PROJECT REPORT (VOLUME I)

SPELL CHECKER IN GURMUKHI

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

BY

ASHWANI KUMAR (2/90)

NEENA TAYAL (13/90)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
PATIALA - 147 001

(DEEMED TO BE A UNIVERSITY)

MAY 1993

ACKNOWLEDGEMENT

We would like to express our immense gratitude to Mr. A.K. Aggarwal, State Informatics Officer(NIC, Punjab State Unit), for allowing us to use the apt facilities of NIC and providing us with his guidance.

We would also like to express our deep gratitude to our guide Mr. Madhu Sudan Dada, Senior System Analyst, for his amiability, wise guidance and able animad version of the project. His organized and systematic approach proved very helpful in solving abtruse problems. He was like a guiding torch to us.

We thank Mr. Narinder Singh Arneja, Mr. K.B.Singh, Mr. I.M.S. Nat and Other members of NIC family for their kind cooperation and generous help during our training at NIC.

Our thanks are due to our teachers at institute who had been source of inspiration throughout our academic period in the institute.

(ASHWANI KUMAR)

(NEENA TAYAL)

CONTENTS

1. INTRODUCTION OF THE ORGANISATION 1
2. ANALYSIS 5
   - a) INTRODUCTION 5
   - b) PROBLEM DEFINITION 6
   - c) EXISTING SYSTEM 8
   - d) PROPOSED SYSTEM 12
3. DESIGN 15
   - a) CREATION OF DICTIONARY 15
   - b) SORTING METHODS 16
   - c) SEARCHING TECHNIQUES 32
   - d) DESIGN PROCESS 38
   - e) DEFINITION OF PROCEDURES 44
   - f) FLOW CHARTS 56-73
4. IMPLEMENTATION & TESTING 74
   - a) LOGICAL IMPLEMENTATION 74
   - b) PSEUDOCODE/PROCESSING LOGIC 77
   - c) ENVIRONMENT 89
5. CONCLUSION & REMARKS 97
6. ANNEXURES 100
7. BIBLIOGRAPHY 105

NATIONAL INFORMATICS CENTRE

A PROFILE

NATIONAL INFORMATICS CENTRE

In 1976, under the Department of Electronics , nucleus of experts was formed for the introduction of modern information technology and this was termed as National Informatics Centre (NIC). During the past decade NIC has been in the forefront in field of Information Technology. Its pioneering efforts for the bringing about a change in the operation of management techniques with reference to taxation administration , agriculture , health and all other areas of national economy.

In addition , NIC has set up a most modern satellite based network of computers , starting from district levels to state , regional and central systems interconnections at every level. The system is operational and this opens up an enormous potential for data collection , data processing , analysis for administrative and decision making purposes. NIC has also been able to make a headway in creating and introducing an awareness in government departments for computer based information systems as an effective tool for decision support. It is the information that will keep the vitality of the economy in future. The economy would become an "Information Movement and Management (IM&M) " economy. Today , it is not only the most sought after costly commodity,

but also the mainstay of decision making process in any organisational set up.

ROLE OF NIC IN GOVERNMENT INFORMATICS

In order to explore and exploit the vast opportunities offered by information technology which is improving and accelerating the planning process and implementation of socio-economic programmes, all-round development growth of the nation, National Informatics Centre, the NODAL organisation of Government Of India to introduce computer based MIS, file-less office concept, electronic mail services and telematic services in the central, state and district govt. departments, has set up a satellite based Computer-Communication Network **(NICNET)** covering all districts, state capitals and the centre . This is facilitating the development of District Information system at District level **(DISNIC)** and essential databases for decision making at various levels at state and central govt. departments.

NICNET comprises of

- very large computers (NEC-S1000) at the NIC - regional centres (Delhi, Pune, Bubneshwar and Hyderabad)

- ND-550 or equivalent super mini computers at state capitals for providing informatics services to the states

- super PC-AT computer systems at each district

- Mother Earth Station at NIC Headquarters ,Delhi and a Micro Earth Station at each of NIC state units

- District Informatics Centres, for exchange of information via satellite (INSAT-1D).

INFORMATION FLOW

Each District Informatics Centre has facilites to process information for monitoring socio-economic development activities within the district. Each District Informatics Centre has communication link with the **NICNET** for the flow of information between any two nodes of **NICNET** centres.

The State Informatics Centre supports and coordinates all the activities of district centres and in addition,

supports the state government departments. It also provides interactive facility to all the state govt. departments and district administrations. The satellite network has high reliability and is available all the time for efficient information flow.

The State, Regional and National Centres have facilities to share loads and also serve as back up to one another. The Regional Centres provide technical support to the states in the respective regions. They provide specialised peripherals and high processing power for high level modelling and simulation studies. It also serves as a repository for compilation of information on the state systems, if required. In addition, the Centre has sophisticated research and development projects to develop relevant software tools to support district, state and regional level requirements.

SYSTEM ANALYSIS

INTRODUCTION

For the successful introduction of computers in the different spheres of life and for use by the common man, it is very essential that computers should work in the language of the users, i.e. in the native language. With the introduction of multilingual terminal, and GIST terminals, now it is possible to develop the applications in the Indian Languages, so that the users do not resist to the computerized system.

Further many word-processing packages are coming up for writing letters/documents in the Indian Languages. A spell checker is the necessary feature in such word-processing packages. This project report discusses the Design and Development of one such spell checker which checks and corrects the Punjabi text.

PROBLEM DEFINITION

There are many spell-checkers available in different word-processing packages like WORDSTAR, LYRIX etc. But these spell-checkers are working at English text only. No spell-checker is commercially available so far to check the spellings of words of a text written in Punjabi or other Indian languages . So our problem was to design and develop a new spell-checker that can work with Punjabi and/or other Indian Languages.

In NIC, Punjab many applications have been developed in Punjabi also. For documenting the applications in Punjabi there was a need of a spell-checker that can work on Punjabi text . The main features required for a spell-checker are:

1.A dictionary that contains a pool of Punjabi words.

2.It must be able to check each word of the text file and if it is incorrect , it must display some choices for the user to replace that word with any one of them.

3.The user should be able to append the word in the dictionary

4.There are some words in a text file which do not belong to Punjabi language, for these words the facility for the user to

skip the words must be there. e.g. A document may have some words written in English also (multilingual document) , the spell checker should skip such words.

5.The spell checker should be user friendly and it must display help for the user to work with it easily.

SPELL CHECKER OF LYRIX WORD PROCESSOR

Although there are many existing spell-checkers in various packages like WORDSTAR, LYRIX etc. But all of them are capable of checking the spellings of English text only.

In NIC (National Informatics Centre) the available spell-checker is in LYRIX word-processor. Though, it is of use for us in developing the spell-checker in Punjabi, but it is necessary to know that how the LYRIX spell-checker works.

In this existing system,the spell command compares the text with the words contained in LYRIX's 80,000 words dictionary. When LYRIX finds a word it does not recognize, it lets you correct the word, delete the word, skip the word, or add the word to your local dictionary. You can also quit spell & return to editing mode in your document.

For checking the spellings, move the cursor to the beginning of the document or to the place from where you want to start the spell checking. Spell checker searches from the current cursor location to the end of document. For search and spell check ,an entire document, the cursor must be at the top of the file.

To access spell, press <Esc>x to call up the Xtra functions menu, and then select spell, LYRIX stops at the first misspelled word and the spell menu is displayed on your screen. The options are (Correct, Delete, Next, Add, (S)kipall, (Q)uit . To correct the spelling of the word, select (C)orrect option.LYRIX places the cursor on the list of suggested correct spellings, located in the dialog box on your screen. Select any word from these choices and press <return>. LYRIX replaces the word with the selected word in your text and displays the message "Incorrectword replaced with correctword" and then moves on to the next misspelled word.

When you know that the word is spelled correctly, you can skip over the word by selecting Next. If you use that same word often in this document , you can choose select (S)kipall, instructing spell to skip all occurrences of the word. Or if you use that word often in multiple documents, you can add it to your local spell dictionary by selecting (A)dd.

LYRIX continues its search for misspelled words, When LYRIX reaches at the end of the file, the message, "requested task is now complete" is displayed.

This is how the spell-checker in LYRIX works to check the spellings in a given text. Brief description of each selection on the spell menu is explained below:

-- The (C)orrect selection replaces the word with the correctly spelled word. LYRIX provides a list of suggested correct words, as well as a prompt that allows you to enter the correct word. Highlight the word that you want to use as a replacement for the misspelled word, or enter the correct word on the prompt line and press <return>. LYRIX makes the correction then moves to the next misspelled word.

