPG
MVC HIUSDRBA-FILECNTL(L*HIUSDRBA,R1),HIUSDRBA
CLSE0 LA R4,IFGPL
L R2,PRIORT
CLSE1 TM FLGS(R2),X*PO*
BZ CLSE2
MODCB RPL=(R4),AREA=(*,FBN(R2)),ABG=(S,RBA(R2))
MI FLGS(R2),X*OP*
PUT RPL=(R4) WRITE OUT ANY MARKED CI'S
CLSE2 L R2,PRND(R2)
LTR R2,R2
BNZ CLSE1

LA R4,IFGACB
CLOSE ((R4))
CLSE3 L R0,LOGBUFF
LTR R0,R0
BZ CLSE4
L R1,BUPRD
FREEMAIN R,A=(1),LV=(0)
CLSE4 LM R14,B15,PREVFPCB
ST R14,PREVFPCB-PCBAREA(R15)
ST R15,NEXTFCB-PCBAREA(R14)
L R0,SPFPCBLNG
FREEMAIN R,A=(R12),LV=(0)
B RTN

CARTINIT ACB MACRF=(ADR,SEQ,NCI,OUT,NUB),EXLST=XTLST
PRPL RPL ACB=CARTINIT,OPTCD=(ADR,SEQ,NUP,MVE),
ARG=XTMDSAVE

SUBPOOLS EQU 17 SUB POOL NUMBER
SPFPCBLNG DC AL1(SUBPOOLS),AL3(PCBLNG)

LTORG
END
APPENDIX B

Subroutine VECTOR

VECTOR is a subroutine written as an implementation of the Schrieter-Thomas method to compute the great elliptic distance and normal section azimuth between two sets of geodetic coordinates on a selected spheroid. The method was obtained from ACIC Technical Report Number 80, "Geodetic Distance and Azimuth Computations for Lines over 500 Miles." The following comments were extracted from that report concerning "Types of Positions".

If the results of a distance and azimuth computation are to have any meaning, the terminal points used as basic data must be geodetically related, i.e., the end points must be derived from field measurements originating from a fixed point and computed along a common surface (ellipsoid). The starting point is usually defined in terms of latitude and longitude, either astronomical or geodetic, and the ellipsoid by the parameters a and b. If the initial point is fixed astronomically, the surfaces have what is known as an astro-orientation. Geometrically, this means that the geoid and ellipsoid surface coincide at that point and the fixed starting position is common to both surfaces. To the geodist it means that the normal to the ellipsoid coincides with the local vertical at that point and the components of the deflection of the vertical are zero. The astro-geodetic orientation differs from the preceding in that it compensates for the surface departure by correcting the angles between the geometrical normals and the true local verticals.

Positions on the earth's surface defined with respect to such initial quantities form a geodetic system or datum. Those derived from different datums are unrelated and consequently are unusable for inverse computations. The results would be in error and the magnitude of the error would correspond to the effect of the differences in the initial quantities of their datum. Certainly, accurate distance and azimuth cannot
be expected if the terminal points of the line are referred to different origins and possibly computed along different surfaces of unequal size.

Generally, the positions available for an inverse computation are of three types:

a. Geodetic positions such as described above.
b. Astronomic positions, latitude and longitude of which have been derived instrumentally by direct observations of celestial bodies.
c. Map positions obtained from cartographic sources.

Type a. are the most accurate although one very seldom finds two points as widely separated as 6000 miles referred to the same datum. The second type, b., astronomic points, refer to positions on the geoid and should not be used since the geoid is not a geometrical surface. To use these for computational purposes is to assume that the two surfaces are coincident and the definition of each point identical on both surfaces. This assumption could easily result in distance errors as large as two kilometers which are as likely to occur on 500 mile lines as for the 6000 mile lines.

Map positions are adequate as basic data for such computations if they have been taken from large scale maps (1:50,000 or greater) of geodetic accuracy. It is difficult to say precisely what effect such points would have on the accuracy of the final results for the length and azimuth of the line. However, assuming the terminal points to be charged with a 25 meter error, the corresponding errors are approximately one second in azimuth and a maximum of fifty meters in distance.

The following derivation has been extracted from the ACIC report, rearranged and expanded to better relate to the actual subroutine. Symbols in capital letters are actual labels of variables as they appear in VECTOR for the most part.

\[
\begin{align*}
\text{PHI}_1 &= \phi_1 & \text{initial latitude} \\
\text{PHI}_2 &= \phi_2 & \text{terminal latitude} \\
\text{LAMDA}_1 &= \lambda_1 & \text{initial longitude} \\
\text{LAMDA}_2 &= \lambda_2 & \text{terminal longitude} \\
\text{DELAND} &= \Delta \lambda = \lambda_2 - \lambda_1
\end{align*}
\]
The report shows $\lambda_1 - \lambda_2$, but the sign convention there is positive west; VECTOR uses positive east.)

\[
\begin{align*}
\text{SINDL} &= \sin(\Delta \lambda) \\
\text{SIN2DL} &= \sin^2(\Delta \lambda) \\
\text{COSDL} &= \cos(\Delta \lambda) \\
\text{TANB1} &= \tan(\beta_1) = (b/a) \cdot \tan(\phi_1) \\
\text{TANB2} &= \tan(\beta_2) = (b/a) \cdot \tan(\phi_2)
\end{align*}
\]

where $a$ is the semi-major ellipsoid axis
$b$ is the semi-minor ellipsoid axis
and $f = (a-b)/a$ is defined as the flattening

(Note that many ellipsoids are defined in terms of $a$ and $1/f$.)

Then $b/a = (a-a+b)/a = a/a - (a-b)/a = 1 - f$.

\[
\begin{align*}
Q &= \tan(\phi_1)/\tan(\phi_2) \\
Q_{INV} &= 1/Q = \tan(\phi_2)/\tan(\phi_1) \\
P &= (b^2/a^2) \cdot \tan(\phi_1) \cdot \tan(\phi_2) \\
&= [(b/a) \cdot \tan(\phi_1)] \cdot [(b/a) \cdot \tan(\phi_2)] \\
&= \tan(\beta_1) \cdot \tan(\beta_2) \\
D_1 &= Q \cdot \cos(\Delta \lambda) \\
D_2 &= Q_{INV} - \cos(\Delta \lambda) \\
S &= Q \cdot (D_2^2 + \sin^2(\Delta \lambda)) = (1/Q) \cdot (D_1^2 + \sin^2(\Delta \lambda)) \\
&= (1/Q) \cdot [(Q - \cos(\Delta \lambda))^2 + \sin^2(\Delta \lambda)] \\
&= (1/Q) \cdot (Q^2 - 2Q \cdot \cos(\Delta \lambda) + \cos^2(\Delta \lambda) + \sin^2(\Delta \lambda)) \\
&= (1/Q) \cdot (Q^2 - 2Q \cdot \cos(\Delta \lambda) + 1) \\
&= Q - \cos(\Delta \lambda) + 1/Q - \cos(\Delta \lambda) \\
&= D_1 + D_2 \\
PS &= P \cdot S
\end{align*}
\]
[Hold in floating point register P6 the value

\[ J^* = \frac{(2\cdot D_1 \cdot D_2)}{(P \cdot \cos(\Delta \lambda))} \]

\[ \cot(\Delta \sigma) = \frac{(P \cdot \cos(\Delta \lambda))}{\sqrt{PS+\sin^2(\Delta \lambda)}} \]

\[ \cot 25\sigma = \frac{\cot^2(\Delta \sigma)}{1+\cot^2(\Delta \sigma)} \]

[then \( H^* = 1.5 \cdot (Q-1/Q) \cdot \frac{PS+\sin^2(\Delta \lambda)}{1+\cot^2(\Delta \sigma)} \)]

given \( \frac{1}{n} = (2 + \frac{1}{n_0}) \cdot \frac{(PS+\sin^2(\Delta \lambda))}{PS - 2} \)

\[ n_0 = \frac{(a-b)}{(a+b)} \]

\[ \frac{1}{n_0} = \frac{(a+b)}{(a-b)} \]

\[ = \frac{(a+b - a-b)/(a-b) - 1}{2a/(a-b) - 1} \]

\[ = \frac{2/f - 1}{1} = \text{ELLIP} \]

\[ \frac{1}{n} = (2 + \text{ELLIP}) \cdot \frac{(PS+\sin^2(\Delta \lambda))}{PS - 2} \]

\[ = [2 + \text{ELLIP} \cdot \frac{(PS+\sin^2(\Delta \lambda))}{PS - 2} \cdot \frac{PS}{PS} \]

\[ = [2 + \text{ELLIP} \cdot \frac{(PS+\sin^2(\Delta \lambda))}{PS - 2} \cdot \frac{PS}{PS} \]

\[ n = \frac{PS/[2 \cdot (PS+\sin^2(\Delta \lambda))] + \text{ELLIP} \cdot (PS+\sin^2(\Delta \lambda)) - 2 \cdot PS} {PS/[\text{ELLIP} \cdot (PS+\sin^2(\Delta \lambda))] + 2 \cdot \sin^2(\Delta \lambda)} \]

\[ I = 1 - n + (5/4) \cdot n^2 \]

\[ = [(5/4) \cdot n - 1] \cdot n + 1 \]

\[ \cot 25\omega = \cot(\Delta \omega) = \cot(\Delta \sigma) \cdot [I - 2 \cdot J^* - (3/2) \cdot H^*] \]

\[ = \cot(\Delta \sigma) \cdot [I - (n/S) \cdot (2 \cdot D_1 \cdot D_2)/(P \cdot \cos(\Delta \lambda))] \]

\[ - (n/S) \cdot \frac{(1.5 \cdot (Q-1/Q) \cdot \frac{PS+\sin^2(\Delta \lambda)}{1+\cot^2(\Delta \sigma)})}{I+\cot^2(\Delta \sigma)} \]

\[ = \sqrt{\cot^2(\Delta \sigma) \cdot [I - (n/S) \cdot (J^* + (n/S) \cdot H^*)]} \]

\[ \Delta \omega = \cot^{-1} (\cot 25\omega) \]

\[ \text{DISTANCE (in meters)} = I \cdot a \cdot \Delta \omega \]
In all of the calculations, $\Delta \lambda$ is to be the polar angle $< \pi$ ($180^\circ$). But since $\cos(2\pi - \alpha) = -\cos(\alpha)$ and distance calculations used only $\sin^2(\Delta \lambda)$, where $\sin(2\pi - \alpha) = -\sin(\alpha)$, the direction of $\Delta \lambda$ has made no difference so far. However, azimuth calculations need the proper sign on $\sin(\Delta \lambda)$. Note first that if $\Delta \lambda$ is zero, the heading is to be determined by comparing the magnitude of initial and terminal latitudes. If $\delta_2 \geq \delta_1$, $\text{azm} = 0^\circ$, else $\text{azm} = 180.0^\circ$. If $\Delta \lambda$ is not zero, but $\sin(\Delta \lambda)$ is zero, i.e., $\Delta \lambda = \pi$, $\text{azm} = 0.0^\circ$.

