EXHIBIT 7

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

BEDROCK COMPUTER TECHNOLOGIES LLC,	§	
	§	
Plaintiff,	§	
	§	**CASE NO. 6:09-cv-269-LED**
v.	§	
	§	**Jury Trial Demanded**
SOFTLAYER TECHNOLOGIES, INC., et al.	§	
	§	
Defendants.	§	

BEDROCK'S OPENING CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Bedrock Computer Technologies LLC ("Bedrock") respectfully submits its Claim Construction Brief regarding U.S. Patent No. 5,893,120 ("the '120 patent" attached as Ex. A.1[1]). This brief first introduces the technology of the patent, and then explains why Bedrock's proposed constructions are proper.

I. TECHNOLOGY OVERVIEW

The '120 patent is directed to "on-the-fly removal of expired data" in an information storage and retrieval system. *See* '120::1:1-5.[2] Though the patent addresses a general problem related to providing constantly-available, high-performance storage/retrieval operations while, at the same time, handing expiring data, *see* '120::2:22-26, the '120 patent's solution and claims are directed to a specific type of information storage and retrieval system—namely, one that (i) uses a hashing technique and (ii) uses external chaining. *See id.*

"Hashing" is the translation, via a mapping function (commonly known as hash or hashing function), of a record key value to a hash table array address. *See* '120::1:34-46; *see also* '120::4:53-62.[3] Hashing is commonly used because it provides a very quick and efficient way to answer the question: Where in the table is the record with a particular key value? One problem, however, is that two different key values can often map to the same hash table address. This is known as a collision. *See id.*

Information storage and retrieval systems that use a hashing technique must handle collisions so that collided records are not lost and are still retrievable using the hash function. "External chaining" is one way to handle such collisions. *See* '120:1:58-59; *see also* '120::5:16-17. As used in the '120 patent's claims, the parties have agreed that "external chaining" is "a

---

[1] References herein to "Ex. A._" are the numbered exhibits accompanying the Declaration of J. Austin Curry.

[2] Citations to "'120::xx:y-z" refer to the '120 patent, col. xx, ll.y-z.

[3] "Typical hashing functions include truncation, folding, transposition, and modulo arithmetic." '120:1:49-50; *see also* '120::5:5-7.

technique for resolving hash collisions *using a linked list(s)*." *See* Dkt. No. 251 at 2 (Joint Claim Construction Statement) (emphasis added).

In a system that uses external chaining, a hash table entry does not contain the record itself; instead, the hash table entry contains a pointer to a linked list of records that have collided at that location. *See* '120::1:59-6; *see also* '120::5:16-20. Thus, when the information system is asked to store a new record, the record is inserted into a linked list that chains off of the hash table entry corresponding to the new record. But while accumulating collided records on linked lists solves the problem of hash collisions, the accumulation of records can, itself, become a problem.

Specifically, the '120 patent teaches that some data records, after a limited period of time, become obsolete, and their presence in the storage system is no longer needed or desired. *See* '120::2:7-10. If these expired records are not removed from the information system, they will seriously degrade the performance of the information system. *See* '120::5:41-44. The '120 patent describes two ways in which expired data can burden the performance of an information storage and retrieval system: (i) "the presence of expired records lengthens search times since they cause the external chains to be longer than they otherwise would be" and (ii) "expired records occupy dynamically allocated memory storage that could be returned to the system memory pool for useful allocation." *See* '120:5:44-49. As stated in the specification of the '120 patent, the objective of the invention, then, is to "provide the speed of access of hashing techniques for large, heavily used information storage systems having expiring data and, at the same time, prevent the performance degradation resulting from the accumulation of many expired records." *See* '120::2:22-26.

Removal of unneeded records in an information system is commonly referred to as garbage collection, which often required that the system be taken off-line while the garbage was being collected. See '120::2:64-67. In contrast, the garbage collection routines disclosed and claimed in the '120 patent execute on-the-fly while other types of access to the linked lists take place. *See* '120::2:54-63. As the patent emphasizes, "[t]his incremental garbage collection

technique has the decided advantage of automatically eliminating unneeded records without requiring that the information storage system be taken off-line for such garbage collection. This is particularly important for information storage systems requiring rapid access and continuous availability to the user population." *See* '120::2:64-3:3.

II. PRINCIPLES OF CLAIM CONSTRUCTION

Bedrock proposes constructions of the '120 patent in accordance with long-established principles of claim construction—giving a claim term its ordinary meaning that one of skill in the art, at the time of the invention and in light of the patent's specification and prosecution history, would have given it, except in two unusual circumstances: (1) where the intrinsic record provides a special definition for the term; or (2) where the patentee disclaims a portion of the term's ordinary meaning. *See, e.g., Phillips v. AWH Corp.*, 415 F.3d 1303, 1316–17 (Fed. Cir. 2005). "[A]lthough the specification often describes very specific embodiments of the invention, [the Federal Circuit has] repeatedly warned against confining the claims to those embodiments." *Phillips*, 415 F.3d at 1323, citing *Nazomi Communications, Inc. v. ARM Holdings, PLC*, 403 F.3d 1364, 1369 (Fed. Cir. 2005). Limitations from the specification should not be read into the claims unless the patentee "acted as his own lexicographer and imbued the claim terms with a particular meaning or disavowed or disclaimed scope of coverage, by using words or expressions of manifest exclusion or restriction." *E-Pass Techs., Inc. v. 3COM Corp.*, 343 F.3d 1364, 1369 (Fed. Cir. 2003) (citations omitted).

The '120 patent's specification is straight-forward, concise, and well-written. Despite a lack of support for their positions, Defendants seek to interject several limitations into the claims beyond the ordinary meaning of the terms as interpreted by one of ordinary skill in the art reading the patent's disclosure in light of the specification.

Because the Court is familiar with the law of claim construction, Bedrock will discuss specific claim construction principles only where applicable to each dispute.

III. LEVEL OF ORDINARY SKILL IN THE ART

Claims are to be construed from the viewpoint of a person of ordinary skill in the art. *Phillips*, 415 F.3d at 1313. The level of ordinary skill in the art is a function of many factors, including "'(1) the educational level of the inventor; (2) type of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field.'" *See Daiichi Sankyo Co. v. Apotex, Inc.*, 501 F.3d 1254, 1256 (Fed. Cir. 2007) (quoting *Envtl. Designs, Ltd. v. Union Oil Co.*, 713 F.2d 693, 696 (Fed. Cir. 1983)).

Considering these factors in the context of the technology of the '120 patent, one of ordinary skill in the art of computer programming would have a Bachelor degree in a Computing Science, where the degree program requires extensive training and practice in computer programming. *See* Decl. of Dr. Mark Jones at ¶¶ 6-7.

IV. DISPUTED CLAIM CONSTRUCTIONS

A. Claims Terms Not Governed by 35 U.S.C. 112 ¶ 6

1. "linked list to store and provide access to records" / "linked list of records" [claims 1, 3, 5, and 7]

Bedrock's Proposed Construction	Defendants' Proposed Construction
a list in which each record contains a pointer to the next record or information indicating that there is no next record	two or more records in which each record contains a pointer to the next record in the list or information indicating that there is no next record

The specification of the '120 patent discusses linked lists:

> Such linked lists are formed by storing the records individually in dynamically allocated storage and maintaining with each record a pointer to the location of the next record in the chain of collided records.

'120::5:20-25. The specification also indicates that, when there is no next record, there will be information indicating as much. *See* '120 at col. 11 (Search Table Procedure containing

psuedocode statement "while p ≠ nil"). In this way, Bedrock's proposed construction is consistent with the intrinsic record and should be adopted by the Court.

Although the Defendants' proposed construction includes most of the language proposed by Bedrock, the Defendants seek to include an extraneous limitation in the definition, namely the requirement that there be "two or more records" in a linked list. But nothing in the common meaning of the term "list" implies a requirement for multiple members. On the contrary, "list" is a common word that simply means "list." *See SuperSpeed Software, Inc. v. Oracle Corp.*, 447 F. Supp. 2d 672, 682-83 (S.D. Tex. 2006).

Presumably, Defendants will emphasize the "to records"/"of records" language to support their plural members "list" construction. However, as the Federal Circuit has recognized, the plural form of a claim term can describe a universe ranging from one to some higher number, rather than requiring more than one item. *See Versa Corp. v. Ag-Bag Int'l Ltd.*, 392 F.3d 1325, 1330 (Fed. Cir. 2004). In *Versa*, the Federal Circuit looked to the "context in which the patentee used the plural" in determining whether the dispute term should be construed to require a plurality. *See id.* Here, the '120 patent specifically contemplates the situation of a linked list containing only a single record:

> The invoking procedure expects the remove procedure, on completion, to have advanced the passed pointer that originally pointed to the now-removed element so that it points to the successor element in that linked list, or NIL *if the removed element was the final element.*

'120::7:34-38 (emphasis added). Accordingly, the full context of "records" in "linked list of records" in the '120 patent dictates that a linked list could have just one record. Also, the "linked list" referenced in the quote above ceased being a linked list when it had just one, final element. The Defendants' proposed construction of "two or more" would thus read out this preferred embodiment, which offends a canon of claim construction. *See Globetrotter Software, Inc. v.*

Elan Computer Group, Inc., 362 F.3d 1367, 1381 (Fed. Cir. 2004) ("A claim interpretation that excludes a preferred embodiment from the scope of the claim is rarely, if ever, correct.") (inner quotation omitted). The Defendants' proposed construction for this term should therefore be rejected.

2. "automatically expiring / expired" [claims 1, 3, 5, and 7]

Bedrock's Proposed Construction	Defendants' Proposed Construction
after a limited period of time or after the occurrence of some event, becoming obsolete and therefore no longer needed or desired in the storage system / obsolete and therefore no longer needed or desired in the storage system	becoming obsolete and no longer needed or desired in the storage system because of some external condition

In each of the independent claims of the patent, the on-the-fly garbage collection takes aim at automatically expiring records. The specification of the '120 patent discusses why automatically expiring records are problematic for information systems:

> Some forms of information are such that individual data items, after a limited period of time, **become obsolete, and their presence in the storage system is no longer needed or desired.** Scheduling activities, for example, involve data that become obsolete once the scheduled event has occurred. An automatically-expiring data item, once it expires, needlessly occupies computer memory storage that could otherwise be put to use storing an unexpired item.

'120::2:7-11 (emphasis added). Bedrock's proposed construction incorporates the specification's explanation of what it means to "expire" (boldfaced above) and what it means to expire "automatically" (underlined above). Bedrock's proposed construction for "automatically expiring" amalgamates these explanations in the following way: "after a limited period of time or after the occurrence of some event, **becoming obsolete and therefore no longer needed or desired** in the storage system." The specification further supports this construction when it states that "records can become obsolete merely by the passage of time or by the occurrence of some event." *See* '120::5:38-41

The Defendants' proposed construction suffers from a variety problems. First, the Defendants' proposed construction improperly engrafts a *test* for expiration that the '120 patent uses in its preferred embodiment, *see* '120::6:5-9, as the definition of expiration itself. Because there was no lexicography, no disavowal, and no disclaimer, this should be rejected. *See E-Pass Techs.*, 343 F.3d at 1369. Second, the Defendants' proposed construction places two requirements on a record for it to be considered "expiring"—becoming obsolete ***and*** no longer needed or desired. The discussion of automatic expiration in the specification of the '120 patent first equates "expired" with "obsolete" and then clarifies what it means to be expired or obsolete from the perspective of an information storage system. *See* '120::2:7-11. The '120 patent does not teach that a record must be obsolete ***and*** no longer needed or desired for it to be considered expired. Third, Defendants have put forth the same construction for "expired" and "automatically expiring." At a minimum, this reads awkwardly when substituted into the claims. For example, "removing at least some of the expired ones of the records" becomes "removing at least some of the *becoming obsolete and no longer needed or desired in the storage system because of some external condition* ones of the records." For these reasons, the Defendants' proposed construction for this term should be rejected.

3. "identifying at least some of the automatically expired ones of the records" [claims 3 and 7]

Bedrock's Proposed Construction	Defendants' Proposed Construction
No construction necessary; however, should the Court construe this term: "identifying at least some of the automatically expired ones of the records when the linked list is accessed for a purpose other than garbage collection, using the same linked list traversal performed for the purpose other than garbage collection"	determining whether a record is expired by comparing some portion of the contents of the record to some external condition[4]

[4] Defendants indicated in their P.R. 4-3 chart that this limitation further applies to claims 1 and 5.

The goal of claim construction process is to explain the meaning of the claims to the jury. If a claim term "has an ordinary meaning that a jury would understand without construction" then no construction is necessary. *See VirnetX Inc. v. Microsoft Corp.*, No. 6:07-CV-80, 2009 U.S. Dist. LEXIS 65667, at **19-20 (E.D. Tex. July 30, 2009) (Davis, J.); *see also Peer Commc'ns Corp. v. Skype Techs. SA*, No. 6:06-CV-370, 2008 U.S. Dist. LEXIS 92683, at **9–10 (E.D. Tex. May 29, 2008) (Love, J.) (declining to construe term that the jury can understand). Here, once the term "automatically expired" is construed, the jury will understand the meaning of the phrase "identifying at least some of the automatically expired ones of the records." To the extent the Court determines that construction of this term is necessary, Bedrock respectfully requests the Court adopt its construction, which merely clarifies that the referenced identification occurs "when the linked list is accessed for a purpose other than garbage collection, using the same linked list traversal performed for the purpose other than garbage collection."

The Defendants' proposed construction attempts to limit the "identifying" to one specific way in which an expired record could be identified, i.e., "by comparing some portion of the contents of the record to some external condition." As support for their proposed construction, the Defendants' rely on a discussion of a preferred embodiment. *See* '120::6:5-20. Here, the patentee, Dr. Richard Nemes, was not acting as his own lexicographer. Nor did Dr. Nemes, in his discussion of the preferred embodiment, disavow or disclaim the ordinary scope of "identifying" to that embodiment. The Defendants' proposed construction for this term, therefore, should be rejected. *See E-Pass Techs.*, 343 F.3d at 1369.

4. "removing at least some of the automatically expired records from the linked list when the linked list is accessed" [claims 3 and 7]

Bedrock's Proposed Construction	Defendants' Proposed Construction
No construction necessary; however, should the Court construe this term: "removing at least some	while traversing the linked list, both adjusting the pointers in the linked list to

of the automatically expired records from the linked list when the linked list is accessed for a purpose other than garbage collection, using the same linked list traversal performed for the purpose other than garbage collection"	bypass the previously identified expired records and de-allocating the memory occupied by those records

For the same reasons stated above, no construction is necessary for this claim term. Once the terms "automatically expired" and "linked list" are construed, the jury will understand the meaning of the phrase "removing at least some of the automatically expired records from the linked list when the linked list is accessed."

The Defendants' proposed construction attempts to confine "removing" to "while traversing the linked list," by "adjusting the pointers in the linked list to bypass the previously identified expired records and de-allocating the memory occupied by those records." Again, as support for their proposed construction, the Defendants' rely on a discussion of a preferred embodiment. *See*, e.g., '120 at Figs. 3 and 4. These limitations from the specification should not be read into the claims because Dr. Nemes was not "act[ing] as his own lexicographer and imbu[ing] the claim terms with a particular meaning or disavow[ing] or disclaim[ing] scope of coverage, by using words or expressions of manifest exclusion or restriction." *See E-Pass Techs.*, 343 F.3d at 1369.

Further, the prosecution history of the '120 patent does not support the Defendants' construction. A "clear and unmistakable disavowal of scope during prosecution may affect the construction of a claim term." *Lucent Techs., Inc. v. Gateway, Inc.*, 525 F.3d 1200, 1211 (Fed. Cir. 2008) (citations omitted). Dr. Nemes made no disavowal that supports the Defendants' proposed construction. Relevantly, Dr. Nemes distinguished the '120 patent from the prior art in part by reiterating that the prior art does not remove records "when the linked list is accessed." *See* Ex. A.2 at BTEX0000367. Plainly, the phrase "when the linked list is accessed" is not the

same as "while traversing the linked list."[5] As such, the Defendants' proposed construction should be rejected.

5. "dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed" [claims 4 and 8]

Bedrock's Proposed Construction	Defendants' Proposed Construction
determining, during program execution, maximum number of expired ones of the records to remove when the linked list is accessed	immediately before the linked list is traversed, determining a single number that serves as an upper limit on the number of records to remove as the linked list is traversed

Bedrock's proposed construction merely clarifies that "dynamically" means "during program execution." "Dynamic" is a term of art that has special meaning in the field of computer science, but it does not differ significantly from a lay understanding of the term, namely, the opposite of static. Bedrock recognizes that extrinsic evidence is "less significant than the intrinsic record in determining the legally operative meaning of claim language" *see Phillips*, 415 F.3d at 1317 (inner quotation removed); still, extrinsic evidence can be useful in understanding what one of ordinary skill in the art would have understood a term to mean at the time of the invention. The *New IEEE Standard Dictionary of Electrical and Electronic Terms* (5th ed. 1990) defines "dynamic" as "[p]ertaining to an event or process that occurs during computer program execution; for example, dynamic analysis, dynamic binding. Contrast with: static." *See* Ex. A.3 (emphasis removed). In sum, one of ordinary skill in the art at the time of the invention would have understood that "dynamically" means "during program execution."

The Defendants' proposed construction attempts to confine "dynamically" to "immediately before the linked list is traversed." As support for their proposed construction, the

[5] Should the Court find disavowal of scope related to this disputed term, Bedrock respectfully requests the Court adopt its alternative construction, which essentially restates any such disavowed scope by clarifying that removal "when the linked list is accessed" is specifically a removal that occurs "when the linked list is accessed for a purpose other than garbage collection, using the same linked list traversal performed for the purpose other than garbage collection."

Defendants' rely on a discussion of a preferred embodiment, *see* '120::6:56-7:15, and extrinsic evidence. But because Dr. Nemes was not acting as his own lexicographer, and further because Dr. Nemes did not disavow or disclaim the ordinary scope of "dynamically," the Defendants' attempt to limit this term to the preferred embodiment should be rejected. *See Phillips*, 415 F.3d at 1323; *see also E-Pass Techs.*, 343 F.3d at 1369. Moreover, the Defendants' extrinsic evidence does not support their proposed construction. The closest extrinsic evidence that the Defendants cite to support their construction is Microsoft Press's *Computer Dictionary* (3rd ed. 1997) (attached as Ex. A.4), which defines "dynamic" as: "[o]ccuring immediately and concurrently. The term is used in describing both hardware and software; in both cases it describes some action or event that occurs when and as needed." *See* Ex. A.4. This definition, however, does not support a construction that requires the "determination" in the disputed term to be "immediately before the linked list is traversed." To the contrary, the two definitions in Microsoft's Computer Dictionary immediately following the definition of "dynamic" provide examples of dynamic actions: "dynamic address translation" happens "when a program is run," and "dynamic allocation" happens "during program execution." *See* Ex. A.4. In this way, Defendants' extrinsic evidence actually supports Bedrock's proposed construction.

Further, the Defendants' proposed construction attempts to limit "dynamically determining maximum number of expired ones of the records" to " . . . determining a single number that serves as an upper limit on the number of records to remove. . . ." Once the Court construes the "dynamically" and "expired" aspects of this disputed claim term, the jury will understand the remainder of the term without construction. As such, no construction is necessary. *See VirnetX*, 2009 U.S. Dist. LEXIS 65667, at **19-20.

As support for their position, the Defendants rely on the prosecution history of the '120 patent, specifically, remarks made in reference to prior-art U.S. Patent No. 5,287,499 (the "'499 patent" attached as Exhibit A.5). As background, Dr. Nemes is also the inventor of, the '499 patent, which is directed to choosing between collision resolution techniques based on the number of records colliding to a particular hash table address. *See* '499::2:50-68. If the number of records colliding is below a certain upper threshold, a storage system practicing the '499 patent uses a collision technique called "linear probing." *See id.* If the number of collisions rises above that threshold, however, the records are collected from the linear probing technique and reorganized using the external chaining technique. *See id.* When the USPTO rejected the claims of the '120 patent because of the '499 patent, Dr. Nemes responded that the '449 patent's "threshold", which is actually a pair of upper and lower thresholds, acts as "two-way signals" in determining whether to use linear probing or external chaining in organizing the records. *See* Ex. A.2 at BTEX0000367. Dr. Nemes further explained that, in the '120 patent, the dynamically determined "maximum number" acts as "a single quantity that serves as an upper limit on the number of records removed." *See id.* In making this distinction, Dr. Nemes neither redefined the term "maximum number" nor provided any clear and unequivocal disavowal of claim scope. Rather, the remarks are merely part of an overall argument that the '499 patent's "threshold" has nothing to do with the "maximum number of expired ones of the records to remove when the linked list is accessed" of the '120 patent. As such, the Defendants' proposed construction should be rejected.[6] *See Lucent*, 525 F.3d at 1211-1212 ("[S]tatements by the applicants must be

[6] Defendants again attempt to limit the "removal" in this step to "as the linked list is traversed." For the same reasons discussed in section IV.A.4 of this brief, that aspect of the Defendants' proposed construction should be rejected.

read in the context of its overall argument distinguishing the claimed method from the method disclosed in [prior art].")

B. Claim Terms Governed by 35 U.S.C. 112 ¶ 6

"In construing a means-plus-function claim, the district court must first determine the claimed function and then identify the corresponding structure in the written description of the patent that performs that function." *See Baran v. Med. Techs., Inc.*, No. 2010 1058, 2010 WL 3178377, at *5 (Fed. Cir. Aug. 12, 2010) (citing *Applied Med. Res. Corp. v. U.S. Surgical Corp.*, 448 F.3d 1324, 1332 (Fed.Cir.2006)). In defining the claimed function, if the preamble of the means-plus-function claim is "not an idle description but a vital function to be performed" by the element, then the claimed function should be construed to include that preamble. *See id.* at *6. In identifying the corresponding structure, the absence of internal circuitry or code within a corresponding structure does not automatically render the claim indefinite; rather, "the specification need only disclose adequate defining structure to render the bounds of the claim understandable to an ordinary artisan." *See Telcordia Techs., Inc. v. Cisco Systems, Inc.*, Nos. 2009-1175, 2009-1184, 2010 WL 2653251, at *10 (Fed. Cir. July 6, 2010).

1. "a record search means utilizing a search key to access the linked list" [claim 1]

Bedrock's Proposed Recited Functiou	**Defendants' Proposed Recited Function**
The recited function is record searching utilizing a search key to access the linked list.	(none disclosed)

2. "a record search means utilizing a search key to access a linked list of records having the same hash address [claim 5]

Bedrock's Proposed Construction	**Defendants' Proposed Recited Function**
The recited function is record searching utilizing a search key to access a linked list of records having the same hash address.	(none disclosed)
Bedrock's Proposed Corresponding Structures (for both terms above)	**Defendants' Proposed Construction (for both terms above)**
The corresponding structure is: (1) Portions of	Indefinite

the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software instructions as illustrated in Boxes 31-36 and Boxes 39-41 of FIG. 3, or as portions of the pseudo-code of Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14), and described in col. 5, line 57-col. 6 line 4 and col. 6 lines 15-20, or the equivalents thereof.	

These claim terms are taken together because Bedrock proposes the same corresponding structures across both disputed terms, and the Defendants propose that both terms are indefinite. Bedrock's identified recited functions mirror, almost verbatim, the language found in the claim for these terms. The structure that performs these functions is the portions of the application software, user access software or operating system software, described at '120::4:30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11 of FIG. 1 as described at '120::3:52-56, where the portion of the software includes executable software instructions (i) as disclosed in flow chart form:

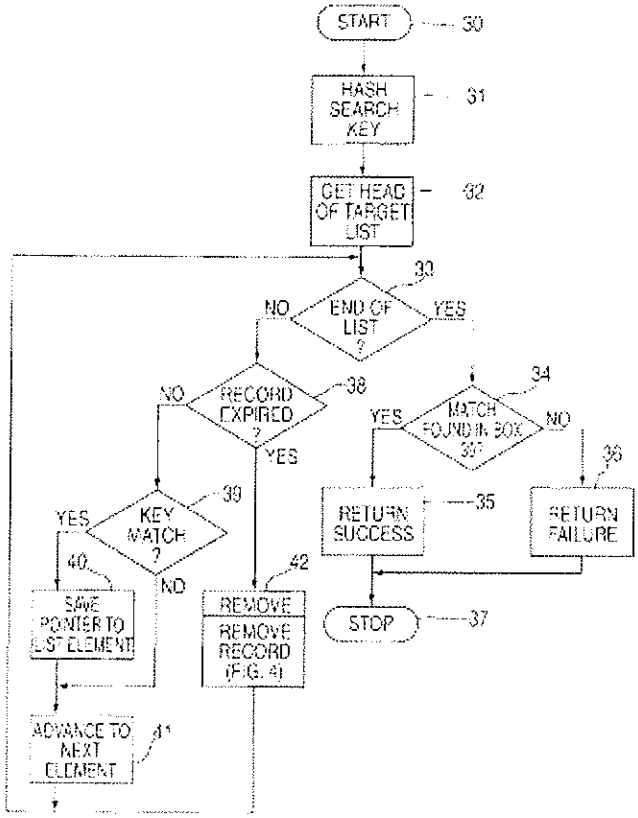


FIG. 3

(ii) as disclosed as pseudocode in Search Table Procedure:

Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size - 1): boolean;
/*  Search table for record_key and delete expired records in target list; if found, position is made to
    point to located record and previous_position to its predecessor, and TRUE is returned; otherwise
    FALSE is returned. index is set to table subscript that is mapped to by hash function in either
    case. */
var p: list_element_pointer;                                        /* Used for traversing chain. */
  previous_p: list_element_pointer;                                 /* Points to p's predecessor. */
begin
  index := hash (record_key);                  /* hash returns value in the range 0 . . . table_size - 1. */
  p := table[index];                                                /* Initialization before loop. */
  previous_p := nil;                                                                        /* Ditto */
  position := nil;                                                                          /* Ditto */
  previous_position := nil;                                                                 /* Ditto */
  while p ≠ nil                          /* HEART OF THE TECHNIQUE: Traverse entire list, deleting */
                                                                    /* expired records as we search. */
    begin
      if p↑.record_contents is expired
        then remove (p, previous_p, index)          /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if position = nil then if p↑.record_contents.key = record_key
                                                                    /* If this is record wanted,*/
            then begin position := p; previous_position := previous_p end;
                                                                    /* save its position. */
          previous_p := p;                                                          /* Advance to */
          p := p↑.next                                                              /* next record. */
        end                                                                         /* else begin */
    end;
      return (position ≠ nil)                  /* Return TRUE if record located, otherwise FALSE. */
end                                                                 /* search_table */
```

(iii) as disclosed as psuedocode in the Alternate Version of Search Table Procedure:

Alternate Version of Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size - 1): boolean;
/*  SAME AS VERSION SHOWN ABOVE EXCEPT THAT THE SEARCH TERMINATES IF
    RECORD IS FOUND, INSTEAD OF ALWAYS TRAVERSING THE ENTIRE CHAIN. */
var p: list_element_pointer;                                        /* Used for traversing chain. */
  previous_p: list_element_pointer;                                 /* Points to p's predecessor. */
begin
  index := hash (record_key);                  /* hash returns value in the range 0 . . . table_size - 1. */
  p := table[index];                                                /* Initialization before loop. */
  previous_p := nil;                                                                        /* Ditto */
  position := nil;                                                                          /* Ditto */
  previous_position := nil;                                                                 /* Ditto */
  while p ≠ nil                          /* HEART OF THE TECHNIQUE: Traverse list, deleting */
                                                                    /* expired records as we search. */
    begin
      if p↑.record_contents is expired
        then remove (p, previous_p, index)          /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if p↑.record_contents.key = record_key                    /* If this is record wanted,*/
            then begin                                              /* save its position. */
              position := p;
              previous_position := previous_p;
              return (true)                          /* We found the record, so terminate search. */
            end;
          previous_p := p;                                                      /* Advance to */
          p := p↑.next                                                          /* next record. */
```

```
        end                                                    /* else begin */
    end;
    return (false)                                             /* Record not found. */
end                                          /* search_table */
```

or the equivalents thereof. *See* 35 U.S.C. 112 ¶ 6. The specification provides a clear link between these terms and the corresponding structures above. *See* '120:: 5:57- 6:4 and 6:15-20.

