Non-Numeric Key Processing for File Indexing Using Hashing Approach

Harihodin Selamat

Institut Sains Komputer
University Technology Malaysia

Abstract

This paper describes the processing tehcnique of non-numeric record key for the purpose of storage and retrieval. Hashing approach is employed in the design and experiments to facilitate the addressing machanism. Link-list approach is adopted to handle synonyms. Basic file organisations are discussed very briefly in order to establish a common basis of terminology and concepts.

Abstrak

Kertas ini menerangkan suatu teknik pemprosesan kunci rekod yang terdiri daripada alpanumerik untuk tujuan penyimpanan dan mendapatkan kembali rekod. Pendekatan kaedah cincangan telah di gunakan dalam rekabentuk dan experimen untuk melengkapkan makanisma pengalamatan. Pendekatan sinomi berpaut juga digunakan untuk mengendalikan sinomi. Perbandingan organisasi fail asas juga dibincangkan secara ringkas untuk mengujudkan keseragaman terminologi dan konsep

***Keywords** : Hashing, Addressing Technique, Non-numeric Key, File, Synonyms, Hash Table, Linked-list*

1. File Organisations

The main classes of file organisation in common usage are Random, Sequential, Indexed and Chained, while the basic accessing techniques are Random and Sequential. [Deen SM, 1977] [Martin J, 1975]. A File records have, almost without exception, a unique identifying field known as a 'key'. As the name implies, randomly organised files have records recorded in a sequence which has no regard for the value of the key field. Sequential files are sometimes called 'sorted' files. For such files, records are stored in ascending-key sequence.

Indexed-sequentiall files require the use of two or more sub-files which are described as follows:

i. The data file, which contains the sorted data records.
ii. One or more index files, which are used to indicate the general location of the storage device for each range of keys.

In practice, a hierarchy of indexes is used; the highest commonly referred to being resident in 'memory'. Indexed-sequential file organisation enables the file to be used sequentially or directly.

Chained files may be in random or partially sorted sequence. Records may contain one or more pointers to other records. such files are usually indexed in some way as a mechanism for finding the 'Header record' in a particular chained sequence (or 'list'). 'Header record' in this paper refers to the first record in the chained sequence. This technique is heavily employed in 'data base' files.

The files structure employed determine the degree of usage of fixed-length or variable-length recordss. Random files tend to use fixed-length records. Indexed-sequential files are usually fixed length. Some file management software supports the use of variable-length records but this would need a complex file managemant routine and put a heavy requirement on overflow areas, especially in high-activity files [J. Martin, 1975], [D. Kroenke, 1977] and [S.M. Deen, 1977].

File design involves synthesizing the collection and association of data to satisfy the information storage, retrieval and reporting requirements of users. The file design is actually an interactive process. It has three major phases which includes logical file design, physical file design, and file creation and operations. Various steps comprise each of these phases. Figure A illustrates the file design process.

2. File Manager Design And Implementation

The case study taken for the purpose of this experiment is the General Medical Practice Information System [Harihodin S., 1983]. The first phase deals with the systems analysis to identify the requirements of the proposed system. The result of the analysis is at a conceptualisation of the file, independent of computer aspects. This phase consisted of transforming the conceptual file to an equivalent file using the particular data model for computer definition, storage and processing. It has been identified that the proposed key composed of two initials (XX) followed by the date of birth which has the format day, month and year (DDMMYY), form the primary access tool to the required patient record in the file.

The second phase is the logical design phase. The easiest way to structure the file is by means of a 'singular' file. This is because most of the data items in a required record are being referred to, or changed after a patient's consultation. This provides a faster access to all data items rather than breaking the records into smaller records, which could be designed by means of hierarchic structure and which may be a two-level tree structure.