It turns out that no adjustment need be made to the sign of $\sin(\Delta \lambda)$. First consider the line on the surface of the earth that is being measured. Since $\Delta \lambda = \lambda_2 - \lambda_1$ and a positive east convention has been assumed, $\Delta \lambda > \pi$ only when the line being measured crosses the international date line. Here $\Delta \lambda > \pi$ would indicate using the identity $\sin(2\pi - \alpha) = -\sin(\alpha)$, since the polar angle of interest is $2\pi - \Delta \lambda$. However, due to crossing the date line, the sign of this angle is wrong according to a positive east convention. Thus the desired angle is actually $-(2\pi - \Delta \lambda)$ or $\Delta \lambda - 2\pi$, but the $-2\pi$ may be dropped. Therefore, we end up with $\sin(\Delta \lambda)$ again and no further adjustments need be made to calculate the azimuth as:

$$\cot(E_{12}) = \frac{\cos(\beta_1) \cdot \tan(\beta_2) - \tan(\beta_1) \cdot \cos(\Delta \lambda) \cdot \sqrt{1-e^2\cos^2(\delta_1)}}{\sin(\Delta \lambda)}$$

where $E_{12}$ is the elliptic arc forward azimuth (heading) and $e^2$ is the major eccentricity squared

$$E_{\text{SQD}} = e^2 = (a^2 - b^2)/a^2$$

$$\cos(\beta_1) = \sqrt{\cos^2(\beta_1)}$$

$$\cos^2(\beta_1) = 1/\sec^2(\beta_1) = 1/(1+\tan^2(\beta_1))$$

$$1 - e^2\cos^2(\beta_1) = 1 - e^2/[1+\tan^2(\beta_1)]$$

$$= (1+\tan^2(\beta_1)-e^2)/(1+\tan^2(\beta_1))$$

$$\cos(\beta_1) \cdot \sqrt{1-e^2\cos^2(\beta_1)} = \sqrt{[\sec^2(\beta_1)-e^2]/\sec^2(\beta_1)}$$

$$E_{12} = \cot^{-1}\left(\frac{\tan(\beta_2) - \tan(\beta_1) \cdot \cos(\Delta \lambda) \cdot \sqrt{\sec^2(\beta_1)-e^2}}{\sin(\Delta \lambda) \cdot \sec^2(\beta_1)}\right)$$

The arccot function returns an angle between $-\pi$ and $\pi$. If $E_{12} < 0$, add $2\pi$ to give a heading between $0^\circ$ and $360^\circ$. 
When the coordinates are expressed in degrees, minutes and seconds, linkage in a calling program is made by:

```
CALL VECTOR (alatd, alats, alats, alond, alonm, alons, alonw, 
             blatd, blats, blats, blond, blonm, blons, blonw, 
             dstnce, [head, ]i)
```

where:

- `alatd, alats, alats` - latitude of the initial point in degrees, minutes, seconds (4-byte arguments)
- `alond, alonm, alons` - longitude of the initial point in degrees, minutes, seconds (4-byte arguments)
- `alonew` - hemisphere of the initial longitude point; 'W' is west. (1-character argument)
- `blatd, etc.` - latitude, longitude and hemisphere of the terminal point
- `dstnce` - the computed distance between point 'a' and point 'b' (single or double precision real/comp-1 or comp-2 (see i below))
- `head` - the forward azimuth measured clockwise from north. If head is omitted or is initialized to a value of 999.0, the azimuth computation is suppressed. (single or double precision real/comp-1 or comp-2 (see i below))
- `i` - the unit of measure that dstnce and head are to be computed in; i is defined as a four byte argument, but is actually interpreted as two halfwords, i' and i" with compatibility to a fullword integer. If the lower (bytes 3 and 4) halfword, i" < 0, then dstnce is returned as a double precision real (comp-2) value, otherwise as a single precision (comp-1) value. The units are based on the absolute value where:

\[
|i"| = 
\begin{array}{ll}
1 & \text{returns nautical miles}, \\
2 & \text{feet,} \\
3 & \text{statute miles,} \\
4 & \text{kilometers,} \\
\text{else} & \text{meters.}
\end{array}
\]
If the upper (bytes 1 and 2) halfword, $i^* < 0$, then head is returned as double precision real (comp-2), otherwise as a single precision value. The units returned are specified by the absolute value where:

$$\left|i^*\right| = 0 \text{ or } 1 \text{ returns degrees,}$$
$$2 \text{ minutes,}$$
$$3 \text{ seconds,}$$
$$\text{else radians.}$$

If coordinates are expressed as degrees, minutes and seconds and are grouped in a 16-word array of 4-byte arguments arranged as:

array(01) alatd
(02) alatm
(03) alats
(04) alatns
(05) alond
(06) alons
(07) alons
(08) alonew
(09) blatd
(10) blatm
(11) blats
(12) blatns
(13) blond
(14) blonm
(15) blons
(16) blonew

then use the calling sequence:

CALL VECTOR (array,dstnce,[head,]i)

Words 4, 8, 12 and 16 of the array are A$^4$ (Hollerith) or PIC X(4) character data with blank fill.
When the coordinates are expressed in radians or composite arc seconds, the linkage is:

CALL VECTOR (alat,alon,alonew,blat,blon,blonew,
dstnce,[head,]i)

where alonew, blonew, dstnce, head and i are as described above and alat,alon,blat and blon are the latitude and longitude of the initial and terminal points in units of:

1) radians if in floating point
2) arc seconds if in binary integer.

A variant of this call is:

CALL VECTOR (alatalon, blat,blon,
dstnce,[head,]i)

where longitude hemisphere indicators are omitted and the latitude and longitude are signed values with north and east as positive.

Known Limitations
Accuracy has been tested only to 6000 statute miles. Due to the ratios of tangents that are calculated, points that are exactly on the equator (0°) and mathematically "close" to the poles (±90°) will cause an abort due to a divide by zero check. However a latitude close to the equator may be specified as approximately in the range of 10^-10 arc seconds to prevent the divide by zero condition.

Remarks
The arguments listed as "4-byte arguments" may be either single precision real/comp-1 or signed binary full-word integer/comp. There is one exception: if the latitude and longitude are being supplied as arc radians, and the distance is being requested in double precision, then the latitude and longitude are also assumed to be double precision values. The results are always returned as floating point values, either single precision/comp-1 or double precision/comp-2 as requested by the signs of i' and i".

The alias RADVEC may be used in place of VECTOR in any of the calls described.
APPENDIX C

VECTOR SOURCE

VECTOR TITLE "*** SUBROUTINE(S) VECTOR/RADVEC ***"
• AUTHOR: MAJ. S. V. PETERSEN, HQ SAC/ADINSD; EXT. 3952
• DATE WRITTEN: 1 NOV 76

• REFERENCE: ACIC TECHNICAL REPORT NUMBER 80,
  "GEODETIC DISTANCES AND AZIMUTH COMPUTATIONS
  FOR LINES OVER 500 MILES"

• DISTANCES ARE CALCULATED AS A GREAT ELLIPTIC, USING THE
  SCHREITER-THOMAS METHOD AS DESCRIBED IN APPENDIX I OF THE
  REPORT. SOME OF THE COMPUTATIONS HAVE BEEN MANIPULATED
  INTO A DIFFERENT FORM TO FACILITATE PROCESSING.
  SOME ERRORS ALSO APPEAR IN THE WRITE-UP, WHICH HOPEFULLY
  HAVE BEEN CORRECTED.

• IF THIS ROUTINE IS ASSEMBLED WITH AN ASSEMBLER THAT ALLOWS
  THE "SYSPARM" OPTION, THE SPHEROID USED FOR A BASE OF
  CALCULATION MAY BE CHANGED AT ASSEMBLY TIME. ENTER THE
  NAME OF THE DESIRED SPHEROID AS THE SYSPARM VALUE AS:
  • SYSPARM(AIRY)
  • SYSPARM(A.M.S.)
  • SYSPARM(BESSEL)
  • SYSPARM(CLARK 1866)
  • SYSPARM(CLARK 1880)
  • SYSPARM(INTERNATIONAL)
  • SYSPARM(HAYFORD) SAME AS INTERNATIONAL
  • SYSPARM(KRASSOVSKY)
  • THE DEFAULT SPHEROID IS THE CLARK 1866 DATUM.
-145-

**GELB**

**GELB360**

SET TO 1 FOR USE ON 360

**GELB360**

SETB 0

**GELB**

&AIRY, &GAMS, &BESSEL, &CLK1866, &CLK1880, &HAYFORD

**GELB**

&KRSVSKY

AIF ($GELB360$) .IREC3A NO &SYSPARM ON 360

**AIF**

&SYSPARM = 'AIRY'. .IREC1

**&AIRY**

SETB 1

AGO .IREC99

**&IREC1**

AIF ('&SYSPARM = 'A.M.S.'). .IREC2

**&GAMS**

SETB 1

AGO .IREC99

**&IREC2**

AIF ('&SYSPARM = 'BESSEL'). .IREC3

**&BESSEL**

SETB 1

AGO .IREC99

**&IREC3**

AIF ('&SYSPARM = 'CLARK 1866'). .IREC4

**&IREC3A**

ANOP CLARK1866 IS THE DEFAULT DATUM

**&CLK1866**

SETB 1

AGO .IREC99

**&IREC4**

AIF ('&SYSPARM = 'CLARK 1880'). .IREC5

**&CLK1880**

SETB 1

AGO .IREC99

**&IREC5**

AIF ('&SYSPARM = 'INTERNATIONAL'). .IREC5A

AIF ('&SYSPARM = 'HAYFORD'). .IREC6

**&IREC5A**

ANOP

**&HAYFORD**

SETB 1

AGO .IREC99

**&IREC6**

AIF ('&SYSPARM = 'KRASSOVSKY'). .IREC3A

**&KRSVSKY**

SETB 1

AGO .IREC99

**&IREC99**

PUNCH * ALIAS RADCRC
VECTOR CSECT
 USING *,R15
 B PASTCONS
 DC A11(L'VCTID)
 VCTID DC C'VECTOR/RADVEC'
 AIP (6IBM360) .SKDT
 DC C'.6SYSDATE..SYS_TIME'
 .SKDT ANOP
 RADVEC EQU VECTOR
 ENTRY RADVEC

SAVEAREA DC 9D'0'

UNIT DC D'1852. '
 DC D'0. 3048'
 DC D'1609. 344'
 DC D'1000. '
 NUMITS EQU (*.UNIT)/8

PI DC D'3. 141592653589793238462643'
 TWOPI DC D'6. 283185307179586476925286'
 RADDEG DC D'57. 2957795130823087679816'
 DC D'3437. 746770784939252607890'
 DC D'206264. 8062470963551564734'
 NAUSMS EQU (*.RADDEG)/8