The Defendant's contention that these terms are indefinite is without merit. The '120 patent is presumed to be valid, and the Defendants bear the burden of proving that an ordinary artisan would not understand the bounds of the claim from the disclosed structure. *See Telcordia,* 2010 WL 2653251, at *10 (citing *Aero Prods. Int'l, Inc. v. Intex Rec. Corp.*, 466 F.3d 1000, 1015 (Fed. Cir. 2006)). This is a burden that Defendants cannot carry here. An ordinary artisan would understand the bounds of the disputed terms in light of the disclosed structures cited above. *See also* Decl. of Dr. Mark Jones at ¶¶ 10-13 and 31-32.

3. "the record search means including a means for identifying and removing at least some [of the] expired ones of the records from the linked list [of records] when the linked list is accessed" [claims 1 and 5]

Bedrock's Proposed Construction	Defendants' Proposed Construction
The recited function is record searching including identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. The corresponding structure is: (1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a	Function:[7] identifying and removing at least some [of the] expired ones of the records from the linked list [of records] when the linked list is accessed. Both identification and removal of an automatically expired record occurs during the same traversal of the linked list.

[7] Before briefing, counsel for Bedrock sought clarification from the Defendants as to whether they intended that *all* sentences and phrases after "Function:" and before "Means:" comprise the recited function. The Defendants responded that all such sentences and phrases "relate to both the scope of the function and the overall claim limitations resulting from the specified structure. The defendants are proposing that the Court adopt the limitations in order to simplify the relevant issues for the jury." *See* Ex. A.6 (email chain between A. Curry and counsel for the Defendants). The Court should not adopt this extraneous commentary in addition to determining the claimed function and then identifying the corresponding structure in the written description of the patent that performs that function. *See Baran*, 2010 WL 3178377, at *5.

computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software as described in Boxes 33-42 of FIG. 3, and/or as pseudo-code in the Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14) including the lines "while ... /*HEART OF THE TECHNIQUE...", and/or as described in col. 5, line 63 - col. 6, line 34, or the equivalents thereof.	For claim 1, the phrase "when the linked list is accessed" refers to the time during which the "utilizing a search key to access the linked list" function in limitation is carried out in claim 1. For claim 5, the phrase "when the linked list is accessed" refers to the time during which the "utilizing a search key to access a linked list of records having the same hash address" function is carried out in claim 5. Removing requires, while traversing the linked list, both adjusting the pointers in the linked list to bypass the previously identified expired records and de-allocating the memory occupied by those records. Means: Boxes 10 and 11 of Fig. 1, Boxes 38 and 42 of Fig. 3, Fig 4 psuedocode in the Search Procedure (cols, 11-14) and Remove Procedure (cols. 13-14) and corresponding portions of the specification. The inclusions of "the records search means," however, renders these limitations indefinite as the "record search means" limitation is indefinite.

The recited function of this claim term is "record searching including identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed." This is a very close restatement of the claim language. In contrast, the Defendants attempt to import additional limitations—some of which are not even recast as language describing functionality—in an improper attempt to confine this term. The structure that performs the recited function is the portions of the application software, user access software or operating system software, described at '120::4:30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11 of FIG. 1 as described at '120::3:52-56,

where the portion of the software includes executable software instructions (i) as disclosed in flow chart form:

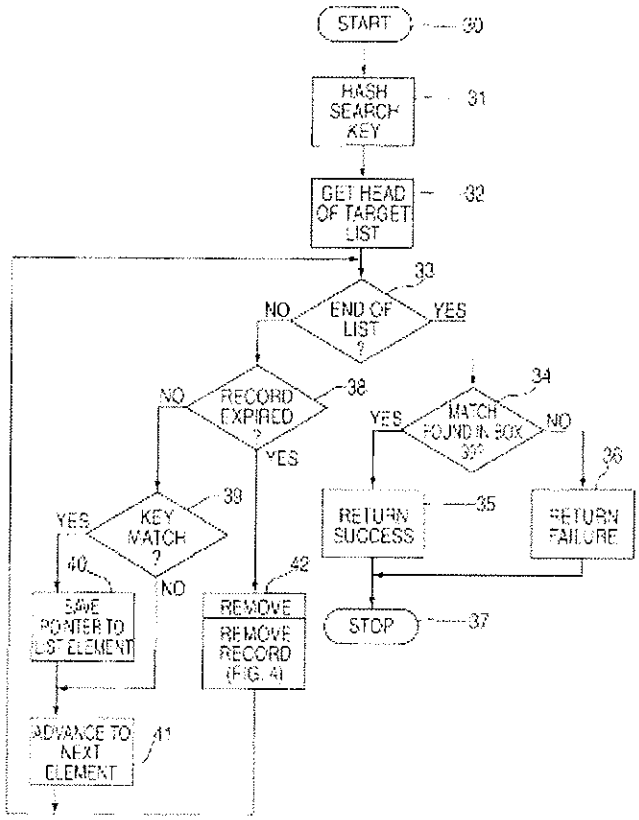


FIG. 3

(ii) as disclosed as pseudocode in Search Table Procedure:

Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size – 1): boolean;
/*  Search table for record_key and delete expired records in target list; if found, position is made to
    point to located record and previous_position to its predecessor, and TRUE is returned; otherwise
    FALSE is returned. index is set to table subscript that is mapped to by hash function in either
    case. */
var p: list_element_pointer;                                        /* Used for traversing chain. */
  previous_p: list_element_pointer;                                 /* Points to p's predecessor. */
begin
  index := hash (record_key);                  /* hash returns value in the range 0 . . . table_size – 1. */
  p := table[index];                                                /* Initialization before loop. */
  previous_p := nil;                                                                    /* Ditto */
  position := nil;                                                                      /* Ditto */
  previous_position := nil;                                                             /* Ditto */
  while p ≠ nil                          /* HEART OF THE TECHNIQUE: Traverse entire list, deleting */
                                                                    /* expired records as we search. */
    begin
      if p↑ .record_contents is expired
        then remove (p, previous_p, index)          /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if position = nil then if p↑ .record_contents.key = record_key
                                                                    /* If this is record wanted,*/
            then begin position := p; previous_position := previous_p end;
                                                                    /* save its position. */
          previous_p := p;                                                  /* Advance to */
          p := p↑ .next                                                     /* next record. */
        end                                                                 /* else begin */
    end;
      return (position ≠ nil)                    /* Return TRUE if record located, otherwise FALSE. */
end                                                                 /* search_table */
```

(iii) as disclosed as psuedocode in the Alternate Version of Search Table Procedure:

Alternate Version of Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size – 1): boolean;
/*  SAME AS VERSION SHOWN ABOVE EXCEPT THAT THE SEARCH TERMINATES IF
    RECORD IS FOUND, INSTEAD OF ALWAYS TRAVERSING THE ENTIRE CHAIN. */
var p: list_element_pointer;                                        /* Used for traversing chain. */
  previous_p: list_element_pointer;                                 /* Points to p's predecessor. */
begin
  index := hash (record_key);                  /* hash returns value in the range 0 . . . table_size – 1. */
  p := table[index];                                                /* Initialization before loop. */
  previous_p := nil;                                                                    /* Ditto */
  position := nil;                                                                      /* Ditto */
  previous_position := nil;                                                             /* Ditto */
  while p ≠ nil                          /* HEART OF THE TECHNIQUE: Traverse list, deleting */
                                                                    /* expired records as we search. */
    begin
      if p↑ .record_contents is expired
        then remove (p, previous_p, index)          /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if p↑ .record_contents.key = record_key                   /* If this is record wanted,*/
            then begin                                              /* save its position. */
              position := p;
              previous_position := previous_p;
              return (true)                          /* We found the record, so terminate search. */
            end;
          previous_p := p;                                                  /* Advance to */
          p := p↑ .next                                                     /* next record. */
```

```
        end                                                    /* else begin */
    end;
    return (false)                                             /* Record not found. */
end                                          /* search_table */
```

or the equivalents thereof. *See* 35 U.S.C. 112 ¶ 6. The specification provides a clear link between these terms and the corresponding structures above. *See* '120::5:63-6:39.

Bedrock's construction, which pairs the recited function with the corresponding structures diselosed in the specifieation, should be adopted, and the Defendants' construction, which ignores disclosed structures for some functions yet includes unnecessary strueture, should be rejected. *See also* Decl. of Dr. Mark Jones at ¶¶ 14-17.

4. "means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" [claim 1]

Bedrock's Proposed Construction	Defendants' Proposed Construction
The recited function is utilizing the record search means, accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The corresponding structure is: (1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software which provides the insert, retrieve, or delete record capability illustrated in the flowchart of FIG. 5, FIG. 6, or FIG. 7, respectively, and/or as pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), or Delete Procedure (cols. 11 and 12), respectively, and/or described in col. 7, line 65 - col. 8, line 32, col. 8, lines 33-44, or col. 8 lines 45-59, or the equivalents thereof.	Function: using the records search means defined above, accessing and during the same traversal of the linked list removing at least some of the expired ones of the records in the linked list. "at the same time" means during the same traversal of the linked list. This limitation requires that the referenced means remove at least one of the expired records in the linked list while utilizing a search key to access the linked list. Removing requires, while traversing the linked list, both adjusting pointers in the linked list to bypass the previously identified expired records and de-allocating the memory occupied by those records. Means: Boxes 10 and 11 of Fig. 1; Figs. 4, 5, 6, and 7, psuedocode in the Search Procedure (cols. 11-14), Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9 and 10), Delete Procedure (cols. 11-12), and Remove Procedure (cols. 13-14), and corresponding

	portions of the specification. Inserting, retrieving, and deleting are all required. The inclusion of "utilizing the records search means," however, renders this limitation indefinite as the "record search means" limitation is indefinite.

5. *"mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records' [claim 5]*

Bedrock's Proposed Construction	Defendants' Proposed Construction
The recited function is utilizing the record search means, inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. The corresponding structure is: (1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software which provides the insert, retrieve, and delete record capability illustrated in the flowchart of FIG. 5, FIG. 6, and FIG. 7, respectively, and/or as pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), and Delete Procedure (cols. 11 and 12), respectively, and/or described in col. 7, line 65 - col. 8, line 32, col. 8, lines 33-44, and col. 8 lines 45-59, or the equivalents thereof.	Function: Using the record search means defined above, inserting, retrieving, and deleting during the same traversal of the linked list removing at least some expired ones of the records in the accessed linked list of records. "at the same time" means during the same traversal of the linked list. This limitation requires that the referenced means remove at least one of the expired ones of the records in the linked list while utilizing a search key to insert, retrieve, and delete records having the same hash address from the system. Removing requires, while traversing the linked list, both adjusting the pointers in the linked list to bypass the previously identified expired records and de-allocating those records from memory. Means: Boxes 10 and 11 of Fig. 1; Figs. 4, 5, 6, and 7, psuedocode in the Search Procedure (cols. 11-14), Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9 and 10), Delete Procedure (cols. 11-12), and Remove Procedure (cols. 13-14), and corresponding portions of the specification. Inserting, retrieving, and deleting are all required. The inclusion of "utilizing the records search means," however, renders this limitation indefinite as the "record search means"

	limitation is indefinite.

These claim terms are taken together because Bedrock proposes the same structures across the two claim terms. The difference between Bedrock's proposed constructions is that, in the first construction above, the identified structures are combined in disjunctive form, i.e., with the conjunction "or," but in the second construction above, the identified structures are combined in the conjunctive form, i.e., with the conjunction "and." Bedrock's proposed recited functions are close restatements of the claim language. In contrast and again, the Defendants attempt to import additional limitations—some of which are not even recast as language describing functionality—in an improper attempt to confine this term.

The structures that performs the recited function in the second term above are the portions of the application software, user access software or operating system software, described at '120::4:30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11 of FIG. 1 as described at '120::3:52-56, where the portion of the software includes executable software instructions which provide the insert, retrieve, and delete record capability illustrated in the flowchart of FIG. 5, FIG. 6, and FIG. 7, respectively, and/or as pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), and Delete Procedure (cols. 11 and 12) or the equivalents thereof. With respect to the first term above, any one of the insert, retrieve, or delete structures can serve as the corresponding structure, and so there the identified structures are joined together as above, but with the conjunction "or." The specification provides a clear link between these terms and the corresponding structures above. *See* '120:::7:65- 8:32, 8:33-44, and 8:45-59. Bedrock's construction, which pairs the recited function with the corresponding structures disclosed in the specification, should be adopted, and the Defendants' construction, which ignores disclosed structures for some functions yet includes

unnecessary structure, should be rejected. *See also* Decl. of Dr. Mark Jones at ¶¶ 18-21 and 28-30.

6. "a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" [claim 5]

Bedrock's Proposed Construction	Defendants' Proposed Construction
The recited function is using hashing to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. The corresponding structure is: (1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software instructions corresponding to pseudo-code "var table: array [0 . . . table_size - 1] of list_element_pointer /* Hash table.*/" which point to records of type "list_element" in cols. 9-10 that allocates in memory an external chaining hash table, and/or as described in col. 5, lines 16-41, or the equivalents thereof.	Indefinite

The recited function is "using hashing to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring." The Defendants have proposed no recited function. The structure that performs this function is the portions of the application software, user access software or operating system software, described at '120::4:30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11 of FIG. 1 as described at '120::3:52-56, where the portion of the software includes executable software instructions as disclosed in the psuedocode:

Definitions

The following formal definitions are required for specifying the insertion, retrieval, and deletion procedures. They are global to all procedures and functions shown below.

```
1. const table_size                                              /* Size of hash table. */
2. type list_element_pointer = ↑ list_element                    /* Pointer to elements of linked list. */
3. type list_element =                                           /* Each element of linked list. */
     record
          record_contents: record_type;
          next: list_element_pointer                             /* Singly-linked list. */
     end
4. var table: array [0 . . . table_size - 1] of list_element_pointer   /* Hash table. */
                                                                 /* Each array entry is pointer to head of list. */
     Initial state of table: table[i] = nil ∀ i 0 ≤ i < table_size      /* Initially empty. */
```

or the equivalents thereof. The specification provides a clear link between this term and the corresponding structures above. *See* '120::1:34-2:5; *see also* '120::4:53-5:33.

The Defendant's contention that this term is indefinite is without merit. An ordinary artisan would understand the bounds of this disputed term in light of the disclosed structures cited above. *See also* Decl. of Dr. Mark Jones at ¶¶ 25-27.

7. "means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records [claims 2 and 6]

Bedrock's Proposed Construction	Defendants' Proposed Construction
The recited function is dynamically determining maximum number of records for the record search means to remove in the accessed linked list of records. The corresponding structure is: (1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56; and (2) Executable software, as described in col. 6, line 56 - col. 7, line 15, that dynamically chooses among removal strategies (e.g., chooses whether to execute Search Table Procedure [cols. 11-12] or Alternate Version of Search Table Procedure [cols. 11-14]) "at the time the record search means is invoked by the caller, thus sometimes removing all expired records, at other times removing some but not all of them, and yet at other times choosing to remove none of them. Such a dynamic decision can be based on factors	Indefinite

such as, for example, how much memory is available in the system storage pool, general system load, time of day, the number of records currently residing in the information system, and other factors both internal and external to the information storage and retrieval system itself" (col. 7, lines 1-10), or the equivalent thereof.	

The recited function of this disputed term is "dynamically determining maximum number of records for the record search means to remove in the accessed linked list of records." This is a close restatement of language in the claims. The Defendants have proposed no recited function. The structure that performs the recited functions is the portions of the application software, user access software or operating system software, described at '120::4:30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11 of FIG. 1 as described at '120::3:52-56, where the portion of the software includes executable software instructions which, at the time the record search means is invoked by the caller, chooses whether to execute the Search Table Procedure or the Alternate Version of Search Table Procedure. *See* '120::6:56-7:15. This choice is based on factors such as how much memory is available in the system storage pool, general system load, time of day, the number of records currently residing in the information system. *See id.*

The Defendant's contention that these terms are indefinite is without merit. To be sure, there is no flowchart or pseudocode in the '120 patent that entirely discloses the corresponding structure for this claim term. There is, however, an algorithm specified in prose form in the specification, which is just as capable a structure. *See Finisar Corp. v. DirecTV Group, Inc.*, 523 F.3d 1323, 1340 (Fed. Cir. 2008) ("This court permits a patentee to express that algorithm in any understandable terms including as a mathematical formula, in prose, or as a flow chart, or in any

other manner that provides sufficient structure.") (inner citation removed). *See also* Decl. of Dr. Mark Jones at ¶¶ 22-24.

C. Miscellaneous Construction Issues

1. Ordering of the claims [claims 3 and 7]

Bedrock proposes that the steps of claims 3 and 7 require no construction. The Defendants propose a rigid ordering of the steps. The Federal Circuit has been clear that "[u]nless the steps of a method actually recite an order, the steps are not ordinarily construed to require one." *See Altiris v. Symantec Corp.*, 318 F.3d 1363, 1369 (Fed. Cir. 2003). The *Altiris* Court went on to hold that the test for determining whether the steps of a method claim that do not recite an order must nonetheless be performed in an order is to look at the logic and grammar of the claim as well as the intrinsic record. *See id.* Here, neither logic nor grammar requires a specific ordering of the steps of claims 3 and 7, with the exception that a first "identifying" step would begin performing before the first "removing" step could be completed. The jury, however, would understand this without construction. Further, the steps of claims 3 and 7 can be performed in a consecutive, repeating, and/or overlapping manner without offending the logic or grammar of the claim. The Defendants' proposed ordering, however, does not permit these possibilities. As such, the Defendants' proposed ordering should be rejected.

2. "identifying . . . and removing . . . when the linked list is accessed" [claims 3 and 7]

The Defendants propose a construction for "identifying . . . and removing . . . when the linked list is accessed" separately from the "identifying" and "removing" terms of claims 3 and 7. This suffers from a variety of problems. First, this is not a claim term; these are two claim terms glued together with ellipses. Second, after claims terms are construed, the claims should be capable of being rewritten with the constructions substituted for the terms. It is not clear, then, how the claims would be rewritten since the Defendants also want the "identifying" and

"removing" claim terms construed separately. Also, the Defendants' proposed construction does not make sense when substituted into the claims. Taking claim 3 as an example:

> accessing the linked list of records,
>
> identifying at least some of the automatically expired ones of the records, and
>
> removing at least some of the automatically expired records from the linked list when the linked list is accessed

would read:

> accessing the linked list of records,
>
> both identification and removal of the automatically expired record(s) occurs during the same traversal of the linked list.

Finally, the requirement that these steps occur "during the same traversal" is not proper for the same reasons given in section IV.A.4 of this brief. As such, the Defendants' proposed construction should be rejected.

V. CONCLUSION

For the foregoing reasons, Bedrock respectfully requests that the Court adopt Bedrock's proposed constructions of the disputed claim terms of the '120 patent, and refuse Defendants' repeated invitations to alter the claim language and import limitations from the preferred embodiments.

DATED: August 27, 2010

Respectfully submitted,
McKOOL SMITH, P.C.

/s/ Douglas A. Cawley

Sam F. Baxter
Texas Bar No. 01938000
McKOOL SMITH, P.C.
sbaxter@mckoolsmith.com
104 E. Houston Street, Suite 300
P.O. Box 0
Marshall, Texas 75670
Telephone: (903) 923-9000
Facsimile: (903) 923-9099

Douglas A. Cawley, Lead Attorney
Texas Bar No. 04035500
dcawley@mckoolsmith.com
Theodore Stevenson, III
Texas Bar No. 19196650
tstevenson@mckoolsmith.com
Jason D. Cassady
Texas Bar No. 24045625
jcassady@mckoolsmith.com
J. Austin Curry
Texas Bar No. 24059636
acurry@mckoolsmith.com
McKOOL SMITH, P.C.
300 Crescent Court, Suite 1500
Dallas, Texas 75201
Telephone: 214-978-4000
Facsimile: 214-978-4044

Robert M. Parker
Texas Bar No. 15498000
Robert Christopher Bunt
Texas Bar No. 00787165
PARKER, BUNT & AINSWORTH, P.C.
100 E. Ferguson, Suite 1114
Tyler, Texas 75702
Telephone: 903-531-3535
Facsimile: 903-533-9687
E-mail: rmparker@pbatyler.com
E-mail: rcbunt@pbatyler.com

ATTORNEYS FOR PLAINTIFF BEDROCK COMPUTER TECHNOLOGIES LLC

CERTIFICATE OF SERVICE

I hereby certify that all counsel of record who are deemed to have consented to electronic service are being served with a copy of the forgoing document via the Court's CM/ECF system pursuant to the Court's Local Rules this 27th day of August, 2010.

/s/ J. Austin Curry
J. Austin Curry

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

BEDROCK COMPUTER TECHNOLOGIES LLC,	§	
	§	
Plaintiff,	§	
	§	**CASE NO. 6:09-cv-269-LED**
v.	§	
	§	**Jury Trial Demanded**
SOFTLAYER TECHNOLOGIES, INC., et al.	§	
	§	
Defendants.	§	

DECLARATION OF J. AUSTIN CURRY IN SUPPORT OF BEDROCK'S OPENING CLAIM CONSTRUCTION BRIEF

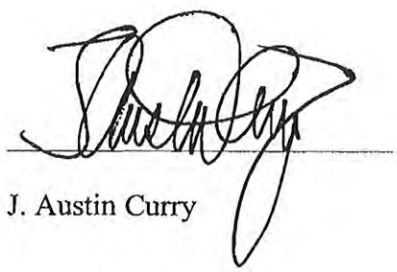
I, J. Austin Curry, declare as follows:

1. I am an attorney with the law firm of McKool Smith P.C., counsel of record for Plaintiff in the above-captioned matter. I submit this declaration based on personal knowledge and following a reasonable investigation. If called upon as a witness, I could and would competently testify to the truth of each statement herein.
2. Attached hereto as Exhibit A.1 is a true and correct copy of the patent in suit, U.S. Patent No. 5,893,120.
3. Attached hereto as Exhibit A.2 is a true and correct copy of selected, Bates-labeled excerpts from the file history of the '120 patent.
4. Attached hereto as Exhibit A.3 is a true and correct copy of selected excerpts from the *New IEEE Standard Dictionary of Electrical and Electronic Terms* (5th ed. 1990).
5. Attached hereto as Exhibit A.4 is a true and correct copy of selected excerpts from Microsoft Press's *Computer Dictionary* (3rd ed. 1997).

6. Attached hereto as Exhibit A.5 is a true and correct copy of U.S. Patent No. 5,287,499.
7. Attached hereto as Exhibit A.6 is a true and correct copy of an email chain between counsel for the Defendants and me.

I DECLARE UNDER PENALTY OF PERJURY UNDER THE LAWS OF THE UNITED STATES OF AMERICA AND THE STATE OF TEXAS THAT THE FOREGOING IS TRUE AND CORRECT.

Dated: August 27, 2010



J. Austin Curry

Exhibit A.1

US005893120A

United States Patent [19]

Nemes

[11] Patent Number: 5,893,120

[45] Date of Patent: Apr. 6, 1999

[54] **METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL USING A HASHING TECHNIQUE WITH EXTERNAL CHAINING AND ON-THE-FLY REMOVAL OF EXPIRED DATA**

[76] Inventor: **Richard Michael Nemes**, 1432 E. 35th St., Brooklyn, N.Y. 11234-2604

[21] Appl. No.: **775,864**

[22] Filed: **Jan. 2, 1997**

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **707/206**; 707/1; 707/100; 707/101; 707/202

[58] **Field of Search** 707/1, 200–206, 707/2, 100–103

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,121,495	6/1992	Nemes	707/3
5,202,981	4/1993	Shackelford	707/1
5,287,499	2/1994	Nemes	707/206

OTHER PUBLICATIONS

D.E. Knuth, *The Art of Computer Programming,* vol. 3, Sorting and Searching, Addison–Wesley, Reading, Massachusetts, 1973, pp. 506–549.

R.L. Kruse, *Data Structures and Program Design,* Second Edition, Prentice–Hall, Englewood Cliffs, New Jersey, 1987, Section 6.5, "Hashing," and Section 6.6, Analysis of Hashing, pp. 198–215.

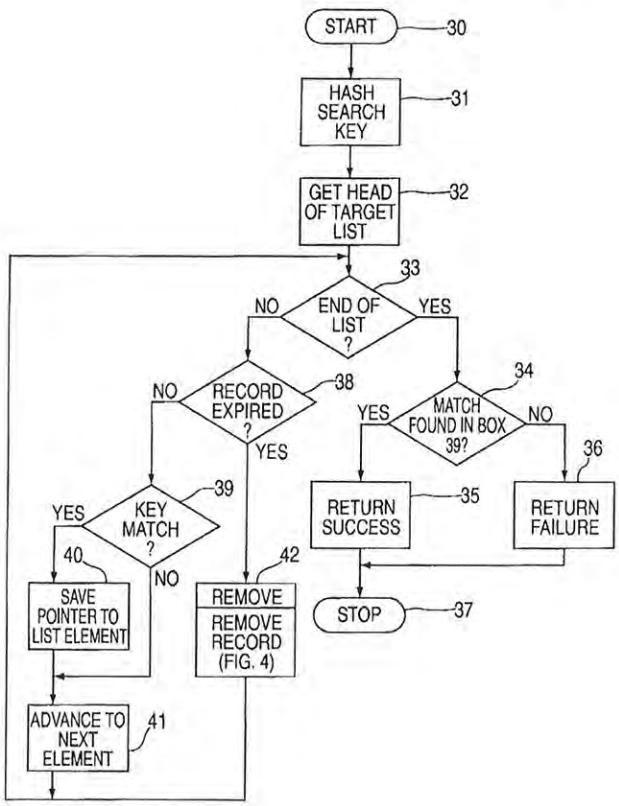
D. F. Stubbs and N.W. Webre, *Data Structure with Abstract Data Types and Pascal,* Brooks/Cole Publishing Company, Monterey, California, 1985, Section 7.4, "Hased Implementations," pp. 310–336.

Primary Examiner—Thomas G. Black
Assistant Examiner—Hosain T. Alam

[57] **ABSTRACT**

A method and apparatus for performing storage and retrieval in an information storage system is disclosed that uses the hashing technique with the external chaining method for collision resolution. In order to prevent performance deterioration due to the presence of automatically expiring data items, a garbage collection technique is used that removes all expired records stored in the system in the external chain targeted by a probe into the data storage system. More particularly, each insertion, retrieval, or deletion of a record is an occasion to search an entire linked-list chain of records for expired items and then remove them. Because an expired data item will not remain in the system long term if the system is frequently probed, it is useful for large information storage systems that are heavily used, require the fast access provided by hashing, and cannot be taken off-line for removal of expired data.