-- The (D)elete selection allows you to remove the misspelled word from your file.

-- The (N)ext selection moves to the next misspelled word without affecting the current one.

-- The (Add) selection places the word in your local dictionary. LYRIX does not recognize some correctly spelled words, such as uncommon surnames, unusual technical jargon, and some slang or colloquial terms. If you save a word in your local dictionary, LYRIX then recognizes it as a correctly spelled word.

-- The (S)kipall selection tells spell to ignore all further occurrences of this word in this editing session. LYRIX leaves the word without changing its spelling.

--The (Q)uit selection exits Spell and returns you to the Edit screen. No changes or corrections are made to your file.

-- The (E)verywhere field allows you to tell LYRIX to correct every occurrence of the misspelled word from the current cursor position to the end of your file. If Everywhere is set to no, the Correct selection only affects the current occurrence of the word.

PROPOSED SYSTEM

After studying the detailed working of the Lyrix spell checker system , which works only for the English text, it is proposed to develop a similar kind of a spell checker which works for the Punjabi language text. The proposed Punjabi language spell checker will have the following facilities/features:-

1. The spell checker will be able to check/correct the Punjabi document/text written with any existing word-processor or editor.

2. The spell checker will have a large database containing numerous Punjabi words which are most oftenly used in the documents/text in the Govt. environment. This is called a "Dictionary".

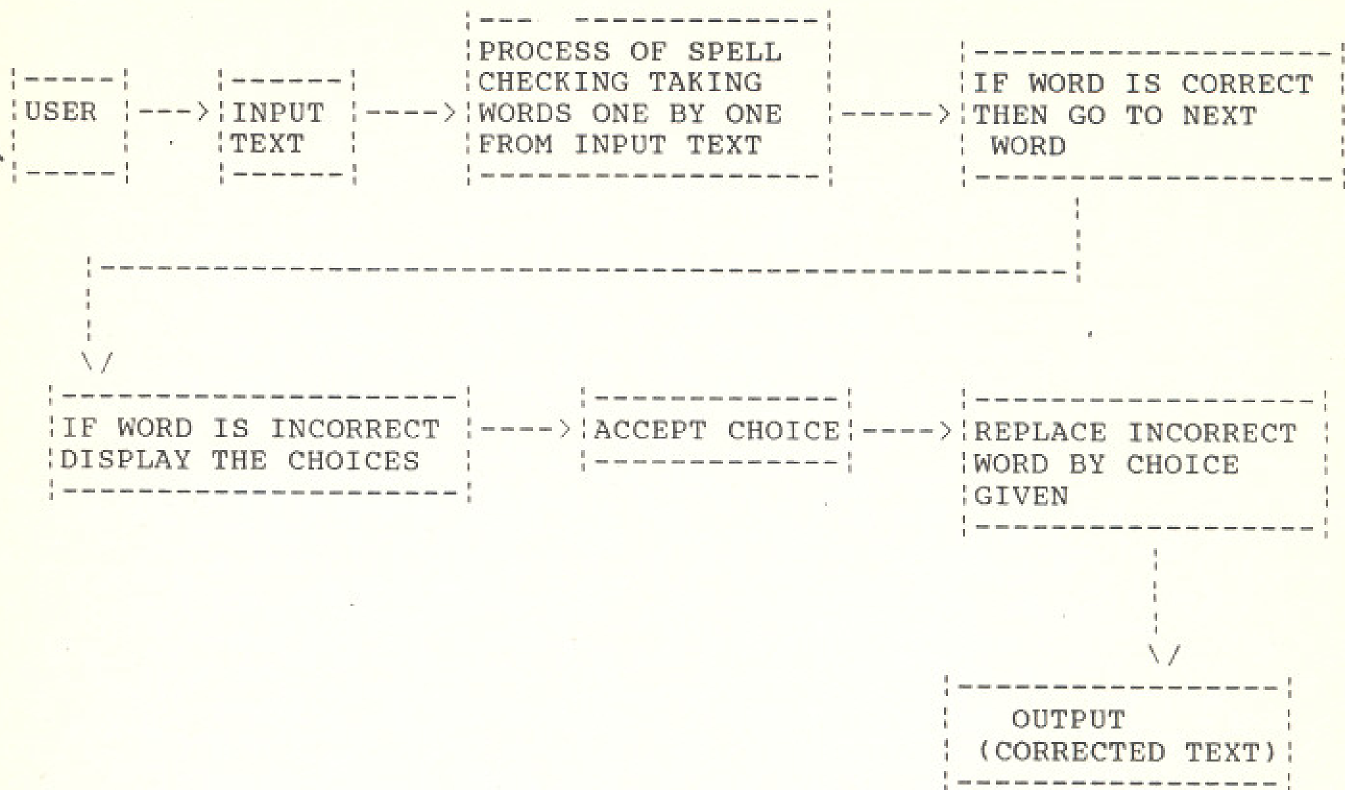
3. User will be able to add the words in the "Dictionary".

4.The spell checker will check each word and wherever it finds a wrong word (or a word which does not matches with the dictionary word), it will stop there and will display the number of similar correct words for replacement and/or skip.

5. The spell checker will be easily invokable by the user.

6. Help facility will be made available wherever the user finds some difficulty.

7. The user will have the option of replacing the original text with all the changes he has made during the spelling checking or make another temporary file of changed text and leave the original file as it is.


USER
INPUT TEXT
PROCESS OF SPELL CHECKING TAKING WORDS ONE BY ONE FROM INPUT TEXT
IF WORD IS CORRECT THEN GO TO NEXT WORD
IF WORD IS INCORRECT DISPLAY THE CHOICES
ACCEPT CHOICE
REPLACE INCORRECT WORD BY CHOICE GIVEN
OUTPUT (CORRECTED TEXT)

SYSTEM DESIGN

DESIGN

After proposing the system, we are to design the system i.e. to develop a spell checker which will work on punjabi as well as on other Indian Languages text to check the spellings of that text.

For design process, the following steps are kept in mind

CREATION OF DICTIONARY

First step in designing the system is to create a database which has a pool of words and is called dictionary. This dictionary is helpful for matching the words of user's input text against it. Thus, the word can be searched . We have taken a dictionary of about thousand Punjabi words as a sample. This Dictionary is given in Annexure-II.We have created the dictionary in two ways:

1.LYRIX: The dictionary is created in lyrix just writing the words one by one. In lyrix next word is seperated from last word by a carriage return. In this type, the dictionary is sorted by Quicksort and then binary search is applied.

2.FOXBASE: In this type, the dictionary is created by creating a dbf file with one field and it is indexed on that only field. This indexed file is used for searching. Search

technique is binary search. In the binary search, the average computing time for searching is (log n).

SORTING METHODS

The next step in design process is to sort these words in dictionary. The words are records of the database. Sorting is necessary for searching the words.

Sorting is the operation of arranging the records of a table into some sequential order according to an ordering criterion . The sort is performed according to the key value of each record. Depending on the makeup of the key, records can be sorted either numerically or, more generally, alphanumerically. In numerical sorting, the records are arranged in ascending or descending order according to the numerical value of the key. An example of this type is the sorting of a symbol table according to the internal numeric value of the alphanumeric representation for each variable name. In general, a key can be any sequence of characters, and the ordering imposed by sorting depends on the collating sequence associated with the particular character set. Hence, algorithms of sorting are applicable to any string of characters when a particular collating sequence is given.

Like in our case, the ASCII codes of Punjabi words are collating sequence given and sorting is done according to the ASCII values of Punjabi letters. The ASCII values of Punjabi letters are given in Annexure-I.

Most of the sorting algorithms involve the movement of records from one place to another in the table. Since records in certain applications can be quite long and consequently expensive to move, the records can be organised in such a manner as to minimise this moving cost while performing a sort.

One method of substantially reducing the cost of moving the records is to arrange the table as a simple linked list. Clearly, the movement of records is efficient when such a representation is used. The additional memory required for a pointer field becomes less significant as the record length increases.

Another method of reducing a record movement is to use a pointer vector, each element of which contains the address of one record.

All sorting methods to be discussed assume that the entire table can be sorted in the computer's main memory. A number of these methods, such as the merge sort and

address-calculation sort, can easily be adapted to tables stored on auxiliary storage devices like disks and drums.

Each sorting method to be discussed will include an approximate quantitative description of the method as to the number of comparisons and the number of record movements required. Since, as previously mentioned, the expense of the record movements can be reduced significantly, the more important factor is the number of comparisons which is required by a particular method.