Each data item in a record in given a symbolic name for use in the program to refer to a particular field. Figure B illustrates the data structure of the patient record, coded in UCSD Pascal Language. The collection of these records form a patient file. The prime retrieval tool to access any required record is a code made up of the first two initials and followed by date of birth. Hence the code is an array of 8 characters in the format (XX999999). It is obviously not possible to access the record directly from the file using the code because the nature of the programming language in handling external file [K.L.

Bowles, 1980] requires a record number which corresponds to the code in the file. Therefore, an index has to be maintained to serve as an access path to the file. This involves the transformation of the code into a number; that is a positive integer value. Therefore transformation will enable direct access to any desired location in the index to obtain the corresponding record key. This record key is used to to access the required record directly from the file.

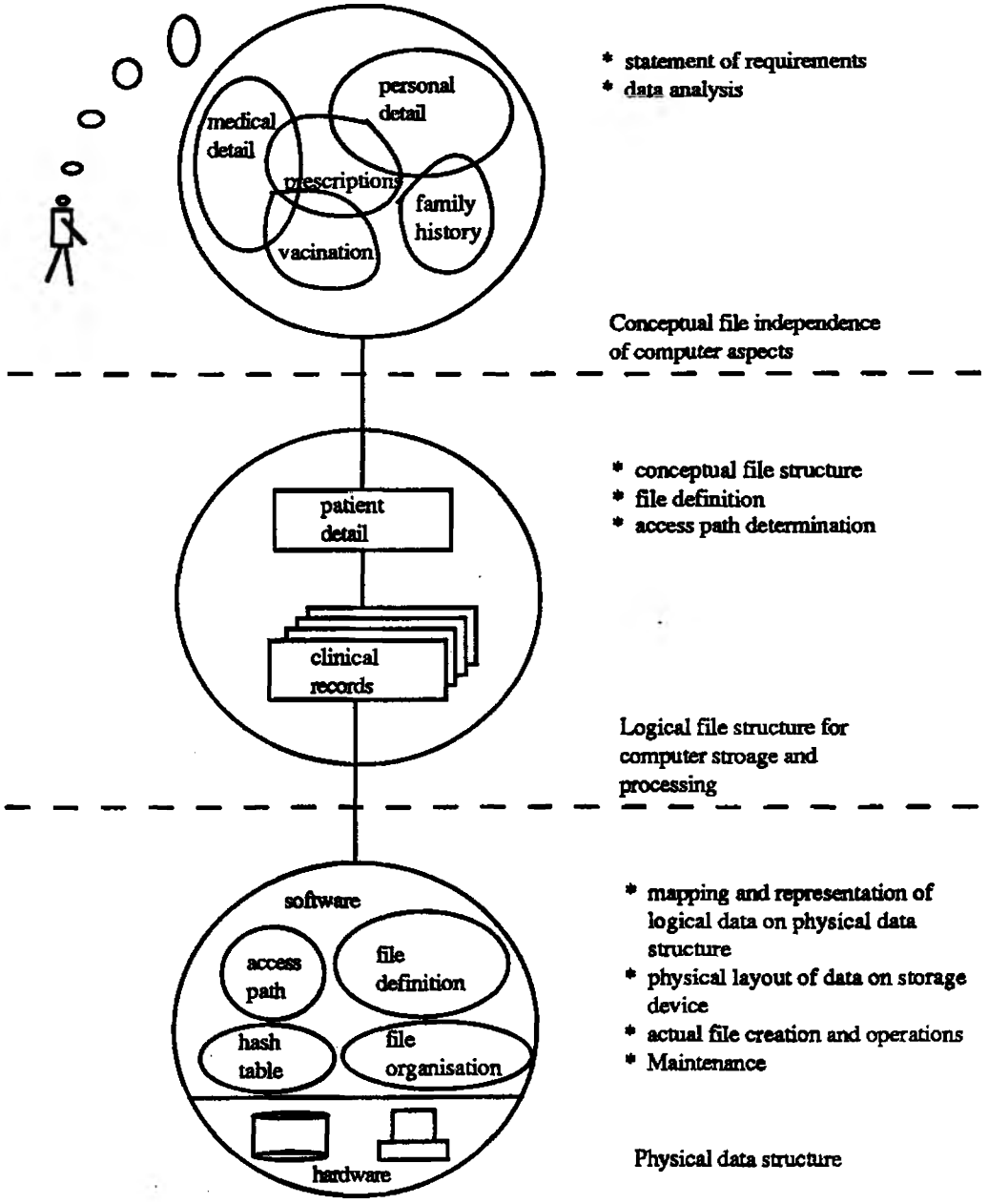

Figure A : File Design Process

2.1 Transformation Process

There are several ways in which the code can be transformed. For instance, convert each character composing the code into an ordinal number and sum up all the numbers. However, this method takes a larger 'base' for the transformation process. 'Base' here refers to the number of possible characters that make up the code. There is another well-known technique called the Soundex Code, developed by Remington-Rand, but this method is undesirable in this application because it ignores all occurrences of A, E, I, O, U, W, H and Y, hence minimising the identification of each code. The technique employed in this system is a combination of both methods designed by Harihodin S. A description of the transformation process as follows:

1. Ignore all occurrences of a, A, -, zero and other special characters; that is, convert them to 0.

2. Convert the following characters to digits:

0, a, A, Special Char - 0

1, b, B, - 1		o, O - 14	
2, c, C, - 2		p, P - 15	
3, d, D, - 3		q, Q - 16	
4, e, E, - 4		r, R - 17	
5, f, F, - 5		s, S - 18	
6, g, G, - 6		t, T - 19	
7, h, H, - 7		u, U - 20	
8, i, I, - 8		v, V - 21	
9, j, J, - 9		w, W - 22	
k, K, - 10		x, X - 23	
l, L, - 11		y, Y - 24	
m, M, - 12		z, Z - 25	
n, N, - 13			