8 Claims, 6 Drawing Sheets



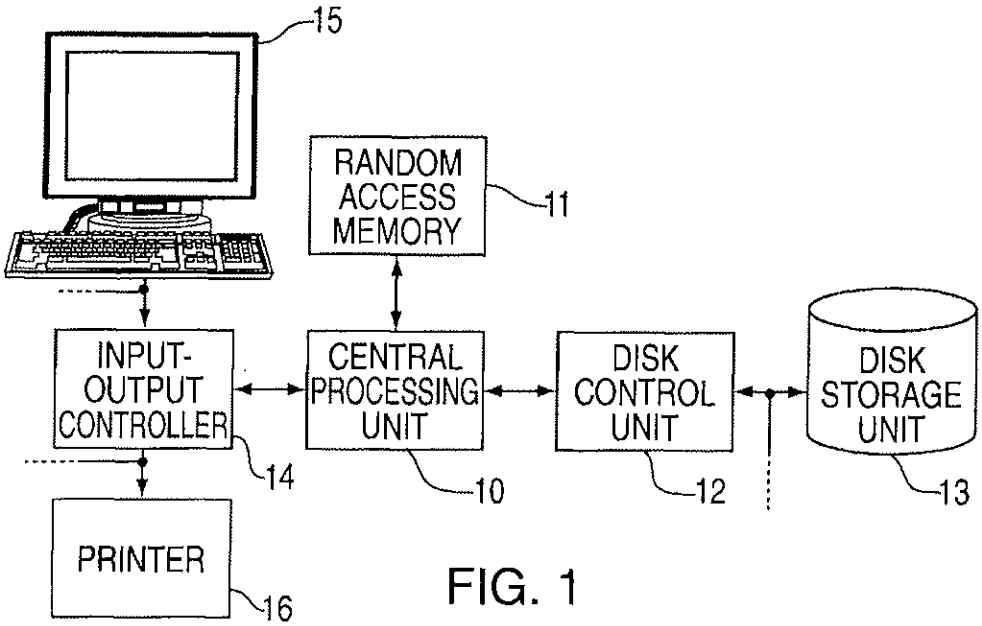

15
RANDOM ACCESS MEMORY
11
INPUT-OUTPUT CONTROLLER
14
CENTRAL PROCESSING UNIT
10
DISK CONTROL UNIT
12
DISK STORAGE UNIT
13
PRINTER
16

FIG. 1

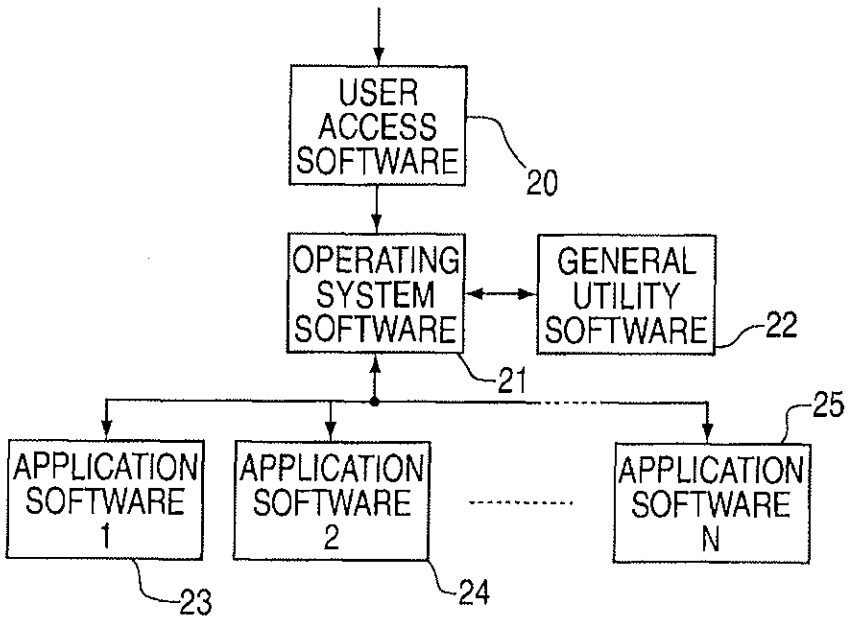

USER ACCESS SOFTWARE
20
OPERATING SYSTEM SOFTWARE
21
GENERAL UTILITY SOFTWARE
22
25
APPLICATION SOFTWARE 1
23
APPLICATION SOFTWARE 2
24
APPLICATION SOFTWARE N

FIG. 2

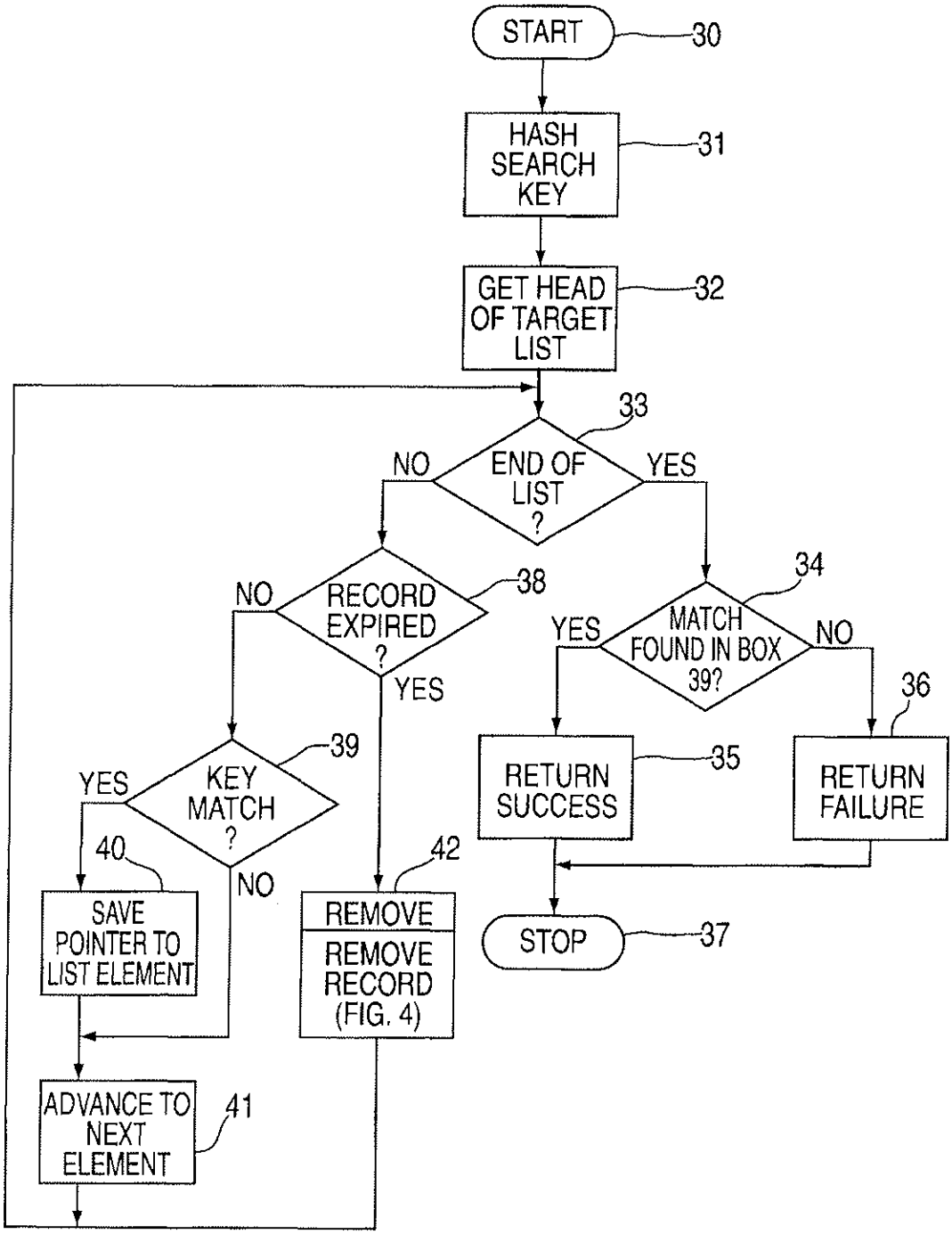

START
30
HASH SEARCH KEY
31
GET HEAD OF TARGET LIST
32
33
END OF LIST ?
NO
YES
RECORD EXPIRED ?
38
NO
YES
MATCH FOUND IN BOX 39?
34
YES
NO
36
KEY MATCH ?
39
YES
NO
RETURN SUCCESS
35
RETURN FAILURE
40
42
SAVE POINTER TO LIST ELEMENT
REMOVE
REMOVE RECORD (FIG. 4)
STOP
37
ADVANCE TO NEXT ELEMENT
41

FIG. 3

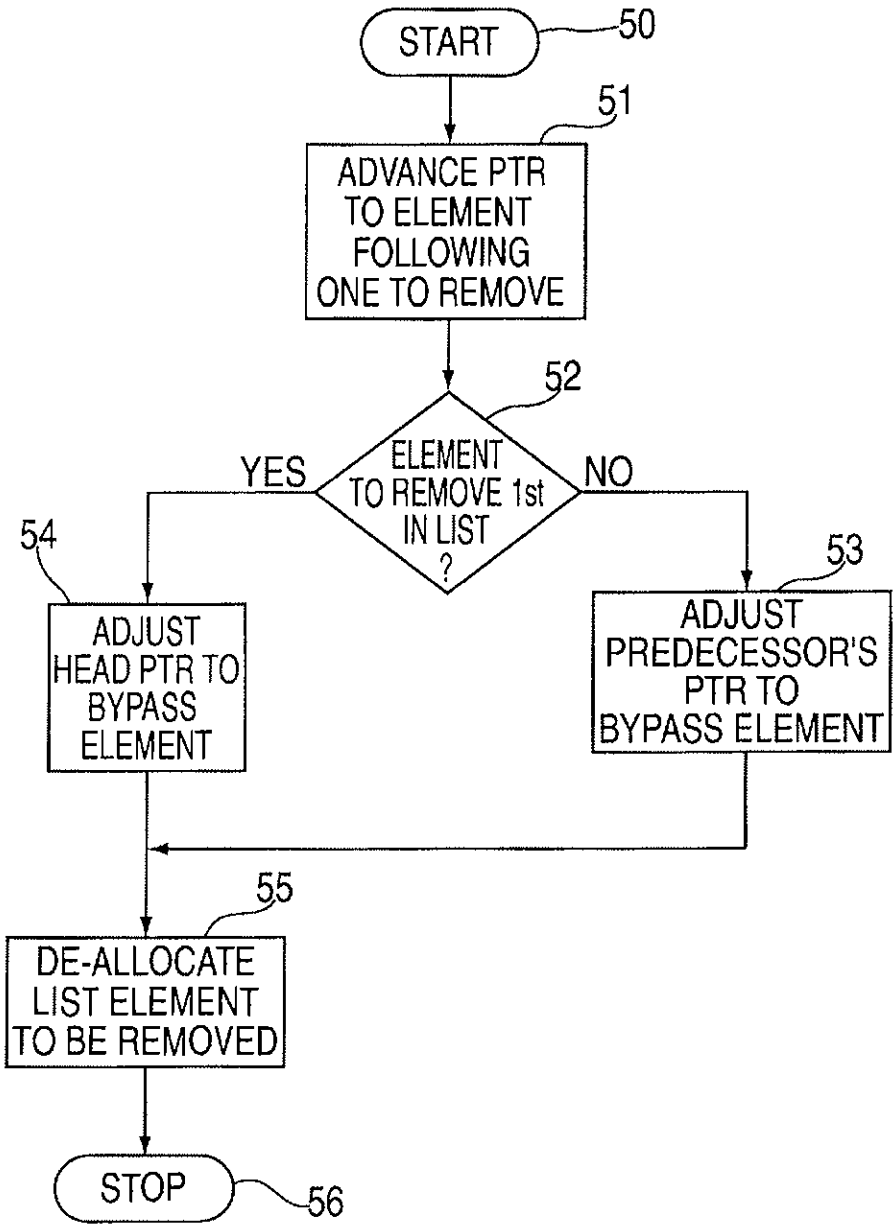

START
50
51
ADVANCE PTR TO ELEMENT FOLLOWING ONE TO REMOVE
52
ELEMENT TO REMOVE 1st IN LIST ?
YES
NO
54
ADJUST HEAD PTR TO BYPASS ELEMENT
53
ADJUST PREDECESSOR'S PTR TO BYPASS ELEMENT
55
DE-ALLOCATE LIST ELEMENT TO BE REMOVED
STOP
56

FIG. 4

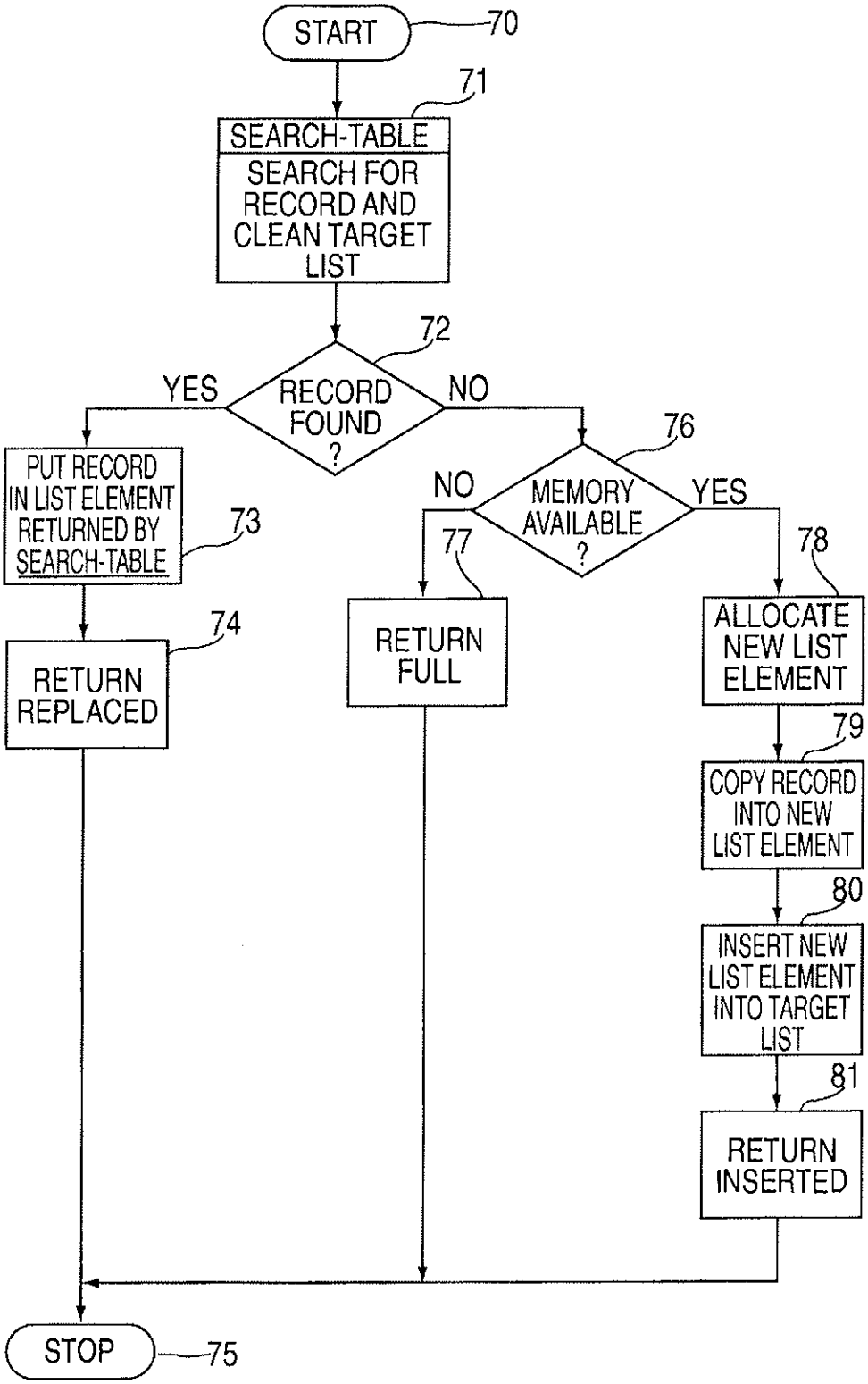

START
70
71
SEARCH-TABLE
SEARCH FOR RECORD AND CLEAN TARGET LIST
72
RECORD FOUND ?
YES
NO
PUT RECORD IN LIST ELEMENT RETURNED BY SEARCH-TABLE
73
RETURN REPLACED
74
76
MEMORY AVAILABLE ?
NO
YES
77
RETURN FULL
78
ALLOCATE NEW LIST ELEMENT
79
COPY RECORD INTO NEW LIST ELEMENT
80
INSERT NEW LIST ELEMENT INTO TARGET LIST
81
RETURN INSERTED
STOP
75

FIG. 5

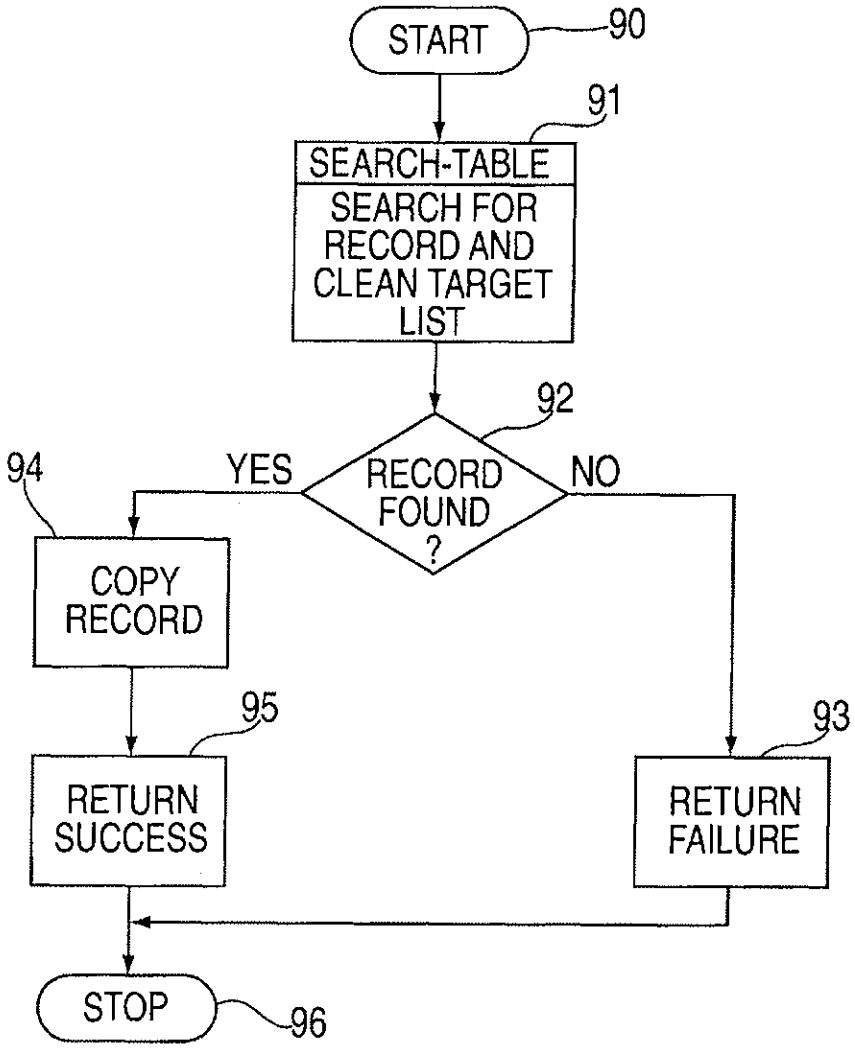

START
90
91
SEARCH-TABLE
SEARCH FOR RECORD AND CLEAN TARGET LIST
92
RECORD FOUND ?
YES
NO
94
COPY RECORD
95
RETURN SUCCESS
93
RETURN FAILURE
STOP
96

FIG. 6

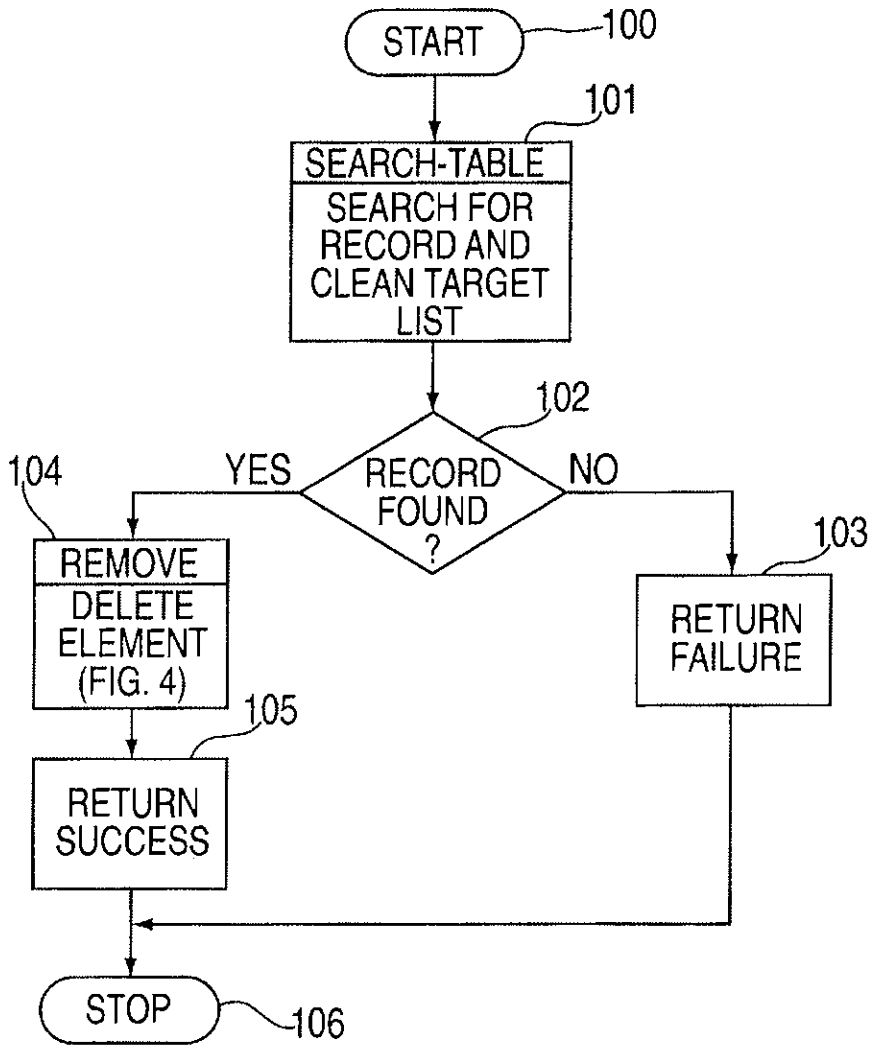

START
100
101
SEARCH-TABLE
SEARCH FOR RECORD AND CLEAN TARGET LIST
102
RECORD FOUND ?
YES
NO
104
REMOVE
DELETE ELEMENT (FIG. 4)
105
RETURN SUCCESS
103
RETURN FAILURE
STOP
106

FIG. 7

METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL USING A HASHING TECHNIQUE WITH EXTERNAL CHAINING AND ON-THE-FLY REMOVAL OF EXPIRED DATA

CROSS-REFERENCE TO RELATED APPLICATIONS

Not Applicable

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not Applicable

REFERENCE TO A MICROFICHE APPENDIX

Not Applicable

BACKGROUND OF THE INVENTION

This invention relates to information storage and retrieval systems, and, more particularly, to the use of hashing techniques in such systems.

Information or data stored in a computer-controlled storage mechanism can be retrieved by searching for a particular key value in the stored records. The stored record with a key matching the search key value is then retrieved. Such searching techniques require repeated access to records into the storage mechanism to perform key comparisons. In large storage and retrieval systems, such searching, even if augmented by efficient search procedures such as the binary search, often requires an excessive amount of time due to the large number of key comparisons required.

Another well-known and much faster way of storing and retrieving information from computer storage, albeit at the expense of additional storage, is the so-called "hashing" technique, also called scatter-storage or key-transformation method. In such a system, the key is operated on by a hashing function to produce a storage address in the storage space, called the hash table, which is a large one-dimensional array of record locations. This storage address is then accessed directly for the desired record. Hashing techniques are described in the classic text by D. E. Knuth entitled *The Art of Computer Programming,* Volume 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973, pp. 506–549.

Hashing functions are designed to translate the universe of keys into addresses uniformly distributed throughout the hash table. Typical hashing functions include truncation, folding, transposition, and modulo arithmetic. A disadvantage of hashing is that more than one key will inevitably translate in the same storage address, causing "collisions" in storage. Some form of collision resolution must therefore be provided. For example, the simple strategy called "linear probing," which consists of searching forward from the initial storage address to the first empty storage location, is often used.

Another method for resolving collisions is called "external chaining." In this technique, each hash table location is a pointer to the head of a linked list of records, all of whose keys translate under the hashing function to that very hash table address. The linked list is itself searched sequentially when retrieving, inserting, or deleting a record. Insertion and deletion are done by adjusting pointers in the linked list. External chaining is discussed in considerable detail in the aforementioned text by D. E. Knuth, in *Data Structures and Program Design,* Second Edition, by R. L. Kruse, Prentice-Hall, Incorporated, Englewood Cliffs, N.J., 1987, Section 6.5, "Hashing," and Section 6.6, "Analysis of Hashing," pp. 198–215, and in *Data Structures with Abstract Data Types and Pascal,* by D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing Company, Monterey, Calif., 1985, Section 7.4, "Hashed Implementations," pp. 310–336.

Some forms of information are such that individual data items, after a limited period of time, become obsolete, and their presence in the storage system is no longer needed or desired. Scheduling activities, for example, involve data that become obsolete once the scheduled event has occurred. An automatically-expiring data item, once it expires, needlessly occupies computer memory storage that could otherwise be put to use storing an unexpired item. Thus, expired items must eventually be removed to reclaim the storage for subsequent data insertions. In addition, the presence of many expired items results in needlessly long search times since the linked lists associated with external chaining will be longer than they otherwise would be. The goal is to remove these expired items to reclaim the storage and maintain fast access to the data.

The problem, then, is to provide the speed of access of hashing techniques for large, heavily used information storage systems having expiring data and, at the same time, prevent the performance degradation resulting from the accumulation of many expired records. Although a hashing technique for dealing with expiring data is known and disclosed in U.S. Pat. No. 5,121,495, issued Jun. 9, 1992, that technique is confined to linear probing and is entirely inapplicable to external chaining. The procedure shown there traverses, in reverse order, a consecutive sequence of records residing in the hash table array, continually relocating unexpired records to fill gaps left by the removal of expired ones.

Unlike arrays, linked lists leave no gaps when items from it are removed, and furthermore it is not possible to efficiently traverse a singly linked list in reverse order. There are significant advantages to external chaining over linear probing that sometimes make it the method of choice, as discussed in considerable detail in the aforementioned texts, and so hashing techniques for dealing with expiring data that do not use external chaining prove wholly inadequate for certain applications. For example, if the data records are large, considerable memory can be saved using external chaining instead of linear probing. Accordingly, there is a need to develop hashing techniques for external chaining with expiring data. The methods of the above-mentioned patent are limited to arrays and cannot be used with linked lists due to the significant difference in the organization of the computer's memory.

BRIEF SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the invention, these and other problems are overcome by using a garbage collection procedure "on-the-fly" while other types of access to the storage space are taking place. In particular, during normal data insertion or retrieval probes into the data store, the expired, obsolete records are identified and removed from the external chain linked list. Specifically, expired or obsolete records in the linked list including the record to be accessed are removed as part of the normal search procedure.

This incremental garbage collection technique has the decided advantage of automatically eliminating unneeded records without requiring that the information storage system be taken off-line for such garbage collection. This is

particularly important for information storage systems requiring rapid access and continuous availability to the user population.

More specifically, a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, is disclosed. The method accesses the linked list of records and identifies at least some automatically expired ones of the records. It also removes at least some automatically expired ones of the records from the linked list when the linked list is accessed. Furthermore, the method provides for dynamically determining maximum number of expired ones of the records to be removed when the linked list is accessed.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

A complete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawing, in which:

FIG. **1** shows a general block diagram of a computer system hardware arrangement in which the information storage and retrieval system of the present invention might be implemented;

FIG. **2** shows a general block diagram of a computer system software arrangement in which the information storage and retrieval system of the present invention might find use;

FIG. **3** shows a general flow chart for a table searching operation that might be used in a hashed storage system in accordance with the present invention;

FIG. **4** shows a general flow chart for a linked-list element remove procedure that forms part of the table searching operation of FIG. **3**;

FIG. **5** shows a general flow chart for a record insertion operation that might be used in a hashed storage system in accordance with the present invention;

FIG. **6** shows a general flow chart for a record retrieval operation for use in a hashed storage system in accordance with the present invention; and

FIG. **7** shows a general flow chart for a record deletion operation that might be used in a hashed storage system in accordance with the present invention.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **1** of the drawings shows a general block diagram of a computer hardware system comprising a Central Processing Unit (CPU) **10** and a Random Access Memory (RAM) unit **11**. Computer programs stored in the RAM **11** are accessed by CPU **10** and executed, one instruction at a time, by CPU **10**. Data, stored in other portions of RAM **11**, are operated on by the program instructions accessed by CPU **10** from RAM **11**, all in accordance with well-known data processing techniques.

Central Processing Unit (CPU) **10** also controls and accesses a disk controller unit **12** that, in turn, accesses a digital data stored on one or more disk storage units such as disk storage unit **13** until required by CPU **10**. At this time, such programs and data are retrieved from disk storage unit **13** in blocks and stored in RAM **11** for rapid access.