Description of different sorting algorithms is as follows :

1. Selection sort :

One of the easiest ways to sort a table is by selection. Beginning with the first record in the table, a search is performed to locate the element which has the smallest key. When this element is found, it is interchanged with the first record in the table.This interchange places the record with the smallest key in the first position of the table. A search for the second smallest key is then carried out. This is accomplished by examining the keys of the records from the second element onward. The element which has the second

smallest key is interchanged with the element located in the second position of the table. The process of searching for the record with the next smallest key and placing it in its proper position (within the desired ordering) continues until all records have been sorted in ascending order. A general algorithm for the selection sort is now presented.

1. Repeat thru step 5 a total of n-1 times
2. Record the portion of the vector already sorted
3. Repeat step 4 for the elements in the unsorted portion of the vector
4. Record location of smallest element in unsorted vector
5. Exchange first element in unsorted vector with smallest element

Performance:

We now turn to the performance of this algorithm. During the first pass, in which the record with the smallest key is found, n-1 records are compared. In general, for the ith pass of the sort, n-i comparisons are required. The total number of comparisons is, therefore, the sum

$$\sum_{i=1}^{n-1} (n-i) = 1/2\, n(n-1)$$

Therefore, the number of comparisons is proportional to (n*n), i.e., O(n*n). The number of record interchanges depends upon how unsorted the table is. Since, during each pass, no more than one interchange is required, the maximum number of interchanges for the sort is n-1.

2. Bubble sort :

Another well-known sorting method is the bubble sort. It differs from the selection sort in that, instead of finding the smallest record and then performing an interchange, two records are interchanged immediately upon discovering that they are out of order.

When this approach is used, there are at most n-1 passes required. During the first pass, K1 and K2 are compared, and if they are out of order, then records R1 and R2 are interchanged; this process is repeated for records R2 and R3, R3 and R4, and so on. This method will cause records with small keys to move or "bubble up". After the first pass, the record with the largest key will be in the nth position. On each successive pass, the records with the next largest key

will be placed in position n-1, n-2, ..., 2, respectively, thereby resulting in a sorted table.

After each pass through the table, a check can be made to determine whether any interchange were made during that pass. If no interchanges occurred, then the table must be sorted and no further passes are required.

A general algorithm for the bubble sort is as follows.

1. Repeat thru step 4 a total of n-1 times
2. Repeat step 3 for elements in unsorted portion of the vector
3. If the current element in the vector > next element in the vector, then

   exchange elements
4. If no exchanges were made

   then return

   else reduce the size of the unsorted vector by one

Performance:

Let us consider the analysis of the bubble sort. The best case involves performing one pass which requires n-1 comparisons. Consequently, the best case is O(n). The worst case performance of the bubble sort is n(n-1)/2 comparisons

and n(n-1)/2 exchanges. The average case is more difficult to analyze than the other cases. It can be shown that the average case analysis is O(n*n). The average number of passes is approximately n-1.25(sqrt(n)). For n=10 the average number of passes is 6. The average number of comparisons and exchanges are both O(n*n).

3.Insertion sort:

The basic step in this method is to insert a record R into a sequence of ordered records, R1, R2,...., Ri, (K1 <= K2 .... <=Ki) in such a way that the resulting sequence of size i+1 is also ordered. The algorithm below accomplishes this insertion. It assumes the existence of an artificial record R0 with key K0 = -maxint (i.e. all keys are >= K0).

1.char afile[10];

2.procedure insert(records r; afile list; int i)

/*insert record r with key r.key into the ordered sequence list[0],...,list[i] in such a way that the resulting sequence is also ordered on the field key. We assume that the list contains a dummy record at index zero such that r.key >= list[0].key for all i /*

```
3.int j;
4.{
5.j=i;
6.while r.key < list[j].key
7.{
8.list[j+1]=list[j];
9.j=j-1;
10.}
11.list[j+1]=r;
12.} /*end of insert */
```

Performance:

In the worst case algorithm insert(r,list,i) makes i+1 comparisons before making the insertion. Hence the computing time for the insertion is O(i). insort invokes procedure insert for i=1,2,...,n-1 resulting in an overall

worst case time of $O(\sum_{i=1}^{n-1} i) = O(n*n)$.

```
1.procedure insort(afile list;   int i);
/*  sort list in nondecreasing value of the file key.   Assume
n>=0 */
2.int j;
```

```
3.{
4.list[0].key= -maxint;
5.for(j=2;j<=n;j++)
6.insert(list[j],list,j-1);
7.} /*end of insort */
```

One may also obtain an estimate of the computing time of this method based upon the relative disorder in the input file. We shall say that the record

$$R_i \text{ is 'left out of order' iff } R_i < \max_{1 \le j < i} \{R_j\} .$$

Clearly, the insertion step has to be carried out only for those records that are 'left out of order' . If k is the number of records 'left out of order', then the computing time is O((k+1)n). The worst case time is still O(n*n). One can also show that the average time is also O(n*n).

4. Quicksort:

In insertion sort the key Ki currently controlling the insertion is placed into the right spot with respect to the sorted subfile (R1,....,R(i-1)). Quicksort differs from insertion sort in that the key Ki controlling the process is placed at the right spot with respect to the whole file. Thus, if key Ki is placed in position s(i), then Kj <= Ks(i) for j < s(i) and Kj >= Ks(i) for j > s(i). Hence after this positioning has been made, the original file is partitioned into two subfiles, one consisting of records R1,...,(Rs(i)-1) and the other of records (Rs(i)+1),...,Rn. Since in the sorted sequence all records in the first subfile may appear to the left of s(i) and all in the second subfile to the right of s(i), these two subfiles may be sorted independently. Procedure 'interchange' (x,y) performs t=x; x=y; y=t.

```
1.procedure qsort(afile list; int m,n);

/*  sort records list[m],....,list[n] into nondecreasing order
on field key. Key k = list[m].key is arbitrarily chosen as
the control key. Pointers i & j are used to partition the
subfile so that at any time list[l].key <= k, l<i and
list[l].key >=k, l>j.  It is assumed that list[m].key <=
list[n+1].key */
```

```
2.int i,j,k;
3.{
4.if (m<n)
5.{
6.i=m;
7.j=n+1;
8.k=list[m].key;
9.do
10.{
11.do
12.{
13.i=i+1;
14.}while (list[i].key >= k);
15.do
16.{
17.j=j-1;
18.} while (list[j].key <= k);
19.if (i<j)
20.interchange(list[i],list[j]);
21.}while (i >= j);
22.interchange(list[m],list[j]);
23.qsort(list,m,j-1);
24.qsort(list,j+1,n);
25.} /*end of if */
26.} /* end of qsort */
```

Performance:

In the worst case of this algorithm, the compile time is $O(n*n)$. However, If we are lucky then each time a record is correctly positioned, the subfile to its left will be of the same size as that to its right. This would leave us with the sorting of two subfiles each of size roughly $n/2$. The time required to position a record in a file of size n is $O(n)$. If $T(n)$ is the time taken to sort a file of n records, then when the file splits roughly into two equal parts each time a record is positioned correctly we have

$T(n)= O(n \log n)$

The average computing time for Quicksort is $O(n \log n)$. Moreover, experimental results show that as far as computing time is concerned, it is the best of the internal sorting methods.

Unlike Insertion sort where the only additional space needed was for one record, Quicksort needs stack space to implement the recursion. In case the files split evenly as in the above analysis, the maximum recursion depth would be log n requiring a stack space of $O(\log n)$. The worst case occurs when the file is split into a left subfile of size n-1 and a right subfile of size 0 at each level of recursion. In this

case, the depth of recursion becomes n requiring stack space of O(n). The worst case stack space can be reduced by a factor of 4 by realizing that right subfiles of size less than 2 need not be stacked. An asymptotic reduction in stack space can be achieved by 'sorting smaller subfiles first'. In this case the additional stack space is at most O(log n).

COMPARISON OF SORTING METHODS

The sorting methods discussed thus far are summarised in table below. Note that the entries in the table are approximate:

Table Computing Time

In this table, there is the comparison of all the sorting methods discussed above. Table shows the average time, the method takes in sorting, average time in the worst case and space usage by a particular sorting method.