UNZR1 DC XL8*4E00000000000000*
 DL40VPI DC XL8*41145F306DC9C883* 4/PI

F0 EQU 0
 F2 EQU 2
 F4 EQU 4
 F6 EQU 6
 R0 EQU 0
 R1 EQU 1
 R2 EQU 2
 R3 EQU 3
 R4 EQU 4
 R5 EQU 5
 R6 EQU 6
 R7 EQU 7
 R8 EQU 8
 B9 EQU 9
 R10 EQU 10
 R11 EQU 11
 R12 EQU 12
 R13 EQU 13
 R14 EQU 14
 R15 EQU 15
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONST</strong></td>
<td><strong>DC</strong></td>
<td>$D*4.848136811095359936E-6$</td>
</tr>
<tr>
<td></td>
<td><strong>DC</strong></td>
<td>$D<em>60.0^</em>$</td>
</tr>
<tr>
<td></td>
<td><strong>DC</strong></td>
<td>$D<em>60.0^</em>$</td>
</tr>
<tr>
<td><strong>ACTC1</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>BF1E31FF17848965^</em>$</td>
</tr>
<tr>
<td><strong>ACTC2</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>C0ACDB34CD1B35D^</em>$</td>
</tr>
<tr>
<td><strong>ACTC3</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>412B7CE45AP5C165^</em>$</td>
</tr>
<tr>
<td><strong>ACTC4</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>C11A8F923B178C7E^</em>$</td>
</tr>
<tr>
<td><strong>ACTC5</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>412AB4FD5D433FF6^</em>$</td>
</tr>
<tr>
<td><strong>ACTC6</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>C0229AB85FD869^</em>$</td>
</tr>
<tr>
<td><strong>ACTC7</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>41154CE8B70CA99^</em>$</td>
</tr>
<tr>
<td><strong>ONE</strong></td>
<td><strong>DC</strong></td>
<td>$D<em>1.0^</em>$</td>
</tr>
<tr>
<td><strong>ACTC9</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>41BB67AE8584CAB^</em>$  $\sqrt{3}$</td>
</tr>
<tr>
<td><strong>ACTD1</strong></td>
<td><strong>DC</strong></td>
<td>$D<em>0.0^</em>$</td>
</tr>
<tr>
<td><strong>PIOV2</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>411921F5442D18^</em>$  $-0.52359884$</td>
</tr>
<tr>
<td><strong>ACTCE</strong></td>
<td><strong>DC</strong></td>
<td>$XL4^<em>0E00000^</em>$</td>
</tr>
<tr>
<td><strong>ACTCF2</strong></td>
<td><strong>DC</strong></td>
<td>$XL4^<em>F200000^</em>$</td>
</tr>
<tr>
<td><strong>ACTC3A</strong></td>
<td><strong>DC</strong></td>
<td>$XL4^<em>3A100000^</em>$</td>
</tr>
<tr>
<td><strong>ACTC40</strong></td>
<td><strong>DC</strong></td>
<td>$XL4^<em>40449851^</em>$</td>
</tr>
<tr>
<td><strong>SCA</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>3778FCE0E5AD1685^</em>$  $\sin$</td>
</tr>
<tr>
<td><strong>SCB</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>B66C992E84B6AA37^</em>$  $\cos$</td>
</tr>
<tr>
<td><strong>SCC</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>3787E31045017594^</em>$  $\sin$</td>
</tr>
<tr>
<td><strong>SCD</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>3B541E0F684527^</em>$  $\cos$</td>
</tr>
<tr>
<td><strong>SCE</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>B869B47B1E41AEF6^</em>$  $\cos$</td>
</tr>
<tr>
<td><strong>SCF</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>B265A599C5CB632^</em>$  $\cos$</td>
</tr>
<tr>
<td><strong>SCG</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>3E3C3EAD306ABC29^</em>$  $\cos$</td>
</tr>
<tr>
<td><strong>PIOV4</strong></td>
<td><strong>EQU</strong></td>
<td>$SCG$  $\pi/4$</td>
</tr>
<tr>
<td><strong>ZERO</strong></td>
<td><strong>EQU</strong></td>
<td>$ACTD1$  $\cos$</td>
</tr>
<tr>
<td><strong>TCTA</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>C41926DBBB1F469B^</em>$</td>
</tr>
<tr>
<td><strong>TCTB</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>4532644B1E45A133^</em>$</td>
</tr>
<tr>
<td><strong>TCTC</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>5B0F82C871A3B68^</em>$</td>
</tr>
<tr>
<td><strong>TCTD</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>C58AFDD0A41992D4^</em>$</td>
</tr>
<tr>
<td><strong>TCTE</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>44EFA6393159226^</em>$</td>
</tr>
<tr>
<td><strong>TCTF</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>325FD4A87357CA9^</em>$</td>
</tr>
<tr>
<td><strong>TCTG</strong></td>
<td><strong>DC</strong></td>
<td>$XL8^<em>422376F171F7282^</em>$</td>
</tr>
</tbody>
</table>
### Reference Ellipsoid Constants

- **A** = Semi-Major Axis (Meters)
- **P** = Flattening = \((A-B)/A\)
- **ESQD** = Major-Eccentricity Squared
  \( = (A^2 - B^2)/A^2 \)
- **BOVRA** = Semi-Minor/Semi-Major = 1 - P
- **W0** = \((A-B)/(A+B)\)
- **ELLIP** = \(1/W0 = 2*PINV - 1\)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CLARK 1866**

<table>
<thead>
<tr>
<th>A</th>
<th>DC</th>
<th>6378206.4000</th>
<th>294.978698</th>
<th>6356583.8000</th>
<th>0.00339007530393</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESQD</td>
<td>DC</td>
<td>6378206.4000</td>
<td>294.978698</td>
<td>6356583.8000</td>
<td>0.00339007530393</td>
</tr>
<tr>
<td>BOVRA</td>
<td>DC</td>
<td>0.00676865799729</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELLIP</td>
<td>DC</td>
<td>0.588957396</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGO</td>
<td>DC</td>
<td>6378206.4000</td>
<td>294.978698</td>
<td>6356583.8000</td>
<td>0.00339007530393</td>
</tr>
</tbody>
</table>

**INTERNATIONAL (HAYFORD)**

<table>
<thead>
<tr>
<th>A</th>
<th>DC</th>
<th>6378388.0000</th>
<th>297.000000</th>
<th>6356911.9461</th>
<th>0.00336700336700</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESQD</td>
<td>DC</td>
<td>6378388.0000</td>
<td>297.000000</td>
<td>6356911.9461</td>
<td>0.00336700336700</td>
</tr>
<tr>
<td>BOVRA</td>
<td>DC</td>
<td>0.00672267002233</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELLIP</td>
<td>DC</td>
<td>0.5930</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGO</td>
<td>DC</td>
<td>6378388.0000</td>
<td>297.000000</td>
<td>6356911.9461</td>
<td>0.00336700336700</td>
</tr>
</tbody>
</table>

**KRASSOVSKI**

<table>
<thead>
<tr>
<th>A</th>
<th>DC</th>
<th>6378245.0000</th>
<th>298.300000</th>
<th>6356863.0188</th>
<th>0.00335232986926</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESQD</td>
<td>DC</td>
<td>6378245.0000</td>
<td>298.300000</td>
<td>6356863.0188</td>
<td>0.00335232986926</td>
</tr>
<tr>
<td>BOVRA</td>
<td>DC</td>
<td>0.00669342162297</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELLIP</td>
<td>DC</td>
<td>0.5956</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGO</td>
<td>DC</td>
<td>6378245.0000</td>
<td>298.300000</td>
<td>6356863.0188</td>
<td>0.00335232986926</td>
</tr>
</tbody>
</table>

**REC1** (NOT &CLK1866)

**REC2**

**REC3** (NOT &HAYFORD)

**REC4** (NOT &KRSVKY)

**REC99**
-149-

.REC4  AIP (NOT &CLARK1880) .REC5
* CLARK 1880
* 6378249.1450 293.465000 6356514.8695 .00340756137870
* .00680351128285
A  DC D'6378249.1450'
ESQD  DC D*.00680351128285'
BOVRA  DC D'0.9965924386213'
ELLIP  DC D'585.930'
AGO  .REC99

.REC5  AIP (NOT 5AIRY) .REC6
* AIRY
* 6376542.0000 299.300000 6355237.1487 .00334112930170
* .00667109545840
A  DC D'6376542.00'
ESQD  DC D*.00667109545840'
BOVRA  DC D'0.9966588706983'
ELLIP  DC D'597.60'
AGO  .REC99

.REC6  AIP (NOT 5AMPS) .REC7
* A.M.S.
* 6378270.0000 297.000000 6356794.3434 .00336700336700
* .00672267002233
A  DC D'6378270.00'
ESQD  DC D*.00672267002233'
BOVRA  DC D'0.996632996632996632996632'
ELLIP  DC D'593.0'
AGO  .REC99

.REC7  AIP (NOT 5BESSEL) .RECDP
* BESSEL
* 6377397.1550 299.152813 6356078.9628 .00334277318503
* .00667437223749
A  DC D'6377397.1550'
ESQD  DC D*.00667437223749'
BOVRA  DC D'0.99665722681497'
ELLIP  DC D'597.305625'

.REC99  ANOP
```
<table>
<thead>
<tr>
<th>WKARZA</th>
<th>DC</th>
<th>D°0'</th>
</tr>
</thead>
<tbody>
<tr>
<td>COORDS</td>
<td>DS</td>
<td>0D</td>
</tr>
<tr>
<td>LANDA2</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>PHI2</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>LANDA1</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>PHI1</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>SINDL</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>SIN2DL</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>COSDL</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>TAMB1</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>TAMB2</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>S</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>PS</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>DELAND</td>
<td>EQU</td>
<td>LANDA1</td>
</tr>
<tr>
<td>COT2SG</td>
<td>EQU</td>
<td>LANDA2</td>
</tr>
<tr>
<td>TB2</td>
<td>EQU</td>
<td>COT2SG</td>
</tr>
<tr>
<td>COTDW</td>
<td>EQU</td>
<td>COT2SG</td>
</tr>
<tr>
<td>TANPH1</td>
<td>EQU</td>
<td>LANDA2</td>
</tr>
<tr>
<td>D1</td>
<td>EQU</td>
<td>LANDA2</td>
</tr>
<tr>
<td>SWITCH</td>
<td>EQU</td>
<td>S</td>
</tr>
<tr>
<td>I</td>
<td>EQU</td>
<td>PS</td>
</tr>
<tr>
<td>IJR</td>
<td>EQU</td>
<td>S</td>
</tr>
<tr>
<td>TEMP2</td>
<td>DC</td>
<td>D°0'</td>
</tr>
<tr>
<td>PCSOSDL</td>
<td>DC</td>
<td>F°0'</td>
</tr>
<tr>
<td>SCQ</td>
<td>EQU</td>
<td>PCSOSDL*3</td>
</tr>
<tr>
<td>MINM</td>
<td>DC</td>
<td>XL4°354000000*</td>
</tr>
<tr>
<td>C24M8</td>
<td>DC</td>
<td>F°24, -8°</td>
</tr>
</tbody>
</table>
```
PASTCONS STM  R14, R12, 12 (R13)
LB    R2, R13
LA    R13, SAVEAREA
DROP R15
USING SAVEAREA, R13
ST    R2, 4 (R13)
ST    R13, 8 (R2)
MVI   SWITCH, 0
LM    R4, R5, C24MB
LA    R6, STORAD
LB    R2, R1
LA    R14, 4
LA    R15, (17-1) * 4 - 8 (R1)
CTPPRMS TM  8 (R2), X * 80
BO    EOLFST
B    R2, R14, CTPPRMS
B    WRNGNBR
EOLFST LM  R10, R12, 0 (R2)
SR    R2, R1
SBL   R2, 2
IC    R14, BTBL (R2)
B    WRNGNBR (R14)

* #ARGS = 3, 4, 5, 6, 7, 8, 9,
BTBL DC AL1 (NOHEAD, ARG4?, 0, NOHEAD, ARG7?, NOHEAD, ARG9?)
DC AL1 (0, 0, 0, 0, 0, 0, NOHEAD, ARG17?, 0)
* 10 ------ 15, 16, 17
WRNGNBR DC X * B2H, H * 32* THIS INVALID OPCODE TERMINATES
DC CL32*WRONG NUMBER OF ARGUMENTS PASSED*
B    RTN

NOHEAD LR  R10, R11  OPTIONAL AZIMUTH PARAMETER MISSING
NOHEAD EQU NOHEAD - WRNGNBR
LA  R11, = E * 9999.0* SUPPRESS THE CALCULATION
IC  R14, BTBL + 1 (R2)
B    WRNGNBR (R14)
VECTOR (ALATD, ALATH, ALATS, ALNGD, ALNGM, ALNGS, AEW,
  BLATD, BLATH, BLATS, BLNGD, BLNGM, BLNGS, BEW,
  DSTWCE, <HEAD,> IUNIT)

ARG17   LA    R14, DMSRAD
ARG17@  EQU   ARG17-WRNGNBR
DMSRAD  LD    F0, ZERO             INDEX
        LA    R3, 16
CNVRT17 L      R15,0(R1)
        LA    R1,4(R1)
        MVC   WKAREA(4),0(R15)    MOVE IN VALUE
        TM    WKAREA,X*PF*
        BM    CV17R                REAL*4
        BZ    CV17POSI              POSITIVE INTEGER*4
        L     R0, WKAREA            NEGATIVE INTEGER*4
        LPR   R0, R0
        ST    R0, WKAREA
        MVI   WKAREA,X*80*         MAKE NEGATIVE
        CV17POSI OI    WKAREA,X*46* INTEGER. MAKE AN UNNORM REAL
        CV17R AD    F0, WKAREA
        MD    F0, CONST(R3)
        BXH   R3, R5, CNVRT17
        BR    R6                  TO CHECK EAST/WEST AND STORE.
VECTOR (LATR1, LNBGR1, <AEW,> LATR2, LNBGR2, <BEW,> DSTNCE, <HEAD,> IUNIT)