3. Multiply each converted character by 26^n, where n is the relative position of the character starting from the rightmost character, and then sum up the numbers.

2.2 Record Addressing

The above method reduces the base to 26 only where the outcome of the number can be made smaller. However, it is dependent of the size of the maximum integer that the computer can hold. If the number is bigger than the largest integer, then automatic truncation by the computer will occur. Table 1 illustrates the transformation technique. However, each transformed key cannot be taken as an access key for the record because (i) the nature of the programming language in handling external file for direct access [K.L. Bowles, 1980], and (ii) there is a possibility that more than one code might produce the same transformed key due to same initial and date of birth.

Keyword	Code of Characters	Transformed Keyword
HS261154	7 18 2 6 1 1 5 4	18142 (truncated)
AB101062	0 1 1 0 1 0 6 2	28314
CD081253	2 3 0 8 1 2 5 3	31516

The transformed keys above are for illustration purposes only. For each occurrence of character in the code, the corresponding number = (code of 1st char. x 26^7) + (code of 2nd char. x 26^6) + ... (code of 8th Char x 26^0)

The formula to obtain the number used in the computation is : [Harihodin S, 1983a)]

$$\sum_{i=1}^{8} code \times base^{(8-i)}$$

where *code* = *converted character, and*
base = *26*

Table 1 : Example Illustrating The Transformation Technique

To overcome the problem, file indexing technique has to be employed. The idea is to derive, in some way, a unique address from the transformed key and record the address in the index where the actual record can be located. The functions to be performed on the index would be to search for an address and insert a new address of the record. Hence the usage frequency of the index is very high. It is therefore desirable to store the index in the internal memory when the system is running. A possible way to do this is by means of hashing or scatter storage techniques. From now on 'hash table' is the term used instead of the index and 'hash address' is used for address as the more meaningful name for the method. [Martin J. 1975], [Harowitz E & Sahni. S, 1977].

There are several hashing techniques available [S.M. Dean, 1977]. The approach to hashing is to divide the problem into two separate sub-problems:-

1. Find a hashing function h (X), whose range of values for the transformed key domain approximates to a uniform distribution over the hash table.