Comparison of sorting methods (entries are approximate)

Algorithm	Average	Worst case	Space usage
Selection	(n*n)/4	(n*n)/4	In place
Bubble sort	(n*n)/4	(n*n)/2	In place
Insertion sort	(n*n)/4	(n*n)/2	extra space for one record
Quicksort	O(n log n)	(n*n)/2	extra log n entries

WHY WE USED QUICKSORT

We used Quicksort method of sorting because of the following reasons:

1. As from our study of different sorting techniques, we came to conclusion that selection or bubble sort can be used if the number of records in the table are small. If number is large, Quicksort can be used. As we have dictionary of much more words to sort, we used quicksort.

2. Quicksort is used when there are insertions and deletions are less in the table. There are very less insertions and deletions in our dictionary and we used this method of sorting.

3. Moreover, experimental results show that as far as average computing time is concerned, it is the best of the internal sorting methods, we have discussed above.

4. Quicksort is also supported by the standard 'C' on Xenix operating system, as there is a function qsort() available in 'C'. The parameters of this function are

```
qsort(*base, size_t nelem, size_t width, int (*fcmp)());
```

Base is the pointer to the first(0th) element of the table to be sorted.

nelem is the number of entries in the table.

width is the size in bytes of each element to be sorted.

fcmp() is a pointer to a user supplied function called the comparison function. fcmp() must be defined so that fcmp(ptr1,ptr2) returns -1 if *ptr1 is less than *ptr2, 0 if *ptr1 equals *ptr2, and +1 if *ptr1 is greater than *ptr2. ptr1 & ptr2 are declared as pointers to elements of the table to be sorted. For simple string sorts we can use strcmp or memcmp variants in string.h or memory.h.

5. Quicksort is the fastest sorting method.

SEARCHING TECHNIQUES

The next step is to search a word from the sorted dictionary. As explained earlier the checking of spellings is to be done by taking one word of the the text each time and then try to find out a match for it from the dictionary words. If a word is found , then it is taken as correct ,otherwise incorrect.

So searching part of the system is very crucial point. As dictionary is of very large size , it may be of 80000 words or more . So a searching technique is required , which will be faster as well as less complicated.For we studied different searching techniques , all of them have some advantages and some disadvantages. We are describing that study here.

SEQUENTIAL SEARCH :

As a file is a collection of records , each record having one or more fields. The fields are used to distinguish among the records, and these fields are called KEY . As in our dictionary file we have one field in the file, so it is key element of file.

Now if we want to locate some record in the file , this can be done by matching it with the key field. for

example in a telephone directory, each record have information like:

TELEPHONE NO: PERSON-NAME : ADDRESS:

Now the TELEPHONE-NO can be key field.

Generally, file is assumed to be serial(a sequence of components), where the basic operations are read, write and reset.Reading and writing operations are done serially (accessing one adjacent component after each other)

Once we have a collection of records , there are at least two ways of storing them : SEQUENTIALLY and NON-SEQUENTIALLY. For the time being let us assume that we have a sequential file "f" and we want to retrieve a record with certain key value "k". If "f" has "n" records with f[i].key and key values for record "i" then one may carry out the retrieval by examining the key values f[n].key ... f[1].key in that order until correct record is located.

Such type of search is called sequential search. Since records are examined sequentially . The algorithm for this search:

1. Procedure seqsrch(f: a file ; var i: int; n,k :int);
2. (search a file f with key values f[1].key,....f[n].key for a record
3. such that f[i].key=k. If there is no such record, i is set to 0.)

```
4.  {
5.  f[0].key = k;
6.  i=n;
7.  while f[i].key <> k
8.  {
9.  i=i-1;
10. }
11. }(end of search)
```

Note that introductin of dummy records with f[0].key = k in "f" simplifies the search by eliminating the need for an end of file test (i<1) in the while loop . While this might appear to be a minor improvement, it actually reduces the running time by 50% for large n.

Performance:

If no record in the file has key value k, then i=0, and above algorithm requires (n+1) comparisons. Number of key comparisons made in case successful search ,depends on the position of the key in the file.So AVG no. of comparisons for successful search is:

((n-i+1)/n = (n+1)/2

1 <= "i" <= n

For large n ,it may be inefficient . So we can not use it for our search procedure as our dictionary is very large.

BINARY SEARCH:

In this method, the search begins by examining the record in the middle of the file rather than the one at one of the ends as in sequential search. Let us assume that the file being searched is ordered by nondecreasing values of the key (i.e. in alphabetical order for strings). Then, based on the results of the comparison with the middle key, f[m].key, one can draw one of the following conclusions:

1. If k < f[m].key then if the record being searched for is in the file, it must be in the lower numbered half of the file;

2. If k = f[m].key then the middle record is the one being searched for;

3. If k > f[m].key then if the record being searched for is in the file, it must be in the higher numbered half of the file.

Consequently, after each comparison either the search terminates successfully or the size of the file remaining to be searched is about one half of the original size (note that in the case of sequential search, after each comparison the size of the file remaining to be searched decreases by only 1). So after j key comparisons the file remaining to be examined is of size at most [n/2] (n is the number of records). Hence, in the worst case, this method requires O(log n) key comparisons to search a file. Algorithm binsrch implements the scheme just outlined.

```
1.procedure binsrch(afile f;  int i;  int n,k);
/* search a file whose n records are ordered such that
f[1].key  <= f[2].key <= ....  <= f[n].key for a record i such
that  f[i].key  <=  k;  i=0 if there is no  such  record else
f[i].key=k.  Throughout the algorithm, l is the smallest index
such  that f[l].key may be k and u the largest index such that
f[u].key may be k.  */
2.boolean done;
3.int l,u,m;
4.{
5.l=1;
6.u=n;
7.done=false;
```

```
8.while((l<=u) and (not done))
9.{
10.m=(l+u)/2;
/* compute index of middle record */
11.switch (compare(k,f[m].key))
12.{
13.'>': l=m+1; /* look in upper half */
14.'=':
15.{
16.i=m;
17.done=true;
18.}
19.'<': u=m-1; /* look in lower half */
20.} /*end of case */
21.} /*end of while*/
22.} /* end of binsrch */
```

WHY WE USED BINARY SEARCH:

So after j key comparisons, the file remaining to be examined is of size at most (n/2**j). For both the worst case and the average case, this method requires O(log n) key comparisons to search a file and it is much better than all the searching methods.

--Its another advantage is that it is very fast in searching which is the requirement for our dictionary.

--'C' also supports the binary search function.

DESIGN PROCESS

First of all, the database file has been put into an array by taking one by one word. Then quicksort function has been used on this array for sorting the dictionary. This function is available in 'C'. There is only one field in our database. So we have sorted on this field. Records of this field are the words of dictionary. The sorted words are in the file named sdic. Then again, we have put the database file i.e. dictionary into an array by taking reverse of each and every word. Now the reverse words are in the array and quicksort function is used on this array. The reverse sorted words are in the file named sdic1.

In the next step, we have opened these two files sdic and sdic1 in read mode for comparing and searching the words.

We have opened another temporary file tr in write mode . In this file, we have put choices by the user in place of incorrect words and also put correct words.

Another file arr have been opened in read mode in which user's input text is there for which he will use the spell checker.

sd(): It is a procedure which have been called after opening all the files. This procedure have put the sorted words in a

two dimensional array w[i][j]. These words are available from sdic file. The words have been put into array until the end of file. Each word is stored in a different row i.e. When there is <return>, the next word is stored in next row.

sd1(): After calling sd() procedure, sd1() procedure has been called. In this procedure, the reverse words of dictionary are put into two dimensional array w1[i][j]. These words are available from sdic1 file which is opened in read mode. This procedure have put the words into array until the file ends and each word is stored in next row i.e. it checks for <return> or blank and goes to next row.

ab(): This procedure is called after sd() and sd1() i.e. when both the files of dictionary words (original and reversed) are put into array. This procedure works for the input text file until it ends i.e. it works for the whole input text. This procedure has taken one word from input text file and it has stored the word into an array w3. This is done by taking characters one by one and if there is any blank, tab or carriage return, it stores the word into array w3. Thus, each word of user's input text is stored in this array, when this procedure is called again and again for each word. After this, the word has been reversed and stored in

array w4 or the mirror image of the word is stored in w4 for comparing the word from end.