ARG7    LA     R6, STVL
       EQU     ARG7-WRGNBR

ARG7a   LA     R14, RADSEC
       EQU     ARG7a-WRGNBR

ARG9    LA     R15, 0 (R1)
       EQU     ARG9-WRGNBR

RADSEC  LA     R1, 4 (R1)
       TM     0 (R15), X*FF*
       BNM     ARGSEC
       SDR     FO, FO
       LE     FO, 0 (R15)
       TM     2 (R12), X*80*
       BNO     R6
       LD     FO, 0 (R15)
       BR     R6

ARGSEC  LA     R0, 0 (R15)
       LPR     R0, 80
       ST     R0, WKAREA
       MVI     WKAREA, X*46*
       TM     0 (R15), X*80*
       BNO     8
       XI     WKAREA, X*80*
       LD     FO, WKAREA
       MD     FO, CONST
       BR     R6

STORAD  IX     SWITCH, 1
       BMZ     STVL
       L     R15, 0 (R1)
       LA     R1, 4 (R1)
       CLI     0 (R15), C*W*
       BNE     STVL
       LCDR    FO, FO

STVL    STD     FO, COORDS (R4)
       BIX     R4, 25, 0 (R14)
       B     DONECVRT

FO = COORDS (0) = LAMDA2
VECTOR (LTLMARR, DSTNCE, <HEAD,> IUNIT)

ARG4 L R15,0 (R1) ARRAY OF 16 WORDS; SAME ORDER AS ARG17 PARMS, BUT ADD A WORD FOR LAT NORTH/SOUTH

ARG4@ EQU ARG4-WRNGNBR

LA R1,4 (R1)

ABBDM3 LD P0, ZEB0

LA R3, 16

CNVRT4 MVC WKAREA (4), 0 (R15)

LA R15, 4 (R15)

TM WKAREA,X*PP*

BM CV4R

BZ CV4POSI

L R0,WKAREA

LPR R0,R0

ST R0,WKAREA

MVI WKAREA,X*80*

CV4POSI OI WKAREA,X*46*

CV4R AD F0,WKAREA

MD F0,CONST(R3)

BIX R3,R5,CNVRT4

CLI 0 (R15),C*S*

BE WORS

CLI 0 (R15),C*W*

BNE **6

WORS LCDR F0,F0

STD F0,COORDS (R4)

LA R15,4 (R15)

BIX R4,R5,ABBDM3

B DONCNVRT

(F0) = COORDS (0) = LAMDA2
-155-

**DOWECVRT DS 0H**

* LD FO, LAMDA2
SD FO, LAMDA1
STD FO, DELMND
BMZ KALLSIN
STD FO, SINIDL
STD FO, SIN2DL
LD FO, PHI1
CD FO, PHI2
BE STDST
LD FO, ONE
B STCOSDL

**KALLSIN LA R15,4**

BM **6
SR R15, B15
BAL R7, SC1
STD FO, SINIDL
MDR FO, FO
STD FO, SIN2DL
LD FO, DELMND
LA R15,2
BAL R7, SC1

**STCOSDL STD FO, COSDL**

LD FO, PHI1
BAL R7, TANG
TM PHI1, X*80
BNO **6
LCCR FO, FO
STD FO, TANPH1
MD FO, BOVRA
STD FO, TANB1
LD FO, PHI2
BAL R7, TANG
TM PHI2, X*80
BNO **6
LCCR FO, FO
LDR F6, FO
LD F6, TANPH1
DDR F6, F4
DDR F4, FO
MD FO, BOVRA
STD FO, TANB2
MD FO, TANB1
LDR F2, F4
SDR F2, F6
SD F4, COSDL
STD F4, D1
SD F6, COSDL
ADR F4, F6
BZ SZERO

**Polar Angle**

**SIN (θ) = 0**

**IS THIS A ZERO DISTANCE CALL?**

**YES**

**COS (θ) = 1.**

**Sine of Negative Value**

**Sine of Positive Value**

**Cosine of Value**

**Parametric Latitude**

**QINV = 1/Q**

**Q = TAN (PHI1)/TAN (PHI2)**

**P = (FO) = P**

**Q2 = Q - 1/Q**

**D1 = (F4) = D1**

**D2 = (F6) = D2**
STD F4,S
HDR F4,F0
STD F4,PS
LDR F4,F4
BMP SZERO
AD F4,SIN2DL
HDR F0,COSDL
STE F0,PCOSDL
DDR F6,F0
MD F6,D1
ADR F6,F6
MDR F0,F0
DDR F0,F4
STD F0,COT2SG
AD F0,ONE
MDR F2,F2
DDR F2,F0
MD F2,=D*1.5*
MD F4,ELLIP
AD F4,SIN2DL
AD F4,SIN2DL
LD F0,PS
DDR F0,F4
LD F4,=D*1.25*
MDR F4,F0
SD F4,ONE
MDR F4,F0
AD F4,ONE
STD F4,1
DD F0,S
MDR F2,F0
ADR F2,F6
MDR F2,F0
SDR F4,F2
LD F2,COT2SG
BAL H7,SQT
MDR F0,F4
LD F2,ONE
BAL H7,ACT
TM PCOSDL,X*80*
BNO CALCL
SD F0,PI
LPFR F0,F0
CALCL MD F0,1
CALCLE MD F0,A
LH R15,2 (R12)
LPR R15,R15
BZ STDST
C R15,=A (MUNITS)
BH STDST
SLA R15,3
DD F0,UNIT-8 (R15)

S = D1 + D2
PS * SIN**2 (DELMAD)
P + COS (DELMAD)
D2/(P + COS (DELMAD))
D1*
2*
(P + COS (DELMAD)) ** 2
/(PS + SIN**2 (DELMAD))
= COT**2 (DELSIGMA)

(Q - 1/0)**2
/(COT2SG+1)
1.5*

(P0) = N

(N)

-1

* N

+ 1

= I

(P0) = N/S

(F0) = DISTANCE IN METERS

CHECK DISTANCE UNITS
-157-

STDST

TM 2(R12),X*80
BNO STDSTE
STD F0,0(R10) RETURN AS "DISTANCE" VALUE REAL*8
B CHKAZM

STDSTE

DS OH
AIF (SIBM360).V1
LRER F0,F0 ON A 370, WE CAN ROUND NICELY
.V1 STE F0,0(R10) RETURN AS "DISTANCE" VALUE REAL*4

CHKAZM

CLC 0(4,R11),=E*999.0* AZIMUTH DESIRED?
BE RTN

LD F4,SINDL
LPDR F0,F4
BNZ CALCHEAD
LD F6,PHI1 SIN (DELAND) = 0
TM COSDL,X*80
BNO CH0

LCER F6,F6 (POLAR ANGLE IS PI)
CD F6,PHI2 IF COS (DELAND) * PHI1 < PHI2
BNH STD D HEAD = 0.0;
LDPI IF ELSE HEAD = 180
LD F0,PI
B STDHPI

CALCHEAD LD F2,TANB1
MDR F2,F2 SINDL*SEC2B1
AD F2,ONE
MDR F4,F2
STD F4,SINDL
SD F2,ESQD
STD F2,TB2
BAL B7,SQT
LD F4,TANB2
LD F6,TANB1
MD F6,COSDL
SDR F4,F6
MDR F0,F4
STD F0,TB2
LD F2,SINDL
LPER F2,F2
LPER F0,F0
BZ CH1
STE F2,TEMP2
L R14,TEMP2
STE F0,TEMP2
S R14,TEMP2
C R14,ACTCE
BNH CH2

CH1 LD F0,PI0V2
B CHSGN
CH2     TM   TB2,X*80
       BNO  CHACT
       C    R14,ACTCP2
       BL   LDPI

CHACT  BAL  R7,ACT

CHSGN  TM   TB2,X*80
       BNO  *=10
       LCDR F0,F0
       AD   F0,PI
       TM   SINDL,X*80
       BNO  *=10
       LCDR F0,F0
       AD   F0,TDPI

STHDP  LH   R15,0 (R12)
       LPR  R15,15
       BZ   STCNV

*     COULD BE

STCNV  BD   R0,RADDEG (R15)

STHD  TM  0 (R12),X*80
       BNO  STHDE
       STD  F0,0 (R11)
       B   RTN

STHDE  DS   OH
        AIF  (618H360).V2
        LRH  F0,F0
        .V2  STE  F0,0 (R11)

RTN    L    R13,4 (R13)
       RETURN (14,12),T,RC=0

SZERO  LD   F0,ZERO
       TM   COSDL,X*80
       BZ   STDST
       LD   F0,=D*3.1362
       B    CALCLE

CHECK AZIMUTH UNITS
GIVE DEGREES ON 0 OR 1

* COULD BE A 1 IF A NEGATIVE FULL WORD WAS GIVEN AS FLAG
RADIANS ON ALL ELSE

ROUND ON A 370

ELLIPIC CIRCUMFERENCE
SQT1

SQT

LPDR P0, P2
BZR R7
SR R14, R14
IC R14, TB2
LA R14, X^{31}* (R14)
SRDL R14, 1
STC R14, TB2
LE P6, TB2
MVC TB2+1 (3), =X^423A2A^6
AE P6, TB2
ME P6, =X^4B385P07^6
LTR R15, R15
BMM SQT1
AER P6, P6
AER P6, P6

SQT1

DER P2, P6
AUR P6, P2
HER P6, P6
LER P2, P0
DER P2, P6
AUR P6, P2
HER P6, P6
LDR P2, P0
DDR P2, P6
AUR P6, P2
HDR P6, P6
DDR P0, P6
SDR P0, P6
HER P0, P0
SU P0, TB2
AU P0, TB2
ADR P0, P6
BR R7

REFINE USING HERON'S METHOD (NEWTON-RAPHSON)
SC1  BAL  R14, OCTANT
     LA  R15, 8
     TM  SCQ, X*03'
     BH  SC5
     SR  R15, R15
SC5  CE  F4, MINH
     BH  SC6
     LD  F0, ZERO
     B  SC7+2 (R15)
SC6  MDR  F0, F0
     LDR  F2, F0
     MD  F0, SCA (R15)
     AD  F0, SCB (R15)
     MDR  F0, F2
     AD  F0, SCC (R15)
     MDR  F0, F2
     AD  F0, SCD (R15)
     MDR  F0, F2
     AD  F0, SCE (R15)
     MDR  F0, F2
     AD  F0, SCP (R15)
     MDR  F0, F2
     AD  F0, SCG (R15)
     B  SC7 (R15)
SC7  MDR  F0, F4
     B  SC8
     NOPR  0
     MDR  F0, F2
     AD  F0, ONE
SC8  TM  SCQ, X*04'
     BZR  R7
     LCDR  F0, F0
     BR  R7