2. Find a scheme for resolving 'collision' to keys that hashed to the same hash address.

Clearly there are problems when collisions occur, that is the number of records directed to a particular location in the hash table is more than one. Some method of resolving collisions becomes necessary. The more commonly used schemes are:

Linear overflow [W.W. Peterson, 1977]
Quadratic overflow [W.D. Maurer, 1968]

Quadriatic quotient overflow [J.R. Bell, 1970]
Random [M.D. McIlroy, 1963]
Chaining [L.R. Johnson, 1961].

These methods are described and compared by E.S. Page and L.B. Wilson (1978). Both noted that the chaining method is markedly superior to the other methods. The only drawback in this method is that it uses more storage requiirement for its overflow records. On the other hand, E. Harowitz and S. Sahni (1977) have described several hashing functions - mid-square, division, folding and digit analysis method. They have made comparisions and recommend that the division method is more superior to the other types of has functions.

The hashing function used in the proposed system is the division method and the technique employed to resolve collision is the chaining method. The hash table is a one-dimensional array of integers of 10007 elements. Table 2 illustrates hash addressing. Addresses are compute from the transformation of the code and taking the remainder on division by 10007, a prome number. The hashing function in the division method is:

h(K) = K MOD M, where K = the transformed key, and M = size of the table (10007, a prime number used in this project).

Keyword	Code of Characters	Transformed by	HashAddress
HS261154	7 18 2 6 1 1 5 4	18142	8135
AB101062	0 1 1 0 1 0 6 2	28314	9300
CD081253	2 3 0 8 1 2 5 3	31516	5095

Table 2 : Example Illustrating Hash Addressing

For example, the transformed key for HS261154, 18142, is divided by 10007 and taking the remainder, that is 8135.

Figure B is an illustration of the indexing method used in this project. The illustration helps the description of the method which follows:

The code in the Patient Table is arranged in terms of their arrival entering the file. The hash addresses in the Patient Table mainly serve as an illustration. The structure of the hash table and other system parameters is shown in Figure B. File Register (FR) is use to keep track of the number of record in the patient file. Initially the hash table and PR are zeroised and the patient file is created with one dummy record. The reason for creating the dummy record is to make the file ready for processing and to make the actual record key to begin with 1 for the first record, 2 for the second record and so on.

Let us look at the illustration in Figure B. When a patient record with the code HS261158 is to be added, increase the current value of FR by 1. Then check the corresponding location of the hash address for HS251158; that is at