Central Processing Unit (CPU) **10** also controls an Input/Output (I/O) controller **14** that, in turn, provides access to a plurality of input devices such as CRT (cathode ray tube) terminal **15**, as well as a plurality of output devices such as printer **16**. Terminal **15** provides a mechanism for a computer user to introduce instructions and commands into the computer system of FIG. **1**, and may be supplemented with other input devices such as magnetic tape readers, remotely located terminals, optical readers, and other types of input devices. Similarly, printer **16** provides a mechanism for displaying the results of the operation of the computer system of FIG. **1** for the computer user. Printer **16** may similarly be supplemented by line printers, cathode ray tube displays, phototypesetters, laser printers, graphical plotters, and other types of output devices.

The constituents of the computer system of FIG. **1** and their cooperative operation are well-known in the art and are typical of all computer systems, from small personal computers to large mainframe systems. The architecture and operation of such systems are well-known and will not be further described here.

FIG. **2** shows a graphical representation of a typical software architecture for a computer system such as that shown in FIG. **1**. The software of FIG. **2** comprises a user access mechanism that, for simple personal computers, may consist of nothing more than turning the system on. In larger systems, providing service to many users, login and password procedures would typically be implemented in user access mechanism **20**. Once user access mechanism **20** has completed the login procedure, the user is placed in the operating system environment **21**. Operating system **21** coordinates the activities of all of the hardware components of the computer system (shown in FIG. 1) and provides a number of utility programs **22** of general use to the computer user. Utilities **22** might, for example, comprise basic file access and manipulation programs, system maintenance facilities, and programming language compilers.

The computer software system of FIG. **2** typically also includes application programs such as application software **23**, **24**, . . . , **25**. Application software **23** through **25** might, for example, comprise a text editor, document formatting software, a spreadsheet program, a database management system, a game program, and so forth.

The present invention is concerned with information storage and retrieval. It can be application software packages **23**–**25**, or used by other parts of the system, such as user access software **20** or operating system **21** software. The information storage and retrieval technique provided by the present invention are herein disclosed as flowcharts in FIGS. **3** through **7**, and shown as PASCAL-like pseudocode in the APPENDIX to this specification.

Before proceeding to a description of one embodiment of the present invention, it is first useful to discuss hashing techniques in general. Many fast techniques for storing and retrieving data are known in the prior art. In situations where storage space is considered cheap compared with retrieval time, a technique called hashing is often used. In classic hashing, each record in the information storage system includes a distinguished field unique in value to each record, called the key, which is used as the basis for storing and retrieving the associated record. Taken as a whole, a hash table is a large, one-dimensional array of logically contiguous, consecutively numbered, fixed-size storage units. Such a table of records is typically stored in RAM **11** of FIG. **1**, where each record is an identifiable and addressable location in physical memory. A hashing function

translates the key into a hash table array subscript, which is used as an index into the array where searches for the data record begin. The hashing function can be any operation on the key that results in subscripts mostly uniformly distributed across the table. Known hashing functions include truncation, folding, transposition, modulo arithmetic, and combinations of these operations. Unfortunately, hashing functions generally do not produce unique locations in the hash table, in that many distinct keys map to the same location, producing what are called collisions. Some form of collision resolution is required in all hashing systems. In every occurrence of collision, finding an alternate location for a collided record is necessary. Moreover, the alternate location must be readily reachable during future searches for the displaced record.

A common collision resolution strategy, with which the present invention is concerned, is called external chaining. Under external chaining, each hash table entry stores all of the records that collided at that location by storing not the records themselves, but instead a pointer to the head of a linked list of those same records. Such linked lists are formed by storing the records individually in dynamically allocated storage and maintaining with each record a pointer to the location of the next record in the chain of collided records. When a search key is hashed to a hash table entry, the pointer found there is used to locate the first record. If the search key does not match the key found there, the pointer there is used to locate the second record. In this way, the "chain" of records is traversed sequentially until the desired record is found or until the end of the chain is reached. Deletion of records involves merely adjusting the pointers to bypass the deleted record and returning the storage it occupied to the available storage pool maintained by the system.

Hashing techniques have been used classically for very fast access to static, short term data such as a compiler symbol table. Typically, in such storage systems, deletions are infrequent and the need for the storage system disappears quickly. In some common types of data storage systems, however, the storage system is long lived and records can become obsolete merely by the passage of time or by the occurrence of some event. If such expired, lapsed, or obsolete records are not removed from the system, they will, in time, seriously degrade the performance of the retrieval system. Degradation shows up in two ways. First, the presence of expired records lengthens search times since they cause the external chains to be longer than they otherwise would be. Second, expired records occupy dynamically allocated memory storage that could be returned to the system memory pool for useful allocation. Thus, when the system memory pool is depleted, a new data item can be inserted into the storage system only if the memory occupied by an expired one is reclaimed.

Referring then to FIG. **3**, there is shown a flowchart of a search table procedure for searching the hash table preparatory to inserting, retrieving, or deleting a record, in accordance with the present invention, and involving the dynamic removal of expired records in a targeted linked list. Starting in box **30** of the search table procedure of FIG. **3**, the search key of the record being searched for is hashed in box **31** to provide the subscript of an array element. In box **32**, the hash table array location indicated by the subscript generated in box **31** is accessed to provide the pointer to the target linked list. Decision box **33** examines the pointer value to determine whether the end of the linked list has been reached. If the end has been reached, decision box **34** is entered to determine if a key match was previously found in decision box **39** (as will be described below). If so, the search is successful and returns success in box **35**, followed by the procedure's termination in terminal box **37**. If not, box **36** is entered where failure is returned and the procedure again terminates in box **37**.

If the end of the list has not been reached as determined by decision box **33**, decision box **38** is entered to determine if the record pointed to has expired. This is determined by comparing some portion of the contents of the record to some external condition. A timestamp in the record, for example, could be compared with the current time-of-day value maintained by all computers. Alternatively, the occurrence of an event can be compared with a field identifying that event in the record. In any case, if the record has not expired, decision box **39** is entered to determine if the key in this record matches the search key. If it does, the address of the record is saved in box **40** and box **41** is entered. If the record does not match the search key, the procedure bypasses box **40** and proceeds directly to box **41**. In box **41**, the procedure advances forward to the next record in the linked list and the procedure returns to box **33**.

If decision box **38** determines that the record under question has expired, box **42** is entered to perform the on-the-fly removal of the expired record from the linked list and the return of the storage it occupies to the system storage pool, as will be described in connection with FIG. **4**. In general, the remove procedure of box **42** (FIG. **4**) operates to remove an element from the linked list by adjusting its predecessor's pointer to bypass that element. (However, if the element to be removed is the first element of the list, then there is no predecessor and the hash table array entry is adjusted instead.) On completion of procedure remove invoked from box **42**, the search table procedure returns to box **33**.

It can be seen that the search table procedure of FIG. **3** operates to examine the entire linked list of records of which the searched-for record is a part, and to remove expired records, returning storage to the storage pool with each removal. If the storage pool is depleted and many expired records remain despite such automatic garbage collection, then the insertion of new records is inhibited (boxes **76** and **77** of FIG. **5**) until a deletion is made by the delete procedure (FIG. **7**) or until the search table procedure has had a chance to replenish the storage pool through its on-the-fly garbage collection.

Though the search table procedure as shown in FIG. **3**, implemented in the APPENDIX as PASCAL-like pseudocode, and described above appears in connection with an information storage and retrieval system using the hashing technique with external chaining, its on-the-fly removal technique while traversing a linked list can be used anywhere a linked list of records with expiring data appears, even in contexts unrelated to hashing. A person skilled in the art will appreciate that this technique can be readily applied to manipulate linked lists not necessarily used with hashing.

The search table procedure shown in FIG. **3**, implemented as pseudocode in the APPENDIX, and described above traverses the entire linked list removing all expired records as it searches for a key match. The procedure can be readily adapted to remove some but not all of the expired records, thereby shortening the linked list traversal time and speeding up the search at the expense of perhaps leaving some expired records in the list. For example, the procedure can be modified to terminate when a key match occurs. (PASCAL-like pseudocode for this alternate version of search table appears in the APPENDIX.) The implementor even has the prerogative of choosing among these strategies dynamically

at the time search table is invoked by the caller, thus sometimes removing all expired records, at other times removing some but not all of them, and yet at other times choosing to remove none of them. Such a dynamic runtime decision might be based on factors such as, for example, how much memory is available in the system storage pool, general system load, time of day, the number of records currently residing in the information system, and other factors both internal and external to the information storage and retrieval system itself A person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.

In FIG. **4** there is shown a flowchart of a remove procedure that removes a record from the retrieval system, either an unexpired record through the delete procedure as will be noted in connection with FIG. 7, or an expired record through the search table procedure as noted in connection with FIG. **3**. In general, this is accomplished by the invoking procedure, being either the delete procedure (FIG. 7) or the search table procedure (FIG. **3**), passing to the remove procedure a pointer to a linked list element to remove, a pointer to that element's predecessor element in the same linked list, and the subscript of the hash table array location containing the pointer to the head of the linked list from which the element is to be removed. In the case that the element to be removed is the first element of the linked list, the predecessor pointer passed to the remove procedure by the invoking procedure has the NIL (sometimes called NULL, or EMPTY) value, indicating to the remove procedure that the element to be removed has no predecessor in the list. The invoking procedure expects the remove procedure, on completion, to have advanced the passed pointer that originally pointed to the now-removed element so that it points to the successor element in that linked list, or NIL if the removed element was the final element. (The search table procedure of FIG. **3**, in particular, makes use of the remove procedure's advancing this passed pointer in the described way; it is made use of in that box **33** of FIG. **3** is entered directly following completion of box **42**, as was described above in connection with FIG. **3**.)

The remove procedure causes actual removal of the designated element by adjusting the predecessor pointer so that it bypasses the element to be removed. In the case that the predecessor pointer has the NIL value, the hash table array entry indicated by the passed subscript plays the role of the predecessor pointer and is adjusted the same way in its stead. Following pointer adjustments, the storage occupied by the removed element is returned to the system storage pool for future allocation.

Beginning, then, at starting box **50** of FIG. **4**, the pointer to the list element to remove is advanced in box **51** so that it points to its successor in the linked list. Next, decision box **52** determines if the element to remove is the first element in the containing linked list by testing the predecessor pointer for the NIL value, as described above. If so, box **54** is entered to adjust the linked list head pointer in the hash table array to bypass the first element, after which the procedure continues on to box **55**. If not, box **53** is entered where the predecessor pointer is adjusted to bypass the element to remove, after which the procedure proceeds, once again, to box **55**. Finally, in box **55** the storage occupied by the bypassed element is returned to the system storage pool and the procedure terminates in terminal box **56**.

FIG. **5** shows a detailed flowchart of an insert procedure suitable for use in the information storage and retrieval system of the present invention. The insert procedure of FIG. **5** begins at staring box **70** from which box **71** is entered. In box **71**, the search table procedure of FIG. **3** is invoked with the search key of the record to be inserted. As noted in connection with FIG. **3**, the search table procedure finds the linked list element whose key value of the record contained therein matches the search key and, at the same time, removes expired records on-the-fly from that linked list. Decision box **72** is then entered where it is determined whether the search table procedure found a record with matching key value. If so, box **73** is entered where the record to be inserted is put into the linked list element in the position of the old record with matching key value. In box **74**, the insert procedure reports that the old record has been replaced by the new record and the procedure terminates in terminal box **75**.

Returning to decision box **72**, if a matching record is not found, decision box **76** is entered to determine if there is sufficient storage in the system storage pool to accommodate a new linked list element. If not, box **77** is entered to report that the storage system is full and the record cannot be inserted. Following that, the procedure terminates in terminal box **75**.

If decision box **76** determines that sufficient storage can be allocated from the system storage pool for a new linked list element, then box **78** is entered where the actual memory allocation is made. In box **79**, the record to be inserted is copied into the storage allocated in box **78**, and box **80** is entered. In box **80**, the linked list element containing the record copied into it in box **79** is inserted into the linked list to which the contained record hashed. The procedure then reports that the record was inserted into the information storage and retrieval system in box **81** and the procedure terminates in box **75**.

FIG. **6** shows a detailed flowchart of a retrieve procedure used to retrieve a record from the information storage and retrieval system. Starting in box **90**, the search table procedure of FIG. **3** is invoked in box **91**, using the key of the record to be retrieved as the search key. In decision box **92** it is determined if a record with a matching key was found by the search table procedure. If not, box **93** is entered to report failure of the retrieve procedure, and the procedure terminates in terminal box **96**. If a matching record was found, box **94** is entered to copy the matching record into a record store for processing by the calling program, box **95** is entered to return an indication of successful retrieval, and the procedure terminates in terminal box **96**.

FIG.**7** shows a detailed flowchart of a delete procedure useful for actively removing records from the information storage and retrieval system. Starting at box **100**, the procedure of FIG. **7** invokes the search table procedure of FIG. **3** in box **101**, using the key of the record to be deleted as the search key. In decision box **102**, it is determined if the search table procedure was able to find a record with matching key. If not, box **103** is entered to report failure of the deletion procedure, and the procedure terminates in terminal box **106**. If a matching record was found, as determined by decision box **102**, the remove procedure of FIG. **4** is invoked in box **104**. As noted in connection with FIG. **4**, the remove procedure causes removal of a designated linked list element from its containing linked list. Box **105** is then entered to report successful deletion to the calling program, and the procedure terminates in terminal box **106**.

The attached APPENDIX contains PASCAL-like pseudocode listings for all of the programmed components necessary to implement an information storage and retrieval system operating in accordance with the present invention. Any person of ordinary skill in the art will have no difficulty implementing the disclosed system and procedures shown in the APPENDIX, including programs for all common hardware and system software arrangements, on the basis of this

description, including flowcharts and information shown in the APPENDIX.

It should also be clear to those skilled in the art that other embodiments of the present invention may be made by those skilled in the art without departing from the teachings of the present invention. It is also clear to those skilled in the art that the invention can be used in diverse computer applications, and that it is not limited to the use of hash tables, but is applicable to other techniques requiring linked lists with expiring records.

Appendix

Functions Provided

The following functions are made available to the application program:

1. insert (record: record_type)
   Returns replaced if a record associated with record.key was found and subsequently replaced.
   Returns inserted if a record associated with record.key was not found and the passed record was subsequently inserted.
   Returns full if a record associated with record.key was not found and the passed record could not be inserted because no memory is available.
2. retrieve (record: record_type)
   Returns success if record associated with record.key was found and assigned to record.
   Returns failure if search was unsuccessful.
3. delete (record_key: record_key_type)
   Returns success if record associated with record_key was found and subsequently deleted.
   Returns failure if not found.

Definitions

The following formal definitions are required for specifying the insertion, retrieval, and deletion procedures. They are global to all procedures and functions shown below.

```
1. const table_size                                              /* Size of hash table. */
2. type list_element_pointer = ↑ list_element                    /* Pointer to elements of linked list. */
3. type list_element =                                           /* Each element of linked list. */
     record
       record_contents: record_type;
       next: list_element_pointer                                /* Singly-linked list. */
     end
4. var table: array [0 . . . table_size − 1] of list_element_pointer   /* Hash table. */
                                                                 /* Each array entry is pointer to head of list. */
     Initial state of table: table[i] = nil ∀ i 0 ≤ i < table_size       /* Initially empty. */
```

Insert Procedure

```
function insert (record: record_type): (replaced, inserted, full);
  var position: list_element_pointer;                            /* Pointer into list of found record, */
                                                                 /* or new element if not found. */
    dummy_pointer: list_element_pointer;                         /* Don't need position's predecessor. */
    index: 0 . . . table_size − 1;                               /* Table index mapped to by hash function. */
  begin
    if search_table (record.key, position, dummy_pointer, index)     /* Record already exist? */
      then begin                                                 /* Yes, update it with passed record. */
        position↑ .record_contents := record;
        return (replaced)
      end
      else                                                       /* No, insert new record at head of list, */
    if no memory available then return (full)                    /* if memory available to do so. */
      else begin                                                 /* Memory is available for a node. */
        new(position);                                           /* Dynamically allocate new node. */
        position↑ .record_contents := record;                    /* Hook it in. */
        position↑ .next := table[index];
        table[index] := position;
        return (inserted)
      end                                                        /* else begin */
  end                                                            /* insert */
```

Retrieve Procedure

```
function retrieve (var record: record_type): (success, failure);
  var position: list_element_pointer;                            /* Pointer into list of found record. */
    dummy_pointer: list_element_pointer;                         /* Don't need position's predecessor. */
    dummy_index: 0 . . . table_size − 1;                         /* Don't need table index mapped to by hash function. */
begin
    if search_table (record.key, position, dummy_pointer, dummy_index)   /* Record exist? */
      then begin                                                 /* Yes, return it to caller. */
        record := position↑ .record_contents;
        return (success)
      end
        else return (failure)                                    /* No, report failure. */
      end                                                        /* retrieve */
```

-continued

Appendix

Delete Procedure

```
function delete (record_key: record_key_type): (success,failure);
  var position: list_element_pointer;                    /* Pointer into list of found record. */
    previous_position: list_element_pointer;             /* Points to position's predecessor. */
    index: 0 . . . table_size - 1;                       /* Table index mapped to by hash function. */
  begin
    if search_table (record_key, position, previous_position, index)   /* Record exist? */
      then begin                                                       /* Yes, remove it. */
        remove (position, previous_position, index);
        return (success)
      end
      else return (failure)                                          /* No, report failure. */
  end                                                  /* delete */
```

Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size - 1): boolean;
/*  Search table for record_key and delete expired records in target list; if found, position is made to
    point to located record and previous_position to its predecessor, and TRUE is returned; otherwise
    FALSE is returned. index is set to table subscript that is mapped to by hash function in either
    case. */
var p: list_element_pointer;                             /* Used for traversing chain. */
  previous_p: list_element_pointer;                      /* Points to p's predecessor. */
begin
  index := hash (record_key);                /* hash returns value in the range 0 . . . table_size - 1. */
  p := table[index];                                     /* Initialization before loop. */
  previous_p := nil;                                                          /* Ditto */
  position := nil;                                                            /* Ditto */
  previous_position := nil;                                                   /* Ditto */
  while p ≠ nil                              /* HEART OF THE TECHNIQUE: Traverse entire list, deleting */
                                                          /* expired records as we search. */
    begin
      if p↑ .record_contents is expired
        then remove (p, previous_p, index)       /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if position = nil then if p↑ .record_contents.key = record_key
                                                                  /* If this is record wanted,*/
            then begin position := p; previous_position := previous_p end;
                                                                  /* save its position. */
          previous_p := p;                                        /* Advance to */
          p := p↑ .next                                           /* next record. */
        end                                                       /* else begin */
    end;
      return (position ≠ nil)                  /* Return TRUE if record located, otherwise FALSE. */
  end                                          /* search_table */
```

Alternate Version of Search Table Procedure

```
function search_table (record_key: record_key_type;
                       var position: list_element_pointer;
                       var previous_position: list_element_pointer;
                       var index: 0 . . . table_size - 1): boolean;
/*  SAME AS VERSION SHOWN ABOVE EXCEPT THAT THE SEARCH TERMINATES IF
    RECORD IS FOUND, INSTEAD OF ALWAYS TRAVERSING THE ENTIRE CHAIN. */
var p: list_element_pointer;                             /* Used for traversing chain. */
  previous_p: list_element_pointer;                      /* Points to p's predecessor. */
begin
  index := hash (record_key);                /* hash returns value in the range 0 . . . table_size - 1. */
  p := table[index];                                     /* Initialization before loop. */
  previous_p := nil;                                                          /* Ditto */
  position := nil;                                                            /* Ditto */
  previous_position := nil;                                                   /* Ditto */
  while p ≠ nil                              /* HEART OF THE TECHNIQUE: Traverse list, deleting */
                                                          /* expired records as we search. */
    begin
      if p↑ .record_contents is expired
        then remove (p, previous_p, index)       /* ON-THE-FLY REMOVAL OF EXPIRED RECORD! */
        else begin
          if p↑ .record_contents.key = record_key                 /* If this is record wanted,*/
            then begin                                            /* save its position. */
              position := p;
              previous_position := previous_p;
              return (true)                      /* We found the record, so terminate search. */
            end;
          previous_p := p;                                        /* Advance to */
          p := p↑ .next                                           /* next record. */
```

-continued

Appendix

```
        end                                                    /* else begin */
    end;
    return (false)                                             /* Record not found. */
  end                                          /* search_table */
```

Remove Procedure