OCTANT  LPDR  F0, F0
        MD  F0, DL40VPI
        CE  F0, ONE
        BL  OCT1
        LDR  F4, F0
        AW  F4, UNZR1
        STD  F4, TEMP2
        AD  F4, UNZR1
        SDR  F0, F4
        AL  R15, TEMP2+4
OCT1  STC  R15, SCQ
        TM  SCQ, X*01'
        BZ  OCT2
        SD  F0, ONE
OCT2  LPDR  F4, F0
        BR  R14
<table>
<thead>
<tr>
<th>TANG</th>
<th>SB</th>
<th>R15, R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAL</td>
<td>R14, OCTANT</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>F2, TCTG</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>F6, ONE</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>F4, MINM</td>
<td></td>
</tr>
<tr>
<td>BL</td>
<td>TCT2</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>P0, P0</td>
<td></td>
</tr>
<tr>
<td>LDR</td>
<td>F6, P0</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>F6, TCTF</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>F6, P0</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>F6, TCTE</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>F2, P0</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>F2, TCTA</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>F2, P0</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>F2, TCTB</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>F2, P0</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>F2, TCTC</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>P0, P6</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>P0, TCTD</td>
<td></td>
</tr>
<tr>
<td>MDR</td>
<td>P0, P4</td>
<td></td>
</tr>
<tr>
<td>TH</td>
<td>SCQ, X°03°</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>TCT3</td>
<td></td>
</tr>
<tr>
<td>DDR</td>
<td>F0, P2</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>TCT4</td>
<td></td>
</tr>
<tr>
<td>DDR</td>
<td>F2, P0</td>
<td></td>
</tr>
<tr>
<td>LDR</td>
<td>P0, P2</td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td>SCQ, X°02°</td>
<td></td>
</tr>
<tr>
<td>BZR</td>
<td>R7</td>
<td></td>
</tr>
<tr>
<td>LCDR</td>
<td>P0, P0</td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>R7</td>
<td></td>
</tr>
</tbody>
</table>

**TANGENT FUNCTION**
ACT CDR F0,F2
BB ACT02
BL ACT01
LD F0,PI0V4  (X) = 1, LOAD PI/4 AND RETURN
BR R7

ACT01 DDR F0,F2
LA R1,16
B ACT03

ACT02 DDR F2,F0
LDR F0,F2
SR R1,R1

ACT03 LA R14,ACTD1
LD F4,ONE
CE F0,ACTC3A
BNH ACT05
CE F0,ACTC40
BNH ACT04
LDR F2,F0
MD F0,ACTC9
SDR F0,F4
AD F2,ACTC9
DDR F0,F2
LA R14,8 (R14)

ACT04 LDR F6,F0
MDR F0,F0
LD F4,ACTC7
ADR F4,F0
LD F2,ACTC6
DDR F2,F4
AD F2,ACTC5
ADR F2,F0
LD F4,ACTC4

ACT05 DDR F4,F2
AD F4,ACTC3
ADR F4,F0
LD F2,ACTC2
DDR F2,F4
AD F2,ACTC1
MDR F0,F2
MDR F0,F6
ADR F0,F6
SD F0,0 (R1,R14)
LPER F0,F0
BR R7

END
APPENDIX D

COPY BOOKS FOR COBOL PROGRAMS USING CARTAM

CARTCB07 - COMMUNICATION BLOCK.

05 DDNAME PIC X(8) VALUE "GEOINDEX".
05 FUNCTION-CODE VALUE 'OPEN'.
   10 FUNCTION-CODE-1 PIC X.
   10 FUNCTION-CODE-2 PIC X.
   10 FUNCTION-CODE-3 PIC X.
   10 FUNCTION-CODE-4 PIC X.
     88 CONTINUE-WALK VALUE 'N'.
     88 DISCARD-SUBTREE VALUE 'T'.
     88 KEEP-ALL-CHILDREN VALUE 'L'.
05 STATUS-CODE PIC XI.
   88 GOOD-CARTAM-OPEN VALUE 'T'.
   88 SUCCESSFUL-CARTAM VALUE 'T'.
   88 MORE-PATH VALUE 'T'.
   88 END-OF-PARENT VALUE 'GE'.
05 NODE-INDICATOR PIC X.
05 USER-DATA-PAD-CHARACTER PIC X VALUE 'T'.
05 MORE-INDICATOR REDEFINES USER-DATA-PAD-CHARACTER PIC X.
   88 NODE VALUE 'N'.
   88 TERMINAL-ELEMENT VALUE 'T'.
   88 TERMINAL-W-SHORT-KEY VALUE 'X'.
05 OPEN-INFO-AREA.
   10 NUMBER-OF-COORDINATES PIC 9(4) COMP SYNC VALUE 2.
   10 MAX-NUMBER-BUFFERS PIC 9(4) COMP SYNC VALUE 32.
05 RECORD-RBA REDEFINES OPEN-INFO-AREA PIC S9(9) COMP SYNC.
05 MAX-USER-AREA-LENGTH PIC 9(4) COMP SYNC VALUE 0.
05 TRUE-USER-DATA-LENGTH PIC 9(4) COMP SYNC VALUE 0.
05 NUMBER-VSAM-READS PIC 9(4) COMP SYNC VALUE 0.
05 NUMBER-VSAM-WRITES PIC 9(4) COMP SYNC VALUE 0.
CARTAN FUNCTION CODES.

01 CARTAN-FUNCTION-CODES.
   03 CARTAN-OPEN PIC XXXX VALUE 'OPEN'.
   03 CARTAN-LOAD PIC XXXX VALUE 'LOAD'.
   03 CARTAN-ISRT PIC XXXX VALUE 'ISRT'.
   03 CARTAN-CHNG PIC XXXX VALUE 'CHNG'.
   03 CARTAN-DLET PIC XXXX VALUE 'DLET'.
   03 CARTAN-CLOSE PIC XXXX VALUE 'CLOSE'.
   03 GR PIC XXXX VALUE 'GR'.
   03 GRL PIC XXXX VALUE 'GR L'.
   03 GM PIC XXXX VALUE 'GM'.
   03 GMP PIC XXXX VALUE 'GMP'.
   03 GNP PIC XXXX VALUE 'GNP'.
   03 GNPT PIC XXXX VALUE 'GNPT'.
   03 GNPL PIC XXXX VALUE 'GNPL'.
   03 SUB-FUNCTIONS.
      05 88-CONTINUE-WALK PIC X VALUE '.
      05 88-DISCARD-SUBTREE PIC X VALUE '.
      05 88-KEEP-ALL-CHILDREN PIC X VALUE '.
      05 FILLER PIC X VALUE '.
   03 GP PIC XXXX VALUE 'GP'.
   03 GPP PIC XXXX VALUE 'GPP'.
   03 GT PIC XXXX VALUE 'GT'.
   03 GTP PIC XXXX VALUE 'GTP'.
   03 GC PIC XXXX VALUE 'GC'.
   03 GCP PIC XXXX VALUE 'GCP'.
   03 GN PIC XXXX VALUE 'GN'.

-164-
APPENDIX E

INDEX LOAD PROGRAM SOURCE

IDENTIFICATION DIVISION.
PROGRAM-ID. NTBNDLIX.
DATE-WRITTEN. NOV 77.
DATE-Compiled.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
SELECT NTB-FILE ASSIGN TO NTBVSAM
ORGANIZATION IS INDEXED
ACCESS IS SEQUENTIAL
RECORD KEY IS V-NTB-KEY
FILE STATUS IS FILE-STATUS.

SELECT NDL-FILE ASSIGN TO NDLVSAM
ORGANIZATION IS INDEXED
ACCESS IS SEQUENTIAL
RECORD KEY IS V-ZBKEY
FILE STATUS IS FILE-STATUS.
DATA DIVISION.

FILE SECTION.

FD NTB-FILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS
RECORD CONTAINS 276 TO 4596 CHARACTERS
DATA RECORD IS VSAM-NTB-RECORD.
COPY VSAMNTB.

66 V-IBLATLNG RENAMES V-IBLAT THRU V-IBLNG-DIR.

FD NDL-FILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS
RECORD CONTAINS 340 TO 1840 CHARACTERS
DATA RECORD IS VSAM-ZB-ZO-RECORD.
COPY JLPVZBZO.

66 V-ZBLATLNG RENAMES V-ZBLAT THRU V-ZBLNGSGN.
WORKING-STORAGE SECTION.

77 EOF-SWITCH PIC 9 VALUE 0.
     88 EOF VALUE 1.

77 RETURN-STATUS PIC X(04) VALUE SPACES.
     88 SUCCESSFUL VALUE '0000'.

77 DISPOSITION PIC X(03) VALUE 'SHR'.

77 FILE-STATUS PIC X(02) VALUE SPACES.

01 COMMUNICATION-BLOCK.
     COPY CARTCB07.

01 USER-DATA-AREA.
     05 KEY-FEEDBACK-AREA.
         10 WDL-KEY.
             15 ISL PIC 9(5).
             15 DGZ PIC X(3).
             15 REV PIC X.
             10 FILLER PIC X(15).
         05 FILLER REDEFINES KEY-FEEDBACK-AREA.
             10 WTB-KEY.
                 15 ISL PIC 9(5).
                 15 CAT PIC 9(5).
                 15 WAK PIC 9(4).
                 15 BEN PIC X(6).
                 15 ELT PIC X.
                 10 FILLER PIC X(3).
         66 WDL-IGZ RENAMES ISL OF WDL-KEY
                        THRU DGZ OF WDL-KEY.
01 COORDINATE-VECTOR.
   05 ND1-LAT PIC S9(9) COMP SYNC.
   05 ND1-LON PIC S9(9) COMP SYNC.
   05 ND1-DELTA PIC S9(9) COMP SYNC.

01 WK-LAT-LNG.
   03 WK-LAT.
      05 WK-LATD PIC 9(02) VALUE 0.
      05 WK-LATN PIC 9(02) VALUE 0.
      05 WK-LATS PIC 9(02) VALUE 0.
      05 WK-LAT-DIR PIC X(01) VALUE SPACE.
   03 WK-LONG.
      05 WK-LONGD PIC 9(03) VALUE 0.
      05 WK-LONGM PIC 9(02) VALUE 0.
      05 WK-LONGS PIC 9(02) VALUE 0.
      05 WK-LONG-DIR PIC X(01) VALUE SPACE.

01 ALLOCATED-DSN.
   03 FILLER PIC X(04) VALUE 'JLP.'.
   03 FILLER PIC X(08) VALUE 'VSAMNDR.'.
   03 FILLER PIC X(05) VALUE 'ZBZO.'.
   03 REV-FOR-DSN PIC X(01) VALUE 'B'.
   03 FILLER PIC X(01) VALUE SPACE.

01 DD-NAME PIC X(08) VALUE 'MDLYSAM'.

01 DUMMY-DD-NAME.
   03 FILLER PIC X(07) VALUE 'DUMMYDD'.
   03 DUMMY-DD-NAME-REV PIC X(01) VALUE 'B'.

01 VALUE-OF-REV-TABLE PIC X(03) VALUE 'BCD'.
01 TABLE-OF-REV-VALUES REDEFINES VALUE-OF-REV-TABLE.
   03 REV-LETTER PIC X OCCURS 3 TIMES INDEXED BY REV-ND1.

01 ACCUMULATORS.
   03 ONE-CON PIC S9(06) COMP SYNC VALUE +1.
   03 TOTAL-ISRTS PIC S9(06) COMP SYNC VALUE +0.
   03 TOTAL-GETS PIC S9(06) COMP SYNC VALUE +0.
   03 TOTAL-PUTS PIC S9(06) COMP SYNC VALUE +0.
-169-

PROCEDURE DIVISION.

000-OPEN-INITIALIZE.
MOVE 24 TO MAX-USER-AREA-LENGTH.
MOVE 'LOAD' TO FUNCTION-CODE.
MOVE 'P' TO MODE-INDICATOR.
OPEN INDEX FILE FOR INTEGER COORDINATES.
CALL 'CARTAM' USING COMMUNICATION-BLOCK.
MOVE 21 TO TRUE-USER-DATA-LENGTH.
MOVE 'ISRT' TO FUNCTION-CODE.

010-OPEN-FILES.
OPEN INPUT NTB-FILE.
PERFORM 100-CONVERT-CALL-NTB THRU 100-EXIT THRU 100-EXIT
UNTIL EOF.
MOVE +9 TO TRUE-USER-DATA-LENGTH.
PERFORM 200-OPEN-CLOSE-NDL-FILES THRU 200-EXIT
VARYING REV-NDX FROM 1 BY 1
UNTIL REV-NDX > 3.