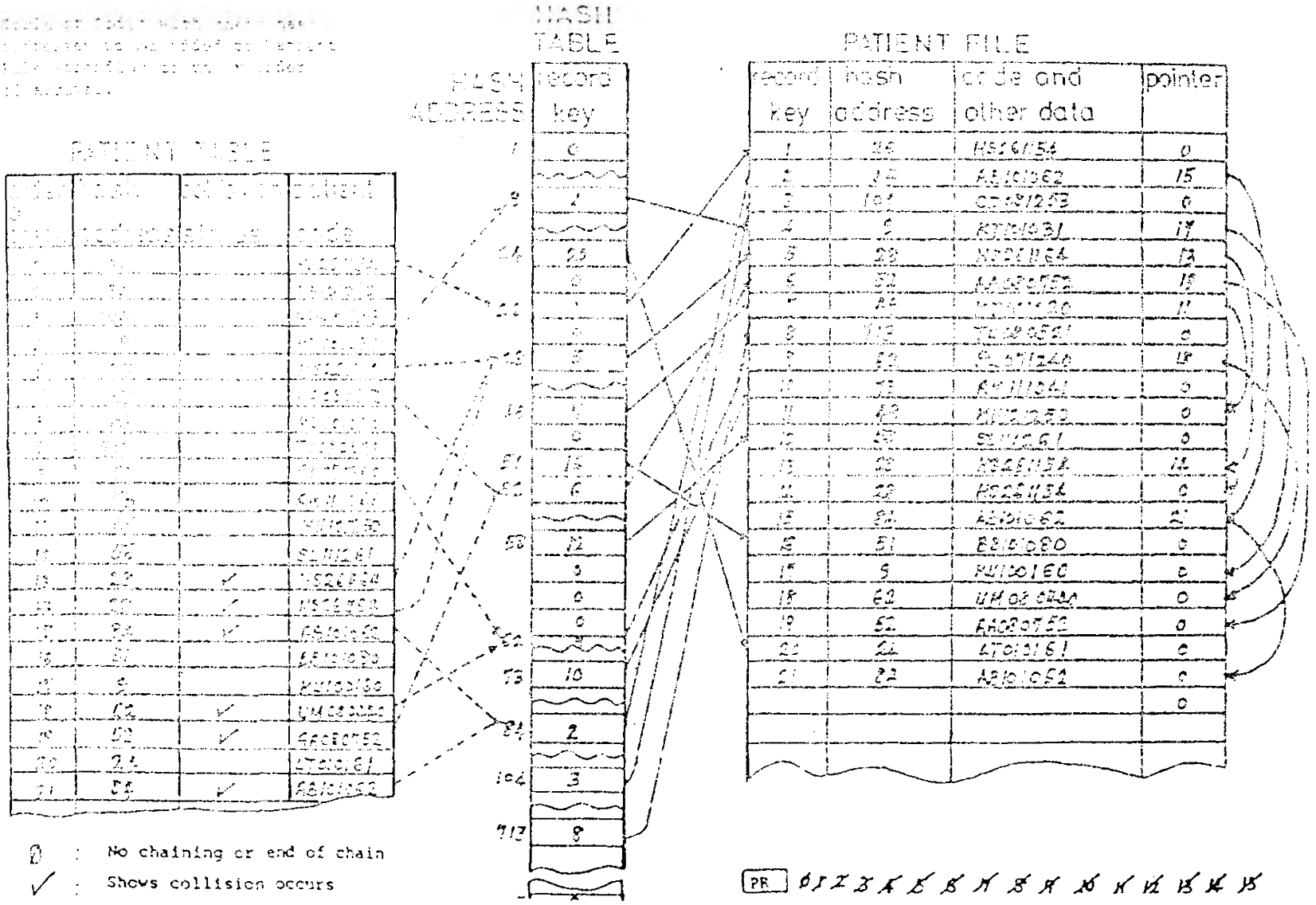
HASH TABLE
PATIENT FILE
HASH ADDRESS
record key
record key
hash address
order and other data
pointer
PR
0 : No chaining or end of chain
✓ : Shows collision occurs

Figure B

location 26 in the hash table. If it is zero, and it is in this case, then insert PR in that location. Then take PR as the record key for that patient record. This creates the access path of the first record in the file, using PR as the record key.

When AB101080 is to be added, increase 1 to content of FR (now FR becomes 2). Then check the corresponding location at hash address in hash table. If it is zero, insert FR in location 84 of hash table. Now take FR as the record key for the second patient record. From the Patient Table, it can be seen that no code is hashed to the same address until the 13th code; that is code HS261164. Now we known that location 28 of the hash table contains the record key for the 5th record. This situation is referred to as a collision and the resulting addresses, 28, are called synonyms.