```
procedure remove (var elem_to_del: list_element_pointer;
                  previous_elem: list_element_pointer;
                  index: 0 . . . table_size − 1);
/*  Delete elem_to_del↑ from list, advancing elem_to_del to next element. previous_elem points to
    elem_to_del's predecessor, or nil if elem_to_del↑ is 1st element in list.*/
var p: list_element_pointer;                          /* Save pointer to elem_to_del for disposal. */
begin
  p := elem_to_del;                    /* Save so we can dispose when finished adjusting pointers. */
  elem_to_del := elem_to_del↑.next;
  if previous_elem = nil                                   /* Deleting 1st element requires changing */
    then table[index] := elem_to_del                              /* head pointer, as opposed to */
    else previous_elem↑.next := elem_to_del;                      /* predecessor's next pointer. */
    dispose (p)                                              /* Dynamically de-allocate node. */
  end                                          /* remove*/
```

I claim:

1. An information storage and retrieval system, the system comprising:

a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,

a record search means utilizing a search key to access the linked list,

the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and

means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.

2. The information storage and retrieval system according to claim **1** further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.

3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:

accessing the linked list of records,

identifying at least some of the automatically expired ones of the records, and

removing at least some of the automatically expired records from the linked list when the linked list is accessed.

4. The method according to claim **3** further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.

5. An information storage and retrieval system, the system comprising:

a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,

a record search means utilizing a search key to access a linked list of records having the same hash address,

the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and

meals, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.

6. The information storage and retrieval system according to claim **5** further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.

7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:

accessing a linked list of records having same hash address,

identifying at least some of the automatically expired ones of the records,

removing at least some of the automatically expired records from the linked list when the linked list is accessed, and

inserting, retrieving or deleting one of the records from the system following the step of removing.

8. The method according to claim **7** further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.

* * * * *

Exhibit A.2

IW 7197768

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

August 27, 2009

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS OF:

APPLICATION NUMBER: *08/775,864*
FILING DATE: *January 02, 1997*
PATENT NUMBER: *5,893,120*
ISSUE DATE: *April 06, 1999*

By Authority of the
Under Seeretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

W. MONTGOMERY
Certifying Officer

GAU 2771

Richard Michael Nemes
2821 Kings Highway, Apartment 1M
Brooklyn, New York 11229

August 10, 1998

Assistant Commissioner of Patents and Trademarks
Washington, D.C. 20231

re: Patent Application No. 08/775,864, Art Unit: 2771, Examiner: H.T. Alam

Dear Sir,

Find enclosed a 6-page response to the Office action dated April 20, 1998, in conjunction with the above patent application. In addition, I am enclosing a petition for extension of time along with a check in the sum of $55, and a self-addressed postcard listing the enclosed items.

Please note my change of permanent address and phone number:

Richard Michael Nemes
2821 Kings Highway, Apartment 1M
Brooklyn, New York 11229
U.S.A.

Telephone: (718) 677-1748

Sincerely yours,

Richard Michael Nemes

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

5/Reg for Reconsid 8-19-98 MP

Application Number: 08/775,864
Art Unit: 2771
Examiner: Hosain T. Alam
Applicant: Richard Michael Nemes

Express Mail No.: EE661456784US

RESPONSE

Assistant Commissioner of Patents and Trademarks
Washington, D.C. 20231

Sir:

In response to the outstanding Office action dated April 20, 1998, please consider the following remarks (a petition for extension of time and payment are enclosed herewith).

REMARKS

1. Formal drawings will be submitted after allowance of the application.

Response to "Part III DETAILED ACTION," Items 3–7: Double Patenting

2. The Office states in items 3–5 that the subject matter claimed in the instant application is fully disclosed in U.S. Patent No. 5,121,495 issued to Nemes (hereinafter '495) and claimed in Claim 1 of '495. Specifically, item 5 states that the term "chain of records" appearing in Claim 1 of '495 (col. 12, line 7) is equivalent to a linked list of pointers/addresses of records, and that "chaining" is equivalent to being linked.

'495 nowhere teaches or claims linked lists or pointers, and is strictly confined to the linear probing technique of hashing, as explicitly stated in Claim 1 of '495 which reads "An information storage and retrieval system using hashing techniques to provide rapid access to the records of said system and utilizing a linear probing technique ..." (col. 11, line 67 through col. 12, line 3).

The linear probing technique is described in pages 506–549 of the classic text by D. E. Knuth entitled *The Art of Computer Programming*, Volume 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973 (Cite No. 1 in the List of Prior Art Cited by Applicant submitted in conjunction with the instant application) and is shown there to be applicable only to the "open addressing" method of collision resolution, which is a nonlinked-list technique. In the same vein, in the DETAILED DESCRIPTION section of the disclose in '495 a hash table is "described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record" (col. 4, line 33–36). This description excludes linked list implementations, which are claimed here. Since the instant application claims linked lists, not suggested by linear probing of the cited art, the subject matter claimed is not disclosed in '495.

The term "chain of records" appearing in '495 is used descriptively, and not as a term of art. In '495 it consistently refers to a sequence of consecutively occupied storage locations, and makes no sense when interpreted to include linked list implementations (col. 1, line 60–63; col. 2, line 11–17). The aforementioned text by Knuth at page 527 is cited by '495 in this context (col. 2, line 17), the discussion in that text being limited to linear probing under open addressing, a strictly nonlinked-list technique. In the same vein, in the DETAILED DESCRIPTION section of the disclosure in '495, all uses of the term "chain" (e.g. col. 5, lines 7, 10, 41, 44; col. 6, line 39) follow and are consistent with the description of a hash table only "as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record" (col. 4, line 33–36.). This definition of "chain" is inconsistent with and does not suggest the linked list technology claimed in the present application. Thus, '495 does not teach or suggest linked list technology claimed in the instant application.

Item 5 states that as to claims 1 and 3, '495 does not recite the term "linked list," but instead recites "chain of records," it being obvious to a person of ordinary skill in the art at the time the invention was made to use a linked list of records because a chain of records generates a linked list. In light of the preceding explanation that "chain" as used in '495 does not suggest linked list, this rejection should be withdrawn.

Item 5 states that as to claims 2 and 4, '495 does not recite the removal based on the determination of a maximum number or expired records, "it being obvious to a person of ordinary skill in the art at the time the invention was made to group a number of records and thus to predetermine the maximum number in the group to facilitate an efficient processing of records ...." Since claims 2 and 4 are dependent on claims 1 and 3, respectively, which have been shown in the previous paragraphs to be not suggested by the subject matter of '495, these claims are also patentable.

3. The Office states in items 6–7 that the subject matter of claims 5–8 is fully disclosed in U.S. Patent No. 5,287,499 issued to Nemes (hereinafter '499) and claimed in Claims 1 and 2 of '499. Specifically, item 6 states that Claims 1 and 2 of '499 are "directed to an apparatus and method for information storage and retrieval wherein the memory addresses are hashed by using a chain of records having same hash address, the chaining of records is external (see claim 1, col. 17, line 1)." It then states that "the 'external chaining of records' is equivalent to a linked list of pointers/addresses of records as claimed and the 'chaining' is equivalent to being linked."

Although it is true that in the instant application "external chaining" and "chaining" are each equivalent to being linked, '499 does not teach or suggest on-the-fly deletion of at least some records based on automatic expiration of data, which is claimed here.

Item 6 states that as to claim 5 and 7, '499 does not recite the terms "linked list," "insert,"

"retrieve," or "delete," but instead recites "external chaining" and "storing," and that "it would have been obvious to a person of ordinary skill in the art at the time the invention was made to use a linked list of records because a chain of records chained by an external chaining generates a linked list" (sic). The '499 patent, however, does not teach means or methods for identifying and removing "at least some expired ones of the records" from the linked list "when the linked list is accessed" (see claims 5 and 7), which is taught by the instant application and is integral to claims 5 and 7. Thus, the rejection should be withdrawn.

Item 6 states that as to claims 6 and 8,[1] '499 does not recite a "maximum number of records" but instead recites "threshold," and that "It would have been obvious to a person of ordinary skill in the art at the time the invention was made to group a number or records for determining the threshold and thus to predetermine the maximum number for the threshold to facilitate an efficient processing of records ...." The "maximum number of records" (in the instant application) and "threshold" (in '499) serve different purposes and are structured and determined differently. In the instant application, the number is a single quantity that serves as an upper limit on the number of records removed from the linked list whenever the linked list is accessed (see claims 6 and 8), whereas in '499 the threshold is a pair of coupled quantities, an upper threshold and a lower threshold, that serve as two-way signals indicating when the system should automatically reorganize a group of records that reside in cells of the hash table into a linked list, and vice versa (col. 6, lines 44–54 and 61–65; APPENDIX). Since neither the maximum number of records nor the upper threshold can be learned from the other by a person of ordinary skill in the art from either '499 or the instant application, the rejection should be removed. Furthermore, the dependent

1. Item 6 reads "... As to claims 7 and 8: The '499 patent does not recite a 'maximum number of records' instead recite a 'threshold' " (sic). This appears to be in error and should read "As to claims 6 and 8: ...." since the term "maximum number" appears only in claims 2, 4, 6, and 8.

claims are patentable because the independent claims on which they depend are patentable.

Item 7 states that there is no apparent reason why claims corresponding to those of the instant application were not presented during prosecution of '499. In light of what has been shown above, that the teachings of the instant application are not included in those of '499, the rejection should therefore be withdrawn.

Response to "Part III DETAILED ACTION," Items 8–11: Claim Rejections – 35 USC § 103

4. In items 8–11, the Office rejects claims 1–8 under 35 U.S.C. § 103 (obviousness) as being unpatentable over U.S. Patent No. 5,287,499 issued to Nemes (hereinafter '499) in view of U.S. Patent No. 5,202,981 issued to Shackelford (hereinafter "Shackelford"). Specifically, item 10 states that with respect to claims 1–8, '499 teaches everything that is claimed (col. 2, line 60–64; col. 6, line 49–51) with the exception that it does not explicitly indicate the determination of "threshold" as being the "maximum number of records," and that Shackelford teaches "maximum number of pointers" (col. 3, line 61 through col. 4, line 2).

Claims 1–8 of the instant application address on-the-fly deletion of at least some records from a linked list based on automatic expiration of data, whereas '499 teaches automatic reorganization of records from linked list structure to sequential storage structure and vice versa to facilitate system efficiency. Nowhere does '499 teach deletion from the system, nor does it teach regarding automatically expiring data.

The instant application teaches and claims (claims 2, 4, 6, and 8) means and method for dynamically determining the maximum number of records to be removed on-the-fly from a linked list when that linked list is accessed. Shackelford, on the other hand, teaches an unrelated quan-

tity, the existence of a stored quantity accompanying the stream class data structure that identifies the maximum number of pointers that are permitted to exist (col. 3, line 61 through col. 4, line 2). Shackelford does not address an application with automatically expiring data, nor does he address how many items to delete. These references separately or in combination do not suggest the claims of the present application. The rejection, therefore, should be withdrawn.

5. Item 11 states that claims 1--8 are rejected under 35 U.S.C. § 103 as being unpatentable over '499 directed to the linked lists and the step of removing, as set forth in the Double Patenting discussion, which is item 6 in the Office action.

Neither '499 nor Shackelford suggest what is recited in claims 1, 3, 5, and 7, for example, means and methods for identifying and removing "at least some expired ones of the records" from the linked list "when the linked list is accessed." In addition, for the reasons explained in detail in the previous discussion, rejection of claims 2, 4, 6, and 8, which are directed to "dynamically determining maximum number," has already been discussed above. Thus, this rejection should be withdrawn.

In view of the foregoing remarks, this application should be allowed to issue as a patent.

Respectfully submitted,

Richard Michael Nemes

Richard Michael Nemes

August 10, 1998

Exhibit A.3

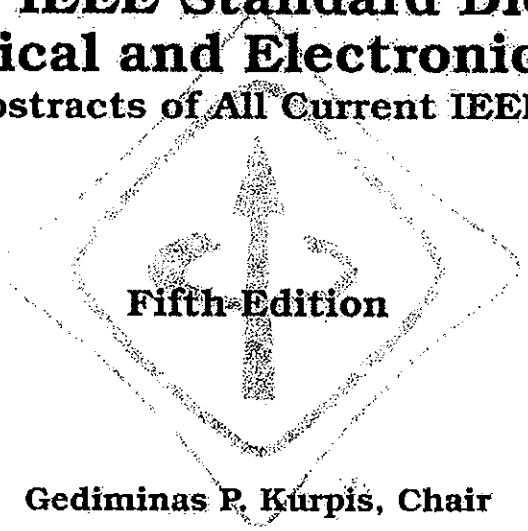
IEEE Std 100-1992

The New IEEE Standard Dictionary of Electrical and Electronics Terms

IEEE Std 100-1992

The New IEEE Standard Dictionary of Electrical and Electronics Terms

[Including Abstracts of All Current IEEE Standards]



Fifth Edition

Gediminas P. Kurpis, Chair

Christopher J. Booth, Editor

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 1-55937-240-0

dyadic (mathematics of computing). Pertaining to an operation involving two operands. *Contrast with:* **monadic.** 610.1

dyadic Boolean operation. A logical operation involving two operands. For example, the equivalence operation. *Contrast with:* **monadic Boolean operation.** 610.1

dyadic operation. An operation involving two operands. *Contrast with:* **monadic operation.** 610.1

dyadic operator. An operator that specifies an operation on two operands. *Syn:* **binary operator.** *Contrast with:* **monadic operator.** 610.1

dyadic selective construct. An if-then-else construct in which processing is specified for both outcomes of the branch. *Contrast with:* **monadic selective construct.** 610.12-1990

dynamic (industrial control) (excitation control systems). A state in which one or more quantities exhibit appreciable change within an arbitrarily short time interval. *Note:* For excitation control systems, this time interval encompasses up to 15-20 sec.; that is, sufficient time to ascertain whether oscillations are decaying or building up with time. *See:* **control system, feedback.** 421A-1978
(2) (software). Pertaining to an event or process that occurs during computer program execution; for example, dynamic analysis, dynamic binding. *Contrast with:* **static.** 610.12-1990

dynamic accuracy (1). Accuracy determined with a time-varying output. Contrast with static accuracy. *See:* **electronic analog computer.** 185-1975w
(2) (analog computers). Accuracy determined with a time-varying output. 165-1977

dynamic allocation (software). *See:* **dynamic resource allocation.** 610.12-1990

dynamically tuned gyro (DTG) (inertial sensor). A two-degree-of-freedom gyro in which a dynamically tuned flexure and gimbal mechanism both supports the rotor and provides angular freedom about axes perpendicular to the spin axis. *See:* **dynamic tuning.** 528-1984w

dynamic analysis (software). The process of evaluating a system or component based on its behavior during execution. *Contrast with:* **static analysis.** *See also:* **demonstration; testing.** 610.12-1990

dynamic analyzer (software). A software tool that aids in the evaluation of a computer program by monitoring execution of the program. Examples include instrumentation tools, software monitors, and tracers. *See:* **computer program; execution; instrumentation tools; program; software monitor; software tool; static analyzer; tracer.** 729-1983

dynamic binding (software). Binding performed during the execution of a computer program. *Contrast with:* **static binding.** 610.12-1990

dynamic braking (rotating machinery). A system of electric braking in which the excited machine is disconnected from the supply system and connected as a generator, the energy being dissipated in the winding and, if necessary, in a separate resistor. [9]

dynamic braking envelope. A curve that defines the dynamic braking limits in terms of speed and tractive force as restricted by such factors asmaximum current flow, maximum permissible voltage, minimum field strength, etcetera. *See:* **dynamic braking.** [119]

dynamic breakpoint. A breakpoint whose predefined initiation event is a runtime characteristic of the program, such as the execution of any twenty source statements. *Contrast with:* **static breakpoint.** *See also:* **code breakpoint; data breakpoint; epilog breakpoint; programmable breakpoint; prolog breakpoint.** 610.12-1990

dynamic buffering. A buffering technique in which the buffer allocated to a computer program varies during program execution, based on current need. *Contrast with:* **simple buffering.** 610.12-1990

dynamic bus sizing. The ability of some microprocessors to adjust the number and the size of data transfers to the amount of data that the responding board can access in one transfer. During the address broadcast portion of the cycle, the slave informs the master how many data lines it actually drives or receives. This information is made available to on-board logic that can then adjust the amount of data that it accesses during the data transfer to the capabilities of the slave. 1096-1988

dynamic characteristic (electron tube) (operating characteristic). *See:* **load (dynamic) characteristic (electron tube).**

dynamic check. *See:* **problem check.** 165-1977

dynamic computer check. *See:* **problem check.**

dynamic cutoff frequency (semiconductor) (nonlinear, active, and nonreciprocal waveguide components). A figure of merit used for varactor diodes. Unlike fixed cutoff frequency measurements at specific bias voltages, dynamic cutoff frequency is a measure of the varactor's total change in Q from a slight forward bias current to reverse breakdown voltage. This dynamic or total figure of merit is useful in evaluating the frequency multiplier performance of fully driven multipliers. 457-1982

dynamic dump (computing systems). A dump that is performed during the execution of a program. [20], [85]

Exhibit A.4

Designed for Microsoft® Windows NT® Windows® 95

CD-ROM Included

The Ultimate Computer Reference

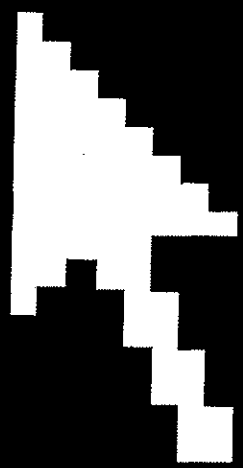
The Comprehensive Standard for Business, School, Library, and Home

Over 7,600 Terms

Additional Terms Available On Line Quarterly

Microsoft Press Computer Dictionary

Third Edition



- Over 300 illustrations and diagrams
- Extensive Internet coverage
- Featured in Microsoft® Bookshelf®
- Covers software, hardware, concepts, and more!

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Microsoft Press Computer Dictionary. -- 3rd ed.
p. cm.
ISBN 1-57231-446-X
1. Computers--Dictionaries. 2. Microcomputers--Dictionaries.
I. Microsoft Press.
QA76.15.M54 1997
004'.03--dc21 97-15489
CIP

Printed and bound in the United States of America.

5 6 7 8 9 QMQM 2 1 0 9 8

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office. Or contact Microsoft Press International directly at fax (425) 936-7329.

Acquisitions Editor: Kim Fryer
Project Editor: Maureen Williams Zimmerman, Anne Taussig
Technical Editors: Dail Magee Jr., Gary Nelson, Jean Ross, Jim Fuchs, John Conrow, Kurt Meyer, Robert Lyon, Roslyn Lutsch

records or keys exist in a file. *See also* key (definition 2). **2.** The use of separate independent calculations to establish the accuracy of a result.

DVD \D`V-D´\ *n. See* digital video disc.

DVD-E \D`V-D-E´\ *n. See* digital video disc–erasable.

DVD-R \D`V-D-R´\ *n. See* digital video disc–recordable.

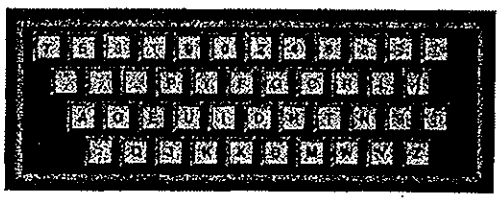
DVD-ROM \D`V-D-rom´, -R-O-M´\ *n. See* digital video disc–ROM.

DVI or **DV-I** \D`V-I´\ *n.* Acronym for **D**igital **V**ideo **I**nterface. A hardware-based compression/decompression technique for storing full-motion video, audio, graphics, and other data on a computer or on a CD-ROM. DVI technology was developed by RCA in 1987 and acquired by Intel in 1988. Intel has since developed a software version of DVI, called Indeo. *Also called* digital video–interactive.

DV-I \D`V-I´\ *n. See* digital video–interactive.

DVMRP \D`V-M-R-P´\ *n. See* Distance Vector Multicast Routing Protocol.

Dvorak keyboard \də-vōr´zhak kē`bōrd, də-vōr´zhäk, dē-vōr´ak\ *n.* A keyboard layout developed by August Dvorak and William L. Dealey in 1936 as an alternative to the overwhelmingly popular QWERTY keyboard. The Dvorak keyboard was designed to speed typing by placing the characters on the keyboard for easiest access to the most frequently typed letters. In addition, pairs of letters that often occur sequentially were separated so that the hands could alternate typing them. See the illustration. *See also* ergonomic keyboard, keyboard. *Compare* QWERTY keyboard.



Dvorak keyboard.

DVST \D`V-S-T´\ *n. See* direct view storage tube.

DXF \D`X-F´\ *n.* Short for drawing interchange format. A computer-aided design file format originally developed for use with the AutoCAD program to facilitate transfer of graphics files between different applications.

dyadic \dī-ad´ik`\ *adj.* Of, pertaining to, or characteristic of a pair—for example, a dyadic processor, which contains two processors controlled by the same operating system. The term is usually limited to describing a system with two microprocessors. Dyadic Boolean operations are those such as AND and OR in which the outcome depends on both values. *See also* Boolean algebra, operand. *Compare* unary.

dye-diffusion printer \dī`di-fyōō-zhən prin´tər\ *n. See* continuous-tone printer.

dye-polymer recording \dī`pol´-ə-mər rə-kōr`-dēng\ *n.* A recording technology used with optical discs in which dye embedded in a plastic polymer coating on an optical disc is used to create minute bumps on the surface that can be read by a laser. Dye-polymer bumps can be flattened and re-created, thus making an optical disc rewritable.

dye-sublimation printer \dī`su-blə-mā`shən prin´-tər\ *n. See* continuous-tone printer.

dynalink \dī´nə-lēnk`\ *n.* Short for **dyna**mic **link.** *See* dynamic-link library.

Dynaload drivers \dī´nə-lōd drī`vərz\ *n.* Device drivers that are supported by Dynaload. Dynaload is a command that can be run from a DOS prompt under IBM's PC DOS 7 and will load compliant device drivers without modification of the CONFIG.SYS file. *See also* CONFIG.SYS.

dynamic \dī-nam´ik`\ *adj.* Occurring immediately and concurrently. The term is used in describing both hardware and software; in both cases it describes some action or event that occurs when and as needed. In dynamic memory management, a program is able to negotiate with the operating system when it needs more memory.

dynamic address translation \dī-nam`ik a´dres tranz-lā`shən, ə-dres´\ *n.* On-the-fly conversion of memory-location references from relative addresses (such as "three units from the beginning of X") to absolute addresses (such as "location number 123") when a program is run. *Acronym:* DAT (dat, D`A-T´).

dynamic allocation \dī-nam`ik a-lə-kā´shən\ *n.* The allocation of memory during program execution according to current needs. Dynamic allocation almost always implies that dynamic deallocation is possible too, so data structures can

Exhibit A.5

US005287499A

United States Patent [19]

Nemes

[11] Patent Number: 5,287,499

[45] Date of Patent: * Feb. 15, 1994

[54] **METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING A METHOD OF HASHING AND DIFFERENT COLLISION AVOIDANCE SCHEMES DEPENDING UPON CLUSTERING IN THE HASH TABLE**

[75] Inventor: **Richard M. Nemes**, Brooklyn, N.Y.

[73] Assignee: **Bell Communications Research, Inc.**, Livingston, N.J.

[*] Notice: The portion of the term of this patent subsequent to Feb. 26, 2008 has been disclaimed.

[21] Appl. No.: **702,444**

[22] Filed: **May 16, 1991**

Related U.S. Application Data

[63] Continuation of Ser. No. 326,976, Mar. 22, 1989, abandoned.

[51] **Int. Cl.**[5] **G06F 12/00**

[52] **U.S. Cl.** **395/600**; 395/700; 371/5.1; 364/962; 364/962.1; 364/963; 364/DIG. 2

[58] **Field of Search** 395/700, 600, 800; 364/900; 371/5.1

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,704,363	11/1972	Salmassy et al.	371/5.1
4,339,657	7/1982	Larson et al.	371/5.1
4,380,067	4/1983	Beardsley et al.	371/11.2
4,564,944	1/1986	Arnold et al.	371/37
4,638,426	1/1987	Chang et al.	364/200
4,663,620	5/1987	Paul et al.	340/825.5
4,680,700	7/1987	Hester et al.	364/200
4,680,703	7/1987	Kriz	364/200
4,695,949	9/1987	Thatte et al.	364/200
4,764,863	8/1988	Silverthorn, III et al.	364/200
4,866,712	9/1989	Chao	371/5.1
4,922,417	5/1990	Churm et al.	364/200
4,961,139	10/1990	Hong et al.	364/200
4,979,105	12/1990	Daly et al.	395/575
4,996,663	2/1991	Nemes	364/900

OTHER PUBLICATIONS

D. Knuth, *The Art of Computer Programming*, vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973, pp. 506-549.

R. L. Kruse, *Data Structures and Program Design*, Prentice-Hall, Englewood Cliffs, N.J., 1984, pp. 112-126.

D. F. Stubbs et al., *Data Structures with Abstract Data Types and Pascal*, Brooks-Cole Publishing, Monterey, Calif., 1985, pp. 310-336.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Paul Harrity
Attorney, Agent, or Firm—Leonard Charles Suchyta; James W. Falk

[57] **ABSTRACT**

An apparatus for performing storage and retrieval in an information storage system is disclosed which uses the hashing technique. In order to provide efficient and graceful operation under varying loading conditions, the system shifts between collision avoidance by linear probing with open addressing when the load is below a threshold, and collision avoidance by external chaining when the load is above a threshold. Insertion, deletion and retrieval operations are arranged to switch dynamically between the two collision avoidance stratagems as the local loading factor on the system, as measured by the number of records hashed to the same address, crosses preselected thresholds.

3 Claims, 4 Drawing Sheets

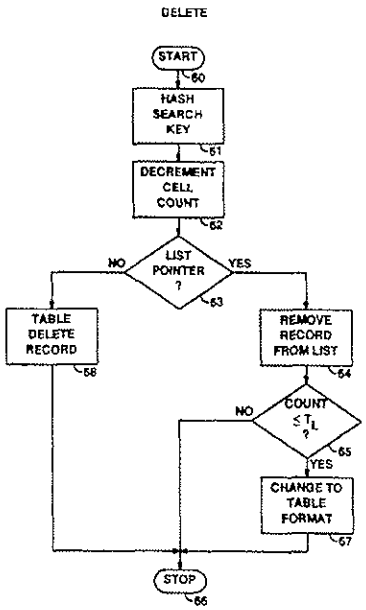


FIG. 1

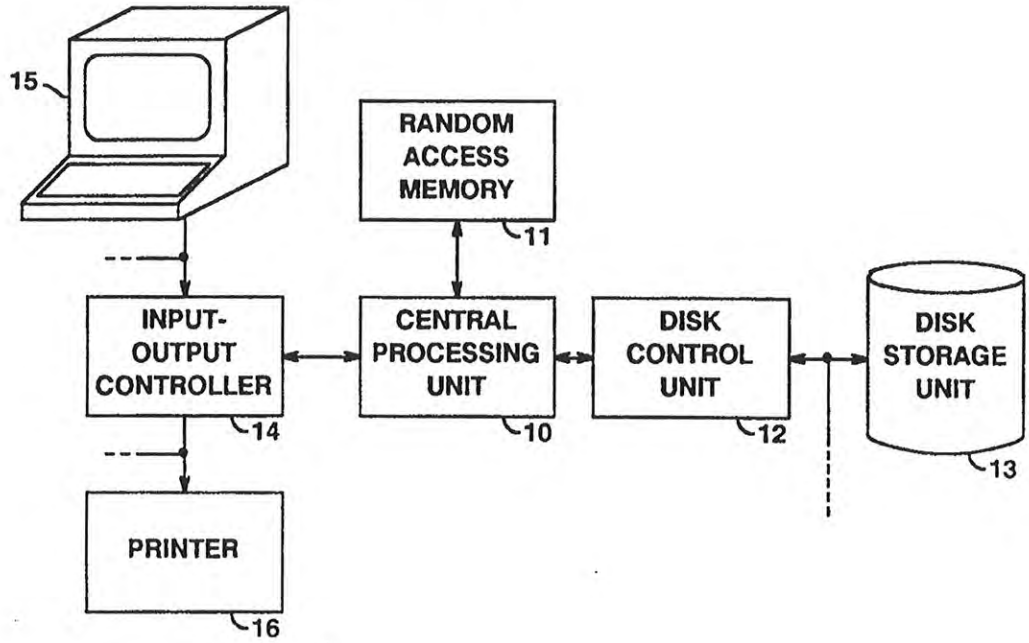

15
RANDOM ACCESS MEMORY
11
INPUT-OUTPUT CONTROLLER
14
CENTRAL PROCESSING UNIT
10
DISK CONTROL UNIT
12
DISK STORAGE UNIT
13
PRINTER
16

FIG. 2

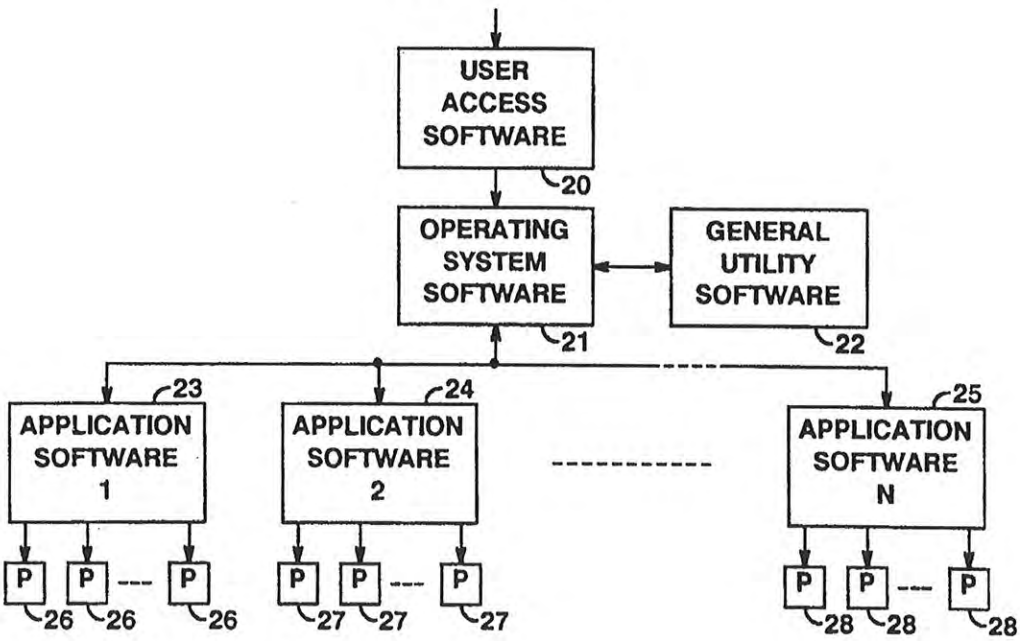

USER ACCESS SOFTWARE
20
OPERATING SYSTEM SOFTWARE
21
GENERAL UTILITY SOFTWARE
22
23
APPLICATION SOFTWARE 1
24
APPLICATION SOFTWARE 2
25
APPLICATION SOFTWARE N
P
26
27
28

FIG. 3

RETRIEVE

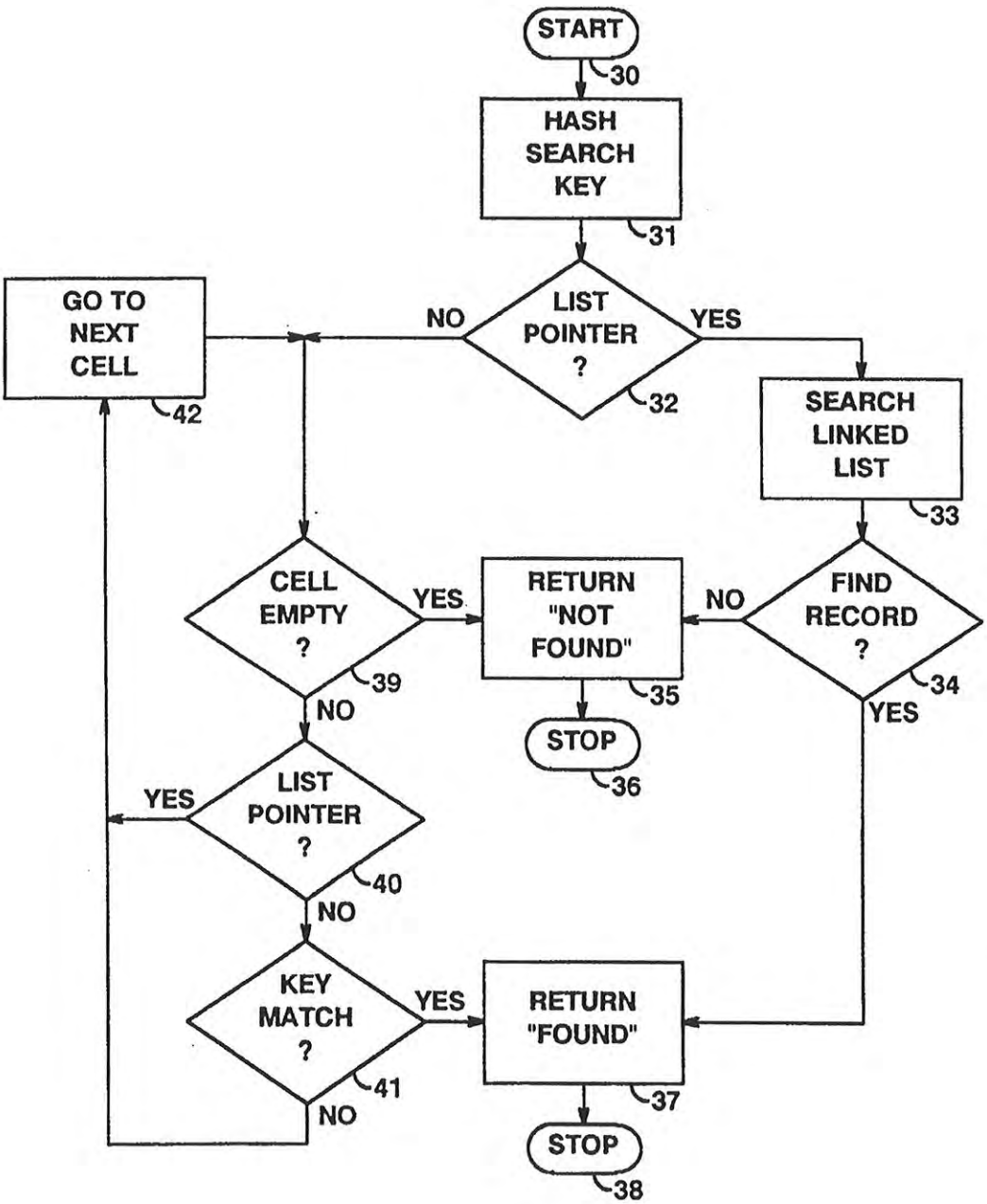


FIG. 4

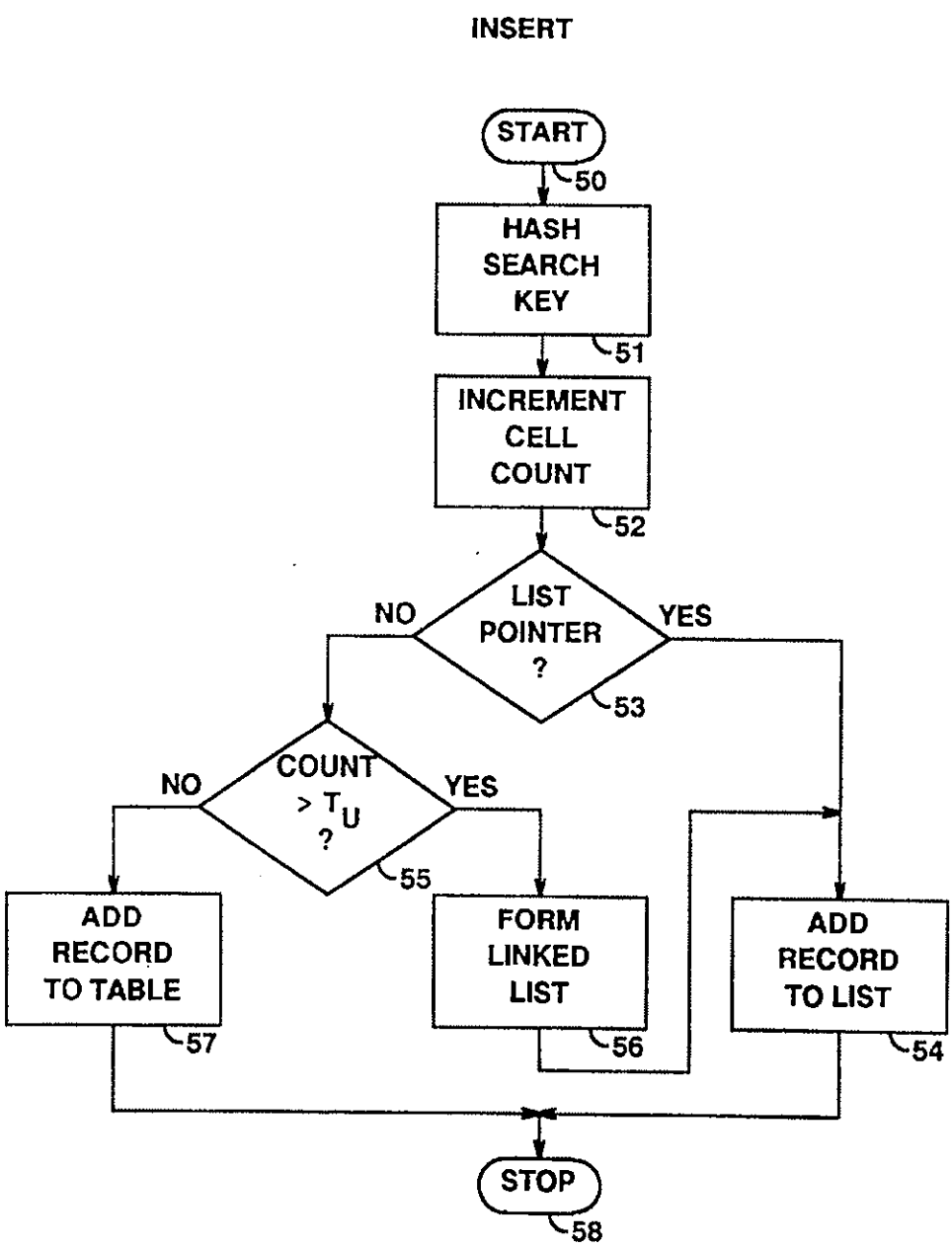
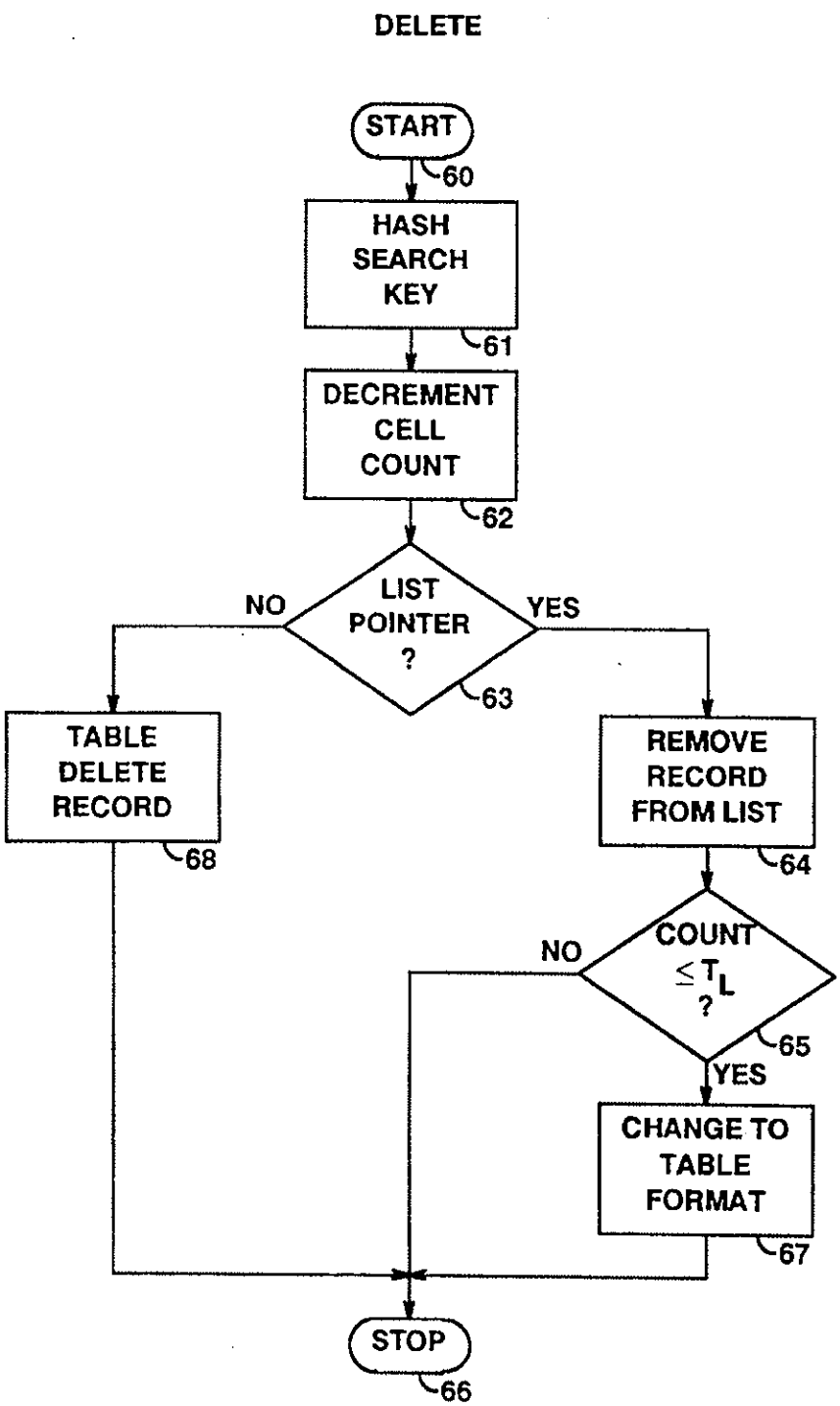

INSERT
START
50
HASH SEARCH KEY
51
INCREMENT CELL COUNT
52
LIST POINTER ?
NO
YES
53
COUNT > T_U ?
NO
YES
55
ADD RECORD TO TABLE
57
FORM LINKED LIST
56
ADD RECORD TO LIST
54
STOP
58

FIG. 5


DELETE
START
60
HASH SEARCH KEY
61
DECREMENT CELL COUNT
62
LIST POINTER ?
63
NO
YES
TABLE DELETE RECORD
68
REMOVE RECORD FROM LIST
64
COUNT ≤ T_L ?
65
NO
YES
CHANGE TO TABLE FORMAT
67
STOP
66

METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING A METHOD OF HASHING AND DIFFERENT COLLISION AVOIDANCE SCHEMES DEPENDING UPON CLUSTERING IN THE HASH TABLE

This application is a continuation of application Ser. No. 07/326,976, filed Mar. 22, 1989, now abandoned.

TECHNICAL FIELD

This invention relates to information storage and retrieval systems and, more particularly, to the dynamic reorganization of the stored information to optimize access in such systems.

BACKGROUND OF THE INVENTION

Information or data stored in a computer-controlled storage mechanism can be retrieved by searching for a particular key in the stored records. The stored record with a key matching the search key is then retrieved. Such searching techniques require repeated accesses or probes into the storage mechanism to perform key comparisons. In large storage and retrieval systems, such searching, even if augmented by efficient search algorithms such as a binary search, often requires an excessive amount of time.

Another well-known and much faster method for storing and retrieving information from computer store involves the use of so-called "hashing" techniques. These techniques are also sometimes called scatter-storage or key-transformation techniques. In a system using hashing, the key is operated upon (by a hashing function) to produce a storage address in the storage space (called the hash table). This storage address is then used to access the desired storage location directly with fewer storage accesses or probes than sequential or binary searches. Hashing techniques are described in the classic text by D. Knuth entitled The Art of Computer Programming, Volume 3, Sorting and Searching, pp. 506–549, Addison-Wesley, Reading, Mass., 1973.

Hashing functions are designed to translate the universe of keys into addresses uniformly distributed throughout the hash table. Typical hashing operations include truncation, folding, transposition and modulo arithmetic. A disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing "collisions" in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called "rehashing") must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the first empty storage location will resolve the collision. This latter technique is called linear probing. If the hash table is considered to be circular so that addresses beyond the end of the table map back to the beginning of the table, then the linear probing is done with "open addressing," i.e., with the entire hash table as overflow space in the event that a collision occurs. Deletion of records is accomplished by marking the record as "deleted" but leaving it in place, or by some deletion algorithm. One such deletion algorithm, known as Knuth's deletion algorithm, operates by recursively moving an appropriate one of the next encountered "occupied" record positions into the now "empty" (deleted) record position and marking that next record position as "empty." Iterating this procedure until the first unoccupied record position is encountered results in removal of the record to be deleted. Deletion problems of this type are discussed in considerable detail in Data Structures and Program Design, by R. L. Kruse, Prentice-Hall, Englewood Cliffs, N.J., 1984, pp. 112–126, and Data Structures with Abstract Data Types and PASCAL, by D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing, Monterey, Calif., 1985, pp. 310–336.

Another technique for resolving collisions is called external chaining. In this technique, each hash table position is able to store all records hashing to that location. More particularly, a linked list is used to store the actual records outside of the hash table. The hash table entry, then, is no more than a pointer to the head of the linked list. The linked list is itself searched sequentially when retrieving or storing a record. Deletion is accomplished by adjusting pointers to eliminate the deleted record from the linked list.

The linear probing with open addressing technique has the advantages of simplicity and minimal storage accesses, but the disadvantages of contamination due to deleted records (if records are merely marked as deleted), the overhead of the more complex deletion algorithms such as Knuth's algorithm, and the precipitous degradation of operation under high load factors. External chaining has the advantages of simple deletion algorithms, readily extendible storage size and graceful operation under high load factors. Thus, neither approach is optimum for all storage and retrieval systems.

The problem, then, is to provide the simplicity and speed of access of linear probing techniques for loads involving little or no collisions, but taking advantage of the more graceful operation of external chaining techniques for loads which cause collisions to rise above some preselected threshold.

It is also well-known that the frequency of retrieval of some records is much higher than others. If this frequency data is known ahead of time, the data can be organized in the storage system to minimize the retrieval time of the most frequently accessed records, for example, by placing such records at the initial hashing position or at the head of the chain. Unfortunately, such optimal organization of the storage system requires an a priori knowledge of the frequency of retrieval statistics. A real problem in storage and retrieval systems is the optimal organization of the storage space when no a priori knowledge is available concerning the frequency of retrieval statistics.

SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the invention, these and other problems are overcome by using dual storage organization techniques which can be selected "on the fly" while data is being stored or accessed in the storage space. In particular, the key for each new record is hashed to a particular position in the hash table. If the number of records hashing to that same position is below a preselected threshold, the collision is resolved by linear probing under open addressing. Once the number of records hashing to that same position rises above the threshold, all of the records hashing to that position are removed from the hash table and linked by external chaining, leaving a pointer to the head of the chain in the hashed position. When the number of records in the external chain drops below a threshold (not necessarily the same threshold that caused external chaining), the external chain is destroyed and the records returned to the hash table and

the records stored there using linear probing under open addressing. Any of the known record deletion techniques can be used in this dynamically combined dual storage system. Each position in the hash table therefore can contain either a record or a pointer to the head of an external chain which can be distinguished, for example, by a one bit flag.

The above system can be simplified by maintaining, in each position of the hash table, a field holding the count of the number of records heretofore hashing to that position in the hash table. This count therefore represents the length of the external chain when the threshold is exceeded.

This dynamic reorganization of the storage space of a storage and retrieval system has the decided advantage of optimizing the retrieval time of records regardless of load factors. Moreover, the higher overhead encountered with external chaining is avoided until the higher load factor (higher number of collisions) suggests that linear probing times will deteriorate substantially. The threshold loadings for switching between the two techniques are, of course, selected to optimize the overall performance of the combined system.

BRIEF DESCRIPTION OF THE DRAWING

A complete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawing, in which:

FIG. **1** shows a general block diagram of a computer system hardware arrangement in which the information storage and retrieval system of the present invention can be implemented;

FIG. **2** shows a general block diagram of a computer system software arrangement in which the information storage and retrieval system of the present invention will find use;

FIG. **3** shows a general flow chart for a record retrieval procedure in a dynamically reorganizable, combined linear probing, external chaining storage and retrieval system in accordance with the present invention;

FIG. **4** shows a general flow chart for a record insertion procedure in the dynamically reorganizable, dual storage technique storage and retrieval system in accordance with the present invention; and

FIG. **5** shows a general flow chart for a record deletion procedure in the dynamically reorganizable, dual storage technique storage and retrieval system in accordance with the present invention.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION

Referring more particularly to FIG. **1** of the drawings, there is shown a general block diagram of a computer hardware system comprising a Central Processing Unit (CPU) **10** and a Random Access Memory (RAM) unit **11**. Computer programs stored in the RAM **11** are accessed by CPU **10** and executed, one instruction at a time, by CPU **10**. Data, stored in other portions of RAM **11**, are operated upon by the program instructions accessed by CPU **10** from RAM **11**, all in accordance with well-known data processing techniques. CPU **10** may, of course, comprise multiple processors and interact with multiple memory units **11** by way of caches for data and/or instructions, all as is also well-known in the data processing art.

Central Processing Unit (CPU) **10** also controls and accesses a disk controller unit **12** which, in turn, accesses digital data stored on one or more disk storage units such as disk storage unit **13**. In normal operation, programs and data are stored on disk storage unit 13 until required by CPU **10**. At this time, such programs and data are retrieved from disk storage unit **13** in blocks and stored in RAM **11** for rapid access.

Central Processing Unit (CPU) **10** also controls an Input-Output (IO) controller **14** which, in turn, provides access to a plurality of input devices such as CRT (cathode ray tube) terminal **15**, as well as a plurality of output devices such as printer **16**. Terminal **15** provides a mechanism for a computer operator to introduce instructions and commands into the computer system of FIG. **1**, and may be supplemented with other input devices such as card and tape readers, remotely located terminals, optical readers and other types of input devices. Similarly, printer **16** provides a mechanism for displaying the results of the operation of the computer system of FIG. **1** for the computer user. Printer **16** may similarly be supplemented by line printers, cathode ray tube displays, phototypesetters, graphical plotters and other types of output devices.

The constituents of the computer system of FIG. 1 and their cooperative operation are well-known in the art and are typical of all computer systems, from small personal computers to large main frame systems. The architecture and operation of such systems are well-known and, since they form no part of the present invention, will not be further described here.

In FIG. **2** there is shown a graphical representation of a typical software architecture for a computer system such as that shown in FIG. **1**. The software of FIG. 2 comprises an access mechanism **20** which, for simple personal computers, may comprise no more than turning the system on. In larger systems, providing service to a larger number of users, login and password procedures would typically be implemented in access mechanism **20**. Once access mechanism **20** has completed the login procedure, the user is placed in the operating system environment **21**. Operating system **21** coordinates the activities of all of the hardware components of the computer system (shown in FIG. **1**) and provides a number of utility programs **22** of general use to the computer user. Utilities **22** might, for example, comprise assemblers and compilers, mathematical routines, basic file handling routines and system maintenance facilities.

The computer software system of FIG. **2** typically also includes a plurality of application programs such as application software **23, 24, . . . 25**. Application software **23–25** might, for example, comprise an editor, a spread sheet program, a graphics package, a data base manager, and so forth. Each of the application programs 23 through **25** includes or provides access to a plurality of programmed processes **26, 27, . . . 28**, respectively. It is the programmed processes **26** through **28** which actually perform the tasks necessary to carry out the purpose of the corresponding application program. In order to make effective use of these application packages, the user must be able to execute the processes **26–28** at the time, and in the sequence, necessary to accomplish the user's goals.

The present invention is concerned with information storage and retrieval systems. Such a system would form one of the application software packages 23, 24, . . . , 25 of FIG. **2**. The various processes (**26,27,28**) which implement the information storage and retrieval system

are herein disclosed as flow charts in FIGS. 3, 4 and 5, and shown as pseudocode in the APPENDIX to this specification. It is believed that the creation and execution of the computer programs necessary to carry out these processes are readily apparent to those skilled in the programming art from the present disclosure.

Many fast techniques for storing and retrieving data are known in the prior art. In situations where storage space is considered cheap relative to retrieval time, a technique called hashing is often used. In classic hashing, each record in the information storage system includes a particular field called the key, which is used as the basis for storing and retrieving the associated record. A mathematical function or map, called a hashing function, translates the key into a cell number or address in the storage space, called the hash table. Taken as a whole, a hash table is a logically contiguous, circular list of consecutively numbered, fixed-size storage units called cells each capable of storing a single data item called a record. The hashing function can be any operation on the key which results in hash table addresses more or less evenly distributed throughout the hash table. Known hashing functions include truncation, folding, transposition, modulo arithmetic, and combinations of these operations. Unfortunately, hashing functions do not always produce unique addresses in the hash table. That is, many distinct keys can map into the same cell number, producing what are called collisions. Some form of collision resolution strategy is therefore required in all hashing systems. In every instance of collision, it is necessary to find an empty storage location somewhere else to store the new record. Moreover, such alternate storage locations must be readily reachable during future probes searching for the displaced record.

Two forms of collision resolution are well-known in the prior art. The first is called open addressing. Under open addressing, whenever a collision occurs due to two different keys hashing to the same cell number, a technique called linear probing is used. Under linear probing, a sequential scanning of storage cells takes place, beginning with the next cell following the cell hashed to, and treating the hash table as circular. The record is stored in the first unoccupied cell encountered in the linear probe. Retrieval of the record is similar. The search key is hashed to the initial cell number. If the record is not found there (the keys do not match), the linear probe is used to access all successive cells until the record is found (the keys match). If an empty cell is encountered during this linear probing, the record sought is not in the data base and the process terminates as an unsuccessful search. The deletion of records under open addressing involves either merely marking the cell as deleted, or physically moving the contents of a cell to fill the deleted cell and maintain the continuity of the probe path. The preferred deletion algorithms (called "garbage collection") are disclosed in the copending applications of the present applicant, Ser. Nos. 151,638 and 151,639, both filed Feb. 2, 1988, and assigned to applicant's assignee, now issued as U.S. Pat. Nos. 4,996,663, Feb. 26,1991, and 5,121,495, Jun. 9, 1992, respectively.

A second general technique for collision resolution is called external chaining. Under external chaining, each cell in the hash table effectively stores all of the colliding records. This is accomplished by making each table entry (each cell) consist of a pointer to the head of a linked list of records. Such linked lists are formed by storing records randomly in any available storage space, but maintaining in each record a pointer to the location of the next record in the chain. When a search key is hashed to the hash table entry, the pointer located there is used to locate the first record. If the search key does not match this record, the pointer therein contained is used to locate the second record. In this way, the "chain" of records is traversed sequentially until the desired record is located or until the end of the chain is reached (no pointer to a next record). Deletion of records simply involves adjusting the pointers to bypass the delected record.

External chaining has numerous advantages over open addressing. The delection procedure is simple and does not leave records in place which must be searched over in future probes, if Knuth's deletion algorithm is not used. The number of records can exceed the size of the hash table, and can be expanded readily without changing the hashing function. Indeed, storage space for new records can be allocated dynamically as needed. Most importantly, the number of probes required to conduct searches for a particular hashed key does not rise precipitously with increases in the table load factor. In an open addressing system, as the table loading grows, the average number of probes necessary to locate a particular record also grows. At some loading level, the successful operation of the retrieval system collapses precipitously.

On the other hand, linear probing under open addressing has distinct advantages over external chaining under more moderate load factors. The addition storage for pointer fields is avoided, along with the processing overhead of following the pointer chains. If the table is implemented in virtual memory, a minimal number of page faults are incurred during record access since the portion of the hash table to be accessed occupies contiguous storage locations on one or two pages.

In accordance with the present invention, the major advantages of both techniques, open addressing and external chaining, are achieved in the same system. More particularly, these two techniques are combined in one system, and the actual storage strategy is selected dynamically, depending on the then current local load factor. Initially, all records are stored in the hash table using the open addressing with linear probing technique. When the local load factor exceeds a preselected threshold, the system shifts dynamically to the external chaining technique. That is, while inserting or deleting a record, the local load factor, as reflected in the number of records hashed to this same hash table cell, is examined. If this number exceeds a threshold, the records hashed to this cell address are reorganized, removed from the hash table itself and organized into an external chain in another part of the store. While such reorganization involves considerable overhead, the payoff comes in subsequent searches where the external chaining greatly reduces the search time. It is assumed, of course, that the frequency of retrievals greatly exceeds the frequency of insertions and deletions, an assumption which holds true for most data storage and retrieval systems. When a deletion from a linked list causes the chain length to fall below a threshold, not necessarily the same threshold that triggered chain formation, the chain is destroyed and the entries reabsorbed into the hash table.

In further accord with the present invention, the dynamic shifting between open addressing and external chaining is facilitated by maintaining a record count

field in each occupied hash table cell. This count is incremented each time a new record is hashed to this same cell, and decremented each time a record, which hashes to this same cell, is deleted from the data base. The count field is then consulted on each access for insertion or deletion, and the value in this field used to dynamically determine the collision resolution strategy to be used. Each entry in the hash table cell also advantageously includes a flag indicating whether the table entry is a record or a pointer to an external chain.

Referring then to FIG. 3, there is shown a flowchart of a retrieve algorithm for retrieving records from a data storage and retrieval system in accordance with the present invention and involving dual collision resolution schemes dynamically selected depending on load factor. In FIG. 3, starting at start box **30**, box **31** is entered where the search key is hashed using any known hashing function. The cell location resulting from the hashing operation is used to access a hash table cell. In decision box **32**, the contents of the cell is examined to determine if the cell contains a record or a pointer to an external chain. As previously noted, a one-bit flag can be reserved for this purpose. Alternatively, the length of the contents can be used to distinguish between records and pointers, or the contents examined to make this decision. If the contents of the cell is a list pointer, box **33** is entered to search the external linked list for a matching key. In decision box **34**, the records in the linked list are examined to ascertain if the keys match, and if they do, box **37** is entered to return the contents of the matching record. The process is then terminated in box **38**. If no matching record is found in the linked list, box **35** is entered to return a message that the search was unsuccessful, and the process terminated in box **36**.

Returning to decision box **32**, if the contents of the initial hash table cell is not a pointer, decision box **39** is entered to determine if the cell is empty. If the cell is empty, box **35** is entered to return an unsuccessful search message and the process terminated in box **36**. If the cell is not empty, decision box **40** is entered to again determine if the contents of the cell is a list pointer. This is necessary because of later iterations of the logic path. If the cell does contain a list pointer, box **42** is entered to advance to the next cell in the hash table. Decision box **39** is then re-entered.

If it is determined in decision box **40** that the contents of the current cell is not a list pointer, decision box **41** is entered where the search key is compared to the key in the current cell. If a match occurs, box **37** is entered to return the matching record and the process terminated in box **38**. If a match does not occur in box **41**, box **42** is entered to access the next cell in the hash table. Thus, the linear probe of the hash table continues until either an empty cell is encountered (box **39**) or a cell with a matching key is encountered (box **41**). Intervening cells containing list pointers are passed over (box **40**).

It can be seen that the retrieve process of FIG. 3 serves to locate the target record whether it is stored under the open addressing process or under the external chaining process. The retrieve process of FIG. 3 assumes that the record has previously been stored using the most efficient storage strategy. The insertion process of FIG. 4 insures that this choice is properly made.

Turning then to FIG. 4, there is shown a flowchart of a record insertion process suitable for carrying out the dual storage scheme of the present invention. In FIG. 4, starting at start box **50**, box **51** is entered where the search key of the record to be inserted is hashed. Using the hash table address produced by the hashing operation, the count field in the cell at that location is incremented by one in box **52**. Decision box **53** is then entered where it is determined whether or not the contents of that cell is a list pointer. If the contents of the cell is a list pointer, box **54** is entered to add the new record to the external chain. This is accomplished by "walking" the chain to its end. The new record is then added at the end of the chain by placing a pointer to the new record in the previously last, but now penultimate, record in the chain. The process then terminates in terminal box **58**.

Returning to decision box **53**, if the contents of the hashed cell is not a pointer, decision box **55** is entered where the cell count is compared to a numerical upper threshold T_U. If the cell count does not exceed this threshold, box **57** is entered where the new record is added to the hash table using standard linear probing techniques. The process then terminates in terminal box **58**. If the cell count does exceed the T_U threshold in decision box **55**, box **56** is entered where all of the records hashed to this same hash table address are retrieved, formed into an external linked chain and a pointer to that chain placed in the hashed cell address. Box **54** is then entered to place the new record at the end of that chain. The process then terminates in terminal box **58**. The process of forming the linked list involves no more than retrieving the hash table records (using FIG. **3**), finding a free storage location for the first record, storing the first record there and placing a pointer to that location in the hash table cell, finding another free storage location for the second record, storing the second record there and placing a pointer to that second location in the first record, and so forth. If the hash table cell originally stored a record that hashes elsewhere (from a previous probe), then that record must be relocated in the hash table to make room for the pointer, again using the open addressing technique.

In FIG. **5** there is shown a flowchart of a record deletion process. Starting at start box **60**, box **61** is entered where the search key is hashed to provide a hash table cell location. In box **62**, the cell count field at that cell location is decremented by one. Decision box **63** is then entered to determine whether or not the contents of that cell is a list pointer. If it is not, box **68** is entered to use any known table deletion algorithm to remove the record from the hash table. As previously noted, the record can merely be marked "deleted" and left in place or can by physically deleted by some algorithm such as Knuth's algorithm. The process terminates in terminal box **66**.

If it is determined in decision box **63** that the contents of the cell is a list pointer, box **64** is entered where the record to be deleted is removed from the linked list. This is easily accomplished by adjusting the pointer in the chain just before the record to be deleted to point the the record following the record to deleted. The storage space of the thus "deleted" record can then be returned to free storage space for future assignment to another record.

Following the removal of the record in box **64**, decision box **65** is entered where the decremented cell count is compared to another lower threshold T_L. If the count is not equal to or less than this T_L threshold, the process terminates in terminal box **6**. If, however, the cell count is less than or equal to the T_L threshold, box **67** is entered where the linked list is disassembled and the re-

cords added to the hash table using linear probing techniques. The process then terminates in terminal box **66**.

It can be seen that the processes of FIGS. **3**, **4** and **5** cooperate to provide a dual collision resolution hashed storage system where the form of collision resolution is determined dynamically "on the fly" depending on the local load factor at the time records are to be added or deleted from the system. Pseudo-code listings for each of these processes, together with pseudo-code for two different forms of deletion are included in the APPENDIX. The correspondence between the listings and FIGS. **3**, **4** and **5** are obvious and will not be further described here.

It should also be clear to those skilled in the art that further embodiments of the present invention may be made by those skilled in the art without departing from the teachings of the present invention.

APPENDIX

Formal Definitions