After storing the word in w3, it calls another procedure prl() for searching the word. Now, we see what prl() does:

prl(): This procedure will do work when there is any word in w3. If array w3 is blank, then this procedure will not work. Suppose there is some word stored in w3, it will be searched from the dictionary by this procedure. Here, we have used binary search for searching the word. In binary search, the whole dictionary file is divided into two parts. The word in w3 is compared with the middle word of dictionary file. If the word is found, then it is O.K. otherwise it will be seen that word in w3 is greater or smaller than middle word. If it is greater than middle word then the search will be in lower half part of dictionary. If it is smaller than middle word then the search will be in upper half part of dictionary because the dictionary is sorted. If next search is in upper half portion of the file, then again this will be divided into two parts and middle word is compared with input word. If the word is found, the search will be stopped. Otherwise, this process will go on. In this method, comparisons are very less as compared to other searching procedures. After j

comparisons the file portion is to be searched is equal to (n/2**j).

If the word is not found in dictionary, now instead of comparing the whole word, we have compared only two or three characters of input word with the dictionary. For this the same binary search has been applied. Resulting words are stored in an array w6. This same process is applied in procedure pr() which is called by pr1().

pr() : In this procedure we have taken the mirror image of the word which is stored in w4 and the dictionary of reverse words is in w1. After comparing two or three characters of input word with this reverse dictionary, the resulting words are stored in d[i][j]. The words in this array have been reversed and now stored in w5 which are original words of dictionary.

The words in w5 and w6 are stored together in w7 for display the choices. The whole work has been done by calling procedure put().

pr1() then calls df() which is designed to get the choice from user i.e. the incorrect word is to be replaced by the chosen word. If the choice given by the user is greater than the

number of choices available, it will again ask for choice by saying that the choice is incorrect. Another option is provided to the user to skip the word by entering the choice 0. If the choice is correct , the word in w3 is replaced by the chosen word from w7. The whole work of replacing the words has been done in temporary file. After that it calls the procedure chn().

chn() : chn() is basically used for initialising the array w7 for next word. Procedure pr1() will work until the user's input text ends. Then it will return to the main program.
In the main program, temporary file and user input file are closed. Then it asks user whether he wants to save the changes or not. If the user says yes then it calls procedure rep().

rep() : In this procedure, temporary file has been opened in read mode and user's input file has been opened in write mode because now the changes are to be made in the user's original input file. Then it writes the words into user's file by taking one by one from temporary file. This procedure goes on until the end of file.

After this it will ask from user whether he wants to append in dictionary or not. If the user will say 'yes' then it will

call procedure append(). This extra facility is given to the user for appending the uncommon words in the dictionary.

append(): This procedure appends a new record in the database file i.e. a new word in the dictionary. In this procedure, first of all the database file (dbf file) is opened in append mode. Then the information in the header of the file is read into a structure. We can get the record size from the structure of the header. We seek the database file at the end of file. After this, a string from user is accepted and it is written to the database file in the end with length equal to record size.

We get the information about time from structure that when the database was updated last time. To update it again after appending the word, we take current system time from time.h and exchanges it with the information regarding time contained in structure. The number of records will be one less than the actual number because we have added one more record but database is not updated. We increase the information about number of records by one. Then structure is written in the beginning of file to update the header of database.

DESCRIPTION OF PROCEDURES

==

PROCEDURE NAME	: SD
PART OF	: SPELL-CHECKER
CALLED BY	: MAIN()
PROCEDURES CALLED	: NONE
PURPOSE	: PUTS THE DICTIONARY OF PANJABI WORDS IN AN ARRAY W[] . WORDS ARE IN THEIR ORIGINAL FORM .
PARAMETERS	: NONE
GLOBALS	: i - IT POINTS TO THE CURRENT WORD WHICH IS TO BE WRITTEN IN THE ARRAY. j - IT POINTS TO THE CURRENT CHARACTER OF THE WORD TO BE WRITTEN. W3[]- ARRAY WHERE THE DICTIONARY IS TO BE STORED TO OPERATE WITH.
LOCALS	: NONE

==

==

PROCEDURE NAME : ab

PART OF : SPELL-CHECKER

CALLED BY : MAIN()

PROCEDURE CALLED : NONE

PURPOSE : TAKES ONE WORD FROM THE INPUT TEXT AND PUT IT IN AN ARRAY W3[] , THEN IT REVERSES THE WORD AND PUT IT INTO ANOTHER ARRAY W4[].

GLOBALS : W3[] - ARRAY IS USED TO STORE THE WORD.
W4[] - ARRAY IS USED TO STORE THE REVERSE OF THE TEXT WORD STORED IN W3[].

LOCALS : c - STORES THE VALUE RETURN BY FGETC()
a - POINTS TO CURRENT CHARACTER OF THE WORD IN W3[] .
y - IT STORES THE NUMBER OF CHARACTERS OF THE WORD STORED IN W3[] , THAT IS TO BE REVERSED.
m - IT IS COUNTER FOR ARRAY W4[].

==

==

PROCEDURE NAME : SD1

PART OF : SPELL-CHECKER

CALLED BY : MAIN()

PROCEDURE CALLED : NONE

PURPOSE : PUTS THE DICTIONARY INTO AN ARRAY W1[]
HERE THE WORDS ARE IN THEIR REVERSE
ORDER .

GLOBALS : i - POINTS TO THE CURRENT WORD OF THE
DICTIONARY WHICH IS BEING INSERTED
HERE WORDS OF DICTIONARY ARE IN REV-
ERSE ORDER (MIRROR IMAGES OF WORDS
ARE STORED).

J - POINTS TO THE CURRENT CHARACTER OF
OF THE WORD IN REVERSE ORDER THAT
IS BEING INSERTED.

LOCALS : NONE.

==

==

PROCEDURE NAME : pr1

PART OF : SPELL-CHECKER

CALLED BY : MAIN()

PROCEDURE CALLED : pr() , put() , df() , chn().

PARAMETERS : NONE

PURPOSE : TO SEARCH THE DICTIONARY FOR MATCH OF THE WORD TAKEN FROM INPUT TEXT AND STORED IN W3[]. AND IF IT IS FOUND DISPLAY CORRECT . OTHERWISE IF IT IS NOT FOUND IT STORES SOME WORDS FROM THE DICTIONARY INTO ARRAY W6[] HAVING SAME STARTING CHA -RACTERS AS THAT OF INPUT TEXT WORD.

GLOBALS : W3[] - HAVING THE WORD FROM INPUT TEXT.

W1[] - HAVING THE DICTIONARY OF PANJABI WORDS IN ORIGINAL ORDER .

W6[] - HAVING THE WORDS FROM DICTIONARY STARTING WITH SAME CHARACTERS AS THAT OF INPUT TEXT WORD.

LOCALS : f - POINTS TO THE WORD FROM THE DICTIONARY FROM WHERE THE SEARCH IS TO BE DONE .

l - POINTS TO THE WORD FROM THE DICTIONARY UPTO WHICH SEARCH IS TO BE DONE.

x1 - POINTS TO THE CURRENT WORD OF DICTIONARY WHICH IS BEING MATCHED WITH THE INPUT TEXT WORD .

s - STORES THE VALUE RETURNED BY THE THE FUNCTION MEMCMP() .

==

==

PROCEDURE NAME : pr()

PART OF : SPELL- CHECKER

CALLED BY : pr1()

PURPOSE : THIS PROCEDURE IS ONLY THEN WHEN WORD OF TEXT IS NOT FOUND IN THE PROCEDURE pr1(). ACTUALLY THIS STORES THE WORDS FROM THE DICTIONARY WH- ICH HAVE SIMILAR TAILING CHARACTERS AS THAT OF INPUT TEXT WORD INTO AN ARRAY NAMED W5[].

PARAMETERS : NONE

GLOBALS : m1 - STORES THE NO. OF WORDS OF THE DICTIONARY

W4[] - IT STORES THE MIRROR IMAGE OF ONE WORD FROM INPUT TEXT.

W1[] - IT STORES MIRROR IMAGES OF ALL THE DICTIONARY WORDS (MEANS STORES DICTIONARY OF INVERTED WORDS) .

d[] - STORES THE MIRROR IMAGES OF WORDS HAVING SAME TAILING CHARACTERS AS THAT OF INPUT TEXT WORD.

W5[] - STORES THE ORIGINAL WORDS WHICH ARE STORED IN d[].

LOCALS : f - IT POINTS TO THE POSITION OF THE FROM WHERE THE SEARCH IS TO BE STARTED.

l - IT POINTS TO THE POSITION OF THE UPTO WHICH THE SEARCH IS TO BE DONE.

s - STORES THE VALUE RETURN BY FUNCTION MEMCMP.

e - SUBSCRIPT FOR THE ARRAY W5[].

k - SUBSCRIPT FOR THE ARRAY W5[].