To resolve collision, the 13th record is chained after the 5th patient record using pointer mechanism. Add 1 to FR and insert FR to the field called POINTER of the 5th record, which means the 13th record is linked from the 5th record. It is seen here how the collision can be overcome using the chaining method which eventually formes a linked list for all the synomyms of a particular hash address. The first record in the list is termed as the 'header' record and the succeeding records are terms as 'overflow' records. Figure B illustrates diagramatically the logical structure of the file in respect of the header records and the overflow records. There may be a number of lists of synonyms and to access the last record in any list involves a table look-up in the hash table to find the access key of the header record and search along the list. The following procedures are established to insert and to find a record:-

1. To insert a record:

   a. Transform the code into a number.

   b. Hash the number and add 1 to PR.

   c. If the location [hash address] in hash table is occupied, do not insert the record. Access the header record using the value at location [hash address] of hash table as the key. Retrieve along the list until the end (Pointer = 0). Update the field called POINTER of the last record by PR. Take PR as the key to the new record and insert the record to the file.

   d. If the hash address location in hash table is empty, insert PR in the location [hash address] of hash table. Take PR as the key to the new record and insert the record to the file and stop.

2. To find a record:

   a. Transform the code into a number.

   b. Hash the number.

   c. If the location hash adfdress in has table is employed, send message 'RECORD NOT FOUND'.

   d. If the location hash address in has table in occupied, access the header record using the value at location hash address as the key.

Compare the code. If match then found. If not, check along the list until the end point. If none match send message 'RECORD NOT FOUND', else found and stop.

3. Design Implementation

This phase involves the initialising and creation of file, file update and file retrieval. File management routine as shown in Figure C is developed to maintain the hash table and file register as well as to serve other application programs which require file access [Figure D] and insertions [Figure E] of new records coded in UCSD Pascal.

```
PROCEDURE file_mgmt_routine;
    found : BOOLEAN;
    rn, ptr : INTEGER;
    ch : CHAR;
    i, j : INTEGER;
BEGIN
  CASE action of
          'a'.'A' : BEGIN
                    WITH index^ DO
                      BEGIN
                        fileregister := fileregistor } 1;
                        IF tabent [hash_key] = 0
                            THEN tabent[hash_key} := fileregister
                        ELSE
                        BEGIN
                            rn := tabent [hash_key];
                            SEEK (patient, rn);
                            GET (patient);
                              BEGIN
                                WHILE patient ^, pointer <> 0 DO
                                  BEGIN
                                    SEEK(patient, patient^, pointer
                                    GET (patient)
                                  END;
                                ptr := patient^, key:
                                WITH patient^ DO
                                  pointer := fileregister;
                                SEEK(patient, ptr);
                                PUT(patient)
                              END;
                        END;
                    END;
                    recno := index^. fileregister
                  END;
    's', 'S' :    BEGIN
                    WITH index^ D0
                      BEGIN
                        IF tabent[hash_key] = 0 THEN
                            reply := FALSE
                        ELSE
                        BEGIN
```

```
                                    rn := tabent[hash_key];
                                    REPEAT
                                        BEGIN
                                            SEEK(patient, rn);
                                            GET(patient);
                                            WITH patient' DO
                                            BEGIN
                                                ID (live = TRUE).THEN
                                                    BEGIN
                                                            position(11,10);
                                                     write('SURNAME:');
                                                    position(11,19);
                                                    write(surname);
                                                    position(13,10);
                                                    write('NAME    :');
                                                    position(13,19);
                                                    write(forename);
                                                    position(23,30);
                                                    write('Right patient?
(Y/N).');

                                                    readln(ch);
                                                    IF (ch = 'y') OR (ch = 'Y')
                                                                THEN found:=
TRUE
                                                                ELSE rn:=
pointer;
                                                        END
                                                    ELSE found := FALSE;
                                                END;
                                        END
                                    UNTIL found OR (patient^.pointer = 0);
                                    IF NOT found THEN reply := FALSE ELSE
                                                    BEGIN
                                                        reply := TRUE;
                                                        recno := patient^. key
                                                    END;
                            END;
                        END;
                    END;
END;
```