```
a. const table_size                 /* size of hash table */
b. const upper_threshold            /* used by insert for conversion to chain structure */
c. const lower_threshold            /* used by delete for conversion to table structure */
                    /* lower_threshold ≤ upper_threshold */
d. type list_element_type = record
                    record_contents: record_type;
                    next: ↑list_element_type  /* pointer to next node in linked list */
                    end
e. type table_element_type = record
                    count: integer;    /* number of stored keys that hash
                                          to this cell */
                    status: (empty, occupied, deleted);
                    case structure: (tbl, list) of  /* type of data structure currently
                                          storing records that hash here */
                    tbl: (record_contents: record_type);
                    list: (list_head: ↑list_element_type)
                    end
f. var table: array[0 .. table_size-1] of table_element_type     /* hash table */
```

Initial state of each element of *table*:

a. *count* = 0

b. *status* = *empty*

c. *structure* = *tbl*

Retrieve Algorithm

```
procedure retrieve (key: key_type);
  var i: 0 .. table_size-1;
     continue: boolean;
```

```
begin
  i := hash (key);
  if table[i].structure = list
    then search linked list pointed to by table[i].list_head
    else begin
      continue := true;
      while (table[i].status ≠ empty) and continue do
        if (table[i].structure = tbl) and (table[i].status = occupied)
          then if table[i].record_contents.key = key
                 then continue := false
                 else i := (i + 1) mod table_size
          else i := (i + 1) mod table_size;
      if table[i].status = empty
        then not found
        else found
    end    /* else begin */
end
```

Insert Algorithm