:==:

==

PROCEDURE NAME : put()

PART OF : SPELL - CHECKER

CALLED BY : pr1()

CALLING PROCEDURE : NONE

PARAMETERS : NONE

PURPOSE : THIS PROCEDURE IS DESIGNED TO PUT ALL THE WORDS WHICH HAVE SIMILAR STARTING AND ENDING CHARACTERS AS THAT OF INPUT WORD FROM THE TEXT WHICH IS NOT FOUND IN THE DICTIONARY.

GLOBALS : q1 - IT STORES THE NUMBER OF WORDS WHICH ARE STORED IN W6[].

p1 - IT STORES THE NUMBER OF WORDS WHICH ARE STORED IN W5[].

W5[]-IT STORES THE WORDS WHICH HAVE SIMILAR ENDING CHARACTERS AS THAT OF INPUT TEXT WORD.

W6[]-IT STORES THE WORDS WHICH HAVE SIMILAR STARTING CHARACTERS AS THAT OF INPUT TEXT WORD.

W7[]-IT STORES ALL THE WORDS THAT ARE IN W5[] AND W6.

LOCALS : h1 - STORES NO OF WORDS IN W6[].

h2 - STORES NO OF WORDS IN W5[].

h3 - STORES NO OF WORDS IN W7[].

==

==

PROCEDURE NAME : chn()

PART OF : SPELL - CHECKER

CALLED BY : prl()

CALLING PROCEDURE : NONE

PARAMETERS : NONE

PURPOSE : THIS PROCEDURE IS DESIGNED TO INITIALISE W7[] ARRAY , AFTER DISPLAYING OPTIONS FOR ONE WORD.

GLOBALS : W7[] - THIS IS THE ARRAY WHICH STORES ALL AVAILABLE OPTIONS FOR THE USER.

LOCALS : i2 - IT POINTS TO iTH ELEMENT OF THE ARRAY W7[].

j2 - IT POINTS TO jTH CHARACTER.

==

==

PROCEDURE NAME	: df()
PART OF	: SPELL - CHECKER
CALLED BY	: pr1()
CALLING PROCEDURES	: NONE
PARAMETERS	: NONE
PURPOSE	: THIS PROCEDURE IS DESIGNED TO TAKE OPTION ENTERED BY THE USER FROM KEY _ BOARD TO REPLACE THE INCORRECT INPUT TEXT WORD WITH THE WORD THAT IS SAID BY THE USER FROM THE GIVEN OPTIONS.
GLOBALS	: b - STORES THE TOTAL NO OF WORDS IN W7[] ARRAY. W7[] - STORES THE WORDS HAVING SIMILAR STARTING AND CHARACTERS AS THAT OF INPUT TEXT WORD. W3[] - STORES THE INPUT TEXT WORD.
LOCALS	: k2 -STORES THE NO OF WORDS THAT ARE IN W7[]. q2 -IT STORE THE OPTION GIVEN BY USER.

==

==

PROCEDURE NAME : rep()

PART OF : SPELL - CHECKER

CALLED BY : MAIN()

PROCEDURES CALLED : NONE

PARAMETERS : NONE

PURPOSE : THIS PROCEDURE IS DESIGNED TO IMPLEMENT ALL CHANGES THAT ARE MADE BY USER AFTER FINDING ITS INPUT TEXT INCORRECT , ACTUALLY CORRECTED FILE IS DIFFERNT FROM TEXT FILE . SO THIS PROCEDURE AFTER ASKING FROM USER TO CHANGE , COPIES THE THAT CORRECTED TEMPORARY FILE TO ORIGINAL TEXT FILE.

GLOBAL : NONE

LOCAL : c1 - TO STORE THE CHARACTER RETURNED BY FUNCTION fget().

==

==

PROCEDURE NAME : append()

PART OF : SPELL-CHECKER

CALLED BY : main()

PROCEDURES CALLED : app_dbf()

PARAMETERS : NONE

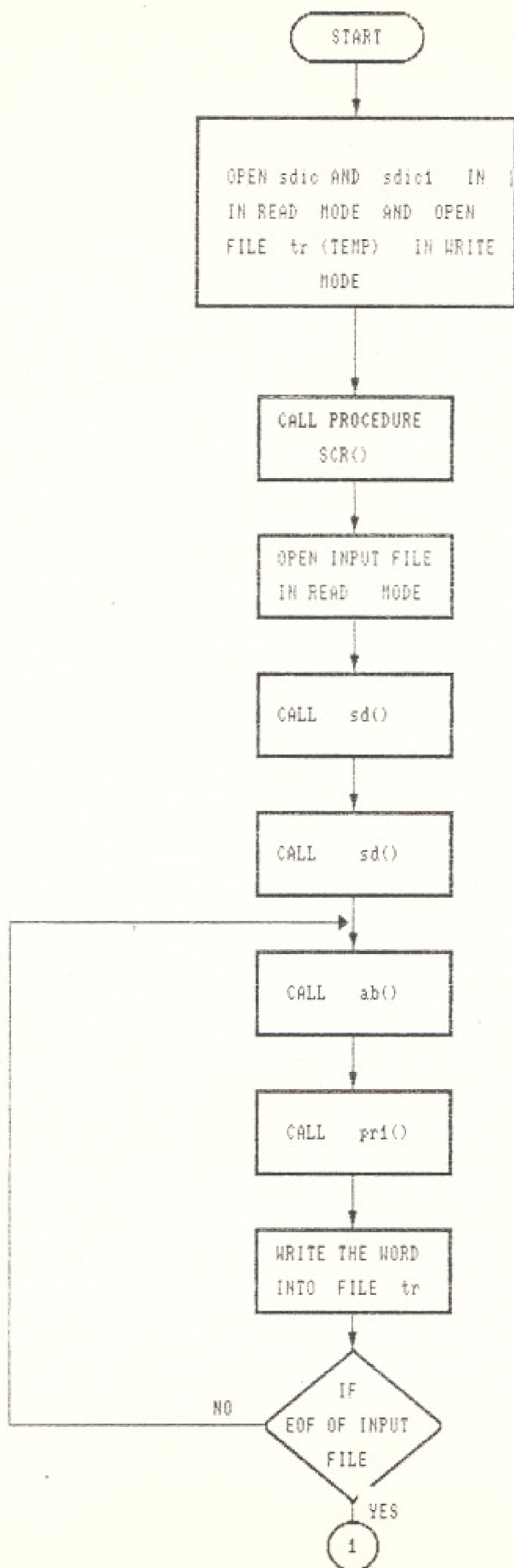
PURPOSE : THIS PROCEDURE IS DESIGNED TO HELP THE USER TO ADD NEW WORDS INTO THE DICTIONARY WHICH ARE ACTUALLY CORRECT BUT AS THEY ARE NOT IN THE DICTIONARY SO THEY CAUSE ERRORS.

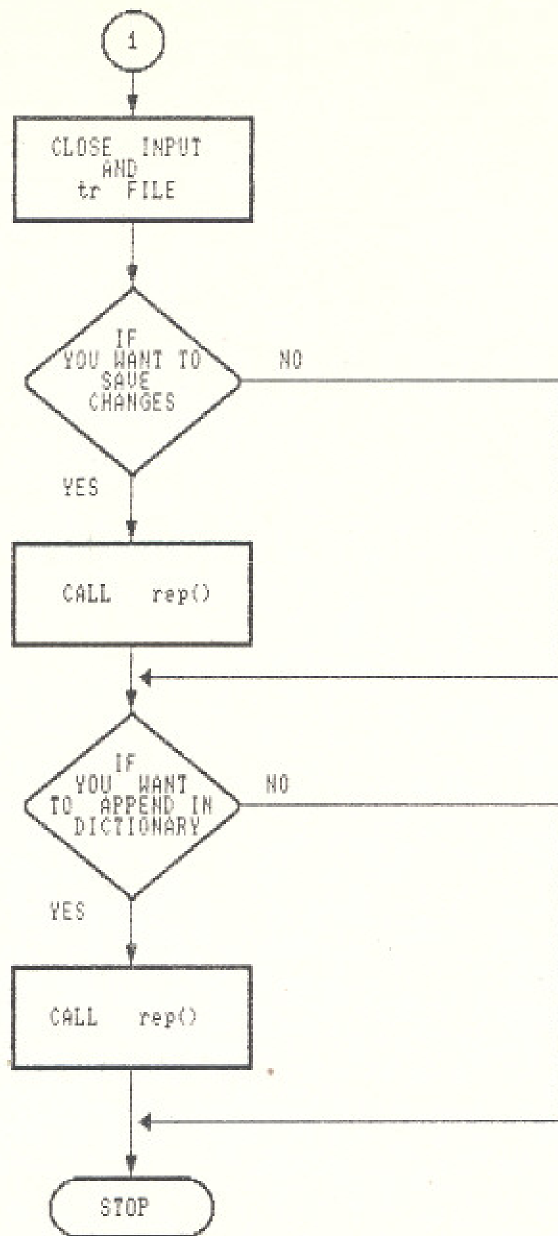
GLOBALS : mn[]- THIS ARRAY STORES THE WORD THAT IS TO BE APPENDED IN THE DICTIONARY.