900-LAST-CALL-TO-CARTOR.
DISPLAY 'TOTAL # READS = ' TOTAL-GETS,
' , TOTAL # WRITES = ' TOTAL-PUTS,
' , TOTAL # INSERTS = ' TOTAL-ISRTS,
MOVE 'CLOSE' TO FUNCTION-CODE.
CALL 'CARTAM' USING COMMUNICATION-BLOCK.

GOBACK.

100-CONVERT-CALL-NTB.
READ NTB-FILE
AT END
MOVE 1 TO EOF-SWITCH
CLOSE NTB-FILE
GO TO 100-EXIT.
MOVE V-IBLATLNK TO WK-LAT-LNG.
MOVE V-NTB-KEY TO NTB-KEY.
PERFORM 500-CONVERT-CALL THRU 500-EXIT.
100-EXIT.
EXIT.
200-OPEN-CLOSE-NDL-FILES.
   MOVE REV-LETTER (REV-NDX) TO REV-FOR-DSN,
       DUMMY-DD-NAME-REV.
   CALL *ALLOCD* USING RETURN-STATUS,
       DD-NAME,
       ALLOCATED-DSN,
       DISPOSITION.
   IF SUCCESSFUL
       MOVE 0 TO EOF-SVITCH
       OPEN INPUT NDL-FILE
       PERFORM 300-CONVERT-CALL-NDL THRU 300-EXIT UNTIL EOF
       CALL *DEALLC* USING RETURN-STATUS,
           DD-NAME
       IF SUCCESSFUL
           NEXT SENTENCE
       ELSE
           DISPLAY 'STATUS = <', RETURN-STATUS,
               '>', DDN = ', DD-NAME
           MOVE '0000' TO RETURN-STATUS
       ELSE
           DISPLAY 'STATUS = <', RETURN-STATUS,
               '>', DDN = ', DD-NAME,
               *$, DSN = ', ALLOCATED-DSN
           MOVE '0000' TO RETURN-STATUS.
           CALL *DEALLC* USING RETURN-STATUS,
               DUMMY-DD-NAME.
           IF NOT SUCCESSFUL
               DISPLAY 'STATUS = <', RETURN-STATUS,
               '>', DDN = ', DUMMY-DD-NAME
               MOVE '0000' TO RETURN-STATUS.

200-EXIT.
   EXIT.
300-CONVERT-CALL-NDL.
READ NDL-FILE
AT END
MOVE 1 TO EOF-SWITCH
CLOSE NDL-FILE
GO TO 300-EXIT.
MOVE V-ZBLATNG TO WK-LAT-LNG.
MOVE V-ZBKEY TO NDL-IGZ.
MOVE V-ZBREV TO REV OF NDL-KEY.
PERFORM 500-CONVERT-CALL THRU 500-EXIT.
300-EXIT.
EXIT.

500-CONVERT-CALL.
COMPUTE NDX-LAT = (60 * WK-LATD + WK-LATM) * 60 + WK-LATS.
IF WK-LAT-DIR = 'S'
  COMPUTE NDX-LAT = -NDX-LAT.
COMPUTE NDX-LON = (60 * WK-LONGD + WK-LONGM) * 60 + WK-LONGS.
IF WK-LONG-DIR = 'W'
  COMPUTE NDX-LON = -NDX-LON.
CALL *CARTAM* USING COMMUNICATION-BLOCK,
  USER-DATA-AREA,
  COORDINATE-VECTOR.
ADD NUMBER-VSAM-WRITES TO TOTAL-PUTS.
ADD NUMBER-VSAM-READS TO TOTAL-GETS.
MOVE ZEROES TO NUMBER-VSAM-WRITES,
  NUMBER-VSAM-READS.
IF SUCCESSFUL-CARTAM
  ADD ONE-COUNT TO TOTAL-ISRTS
ELSE
  DISPLAY *STATUS CODE = '<' STATUS-CODE,
    >', KEY = '<',
    KEY-FEEDBACK-AREA '>', '.'.
500-EXIT.
EXIT.
APPENDIX F

VSAM FILE DEFINITION EXAMPLE

//DFDLGEO EXEC PGM=IDCAMS, REGION=256K
//STEPCAT DD DISP=SHR, DSN=A\%AMAST\%
//SYSPRINT DD SYSOUT=A
//VSNTB DD UNIT=3330, VOL=SER=VSAH02, SPACE=(TRK,1)
//SYSIN DD *
DEFINE CLUSTER (-
   NAME(VSAM\%NTB\%GEONDX) -
   FILE(VSNTB) -
   VOLUME(VSAH02) -
   CYLINDERS(15) -
   SHAREOPTIONS(1) -
   CISZ(4096) -
   NONINDEXED -
   RECORDSIZE(4089 4089) -
   SPEED -
   UNIQUE -
   OWNER(ADUNSD) -
   DATA (-
      NAME(VSAM\%NTB\%GEONDX\%DATA)) -
   CATALOG(A\%AMAST\%CAT)
/

APPENDIX G

CIRCLE SEARCH PROGRAM SOURCE

ID DIVISION.
PROGRAM-ID. ONETENE.
DATE-WRITTEN. MAY 77.
DATE-COMPiled.
REMARKS.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
SELECT COORD-FILE ASSIGN TO UT-S-DATAIN.
SELECT PRINT-FILE ASSIGN TO UT-S-PRINTER.

DATA DIVISION.

FILE SECTION.

FD COORD-FILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS.
01 FILLER PIC X(80).

FD PRINT-FILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS.
01 PRINT-REC PIC X(132).
WORKING-STORAGE SECTION.

01 COMMUNICATION-BLOCK.
   COPY CARTCB07.

01 CONTROL-CARD.
   03 CNTRL-RADIUS COMP-1 SYNC VALUE +3.0E+3.
   03 CNTRLRADIUS-SECS COMP-1 SYNC.
   03 CNTRLRADIUS-METERS COMP-1 SYNC.
   03 CNTRL-UNITS PIC XX VALUE 'MT'.
      88 NAUT-MILES VALUE 'NM'.
      88 KILO-METERS VALUE 'KM'.
      88 FEET VALUE 'FT'.
      88 METERS VALUE 'MT'.

COPY CARTFNCS.

01 COORD-WORK-AREA.
   03 FILLER PIC X(8) VALUE SPACES.
   03 ADN-NUMBER PIC X(4) VALUE SPACES.
   03 FILLER PIC X(21) VALUE SPACES.
   03 LAT-IN.
      05 LAT-DEG PIC 99 VALUE ZEROS.
      05 LAT-MIN PIC 99 VALUE ZEROS.
      05 LAT-SEC PIC 99 VALUE ZEROS.
      * 05 LAT-NS PIC X VALUE SPACES.
      * 88 SOUTH VALUE 'S'.
   03 LON-IN.
      05 LON-DEG PIC 999 VALUE ZEROS.
      05 LON-MIN PIC 99 VALUE ZEROS.
      05 LON-SEC PIC 99 VALUE ZEROS.
      05 LON-EW PIC X VALUE SPACES.
      88 WEST VALUE 'W'.
   03 FILLER PIC X(33) VALUE SPACES.

01 KEY-FEEDBACK-AREA.
   05 MDL-KEY.
      10 ISL PIC 9(5).
      10 DGZ PIC X(3).
      10 REV PIC X.
   05 FILLER PIC X(15).
01 RESULT-AREA.
  03 INPUT-TO-OUTPUT.
    05 FILLER PIC X(8) VALUE SPACES.
    05 ADN-OUT PIC X(4) VALUE SPACES.
    05 FILLER PIC X(68) VALUE SPACES.
  03 FILLER PIC X(2) VALUE SPACES.
  03 IGZ-OUT.
    05 REV PIC X.
    05 FILLER PIC X.
    05 ISL PIC ZZZ9.
    05 DGZ PIC XXX.
    03 FILLER PIC X(3) VALUE SPACES.
  03 DIST-OUT PIC ZZZZ9.9 VALUE 0.0.
  03 FILLER PIC X VALUE SPACES.
  03 DIST-UNITS PIC X VALUE SPACES.
  03 FILLER PIC X(26) VALUE SPACES.

01 LIMIT-VECTORS.
  03 LOW-LIMITS.
    05 LOW-LAT PIC S9(8) COMP SYNC.
    05 LOW-LON PIC S9(8) COMP SYNC.
  03 HIGH-LIMITS.
    05 HIGH-LAT PIC S9(8) COMP SYNC.
    05 HIGH-LON PIC S9(8) COMP SYNC.

01 WORK-AREA.
  03 LATR COMP-2 SYNC VALUE ZERO.
  03 LATO PIC S9(8) COMP SYNC VALUE ZERO.
  03 LONO PIC S9(8) COMP SYNC VALUE ZERO.
  03 CARTAN-COORDINATE-VECTOR.
    05 LAT1 PIC S9(8) COMP SYNC VALUE ZERO.
    05 LON1 PIC S9(8) COMP SYNC VALUE ZERO.
  03 DSTNCE1 COMP-1 SYNC VALUE ZERO.
  03 AZIMUTH1 COMP-1 SYNC VALUE 9.99E+02.
  03 DSTNCE2 COMP-1 SYNC VALUE ZERO.
  03 ESTIMATOR COMP-1 SYNC VALUE 4.5E+01.
  03 NDX-Delta PIC S9(9) COMP SYNC.
  03 ANSWER-FACTOR COMP-1 SYNC VALUE ZERO.
  03 IFLAG PIC S9(8) COMP SYNC VALUE +5.
  03 ONEWON PIC S9(8) COMP SYNC VALUE +1.
  03 MAX-H-C-CELLS PIC S9(8) COMP SYNC VALUE +100.
  03 SECRADE COMP-1 SYNC VALUE 48481368E-05.
  03 NUM-ADMS PIC S9(4) COMP VALUE +1000.
  03 HOME-FLAG PIC X VALUE LOW-VALUES.
  88 NONE-IN VALUE HIGH-VALUES.
01 HISTO-GRAM SYNC.
   03 H-G-MIN PIC S9(8) COMP.
   03 H-G-MAX PIC S9(8) COMP.
   03 H-G-CELL-ZERO PIC S9(8) COMP.
   03 H-G-CELLS PIC S9(8) COMP OCCURS 100 TIMES.
   03 H-G-CELL-MAX PIC S9(8) COMP.

LINKAGE SECTION.
01 PARM-FIELD.
   03 PARM-LENGTH PIC 9(4) COMP.
     88 VALID-PARM-PASSED VALUE 7.
   03 PARM-RADIUS PIC 9(5).
   03 PARM-UNITS PIC XX.
   03 PARM-BUFFERS PIC 99.
   03 PARM-NUM-ADNS PIC 999.
PROCEDURE DIVISION USING PARM-FIELD.