```
procedure insert (new_record: record_type);
  var i, j: 0 .. table_size-1;
    p: ↑list_element_type;     /* used for constructing chain */
  begin
    i := hash (new_record.key);
    table[i].count := table[i].count + 1;
    if table[i].structure = list
      then list_insert (table[i].list_head, new_record);
      else if table[i].count > upper_threshold
          then begin       /* convert to linked list */
            p := nil;      /* initialize for while loop */
            j := i;
            while table[j].status ≠ empty do   /* traverse sequence of records
                                                  and add to linked list */
              begin
                if (table[j].structure = tbl) and (table[j].status = occupied)
                  then if hash (table[j].record_contents.key) = i
                         then begin
                           list_insert (p, table[j].record_contents);
```

```
                knuth_delete (j)
              end
              else j := (j + 1) mod table_size
            else j := (j + 1) mod table_size
        end;   /* while */
        list_insert (p, new_record);
        if table[i].status = occupied
          then table_insert ( hash (table[i].record_contents.key),
                              table[i].record_contents)
          else table[i].status := occupied;
        table[i].structure := list;
        table[i].list_head := p
      end   /* if else if then begin (convert to linked list) */
      else table_insert (i, new_record)
end
```

Delete Algorithm

```
procedure delete (key: key_type);
  var i: 0 .. table_size-1;
    p: ↑list_element_type;        /* used for traversing chain */
    continue: boolean;
  begin
    i := hash (key);
    table[i].count := table[i].count - 1;
    if table[i].structure = list
      then begin    /* delete from linked list */
        search linked list pointed to by table[i].list_head
         and remove record whose key matches key;
        if table[i].count ≤ lower_threshold
          then begin    /* convert linked list to table resident entries */
            p := table[i].list_head;
            table[i].structure := tbl;
            knuth_delete (i);
            while p ≠ nil do
              begin
                table_insert ( hash (p↑.record_contents.key), p↑.record_contents);
                remove p↑ from linked list, dispose of element pointed to by p,
                and advance p to next list element
```

```
            end
        end    /* then begin (convert linked list to table resident entries) */
      end    /* if then begin (delete from linked list) */
      else begin    /* delete table resident entry */
        continue := true;
        while continue do
          if (table[i].structure = tbl) and (table[i].status ≠ deleted)
            then if table[i].record_contents.key = key
                   then continue := false
                   else i := (i + 1) mod table_size
            else i := (i + 1) mod table_size;
        invoke mark_deleted (i) or knuth_delete (i)
      end    /* else begin (delete table resident entry) */
  end
```

Mark Deleted Algorithm

```
procedure mark_deleted (i: 0 .. table_size-1);
  begin
    table[i].status := deleted
  endR
```

Knuth Delete Algorithm

```
procedure knuth_delete (i: 0 .. table_size - 1);
      /* Delete cell i from hash table */
  procedure recursive_delete (j, k: 0 .. table_size - 1);
        /* Delete cell k instead of cell j if required */
    begin  /* recursive_delete */
      if cell k is marked empty
        then mark cell j empty
        else if record in cell k hashes at or before position j
               then begin
                 contents(j) := contents(k);
                 recursive_delete (k, (k + 1) mod table_size)
               end /* then */
               else recursive_delete (j, (k + 1) mod table_size)
    end; /* recursive_delete */

  begin  /* knuth_delete */
```

```
  recursive_delete (i, (i + 1) mod table_size)
end /* knuth'_delete */
```

List Insert Algorithm

```
procedure list_insert (var p: ↑list_element_type; new_record: record_type);
  /* Allocate list element, put new_record in it, and link to list pointed to by p */
  var q: ↑list_element_type;
  begin
    new (q);    /* allocate list element */
    q↑.record_contents := new_record;
    q↑.next := p;
    p := q
  end
```

Table Insert Algorithm

```
procedure table_insert (i: 0 .. table_size-1; new_record: record_type);
  /* Store new_record in table at or ahead of position i */