==

SYSTEM FLOW CHARTS

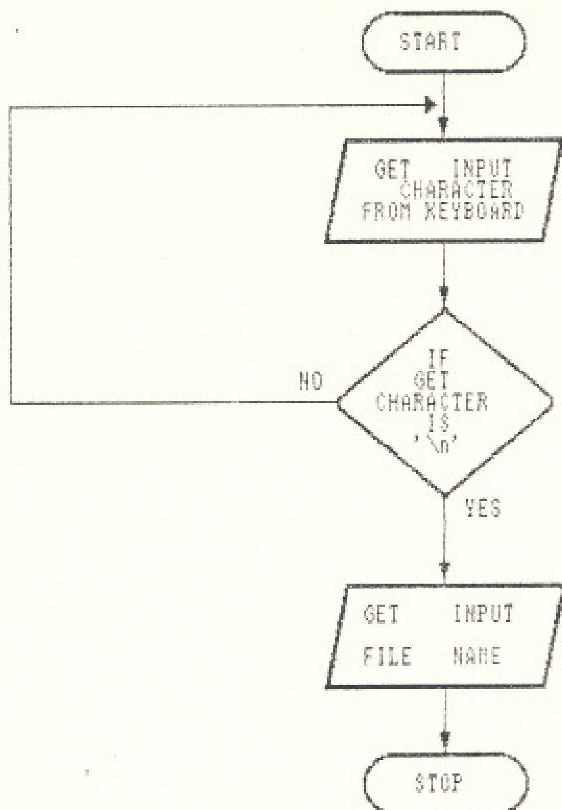
main()



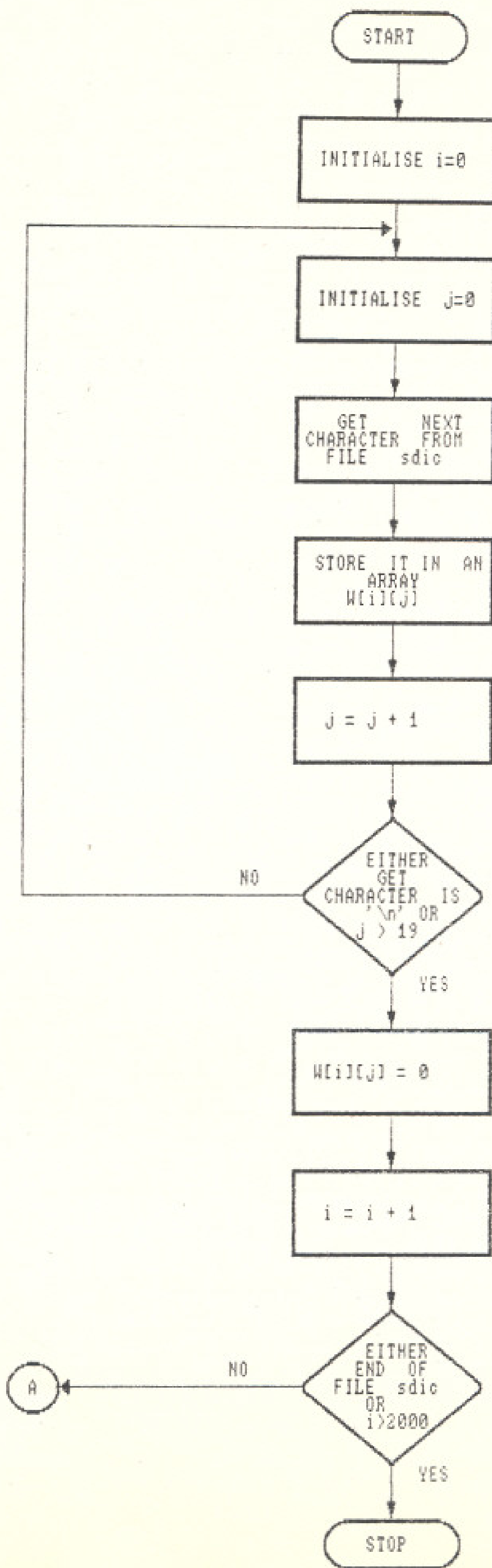


1
CLOSE INPUT
AND
tr FILE
IF
YOU WANT TO
SAVE
CHANGES
NO
YES
CALL rep()
IF
YOU WANT
TO APPEND IN
DICTIONARY
NO
YES
CALL rep()
STOP

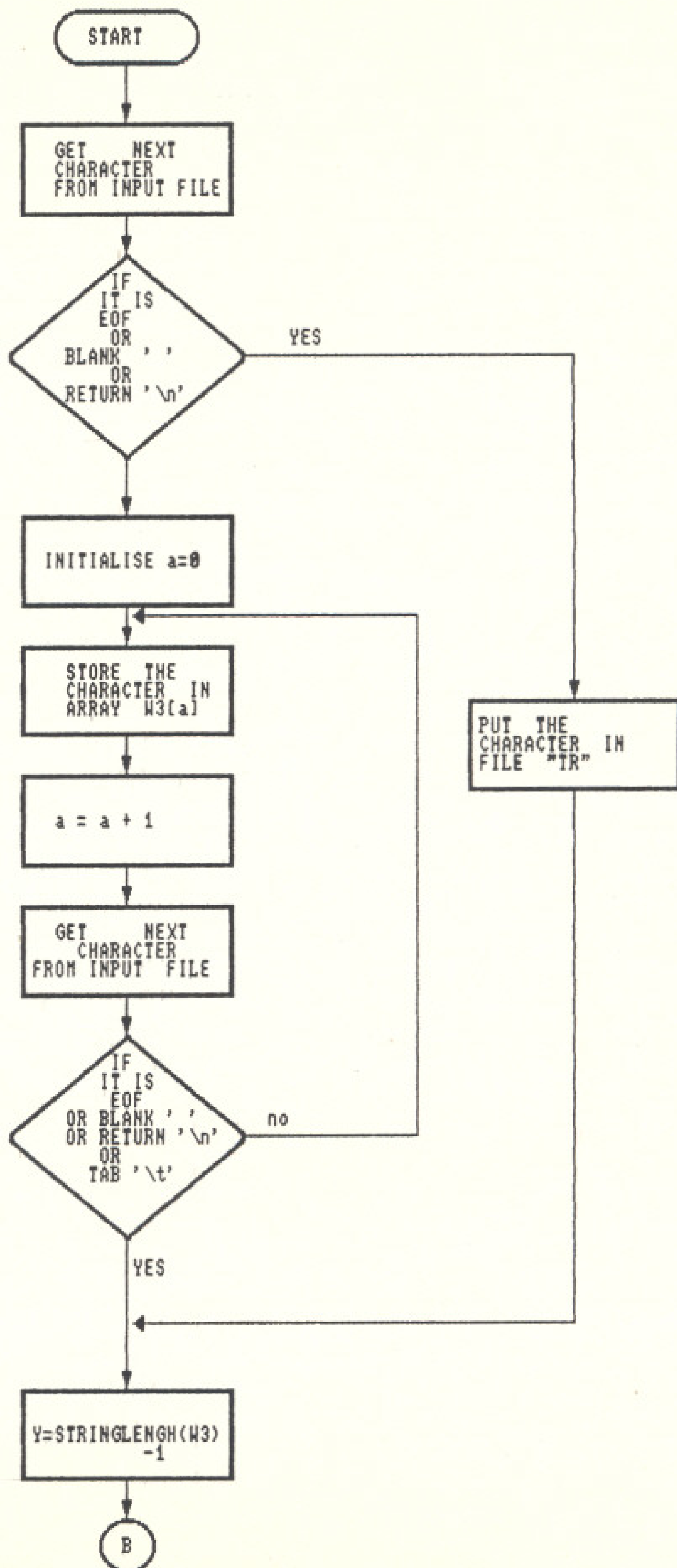
scr()