0000-DRIVER.
MOVE 24 TO MAX-USER-AREA-LENGTH.
MOVE CARTAN-OPEN TO FUNCTION-CODE.
IF PARM-LENGTH NOT < 9
MOVE PARM-BUFFERS TO MAX-NUMBER-BUFFERS.
CALL 'CARTAM' USING COMMUNICATION-BLOCK.
IF NOT GOOD-CARTAN-OPEN
DISPLAY 'BAD OPEN RETURN CODE'
GOBACK.
OPEN INPUT COORD-FILE
OUTPUT PRINT-FILE.
MOVE ALL LOW-VALUES TO HISTO-GRAM.
MOVE +1000000 TO H-G-MIN.
IF PARM-LENGTH NOT < 7
MOVE PARM-RADIUS TO CNTRL-RADIUS
MOVE PARM-UNITS TO CNTRL-UNITS.
IF PARM-LENGTH NOT < 12
MOVE PARM-NUM-ADNS TO NUM-ADNS.
IF NAUT-MILES
COMPUTE CNTRLCRD-RADIUS-SECS = 60.0 * (CNTRL-RADIUS)
MOVE +1852.0 TO ANSWER-FACTOR
ELSE
IF KILO-METERS
COMPUTE CNTRLCRD-RADIUS-SECS = 60.0 * (CNTRL-RADIUS / 1.852)
MOVE +1000.0 TO ANSWER-FACTOR
ELSE
IF FEET
COMPUTE CNTRLCRD-RADIUS-SECS = 60.0 * (CNTRL-RADIUS / 6080.0)
MOVE +0.3048 TO ANSWER-FACTOR
ELSE
COMPUTE CNTRLCRD-RADIUS-SECS = 60.0 * (CNTRL-RADIUS / 1852.0)
MOVE +1.0 TO ANSWER-FACTOR.
COMPUTE CNTRLCRD-RADIUS-IN-METERS =
CNTRL-RADIUS * ANSWER-FACTOR.
0100-PROCESS-LOOP.
READ COORD-FILE INTO COORD-WORK-AREA
AT END GO TO 0100-FINISH-UP.
MOVE CNTRL-RADIUS-SECS TO HIGH-LON.
MULTIPLY HIGH-LON BY +1.1 GIVING HIGH-LAT.
COMPUTE LAT0 = (LAT-DEG * 60 + LAT-MIN) * 60
  + LAT-SEC.
  * IF SOUTH COMPUTE LAT0 = - LAT0.
COMPUTE LONG = (LOW-DEG * 60 + LOW-MIN) * 60
  + LOW-SEC.
  IF WEST COMPUTE LONG = - LONG.
COMPUTE LATR = LAT0 * SEC-RAD.
CALL "HAPSID" USING LATR, HIGH-LON.
COMPUTE LOW-LAT = LAT0 - HIGH-LAT.
COMPUTE LOW-LONG = LONG - HIGH-LONG.
COMPUTE HIGH-LAT = LAT0 + HIGH-LAT.
COMPUTE HIGH-LONG = LONG + HIGH-LONG.
WRITE PRINT-REC FROM COORD-WORK-AREA
AFTER ADVANCING 3 LINES.
MOVE SPACES TO RESULT-AREA.
MOVE CNTRL-UNITS TO DIST-UNITS.
MOVE ADN-NUMBER TO ADN-OUT.
MOVE HIGH-VALUES TO NONE-FLAG.
MOVE ZERO TO NUMBER-VSAM-READS.
MOVE GR TO FUNCTION-CODE.
CALL "CARTAM" USING COMMUNICATION-BLOCK,
  KEY-FEEDBACK-AREA,
  CARTAM-COORDINATE-VECTOR,
  ND-DELTA,
  LOW-LIMTS,
  HIGH-LIMITS.
PERFORM 0200-WALK-PATH THRU 0200-WALK-PATH-EXIT
UNTIL NOT MORE-PATH.
IF NONE-IN
  MOVE CNTRL-RADIUS TO DIST-OUT
  MOVE 'NONE IN ' TO IGZ-OUT
  WRITE PRINT-REC FROM RESULT-AREA.
IF NUMBER-VSAM-READS > H-G-MAX
  MOVE NUMBER-VSAM-READS TO H-G-MAX.
IF NUMBER-VSAM-READS < H-G-MIN
  MOVE NUMBER-VSAM-READS TO H-G-MIN.
IF NUMBER-VSAM-READS < ONE-CON
  ADD ONE-CON TO H-G-CELL-ZERO
ELSE
  IF NUMBER-VSAM-READS > MAX-H-G-CELLS
    ADD +1 TO H-G-CELL-MAX
  ELSE
    ADD +1 TO H-G-CELLS (NUMBER-VSAM-READS).
SUBTRACT 1 FROM NUM-ADNS.
IF NUM-ADNS > 0
  GO TO 0100-PROCESS-LOOP.
0100-FINISH-UP.
  DISPLAY 'MIN # READS = ', H-G-MIN,
   ': MAX # READS = ', H-G-MAX,
   ': CELL(0) = ', H-G-CELL-ZERO,
   ': CELL(101) = ', H-G-CELL-MAX.
IF H-G-MAX > 100
  MOVE +100 TO H-G-MAX.
PERFORM H-G-DISPLAY VARYING NUMBER-VSAM-READS
   FROM 1 BY 1 UNTIL NUMBER-VSAM-READS > H-G-MAX.
MOVE CARTAM-CLOSE TO FUNCTION-CODE.
CALL CARTAM USING COMMUNICATION-BLOCK.
CLOSE COORD-FILE
   PRINT-FILE.
GOBACK.

H-G-DISPLAY.
  DISPLAY ' CELL(\#, NUMBER-VSAM-READS, \#) = \#',
   H-G-CELLS (NUMBER-VSAM-READS).
0200-WALK-PATH.
MOVE GNP TO FUNCTION-CODE.
MULTIPLY NDX-DELTA BY ESTIMATOR GIVING DSTNCE2.
CALL 'VECTOR' USING LAT1 LON1
LAT0 LONO
DSTNCE1 IFLAG.
SUBTRACT CNTRLCRD-RADIUS-IN-METERS FROM DSTNCE1.
IF DSTNCE2 < DSTNCE1
MOVE 88-DISCARD-SUBTREE TO FUNCTION-CODE-4
ELSE
IF DSTNCE2 NOT > - DSTNCE1
MOVE 88-KEEP-ALL-CHILDREN TO FUNCTION-CODE-4
PERFORM 0300-KEEP-ALL THRU 0300-KEEP-ALL-EXIT
UNTIL NOT MORE-PATH
MOVE 88-CONTINUE-WALK TO FUNCTION-CODE-4.
CALL 'CARTAM' USING COMMUNICATION-BLOCK,
KEY-FEEDBACK-AREA,
CARTAM-CORDINATE-VECTOR,
NDX-DELTA.

0200-WALK-PATH-EXIT.
EXIT.

0300-KEEP-ALL.
IF TRUE-USER-DATA-LENGTH = 9
CALL 'VECTOR' USING LAT0 LONO
LAT1 LON1
DSTNCE1 IFLAG
MOVE CORR NDL-KEY TO IGZ-OUT
DIVIDE DSTNCE1 BY ANSWER-FACTOR
GIVING DIST-OUT
MOVE LOW-VALUES TO NONE-FLAG
WRITE PRINT-REC FROM RESULT-AREA
AFTER ADVANCING 1 LINE.
CALL 'CARTAM' USING COMMUNICATION-BLOCK,
KEY-FEEDBACK-AREA,
CARTAM-CORDINATE-VECTOR,
NDX-DELTA.

0300-KEEP-ALL-EXIT.
EXIT.
APPENDIX H

INCLUSION/EXCLUSION AREA SEARCH PROGRAM SOURCE

ID DIVISION.
PROGRAM-ID. XCLUDOR2.
DATE-WRITTEN. MAY 77.
DATE-COMPILED.
REMARKS.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
   SELECT CNTRLCRD ASSIGN TO UT-S-CONTROL.
   SELECT LAUNCH-POINT-FILE ASSIGN TO UT-S-LAUNCH.
   SELECT SORTED-FILE ASSIGN TO UT-S-SRTNULL.
   SELECT SORTED-OUTPUT-FILE ASSIGN TO UT-S-NTBS.
DATA DIVISION.

FILE SECTION.

SD SORTED-FILE.
01 SELECTED-RECORD.
   03 PRIMARY-KEY PIC X(21).
   03 FILLER PIC X(15).

FD CNTRLCDR
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS.
01 FILLER PIC X(80).

FD LAUNCH-POINT-FILE
LABEL RECORDS ARE STANDARD
RECORD CONTAINS 21 CHARACTERS
BLOCK CONTAINS 0 RECORDS.
01 LP-DATA PIC X(21).
* READ INTO LP-DATA-AREA.

FD SORTED-OUTPUT-FILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 0 RECORDS.
01 OUT-REC-S PIC X(36).

WORKING-STORAGE SECTION.

01 SIXTY PIC S9(8) COMP SYNC VALUE +60.
01 COMMUNICATION-BLOCK COPY CARTCB07.

01 NDX-VECTORS.
   05 NDX-LAT PIC S9(8) COMP SYNC.
   05 NDX-LON PIC S9(8) COMP SYNC.
   05 NDX-Delta PIC S9(8) COMP SYNC.

01 LIMIT-VECTORS.
   05 LOW-LIMITS.
      10 LOW-LAT PIC S9(8) COMP SYNC.
      10 LOW-LON PIC S9(8) COMP SYNC.
   05 HIGH-LIMITS.
      10 HIGH-LAT PIC S9(8) COMP SYNC.
      10 HIGH-LON PIC S9(8) COMP SYNC.
01 CNTRLCDR-TRANSFORM REDEFINES CNTRLCDR-IN PIC X(80).
COPY CARTFNCS.
01 RESULT-AREA.
  03 KEY-OUT.
    05 ISL PIC 9(5).
    05 FILLER PIC X(16).
  03 LAT-OUT.
    05 LAT-DEG PIC 99 VALUE ZEROS.
    05 LAT-MIN PIC 99 VALUE ZEROS.
    05 LAT-SEC PIC 99 VALUE ZEROS.
    05 LAT-NS PIC X VALUE SPACES.
  03 LON-OUT.
    05 LON-DEG PIC 999 VALUE ZEROS.
    05 LON-MIN PIC 99 VALUE ZEROS.
    05 LON-SEC PIC 99 VALUE ZEROS.
    05 LON-EW PIC X VALUE SPACES.

01 WORK-AREA.
  03 LATR COMP-2 SYNC VALUE ZERO.
  03 MAXIMUM-RADIUS-IN-METERS COMP-1 SYNC.
  03 CONTROL-RADIUS-IN-METERS COMP-1 SYNC.
  03 ABS-LAT PIC 9(8) COMP SYNC VALUE ZERO.
  03 DSTNCE1 COMP-1 SYNC VALUE ZERO.
  03 SECRAD COMP-1 SYNC VALUE .48481368E-05.
  03 DSTNCE2 COMP-1 SYNC VALUE ZERO.
  03 ESTIMATOR COMP-1 SYNC VALUE 4.5E+01.
  03 LAT-LNG-WORK-AREA PIC S9(8) COMP SYNC VALUE ZERO.
  03 IFLAG PIC S9(8) COMP SYNC VALUE +5.
  03 TOTAL-NUMBER-READS PIC S9(6) COMP SYNC VALUE ZERO.
  03 MIN-ISLE-NUMBER PIC 9(5) COMP-3 VALUE ZERO.
  03 MAX-ISLE-NUMBER PIC 9(5) COMP-3 VALUE ZERO.
  03 NUMBER-RECORDS PIC 9(5) COMP-3 VALUE ZERO.
  03 NONE-FLAG PIC X VALUE LOW-VALUES.
     88 NONE-IN VALUE HIGH-VALUES.
  03 OUTSIDE-ALL-CIRCLES PIC X VALUE SPACE.
  03 INSIDE-A-CIRCLE PIC X VALUE SPACE.
  03 LP-END-FLAG PIC XXX VALUE SPACES.
     88 END-OF-LPS VALUE "END".
  03 NUMBER-OF-LAUNCH-POINTS USAGE INDEX.

01 LAUNCH-POINT-DATA SYNC.
  03 LP-TABLE OCCURS 100 TIMES INDEXED BY LAUNCH-POINT.
    05 LP-LAT PIC S9(8) SYNC COMP.
    05 LP-LON PIC S9(8) SYNC COMP.
    05 LP-DELTA-LAT PIC S9(8) SYNC COMP.
    05 LP-DELTA-LON PIC S9(8) SYNC COMP.
    05 LP-RADIUS-IN-METERS SYNC COMP-1.
PROCEDURE DIVISION.