  begin
    while table[i].status = occupied do i := (i + 1) mod table_size;
    table[i].record_contents := new_record;
    table[i].status := occupied
  end
```

What is claimed is:

1. An information storage and retrieval system for data records using a portion of each said data record for generating a hashed storage address in said system, said system comprising

storage means for storing a collision count for each set of said data records having identical hashed storage addresses,

first means responsive to said storage means for locally resolving collisions by open addressing when said collision count in said storage means is below a preselected threshold, and

second means responsive to said storage means for locally resolving collisions by external chaining when said collision count in said storage means is equal to or greater than said preselected threshold.

2. The information storage and retrieval system according to claim 1 further comprising

means for storing one of said data records at said hashed storage address when said collision count is below said preselected threshold.

3. The information storage and retrieval system according to claim 1 further comprising

means for storing a pointer to one of said data records at said at said hashed storage address when said collision count is equal to or greater than said preselected threshold.

* * * * *

Exhibit A.6

Austin Curry

From: Todd Briggs [toddbriggs@quinnemanuel.com]
Sent: Thursday, August 26, 2010 1:55 PM
To: 'Gardner, Steve'; Austin Curry
Cc: Diane Hughes; Jason Cassady; 'Williams, Danielle'; 'Korn, Russ'; 'Lee, John'; 'Bright, Christopher'; 'Whitehurst, Alan'; 'Ducca, Marissa'; Antonio Sistos; Henry Lien
Subject: RE: Bedrock - Defendants' JCCS positions

Austin,

The statements you refer to in your email relate to both the scope of the function and the overall claim limitations resulting from the specified structure. The defendants are proposing that the Court adopt the limitations in order to simplify the relevant issues for the jury.

Best Regards, Todd

From: Gardner, Steve [mailto:SGARDNER@KilpatrickStockton.com]
Sent: Wednesday, August 25, 2010 4:52 PM
To: 'Austin Curry'
Cc: Diane Hughes; Jason Cassady; Williams, Danielle; Korn, Russ; Todd Briggs; Lee, John; Bright, Christopher; Whitehurst, Alan; Ducca, Marissa
Subject: RE: Bedrock - Defendants' JCCS positions

Dear Austin, I apologize for not responding earlier and letting you know that I received your e-mail but I am in the office only briefly this week and one of us will respond as soon as we can. Thanks, Steve

Steve Gardner
Kilpatrick Stockton LLP
1001 West Fourth Street | Winston-Salem, NC 27101-2400
office 336 607 7483 | fax 336 734 2650
sgardner@kilpatrickstockton.com | My Profile

From: Austin Curry [mailto:acurry@McKoolSmith.com]
Sent: Monday, August 23, 2010 4:12 PM
To: Gardner, Steve
Cc: Diane Hughes; Jason Cassady; Williams, Danielle; Korn, Russ; Todd Briggs; Lee, John; Bright, Christopher; Whitehurst, Alan; Ducca, Marissa
Subject: Bedrock - Defendants' JCCS positions

Steve *et al.*,

I write regarding Defendants' claim construction positions for the 112(6) terms. In your chart (Dkt. No. 251-2), are all of the statements for 112(6) terms that follow "Function:" and precede "Means disclosed:" part of your proposed recited function, or is

your proposed function only what is identified in the paragraph labeled "Function:"?

As an example, see your chart on row 9 for the term "means for identifying and removing at least some [of the] expired ones of the records from the linked list [of records] when the linked list is accessed." You first recite the claim language for the function, but then you also say the following:
- "Both identification and removal of an automatically expired record occurs during the same traversal of the linked list.
- For claim 1, the phrase "when the linked list is accessed" refers to the time during which the "utilizing a search key to access the linked list" function in limitation is carried out in claim 1.
- For claim 5, the phrase "when the linked list is accessed" refers to the time during which the "utilizing a search key to access a linked list of records having the same hash address" function is carried out in claim 5.
- Removing requires, while traversing the linked list, both adjusting the pointers in the linked list to bypass the previously identified expired records and de-allocating the memory occupied by those records."

Is it Defendants' position that all of this constitutes the "function" of the claim term? If not, are you proposing that the Court adopt these as additional limitations to the term in addition to a construction that identifies the recited function and corresponding structure?

I'm not looking to get into the merits one way or another; I just need to know your position.

Thanks,
Austin Curry

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

BEDROCK COMPUTER TECHNOLOGIES LLC,	§	
	§	
Plaintiff,	§	
	§	**CASE NO. 6:09-cv-269-LED**
v.	§	
	§	**Jury Trial Demanded**
SOFTLAYER TECHNOLOGIES, INC., et al.	§	
	§	
Defendants.	§	

DECLARATION OF MARK T. JONES, PH.D

1. I, Mark Jones, declare under penalty of perjury that the following is true and correct.

2. I have been retained by McKool Smith, P.C., counsel for Bedrock Computer Technologies, LLC, as an expert on the lawsuit as captioned above.

3. I am a Professor of Electrical and Computer Engineering at Virginia Tech in Blacksburg Virginia. I graduated *summa cum laude* from Clemson University in 1986 with a B.S. in Computer Science and a minor in Computer Engineering while holding a National Merit Scholarship and the R. F. Poole Scholarship. I then graduated from Duke University in 1990 with a PhD in Computer Science while holding the Von Neumann Fellowship. A detailed record of my professional qualifications is set forth in the attached Appendix B to this Declaration, which is a curriculum vitae, including a list of publications, awards, research grants, and professional activities.

4. I understand that Bedrock Computer Technologies ("Bedrock") has sued the defendants for infringement of United States Patent No. 5,893,120 (the "'120 patent").

5. I have reviewed the '120 patent and the Patent Office file history for this patent. In addition, I have reviewed each party's Preliminary Claim Construction filings and the parties' Joint Claim Construction and Pre-Hearing Statement. In forming the opinions given below, I have considered these documents from the perspective of one of ordinary skill-in-the-art at the time of the invention.

6. I understand that the specification is to be interpreted from the point of view of a person of ordinary skill in the art. I further understand that the level of ordinary skill in the art is a function of many factors, including (1) the educational level of the inventor; (2) type of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field.

7. I find the pertinent art to lie generally in the field of computer-based information storage and retrieval systems. Considering all of the factors in the context of the technology of the '120 patent, I believe that a person of ordinary skill in the art would have a Bachelor of Science degree in computer science or computer engineering, including practical experience writing computer programs.

8. I have been asked to give my opinions regarding several of the terms in the '120 patent. In particular, I have been asked to give opinions regarding several means-plus-function claim terms. I understand that a patent may claim a means for performing a specified function without claiming the structure that performs the function.

I understand that this type of claim term is referred to as a "means-plus-function" term. I further understand that the specification must recite some structure corresponding to the claimed means. I understand that the corresponding structure of a means-plus-function limitation must be disclosed in the written description in such a manner that one skilled in the art will know and understand what structure corresponds to the means limitation.

9. The patent includes flow charts in Figures 3-7. One of ordinary skill in the art would view these flowcharts (and corresponding descriptions in the specification) as detailed descriptions of algorithms and understand how to implement them as computer software. The patent includes an Appendix with pseudocode in columns 9-14. One of ordinary skill in the art would view the pseudocode as detailed descriptions of algorithms and understand how to implement them as computer software. Both flowcharts and pseudocode are commonly used in computing to describe algorithms.

Term 3: "a record search means utilizing a search key to access the linked list" [Claim 1]

10. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "record searching utilizing a search key to access the linked list" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software instructions as illustrated in Boxes 31-36 and Boxes 39-41 of FIG. 3, or as portions of the pseudo-code of Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14), and described in

col. 5, line 57-col. 6 line 4 and col. 6 lines 15-20, or the equivalents thereof." The Defendants contend that this term is indefinite. For the reasons given below, I agree that Bedrock's construction is proper and disagree that the term is indefinite.

11. To one of ordinary skill in the art, the '120 patent contains multiple disclosures of a structure that is "a record search means utilizing a search key to access the linked list." The specification discloses that the invention can be implemented in application software packages, user access software, or operating system software that is part of a computer system as illustrated in Figures 1 and 2. [e.g., 3:53-56, 4:22-24, 4:45-48] The specification describes these types of software. [e.g., Figure 2, 4:24-44]. Further, the specification describes this software as running on a computer system such as the one illustrated in Figure 1. [e.g., 4:22-24] The specification makes it clear that the two components of the computer system that are required to run the software are a Central Processing Unit (CPU) and a Random Access Memory (RAM) unit. The specification indicates that programs are stored in RAM, they are accessed and executed by the CPU, and that data is also stored in RAM and is operated upon by the program instructions. [e.g., 3:53-61] This is consistent with the understanding of one of ordinary skill in the art.

12. Note that portion (1) of the structure is common to Bedrock's constructions for each of the disputed "means-plus-function" terms. One of ordinary skill in the art would understand that this because the specification describes the present invention as being implemented in such a structure as described above.

13. The specification discloses accessing an external chain (a linked list) using a search key, where a hash function is computed upon the search key to arrive at a hash

address and access the proper external chain. See, for example, Figure 3, boxes 31 and 32; the first two lines after the first "begin" in the "Search Table Procedure" and "Alternate Version of the Search Table Procedure" in columns 11, 12, 13, and 14; and described in col. 5:57-63. In each of these examples, one of ordinary skill in the art would understand that a design for executable software is being described, in the form of text, flowcharts, and detailed pseudocode. Hash functions were well-known to one of ordinary skill in the art. One of ordinary skill in the art would have been familiar with the term, concept, implementation, and use of a hash function, as well as the fact that multiple types of hash functions were available to practitioners. The Background of the Invention makes this clear and cites to specific pages of textbooks. [1:23-2:6] One such textbook, "The Art of Computer Programming" by Knuth is considered a classic work in computer science.[1] Another reference is a textbook that was used (in an earlier edition) when I took a data structures course early in my undergraduate curriculum (such a course was often offered at the sophomore level in a computer science curriculum). When describing (or implementing) an algorithm that uses a hash function, it was (and remains) standard practice to embody the hash function as a separate function because, for example, (a) a range of hash functions are available, (b) the module using the hash function can be described (and implemented) in a way that is independent of the particular hash function chosen, and (c) one would want the convenience of quickly substituting one hash function for another. Figure 3, as well as the "Search Table Procedure" and "Alternate Version of the Search Table Procedure," disclose loops in

[1] *American Scientist* (1999) named this one of the "100 or so Books that shaped a Century of Science." See http://www.americanscientist.org/bookshelf/pub/100-or-so-books-that-shaped-a-century-of-science

which the linked list indicated by the hash function is searched for a record that matches the search key. In sum, it is my opinion that these structures, together with the structures for the rest of the claim terms, would render the bounds of the claim understandable to an ordinary artisan.

Term 4: "the record search means including a means for identifying and removing at least some [of the] expired ones of the records from the linked list [of records] when the linked list is accessed" [Claims 1 and 5]

14. This term provides a further limitation on the record search means. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "record searching including identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software as described in Boxes 33-42 of FIG. 3, and/or as pseudo-code in the Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14) including the lines "while … /*HEART OF THE TECHNIQUE…", and/or as described in col. 5, line 63 - col. 6, line 34, or the equivalents thereof." The Defendants construe the function to be similar to Bedrock's but excluding the "record searching including" that is part of the function as described in the claim. The Defendants construe the means as "Boxes 10 and 11 of Fig 1, Boxes 38 and 42 of Fig. 3, Fig 4, pseudocode in the Search

Procedure (cols. 11-14) and Remove Procedure (cols. 13-14), and corresponding portions of the specification."[2] For the reasons given below, I agree that Bedrock's construction is proper and disagree that the Defendants' construction is proper.

15. I have already given the reasons above for why I agree with part (1) of the structure that Bedrock has put forward in its construction. Further, it appears that the Defendants agree at least with the portion of Bedrock's construction that requires a CPU and RAM.

16. Bedrock's construction properly includes in the structure the portion of the disclosed algorithms that includes the traversal of the linked list and the comparison to the search key. [e.g., Boxes 33-42 of Figure 3] Included in record searching is the identification of expired records and the removal of those records. [e.g., Boxes 38 and 42 of Figure 3]

17. To the extent that the Defendants argue that the "record search means" in claims 1 and 5 does not require a means to search for records, I disagree. My discussion of these terms makes it clear that the specification discloses a "record search means" and that this record search means includes searching for records using a search key to (a) locate the linked list and (b) to search for records while traversing the linked list. Construing the claims to read this out would be inconsistent with the disclosed algorithms in the specification as well as the language of the claims.

[2] The Defendants conclude that due to the inclusion of "the record search means" the claim is indefinite because they have concluded that this term is indefinite. I disagree for the reasons given in my discussion of Terms 3 and 12.

Term 5: "means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" [Claim 1]

18. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "utilizing the record search means, accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software which provides the insert, retrieve, or delete record capability illustrated in the flowchart of FIG. 5, FIG. 6, or FIG. 7, respectively, and/or as pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), or Delete Procedure (cols. 11 and 12), respectively, and/or described in col. 7, line 65 - col. 8, line 32, col. 8, lines 33-44, or col. 8 lines 45-59, or the equivalents thereof." For the reasons given below, I agree that Bedrock's construction is proper.

19. I have already given the reasons above for why I agree with part (1) of the structure that Bedrock has put forward in its construction. Further, it appears that the Defendants agree at least with the portion of Bedrock's construction that requires a CPU and RAM.

20. Bedrock's construction properly includes in the structure the algorithms that use the search means to perform inserting, retrieving, or deleting records. [e.g., the insert(), retrieve(), or delete() functions in columns 9-12] These algorithms make use of

the search_table() function (portions of which are identified as the record search means in Bedrock's construction) to access the linked list for the purpose of finding a record and, when the linked list is accessed, remove expired records from the linked list (as described above). Note that the search_table() function may be the one in "Search Table Procedure" in columns 11 and 12 or the one in "Alternate Version of Search Table Procedure" in columns 11-14.

21. The Defendants' construction is not proper because it suffers from several flaws.[3] Most notably, the Defendants' seem to ignore a bulk of structure from the specification as alternative structures to each other. One of ordinary skill in the art would understand that claim 1 merely requires that one of these be present, rather than all three, particularly in light of the language in claim 5.

Term 6: "means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records" [Claims 2 and 6]

22. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "dynamically determining maximum number of records for the record search means to remove in the accessed linked list of records" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a

[3] The Defendants conclude that due to the inclusion of "the record search means" the claim is indefinite because they have concluded that this term is indefinite. I disagree for the reasons given in my discussion of Terms 3 and 12.

CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software, as described in col. 6, line 56 - col. 7, line 15, that dynamically chooses among removal strategies (e.g., chooses whether to execute Search Table Procedure [cols. 11-12] or Alternate Version of Search Table Procedure [cols. 11-14]) "at the time the record search means is invoked by the caller, thus sometimes removing all expired records, at other times removing some but not all of them, and yet at other times choosing to remove none of them. Such a dynamic decision can be based on factors such as, for example, how much memory is available in the system storage pool, general system load, time of day, the number of records currently residing in the information system, and other factors both internal and external to the information storage and retrieval system itself" (col. 7, lines 1-10), or the equivalent thereof." The Defendants' argue that the term is indefinite. For the reasons given below, I agree that Bedrock's construction is proper and disagree that the term is indefinite.

23. I have already given the reasons above for why I agree with part (1) of the structure that Bedrock has put forward in its construction.

24. Bedrock cites to col. 6, line 56 – col.7, line 15 for the description of the algorithm for the executable software in part (2) of the means in its construction. The Defendants cite to the same section, presumably arguing that no algorithm is disclosed in this text. Note that one of ordinary skill in the art does not require pseudocode and/or flowcharts to teach (or understand) an algorithm, an algorithm may be described in text form as is the case in this section of the specification. The result of the function for this term is a maximum number of expired records to be removed. The algorithm for arriving at this result can be quite simple. The specification teaches several ways to find this

result. For example, the specification teaches at col. 6, line 66 – col. 7, line 4[4] that a dynamic decision among several strategies can be made regarding the number of expired records to remove (all, some, or none). It then explains that the way in which this decision can be made at col. 7 lines 4-10. The way the decision is made is to base it upon a factor(s) internal and/or external to the information storage and retrieval system. Examples of such factors are given, including available memory, system load, and the number of records.[5] One of ordinary skill in the art would understand that there is a tradeoff to be made between "shortening the linked list traversal time and speeding up the search" and "the expense of perhaps leaving some expired records in the list." [col. 6, lines 61-63] The patent discusses search times and system memory elsewhere in the specification. [e.g., col. 5, lines 41-52] One of ordinary skill in the art would have understood that the algorithm disclosed here chooses a limitation on the record deletion based on a tradeoff between using processor time to traverse more of the list (and spend time deleting records) versus (potentially) deleting more records to free up more space. One of ordinary skill in the art would have understood that the specific limitation would depend on the details of the particular information storage and retrieval system. The specific combination of factors and weights accorded to each factor would be very system dependent. A simple example of pseudocode for such an algorithm is as follows:

if (available system memory is greater than 25MB) **then** max_to_delete = 0

[4] As well as col. 7 lines 10-15, where it is explained that all records, no records, or a specific number of records may be deleted.

[5] Note that "time of day" is also included. One of ordinary skill in the art would understand that for many computer systems, the system processing load and the memory usage is highly correlated with the time of day.

In sum, it is my opinion that these structures, together with the structures for the rest of the claim terms, would render the bounds of the claim understandable to an ordinary artisan.

Term 10: "a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" [Claim 5]

25. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "using hashing to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software instructions corresponding to pseudo-code "var table: array [0 . . . table_size - 1] of list_element_pointer /* Hash table.*/" which point to records of type "list_element" in cols. 9-10 that allocates in memory an external chaining hash table, and/or as described in col. 5, lines 16-41, or the equivalents thereof." The Defendants contend that this term is indefinite. For the reasons given below, I agree that Bedrock's construction is proper and disagree that the term is indefinite.

26. I have already given the reasons above for why I agree with part (1) of the structure that Bedrock has put forward in its construction.

27. To one of ordinary skill in the art, this function identified for this term is performed by the hash table data structure pseudocode[6] identified in Bedrock's construction. The pseudocode and the description provide a specific description of a hash table data structure that uses external chaining. One of ordinary skill in the art could readily implement this algorithmic description using a programming language. In sum, it is my opinion that these structures, together with the structures for the rest of the claim terms, would render the bounds of the claim understandable to an ordinary artisan.

Term 11: "mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records" [Claim 5]

28. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "utilizing the record search means, accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software which provides the insert, retrieve, and delete record capabilities illustrated in the flowcharts of FIG. 5, FIG. 6, and FIG.7, respectively, and/or as pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), and Delete Procedure (cols. 11 and 12), respectively, and/or described in col. 7, line 65 -

[6] Or the description in col. 5, lines 16-41.

col. 8, line 32, col. 8, lines 33-44, and col. 8, lines 45-59, or the equivalents thereof." For the reasons given below, I agree that Bedrock's construction is proper.

29. I have already given the reasons above for why I agree with part (1) of the structure that Bedrock has put forward in its construction. Further, it appears that the Defendants agree at least with the portion of Bedrock's construction that requires a CPU and RAM.

30. Bedrock's construction properly includes in the structure the algorithms that use the search means to perform inserting, retrieving, and deleting records. [e.g., the insert(), retrieve(), and delete() functions in columns 9-12] These algorithms make use of the search_table() function (portions of which are identified as the record search means in Bedrock's construction) to access the linked list for the purpose of finding a record and, when the linked list is accessed, remove expired records from the linked list (as described above). Note that the search_table() function may be the one in "Search Table Procedure" in columns 11 and 12 or the one in "Alternate Version of Search Table Procedure" in columns 11-14.

Term 12: "a record search means utilizing a search key to access a linked list of records having the same hash address" [Claim 5]

31. The parties appear to agree that this claim element is subject to interpretation under 35 U.S.C. §112, ¶6. Bedrock identifies the function as "record searching utilizing a search key to access a linked list of records having the same hash address" and the corresponding structure as "(1) Portions of the application software, user access software or operating system software, as described at col. 4, lines 30-48 and

illustrated in FIG. 2, of a computer system that includes at least a CPU 10 and RAM 11, see FIG. 1 and col. 3 lines 52-56. (2) Executable software instructions as illustrated in Boxes 31-36 and Boxes 39-41 of FIG. 3, or as portions of the pseudo-code of Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14), and described in col. 5, line 57-col. 6 line 4 and col. 6 lines 15-20, or the equivalents thereof." The Defendants contend that this term is indefinite. For the reasons given below, I agree that Bedrock's construction is proper and disagree that the term is indefinite.

32. The primary difference between Term 3 (in claim 1) and this term is that the records in the linked list have the same hash address. The specification describes the hash address as the index that results from computing the hash function. [e.g., 2:58-62] The structure given in Bedrock's construction for this term is the same structure given for Term 3. Each of the elements in (2) of Bedrock's structure describes a technique in which all records in the linked list have the same address. Note that each record in the list can have a different search key, but every record in the same linked list will have the same hash address. See the discussion regarding Term 3 for further reasons why Bedrock's structure is proper. In sum, it is my opinion that these structures, together with the structures for the rest of the claim terms, would render the bounds of the claim understandable to an ordinary artisan.

I DECLARE UNDER THE PENALTY OF PERJURY UNDER THE LAWS OF THE UNITED STATES OF AMERICA THAT THE FOREGOING IS TRUE AND CORRECT TO THE BEST OF MY KNOWLEDGE.

Date: August 27, 2010

Mark T. Jones, Ph.D

EXHIBIT B

Curriculum Vitae

Mark T. Jones

Work Address:
302 Whittemore Hall
ECE Department
Virginia Tech
Blacksburg VA 24060

US Citizen
DOB 3/15/1965
(540) 231-8849
mtj@vt.edu

EDUCATION

- Ph.D., Computer Science, Duke University, May 1990
  - Von Neumann Fellowship, Duke University, 1986-1989
- B.S. *summa cum laude*, Computer Science, Clemson University, May 1986
  - R. F. Poole Scholarship, Clemson University, 1983-1986
  - National Merit Scholar, 1983

EMPLOYMENT

- Professor of Electrical and Computer Engineering, Virginia Tech, 6/07 – present
- Associate Professor of Electrical and Computer Engineering, Virginia Tech, 4/00 - present
- Assistant Professor of Electrical and Computer Engineering, Virginia Tech, 8/97 – 3/00
- Assistant Professor of Computer Science, University of Tennessee, 8/93 – 7/97
- Assistant Computer Scientist, Mathematics and Computer Science Division, Argonne National Laboratory, 6/90 – 7/93

PROFESSIONAL HONORS AND AWARDS

Research Recognition

- 2003-2004 VBI Faculty Fellow
- 1st prize and Honorable Mention at AOL-CIT Research Day 2002
- 1995 NSF CAREER Award
- 1995 & 1996 University of Tennessee (Knoxville) Science Alliance Award
- 1994 Atanasoff Best Paper Award, 3rd National Symposium on Large-Scale Structural Analysis
- 1992 Gordon Bell Prize, awarded by IEEE Computer Society
- 1992 Honorable Mention, Intel Grand Challenge Computing Award

Teaching Recognition

- *College of Engineering Dean's List*, Virginia Tech: Fall 1997, Spring 1998, Fall 2002, Fall 2002, Spring 2005

PAPERS IN REFEREED JOURNALS

- Jian Liu, T. E. Lockhart, M. Jones, and T. Martin, "Local Dynamic Stability Assessment of Motion Impaired Elderly Using Electronic Textile Pants," *IEEE Transactions on Automation Science and Engineering*, vol. 5, issue 4, pp. 696-702, October 2008.
- Madhup Chandra, Mark Jones, and Thomas Martin, "E-Textiles for Autonomous Location Awareness," *IEEE Transactions on Mobile Computing*, vol. 6, issue 4, pp. 367-380, April 2007.
- Zahi Nakad, Mark Jones, Thomas Martin, and Ravi Shenoy, "Using Electronic Textiles to Implement an Acoustic Beamforming Array: A Case Study," *Pervasive and Mobile Computing Journal*, vol. 3, issue 5, pp. 581-606, October 2007.
- Mark Jones, Zahi Nakad, Paul Plassmann, Yanhua Yi, "The Use of Configurable Computing for Computational Kernels in Scientific Simulations," *Intern. Journal of Future Generation Computer Systems*, 22 (1-2), pp. 67-79 (2006).

- J.-R. Cheng, M. T. Jones, and P. E. Plassmann, "A Portable Software Architecture for Mesh-Independent Particle Tracking Algorithms," *Journal of Parallel Algorithms and Applications*, 19 (2-3), 145-161, 2004.
- Jae H. Park, Gary Friedman and Mark Jones, "Geographical Feature Sensitive Sensor Placement," *Journal of Parallel and Distributed Computing*, volume 64, 2004, pp. 815-825.
- D. Marculescu, R. Marculescu, N. Zamora, P. Stanley-Marbell, P. K. Khosla, S. Park, S. Jayaraman, S. Jung, C. Lauterbach, W. Weber, T. Kirstein, D. Cottet, J. Grzyb, G. Tröster, M. Jones, T. Martin, Z. Nakad, "Electronic Textiles: A Platform for Pervasive Computing," *Proceedings of the IEEE*, volume 91, number 12, December 2003, pp. 1995-2018.
- Kiran Puttegowda, David I. Lehn, Jae H. Park, P. Athanas and Mark Jones, "Context Switching in a Run-Time Reconfigurable System," *Journal of Supercomputing,* Kluwer Academic Press, June 2003, pp 239-257.
- Mark Jones, Shashank Mehrotra, and Jae Hong Park, "Tasking Distributed Sensor Networks," *Journal of High Performance Computing Applications*, Vol 16, pp. 243-257, 2002.
- Eloise Coupey and Mark Jones, "A Script-Based Approach for E-Commerce Applications," *Quarterly Journal of Electronic Commerce*, to appear.
- Eloise Coupey and Mark Jones, "Decision Making in the Electronic Commerce Environment: Issues and Approaches for Tool Development," *Quarterly Journal of Electronic Commerce*, Vol 1, pp. 215-228, 2000.
- Mark Jones and Karthik Ramachandran, "Unstructured Mesh Computations on CCMs," *Advances in Engineering Software*, Vol. 31, pp. 571-580, 2000.
- Mark Jones and Paul Plassmann, "Unstructured Mesh Computations on Networks of Workstations," *Computer-Aided Civil and Infrastructure Engineering*, Vol. 15, 196-208, 2000.
- Lori Freitag, Mark Jones, and Paul Plassmann, "A Parallel Algorithm for Mesh Smoothing," *SIAM Journal on Scientific Computing*, Vol 20, pp 2023-2040, 1999.
- William Barry, Mark Jones, and Paul Plassmann, "Parallel Adaptive Mesh Refinement Techniques for Plasticity Problems," *Advances in Engineering Software*, Vol. 19, pp. 217-229, 1998.
- Mark Jones and Paul Plassmann, "Adaptive Refinement of Unstructured Finite-Element Meshes," *Journal of Finite Elements in Analysis and Design*, Vol. 25, pp. 41-60, 1997.
- Mark Jones and Paul Plassmann, "Parallel Algoritbms for Adaptive Mesh Refinement," *SIAM Journal of Scientific Computing*, Vol. 18, pp. 686-708, 1997.
- Robert Gjertsen, Mark Jones, and Paul Plassmann, "Parallel Heuristics for Improved, Balanced Graph Colorings," *Journal of Parallel and Distributed Computing*, Vol. 37, pp. 171-186, 1996.
- Mark Jones and Daniel Szyld, "Two-stage Multisplitting Methods with Overlapping Blocks," *Numerical Linear Algebra with Applications*, Vol. 3, pp. 113-124, 1996.
- Mark Jones and Paul Plassmann, "An Improved Incomplete Cholesky Factorization," *ACM Trans. on Mathematical Software*, Vol. 21, pp. 5-17, 1995.
- Mark Jones and Paul Plassmann, "Algorithm 740: Fortran Subroutines to Compute Improved Incomplete Cholesky Factorizations," *ACM Trans. on Mathematical Software*, Vol. 21, pp. 18-19, 1995.
- Mark Jones and Paul Plassmann, "Results for Parallel Unstructured Mesh Computations," *Computing Systems in Engineering*, Vol. 5, pp. 297-309, 1994.
- Mark Jones and Paul Plassmann, "Scalable Iterative Solution of Sparse Linear Systems," *Parallel Computing*, Vol. 20, pp. 753-773, 1994.
- Mark Jones and Merrell Patrick, "Factoring Indefinite Matrices on High-Performance Architectures," *SIAM Journal on Matrix Analysis and Applications*, Vol. 15, pp. 273-283, 1994.
- Mark Jones and Paul Plassmann, "Computation of Equilibrium Vortex Structures for Type-II Superconductors," *Int. J. Supercomputing Applications*, Vol.7.2, pp. 129-143, 1993.
- Mark Jones and Paul Plassmann, "A Parallel Graph Coloring Heuristic," *SIAM J. on Scientific and Statistical Computing*, Vol. 14, pp. 654-669, 1993.
- Mark Jones and Merrell Patrick, "Bunch-Kaufman Factorization for Real Symmetric Indefinite Banded Matrices, *SIAM Journal of Matrix Analysis and Applications*, Vol. 14, pp. 553-559, 1993.
- Mark Jones and Merrell Patrick, "The Lanczos Algorithm for the Generalized Symmetric Eigenproblem on Shared-Memory Architectures," *Applied Numerical Mathematics*, Vol. 12, pp. 377-389, 1993.

- Daniel Szyld and Mark Jones, "Two-stage and Multi-splitting Methods for the Parallel Solution of Linear Systems," *SIAM Journal of Matrix Analysis and Applications*, Vol.13, pp. 671-679, 1992.

REFEREED PAPERS IN CONFERENCE PROCEEDINGS

- M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: Using FPGA Configuration Circuitry to Emulate a Network-on-Chip", *Field Programmable Logic and Applications*, September 2008, pp. 257-262.
- M. Jones, T. Martin, and B. Sawyer, "An Architecture for Smart Textiles", Third International Conference on Body Area Networks, March 2008, to appear.
- David Graumann, Giuseppe Raffa, Meghan Quirk, Braden Sawyer, Justin Chong, Mark Jones, Thomas Martin, "Large Surface Area Electronic Textiles for Ubiquitous Computing: A Systems Approach", *MobiQuitous '07*, August 2007, pp. 1-8.
- George Eichinger, Tom Martin, and Mark Jones, "From Circuit to Sewing in One Click," *ISWC 2007*.
- J. Edmison, D. Lehn, M. Jones, and T. Martin, An E-Textile Based Automatic Activity Diary for Medical Annotation and Analysis, *2006 Workshop on Body Sensor Networks*, April 2006, pp. 131-134.
- C. Einsmann, M. Quirk, B. Muzal, B. Venkatramani, T. Martin, and M. Jones, "Modeling a Wearable Full-body Motion Capture System," *Proceedings of the 2005 IEEE International Symposium on Wearable Computers (ISWC)*, October 2005, pp. 144-151.
- J. Edmison, D. Lehn, M. Jones, and T. Martin, "Users' Perceptions of an Automatic Activity Diary for Medical Annotation and Analysis," *Proceedings of the 2005 IEEE International Symposium on Wearable Computers (ISWC)*, October 2005, pp. 192-193.
- M. Chandra, M. Jones, and T. Martin, "E-Textiles for Autonomous Location Awareness," *Proceedings of the 2004 International Symposium on Wearable Computers*, Arlington, VA, Oct. 31-Nov. 3, 2004, pp. 48-55.
- Zahi Nakad, Mark Jones, and Tom Martin, "Fault Tolerant Networks for Electronic Textiles," *CIC 2004,* Las Vegas, June 2004, pp. 100-106.
- Zafer Gurdal, Tom Hartka, Mark Jones, and Sun Wook Kim, "A Reconfigurable Approach to Structural Engineering Design Computations," *ERSA 2004*, Las Vegas, June 2004.
- Jones, Mark T., and Eloise Coupey, "An Agent-based Simulation Prototype for Evaluating Health Behavior Interventions," *METMBS 2004*, Las Vegas, June 2004.
- Thomas Martin, Mark Jones, Joshua Edmison, Tanwir Sheikh, and Zahi Nakad, "Modeling and Simulating E-Textile Applications *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems*, June 11-13, 2004, pp. 10-19.
- Lehn, D., C. Neely, K. Schoonover, T. Martin, and M. Jones, "e-TAGS: e-Textile Attached Gadgets." *Communication Networks and Distributed Systems Modeling and Simulation Conference*, January 2004.
- M. Abdalla, S. W. Kim, Z. Gürdal, and M. T. Jones, "Multigrid Accelerated Cellular Automata for Design Optimization of Continuum Structures: A 1-D Implementation", 45th *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, April 2004.
- Zahi Nakad, Mark Jones, and Tom Martin, "Communication in Electronic Textile Systems," *2003 International Conference on Communications in Computing (CIC 2003)*, pp. 37-43.
- Mark Jones, Paul Plassmann, Zahi Nakad, and Yanhua Yi, "The Use of Configurable Computing in Scientific Simulations," *2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '03)*, pp. 520-524.
- Mark Jones, Mukta Nandwani, Jae Park, Paul Plassmann, and Yanhua Yi, "Examining the Communication Requirements of Remote Scientific Visualization," *2003 International Conference on Communications in Computing (CIC 2003)*, pp. 31-36.
- T. Martin, M. Jones, J. Edmison, and R. Shenoy, "Towards a Design Framework for Wearable Electronic Textiles," *International Society for Wearable Computers 2003*, pp. 190-199.
- M. Jones., T. Martin, Z. Nakad, R. Shenoy, T. Sheikh, D. Lehn, and J. Edmison, "Analyzing the Use of E-textiles to Improve Application Performance," *IEEE Vehicular Technology Conference 2003, Symposium on Wireless Ad hoc, Sensor, and Wearable Networks.*

- Mark Jones, Tom Martin, and Zahi Nakad, "A Service Backplane for e-Textiles," *MAMSET 2002: Proc. Workshop on Modeling, Analysis and Middleware Support for Electronic Textiles*, 6 October 2002, pp. 15-22.
- J. Edmison, M. Jones, Z. Nakad and T. Martin "Using piezoelectric materials for wearable electronic textiles," *Wearable Computers, 2002. (ISWC 2002). Proceedings. Sixth International Symposium on* pp. 41-48, 2002.
- Mark Jones, Lucas Scharf, Jon Scott, Christian Twaddle, Matthew Yaconis, Kuan Yao, Peter Athanas, and Brian Schott, "Implementing an API for Distributed Adaptive Computing Systems," *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, April, 1999).
- Jason Hess, David Lee, Scott Harper, Peter Athanas, and Mark Jones, "Implementation of a Prototype Reconfigurable Router," *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, April, 1999, to appear 7 pages).
- David Lee, Mark Jones, Scott Midkiff, and Peter Athanas, "Towards Active Hardware," Lecture Notes in Computer Science 1653, Springer-Verlag, pp. 180-187, 1999.
- Lori Freitag, Mark Jones and Paul Plassmann, "The Scalability of Mesh Improvement Algorithms," *The IMA Volumes in Mathematics and its Applications*, Vol. 105, pp. 185-211, 1998.
- Lori Freitag, Mark Jones, and Paul Plassmann, "An Efficient Parallel Algorithm for Mesh Smoothing," *Proceedings of the 4th International Meshing Roundtable*, Albuquerque, NM, pp. 47-58, 1995.
- Mark Jones and Paul Plassmann, "The efficient parallel iterative solution of large sparse linear systems," *The IMA Volumes in Mathematics and its Applications*, Vol. 56, pp. 229-245, 1993.
- Mark Jones and Paul Plassmann, "Solution of Large, Sparse Systems of Linear Equations in Massively Parallel Applications," *Supercomputing '92 Proceedings*, IEEE Computer Society, pp. 551-560, 1992.
- Mark Jones, Merrell Patrick, and Robert Voigt, "A Language Comparison for Scientific Computing on MIMD Architectures," *Proceedings of the IFIP Working Conference: Aspects of Computation on Asynchronous Parallel Processors*, M. H. Wright (Editor), Elsevier Science Publishers B. V. (North-Holland), IFIP, pp. 55-67, 1989.

PUBLISHED PAPERS IN CONFERENCE PROCEEDINGS

- J. Edmison, M. Jones, T. Lockhart, and T. Martin, "An E-Textile System for Motion Analysis," Wearable eHealth Systems for Personalised Health Management: State of the Art and Future Challenges, Studies in Health Technology and Informatics, vol. 108, August 2004, pp. 292-301
- Eloise Coupey and Mark Jones, Developing Dynamic Decision Support: Opportunities, Issues and Approaches, Twenty-Third Annual International Computer Software and Applications Conference, October 1999.
- Mark Jones, Michael Langston, and Padma Raghavan, "Tools for Mapping Applications to CCMs," *Proceedings of SPIE: Configurable Computing: Technology and Applications*, Boston, MA, SPIE, pp. 72-80, Nov. 1998.
- Lori Freitag, Mark Jones, and Paul Plassmann, "Mesh Component Design and Implementation within SUMAA3d," *Proceedings of SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, SIAM Publications, to appear (page number/total unknown until typesetting).
- Lori Freitag, Mark Jones, and Paul Plassmann, "A Parallel Algorithm for Mesh Smoothing," *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 1997.
- Lori Freitag, Mark Jones, and Paul Plassmann, "Parallel Adaptive Mesh Refinement with the SUMAA3d Project," *Proceedings of the ICASE/LaRC Workshop on Adaptive Grid Methods*, 1995.
- Lori Freitag, Mark Jones, and Paul Plassmann, "Parallel Algorithms for Unstructured Mesh Computation," *Proceedings of Fifth SIAM Applied Linear Algebra Conference*, SIAM Publications, pp. 123-127, 1994.

- Mark Jones and Paul Plassmann, "Parallel Algorithms for the Adaptive Refinement and Partitioning of Unstructured Meshes," *Proceedings of the Scalable High-Performance Computing Conference*, IEEE, Ed. Dongarra and Walker, pp. 478-485, 1994.
- Lori Freitag, Mark Jones, and Paul Plassmann, "New Techniques for Parallel Simulation of High-Temperature Superconductors," *Proceedings of the Scalable High-Performance Computing Conference*, IEEE, Ed. Dongarra and Walker, pp. 726-733, 1994.
- Lori Freitag, Mark Jones, and Paul Plassmann , "New Advances in the Modeling of High-Temperature Superconductors," *Proceedings of the 1994 International Simulation Conference – Grand Challenges in Computer Simulation*, The Society for Computer Simulation, pp. 208-213, 1994.
- Mark Jones and Paul Plassmann, "Software for the Generalized Eigenproblem on Distributed Memory Architectures," *Proceedings of the Lanczos Centenary Conference*, Ed. Chu, et. al., pp. 322-325, 1994.
- Lori Freitag, J. Garner, Mark Jones, Paul Plassmann, "Recent Computational Results on the Equilibrium Vortex Configurations on Type-II Superconductors," *Proceedings of the Second DELTA Applications Workshop*, Ed. P. Messina, pp. 93-98, March 1993.
- Tom Canfield, Mark Jones, Paul Plassmann, and Michael Tang, "Modeling Piezoelectric Crystals on the Intel DELTA," *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pp. 156-159, 1993.
- Mark Jones and Paul Plassmann, "Recent Results in the Modeling of Type-II Superconductors on Massively Parallel Computers," *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pp. 147-151, 1993.
- Mark Jones and Paul Plassmann, "Parallel Solution of Unstructured, Sparse Systems of Linear Equations," *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pp. 471-475, 1993.
- Tom Canfield, Mark Jones, Paul Plassmann, and Michael Tang, "Thermal Effects on the Frequency Response of Piezoelectric Crystals," *New Methods in Transient Analysis*, Eds. P. Smolinksi, W. K. Liu, G. Hulbert, and K. Tamma, PVP-Vol. 246 and AMD-Vol. 143, pp. 103-108. ASME, New York, 1992.
- Mark Jones and Paul Plassmann, "The Effect of Many Color Orderings on the Convergence of Iterative Methods," *Proceedings of the Copper Mountain Conference on Iterative Methods*, April 1992.
- Mark Jones and Paul Plassmann, "Modeling piezoelectric crystals on the Intel DELTA," *Proceedings of the Intel DELTA Applications Workshop*, Cal Tech, pp. 97-105, 1992.

GRANTS/CONTRACTS

(Principal investigator responsibilities on the following grants, percentage of responsibility is listed.)

- DARPA, "RECE," With BBN, Responsibility 50%, $30,000 , 4/16/09-10/16/09.
- NSF, "Investigating a Novel Embedded Processor Architecture for Electonic Textiles in Wearable and Pervasive Computing," Responsibility 50%, $220,000, 12/1/08-11/30/10.
- NSF, "Fundamental Algorithms to Enable the Simulation of Multi-Scale Biological Systems," Responsibility 50%, $200,000, 9/1/07-8/31/10.
- AFRL, "Phase II: Amorphous Soft-Core Processor for Hardware Anti-Tamper," Responsibility 33%, ~$230,000, 11/1/06-10/31/08.
- Intel, "E-Textile Rug for Gait Analysis, People Tracking and Emergency Directions," Responsibility 50%, $15,000, 6/1/06-12/31/06.
- Carilion, "Quantitative Measurement and Modeling of Early Events in Influenza Pathogenesis," Responsibility 20%, $20,000, 7/1/05-6/30/06.
- AFRL, "Phase I: Wearable Computer for Enhanced Situation Awareness," Responsibility 33%, $30,000, 5/1/06-1/1/07.
- AFRL, "Phase I: Amorphous Soft-Core Processor for Hardware Anti-Tamper," Responsibility 33%, $35,000, 1/1/06-11/1/06.
- MDA, "Phase I: Secure Software Platform for Real-Time Software Anti-Tamper," Responsibility 33%, $29,994, 6/1/06-11/1/06.

- *National Science Foundation* (NSF), "CRI", (e-textiles equipment) Responsibility 50%, ~$80,000, 8/05-7/07.
- *Harris Corporation,* "Exploiting the Reconfigurability of a Software-Defined Radio Platform," $161,070, Responsibility 50%, 9/1/05-6/30/06.
- *DARPA,* "Phase I: Technology for Trusted Circuits," $31,000, 3/1/05-9/15/05, Responsibility, 33%.
- *Harris Corporation,* "Partial Reconfiguration Support for the Harris Programmable Modem Platform," $73,448, Responsibility 50%, 12/01/04-6/30/05.
- *NSF,* "Phase I: An Electronic Textile System for Gait Analysis", $33,000, 1/1/2005-6/30/2005, Responsibility 50%.
- *Office of Naval Research* (ONR). "AWINN." $452,300, 12/20/04-7/31/06, Responsibility 50%.
- *Office of Secretary of Defense,* "Phase II: Reconfigurable Processor Technology for Software Protection," SBIR Subcontract with Luna Innovations, Responsibility 33%, $237,581, 6/04-5/06.
- *NSF*, REU Supplement to ITR: Tailor-Made: Design of e-Textile Architectures for Wearable Computing, NSF, $12,000, 6/1/2005 – 8/31/2005, 50%.
- *Office of Secretary of Defense,* "Phase I: Reconfigurable Processor Technology for Software Protection," SBIR Subcontract with Luna Innovations, Responsibility 50%, $33,000, 8/03-2/04.
- *National Science Foundation* (NSF), "ITR: Tailor-Made: Design of e-Textile Architectures for Wearable Computing", Responsibility 50%, $399,000, 8/02-8/06.
- *NSF,* REU Supplement to above grant, 50%, $10,000.
- *NSF,* REU Supplement to above grant, 50%, $6,000.
- NSF, "A Toolbox of Scalable Algorithms and Software for Advanced Scientific Computing Applications", Responsibility 50%, $300,000, 4/03-4/06.
- *Defense Advanced Research Projects Agency* (DARPA). Computational Fabrics, $217,000, 3/01-12/02. Responsibility 85%. Subcontract from ISI/University of Southern California.
- *Office of Naval Research* (ONR). Secure Configurable Radio. $751,000, 4/00-4/04, Responsibility 50%.
- NSF, SUCCEED. Responsibility 100%. $33,000, 9/1/01-8/31/02.
- *National Science Foundation (NSF).* Amount: $145,023, 10/99-9/02. Responsibility 100%. Subcontract from Penn State. Research into unstructured mesh algorithms.
- *DARPA.* Amount $411,625, 6/99-6/02. Responsibility 50%. Subcontract from ISI/University of Southern California. Development of algorithms and software for the control of distributed, dynamic sensor networks.
- *DARPA.* Amount $160,000, 10/00-10/01. Responsibility 50%. Subcontact with USC/ISI. Development of algorithms and runtime software for systems of configurable computers.
- *Department of Energy.* Amount: $145,000, 12/98-11/01. Responsibility 100%. Subcontract from Pennsylvania State University. Development of algorithms and software for massively parallel architectures.
- *DARPA.* Amount: $345,574, 9/99-9/01. Responsibility 50%. Subcontract from Xilinx. Development of Java-based tools and applications for fast, flexible run-time reconfiguration of configurable computing devices.
- *DARPA.* Amount: $250,000, 6/99-6/01. Responsibility 50%. Subcontract from Lockheed-Martin/Sanders. Development of an application programming interface, algorithms, and applications for run-time reconfiguration on novel adaptive computing architectures.
- Lockheed-Martin/Sanders, $125,000, 5/00-4/01, Responsibility 50%. A Wireless Link for Remote Telemetry of Compact Airborne System.
- *DARPA.* Amount $169,792, 11/99-10/00. Responsibility 50%. Subcontract from ISI/University of Southern California. Development of algorithms and runtime software for systems of configurable computers.
- *National Security Agency (NSA).* Amount $75,000, 9/99-7/00. Responsibility 50%. Software for the translation of JBits FPGA programs to EDIF programs.
- *DARPA.* Amount $162,886, 9/98-10/99. Responsibility 50%. Subcontract from ISI/University of Southern California. Construction of an API & applications for the control of distributed systems of configurable computing nodes.
- *NSF.* Amount: $125,881, 7/96-7/99. Responsibility 100%. Career Award: Parallel algorithms and software for unstructured mesh computations.

- *Air Force Research Laboratory*. Amount: $97,425, 11/97-5/99. Responsibility 50%. Research and development of an Internet-based, interactive, decision making program.
- *NSF*. Amount: $85,000, 12/97-12/98. Responsibility 20%. Equipment funding for a 16-node cluster of configurable computers (Tower of Power).
- *NSF*. Amount: $318,807, 7/95-7/98. Responsibility 50%. Scientific applications in a distributed computing environment.
- *NSF*. Amount, $100,000, 1/96-12/96. Responsibility 20%. Equipment funding for an ATM-based cluster of workstations.
- *NSF*. Amount, $10,000, 1996. Responsibility 100%. Funding to support a workshop organized at Argonne National Laboratory.

SOFTWARE

- *SLAAC/ACS API (4/99):* Software for the control of complex systems of configurable computing nodes and high-speed networks. This code is being distributed to the ACS community. Developed in Configurable Computing Laboratory with Peter Athanas.
- *BlockSolve (4/93 & 1/96):* Software for solving large sparse linear systems on distributed memory architectures. This code has been distributed via netlib and ANL ftp and has been accessed by several hundred people as of 4/4/99. Developed at Argonne National Laboratory with Paul Plassmann.
- *LANZ (10/90):* Software for solving the generalized eigenproblem on shared memory architectures. This code has been distributed via netlib and NASA and has been accessed by over 10,000 people as of 4/4/99. Developed at Duke University and Argonne National Laboratory with Merrell Patrick.

TEACHING RESPONSIBILITIES

- Computer Programming
  ECE 2574, *Data Structures*
  ECE 1574, *Engineering Problem Solving with C++*
- Computer Simulation and Modeling
  CS 371, Numerical Analysis
  CS 594, Computational Modeling
- Computer Network, Architecture, and Organization
  ECE 2504, *Introduction to Computer Engineering*
  CS 594, Internetworking with TCP/IP
  ECE 4504, *Computer Organization*
  ECE 4534, *Design of Embedded Systems*
  ECE 5504 (formerly ECE 5515), *Computer and Network Architectures*
  ECE Special Studies: *Configurable Computing*, E-Textiles
  ECE 5984, *Java-Based Configurable Computing*
  ECE 6504, *Applications of Parallel and Distributed Computing*, TV course.
  CS 530, *Computer Systems Organization*
- CEO of DISC, a student-run virtual corporation, for Fall 1998 and Spring 1999.

COURSE, CURRICULUM, AND PROGRAM DEVELOPMENT

Spring 2005 (Virginia Tech): Redesigned the senior level embedded systems design course to focus on modern embedded processors and systems and incorporated a capstone design experience in the course. This course has been incorporated into the curriculum as a required course for computer engineering majors. I have taught the course three semesters, achieving a rating of 3.9/4.0 during the Spring 2006 semester.

Fall 2005 (Virginia Tech): Revitalized the freshman programming course for computer and electrical engineering majors. The new course focused on the design and development of programs in the context of embedded systems. While the course addressed the basics of

object-oriented programming, significant hands-on experiences with an embedded systems platform were incorporated into laboratories and programming projects. In particular, there was a strong emphasis on designing applications that interact with sensors and actuators. The new course was well-received by the students, with a significant improvement in the instructor evaluation over past semesters.

Fall 2001 (Virginia Tech): Designed (and had approved) a course on computational fabrics (e-Textiles). This course focuses on research issues in computational fabrics as well as the background necessary to pursue research in the area. As part of the course, students will create a functioning computational fabric application.

Fall 2001 (Virginia Tech): Designed (and had approved) a course on the design of Internet-based decision support systems. The goal of this course is an object-oriented design and implementation process for DSS in which the students actively participate in the process. Students learn the basics of DSS as well as the software engineering process.

Fall 2000 (Virginia Tech): All freshman wishing to enter the Computer Engineering and Electrical Engineering majors at Virginia Tech must take an introductory object-oriented programming course. Analysis showed that students in this course were not learning to program adequately and had a high percentage of honor code violations. I led an effort to completely revamp the course to address these concerns; modifications included interactive labs and active learning techniques as well as a redesign of the curriculum. Analysis this semester indicates that the students who progressed on in the computer engineering major are now proficient programmers and the number of honor code violations was reduced by over an order of magnitude.

Fall 1999 (Virginia Tech): Developed a new course based on the configurable computing course that I taught in Fall 1998. This new course is refocused to use Java-based tools for developing configurable computing applications. The short project has been changed to be implemented using Java-based development tools and include a short project report. The large application project has also been changed to Java-based development tools. Also, in collaboration with David Lee of 3Com, many of the large application projects in the class will focus on designing and implementing configurable, high-speed network routers.

Spring 1999 (Virginia Tech): Developed a graduate level course on parallel & distributed algorithms based on the textbook by Kumar, et.al. This was a TV course, so I developed presentation slides suitable for that medium. I also developed several programming exercises for parallel implementation on three different parallel computing platforms: a network of workstations, the IBM SP-2, and the Silicon Graphics "Crunch" computer.

Fall 1998 (Virginia Tech): Developed an upper level undergraduate/graduate course on configurable computing. The lecture materials were drawn from research papers and materials from faculty at other institutions. The students were required to complete a small project on configurable computing hardware, followed by a significant application project on configurable hardware.

Spring 1995 (University of Tennessee): Developed a graduate level course on internetworking based on the *Internetworking with TCP/IP* textbooks by Comer. I developed laboratories and lecture notes for the course.

Spring 1994 (University of Tennessee): Developed a graduate level course on scientific computing. As part of the development process, I created lecture materials, reading materials, and laboratories.

PROFESSIONAL SERVICE

- Technical Program Committee: MAMSET 2002, VTC 2003, RAW 2003/2004.
- Reviewer and/or panel participant for various journals, conferences, and agencies, including *SIAM J. on Scientific Computing, SIAM J. on Matrix Anal. & Appl., SPDP, Supercomputing, Journal of Parallel & Distributed Computing, CONPAR 94, HICSS, Int. J. of Supercomputing Applications, ICPP, IPPS, IEEE Tran. on Parallel & Distributed Systems, ACM Trans. on Math. Software, Symp. on Large-Scale Structural Analysis, Applied Numerical Mathematics*, Department of Energy, NIOSH, and the National Science Foundation.

- *Department*: Curriculum Committee (chair and member), Computer Area Committee, Resource Committee, Undergraduate AdCom Committee, Computer Systems Area Committee (chair and member), Social Committee, Web Committee (chair)
- *Centers:* Director for Virginia Tech Information Systems Center (VISC) Spring 1999-Fall 2000