Figure C : File Management Procedure coded in UCSD Pascal

```
PROCEDURE extract;
{* To search required record in the patient file*}
BEGIN
      action := 's';
      keyword := code;
      hashed_key := 0;
      hash_routine(hashed_key);
      file_mgmt_routime(hashed_key,action,recnum,reply);
      IF reply = FALSE THEN
        BEGIN
```

```
                position(23,20);
            WRITELN('PATIENT NOT IN FILE PRESS
        <RETURN> TO CONTINUE');
            READLN
          END
        ELSE
        BEGIN
          SEEK(patient, recnum);
          GET(patient);
          IF opptions = '1' THEN BEGIN
                                    patfie
                                    first
                                END
          ELSE BEGIN
                    static;
                displayone
              END
        END
DO;
```

Figure D : Extract Procedure coded in UCSD PASCAL

```
PROCEDURE insert_rec;
VAR
    i:          INTEGER;
BEGIN
    action := 'a';
    BEGIN
            {* form code *}
            WITH patbuf DO
            BEGIN
                    code:='          ';
                    code[1] := ssurname[1];
                    code[2] := forename[1];
            •       FOR i := 1 to 8 DO code [i] := birth[i-2]
            END
      keyword := patbuf.code;
      hash_routine(hashed_key);
      file_mgmt_routine(hashed_key,action,recnum,found);
      SEEK(patient,recnum);
      GET(patient) THEN
      IF EOF(patient) THEN
            BEGIN
                    patbuf.key := recnum;
                    patbuf.pointer := 0;
                    patient^ := patbuf;
                    SEEK(patient,recnum);
                    PUT(patient)
            END
      ELSE
      BEGIN
            position(23,20);
            WRITE('Indexing error')
```

```
            END;
        END;
    END;
```

Figure E : Insert Procedure coded in UCSD PASCAL

4. Concluding Remarks

The experiment which was carried out was based on indexing of bibliography and documents using secondary keys of alphanumeric nature. The hashing approach employed in the design of the file handling system shows that the non-numeric key transformation model gives a convincing response time. The transformation model that has been proposed by the author can be used practically for document retrieval system where records with non-numeric secondary keys could be accessed directly using hashing approach.

References and Bibiography

Bell J.R., 1970. The Quadratic Quotient Method. A hash code eliminating secondary clustering, CACM , Vol. 13, No. 2, 1970, pp 107-109.

Bowles K.L., 1980. Beginners Guide for the UCSD Pascal System. McGraw-Hill (1980).

Deen S.M., 1977. Fundamentals of Data Base System, MacMillan Press Ltd. (1977).

Gotleib C.C, 1978. Data Types and Structures. Prentice-Hall Inc. (1978).

Harihodin S, 1987. "Micro Computer Based Keyword Indexing of Bibliography." Jurnal Teknologi, Universiti Teknologi Malaysia, Bil. 10, Dec., 1987.

HarihodinS.etl 1983. "Medical General Practice Information Systems." Department of OR, Cranfield Institute of Technology, 1983.

Harihodin S, 1983a. Development of Cranfield Institute Keyword Indexing Systems.

Harowitz E & Sahni S., 1977. Fundamentals of Data Structures. A Pitman Int. text (1977, 1981).

Johnson L.R., 1961. Indirect Chaining Method of Addressing on Secondary Keys, CACM, Vol 4, No. 5, 1961 pp 218-222.

Kroenke D., 1977. Data Base Processing - Fundamentals, Modelling, Applications. Science Research Inst. (1977)

Maurer W.D., 1968. An Improved Hash Code for Scatter Storage. CACMM, Vol 11, No. 1, Jan 1968. pp 35-39.

McIlroy M.D., 1963. A Variant method of file searching. CACM, Vol 6, No. 3, 1963, pp 101.

Martin J., 1975. Computer Data Base Organisation. Prentice-Hall, New Jersey (1975). Series in automatic computations.

Paice C.D., 1978. Information Retrieval and Computer. McDonald Computer Monograph (1978).

Page E.S.& Wilson L.B., 1978. Information Representation and Manipulation in a computer, 2nd Ed. Cambridge University Press (1978).

Peterson W.W, 1957. Addressing for Random Access Storage. Journal of IBM, April 1957, pp 130 - 146.