0000-DRIVER.
   CALL "TIMEAX" USING INTERVAL.
   MOVE 21 TO MAX-USER-AREA-LENGTH.
   MOVE CARTAM-OPEN TO FUNCTION-CODE.
   CALL "CARTAM" USING COMMUNICATION-BLOCK.
   IF NOT GOOD-CARTAM-OPEN
      DISPLAY "BAD OPEN RETURN CODE"
      GOBACK.
   OPEN INPUT CNTRLCRD.

0000-CNTL-LOOP.
   READ CNTRLCRD INTO CNTRLCRD-IN
   AT END MOVE CARTAM-CLOSE TO FUNCTION-CODE
      CALL "CARTAM" USING COMMUNICATION-BLOCK
      CLOSE CNTRLCRD
      GOBACK.
   TRANSFORM CNTRLCRD-TRANSFORM FROM SPACES TO ZEROES.
   MOVE MIN-ISLE TO MIN-ISLE-NUMBER.
   MOVE MAX-ISLE TO MAX-ISLE-NUMBER.
   MULTIPLY CNTRL-CENTER-LAT-DEG BY 3600 GIVING NDX-LAT.
   IF CNTRL-SOUTH COMPUTE NDX-LAT = - NDX-LAT.
   MULTIPLY CNTRL-DELTA-LAT BY 3600 GIVING NDX-DELTA.
   COMPUTE LOW-LAT = NDX-LAT - NDX-DELTA.
   COMPUTE HIGH-LAT = NDX-LAT + NDX-DELTA.
   MULTIPLY CNTRL-CENTER-LON-DEG BY 3600 GIVING NDX-LON.
   IF CNTRL-WEST COMPUTE NDX-LON = - NDX-LON.
   MULTIPLY CNTRL-DELTA-LON BY 3600 GIVING NDX-DELTA.
   COMPUTE LOW-LON = NDX-LON - NDX-DELTA.
   COMPUTE HIGH-LON = NDX-LON + NDX-DELTA.
   MOVE CNTRL-RADIUS TO LP-RADIUS.
   MOVE ZEROS TO CNTRLCRD-RADIUS-IN-METERS,
      MAXIMUM-RADIUS-IN-METERS,
      NUMBER-RECORDS.
   IF INCLUSION-AREA-SEARCH
      MOVE 88-DISCARD-SUBTREE TO OUTSIDE-ALL-CIRCLES
      MOVE 88-KEEP-ALL-CHILDREN TO INSIDE-A-CIRCLE
   ELSE
      MOVE 88-KEEP-ALL-CHILDREN TO OUTSIDE-ALL-CIRCLES
      MOVE 88-DISCARD-SUBTREE TO INSIDE-A-CIRCLE.
   SET LAUNCH-POINT TO 1.
   PERFORM 0010-CNVRT-COORDS THRU 3010-EXIT.
   MOVE MAXIMUM-RADIUS-IN-METERS
      TO CNTRLCRD-RADIUS-IN-METERS.
   MOVE ZERO TO MAXIMUM-RADIUS-IN-METERS.
   IF LP-LAT (1) = ZERO
      OPEN INPUT LAUNCH-POINT-FILE
PERFORM 0010-READ-LAUNCH-POINTS THRU 0010-EXIT
VARYING LAUNCH-POINT FROM 1 BY 1
UNTIL (LAUNCH-POINT > 100) OR END-OF-LPS
CLOSE LAUNCH-POINT-FILE.
MOVE HIGH-VALUES TO NONE-FLAG.
MOVE GR TO FUNCTION-CODE.
SORT SORTED-FILE ON ASCENDING KEY PRIMARY-KEY
INPUT PROCEDURE CARTAM-RETRIZEVAL
GIVING SORTED-OUTPUT-FILE.

DISPLAY 'FINAL STATUS = ', STATUS-CODE,
'; NUM READS = ', NUMBER-VSAM-READS,
'; # INSTS = ', NUMBER-RECORDS.
GO TO 0000-CNTL-LOOP.

0010-READ-LAUNCH-POINTS.
READ LAUNCH-POINT-FILE
AT END
MOVE 'END' TO LP-END-FLAG
GO TO 0010-EXIT.
TRANSFORM LP-DATA FROM SPACES TO ZEROS.
MOVE LP-DATA TO LP-DATA-AREA.
0010-CNVRT-COORDS.
SET NUMBER-OF-LAUNCH-POINTS TO LAUNCH-POINT.
IF LP-RADIUS = ZERO
    MOVE CTRLCD-RADIUS-IN-METERS TO
    LP-RADIUS-IN-METERS (LAUNCH-POINT)
ELSE
    IF MILES
        COMPUTE LP-RADIUS-IN-METERS (LAUNCH-POINT) =
        LP-RADIUS * 1852.0
    ELSE
        IF KILO-METERS
            COMPUTE LP-RADIUS-IN-METERS (LAUNCH-POINT) =
            LP-RADIUS * 1000.0
        ELSE
            IF FEET
                COMPUTE LP-RADIUS-IN-METERS (LAUNCH-POINT) =
                LP-RADIUS * 0.3048
            ELSE
                MOVE LP-RADIUS
                TO LP-RADIUS-IN-METERS (LAUNCH-POINT).
            IF LP-RADIUS-IN-METERS (LAUNCH-POINT) > MAXIMUM-RADIUS-IN-METERS
                MOVE LP-RADIUS-IN-METERS (LAUNCH-POINT) TO MAXIMUM-RADIUS-IN-METERS.
            COMPUTE LP-LAT (LAUNCH-POINT) = ((LATD * 60 + LATS) * 3600 + LATS).
            IF LP-SOUTH
                COMPUTE LP-LAT (LAUNCH-POINT) = - LP-LAT (LAUNCH-POINT).
            COMPUTE LP-LON (LAUNCH-POINT) = ((LOWD * 60 + LOWR) * 3600 + LOWR).
            IF LP-WEST
                COMPUTE LP-LON (LAUNCH-POINT) = - LP-LON (LAUNCH-POINT).
            COMPUTE LP-DELTA-LAT (LAUNCH-POINT) ROUNDED =
            34 * LP-RADIUS-IN-METERS (LAUNCH-POINT).
            MOVE LP-LAT (LAUNCH-POINT) TO ABS-LAT.
            IF ABS-LAT + LP-DELTA-LAT (LAUNCH-POINT) < 324000
                COMPUTE LATR ROUNDED =
                LP-LAT (LAUNCH-POINT) * SECRAD
                CALL 'HAPSID' USING LATR,
                LP-DELTA-LON (LAUNCH-POINT).
            ELSE
                MOVE 1500000 TO LP-DELTA-LON (LAUNCH-POINT).
    0010-EXIT.
EXIT.
CARTAH-RETRIEVAL SECTION.

WALK-RETRIEVAL-PATH.
CALL 'CARTAH' USING COMMUNICATION-BLOCK,
  KEY-OUT,
  NDX-VECTORS,
  NDX-Delta,
  LOW-LIMITS,
  HIGH-LIMITS.

IF NOT MORE-PATH
  GO TO CARTAH-RETRIEVAL-EXIT
ELSE
  MOVE GNP TO FUNCTION-CODE
  MOVE NDX-LAT TO ABS-LAT
  IF (ABS-LAT + NDX-Delta) NOT > 324000
    INITIALIZE TO OUTSIDE-ALL
    MOVE OUTSIDE-ALL-CIRCLES TO FUNCTION-CODE-4
    MULTIPLY NDX-Delta BY ESTIMATOR GIVING DSTMCE2
    PERFORM 0200-CHECK-LPS THRU 0200-CHECK-LPS-EXIT
    VARYING LAUNCH-POINT FROM 1 BY 1 UNTIL
    (LAUNCH-POINT > NUMBER-OF-LAUNCH-POINTS)
  IF KEEP-ALL-CHILDREN
    PERFORM 0300-KEEP-ALL THRU
    0300-KEEP-ALL-EXIT UNTIL NOT MORE-PATH
  IF STATUS-CODE = 'GH'
    MOVE 88-CONTINUE-WALK TO
    TO FUNCTION-CODE-4
    MOVE SPACES TO STATUS-CODE.

GO TO WALK-RETRIEVAL-PATH.
0200-CHK-LPS.
COMPUTE ABS-LAT = NDI-LAT - LP-LAT (LAUNCH-POINT).
  IF ABS-LAT NOT >
    NDI-DELTA + LP-DELTA-LAT (LAUNCH-POINT)
    COMPUTE ABS-LAT = NDI-LOW - LP-LOW (LAUNCH-POINT)
    IF ABS-LAT NOT >
      NDI-DELTA + LP-DELTA-LOW (LAUNCH-POINT)
      CALL *VECTOR* USING NDI-LAT
      NDI-LOW
      LP-LAT (LAUNCH-POINT)
      LP-LOW (LAUNCH-POINT)
      DSTNCE1 IFLAG
      SUBTRACT LP-RADIUS-IN-METERS (LAUNCH-POINT)
      FROM DSTNCE1
    IF DSTNCE2 NOT > - DSTNCE1
      TOTALLY INSIDE A RANGE CIRCLE
      MOVE INSIDE-A-CIRCLE TO FUNCTION-CODE-4
      SET LAUNCH-POINT
      TO NUMBER-OF-LAUNCH-POINTS
  ELSE
    IF DSTNCE2 > DSTNCE1
      OVERLAPS A RANGE CIRCLE
      MOVE 88-CONTINUE-WALK
      TO FUNCTION-CODE-4
      IF DSTNCE2 > MAXIMUM-RADIUS-IN-METERS
      SET LAUNCH-POINT TO
      NUMBER-OF-LAUNCH-POINTS.
  0200-CHK-LPS-EXIT.
  EXIT.
0300-KEEP-ALL.
IF (NOT MODE) AND (ISL NOT < MIN-ISLE-NUMBER AND NOT > MAX-ISLE-NUMBER)
    MOVE LOW-VALUES TO LOWE-FLAG
    PERFORM 0350-EXPAND-COORDS
    THRU 0350-EXPAND-COORDS-EXIT
    RELEASE SELECTED-RECORD FROM RESULT-AREA
    ADD +1 TO NUMBER-RECORDS.
    CALL 'CANTAH' USING COMMUNICATION-BLOCK,
       KEY-OUT,
       NDX-VECTORS,
       NDX-DELTA.
0300-KEEP-ALL-EXIT.
EXIT.

0350-EXPAND-COORDS.
IF NDX-LAT < 0
   COMPUTE LAT-LNG-WORK-AREA = - NDX-LAT
   MOVE '5' TO LAT-NS OF LAT-OUT
ELSE
   MOVE NDX-LAT TO LAT-LNG-WORK-AREA
   MOVE 'W' TO LAT-NS OF LAT-OUT.
   DIVIDE LAT-LNG-WORK-AREA BY SIXTY
   GIVING LAT-LNG-AREA
   REMAINDER LAT-SEC OF LAT-OUT.
   DIVIDE LAT-LNG-WORK-AREA BY SIXTY
   GIVING LAT-DEG OF LAT-OUT
   REMAINDER LAT-MIN OF LAT-OUT.
IF NDX-LON < 0
   COMPUTE LAT-LNG-WORK-AREA = - NDX-LON
   MOVE 'W' TO LOW-EW OF LOW-OUT
ELSE
   MOVE NDX-LON TO LAT-LNG-WORK-AREA
   MOVE 'W' TO LOW-EW OF LOW-OUT.
   DIVIDE LAT-LNG-WORK-AREA BY SIXTY
   GIVING LAT-LNG-AREA
   REMAINDER LAT-SEC OF LOW-OUT.
   DIVIDE LAT-LNG-WORK-AREA BY SIXTY
   GIVING LOW-DEG OF LOW-OUT
   REMAINDER LOW-MIN OF LOW-OUT.
0350-EXPAND-COORDS-EXIT.
EXIT.

CANTAH-RETRIEVAL-EXIT.
EXIT.
SUBROUTINE HAPSID (ALAT, ISID)
ISID = ABS(1.1*ISID/COS(ALAT))
RETURN
END