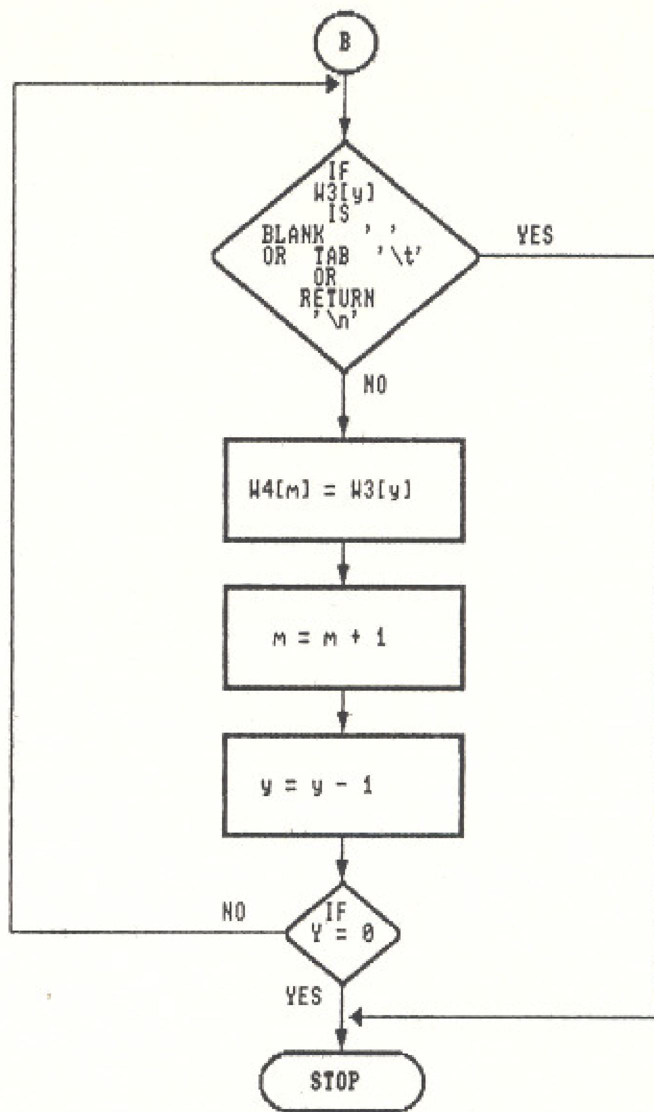


sd( )



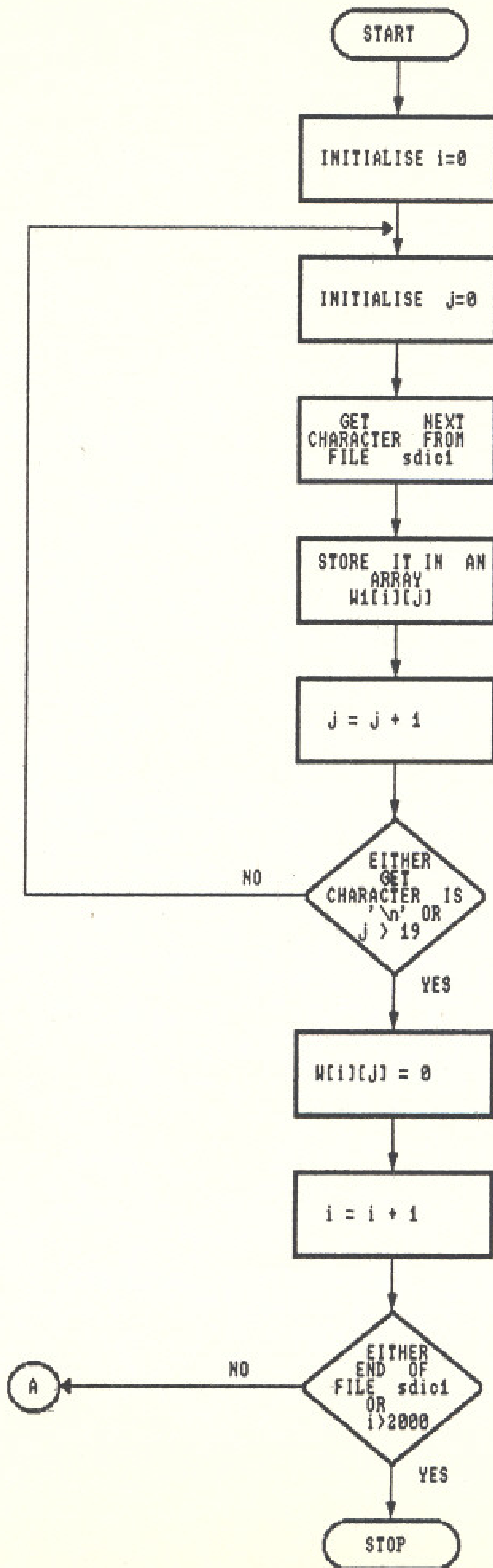
ab( )



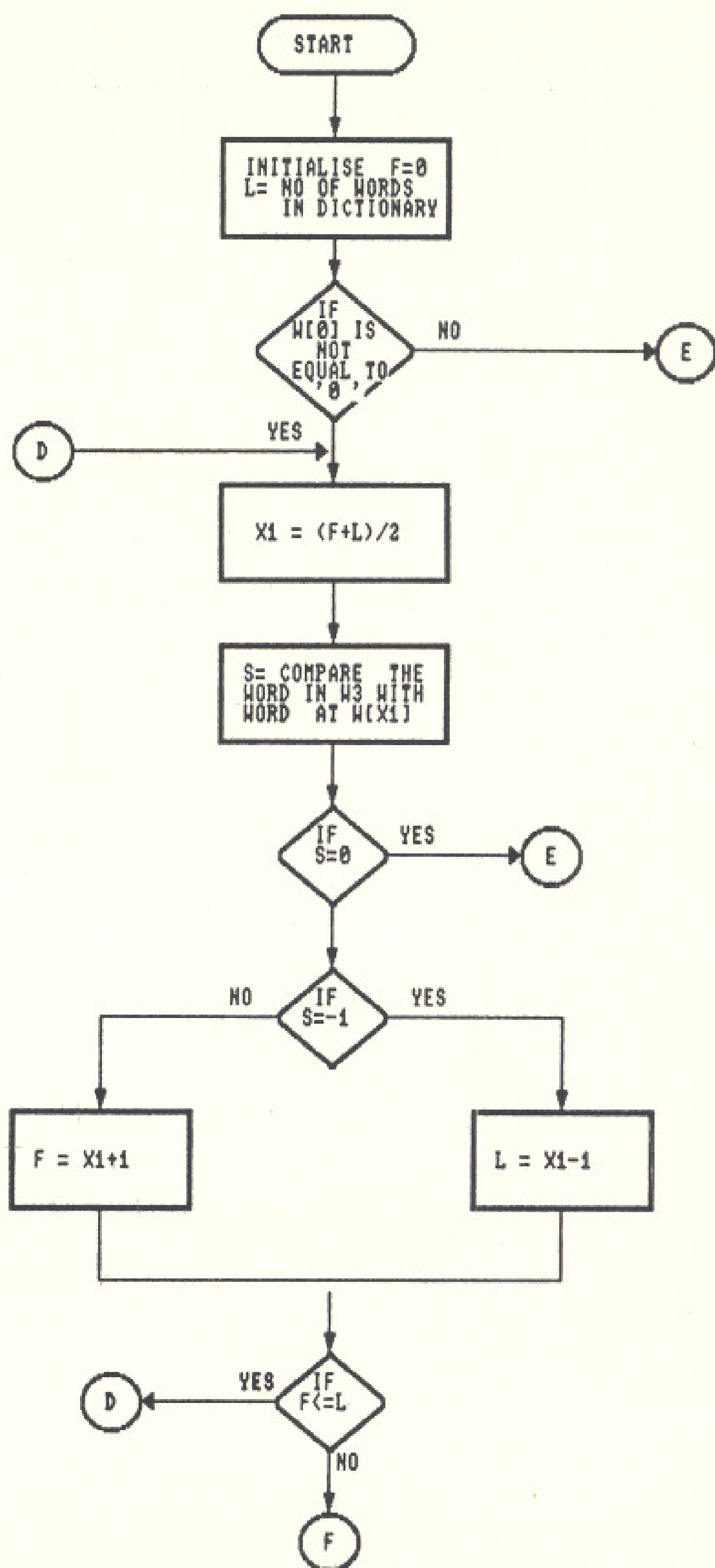


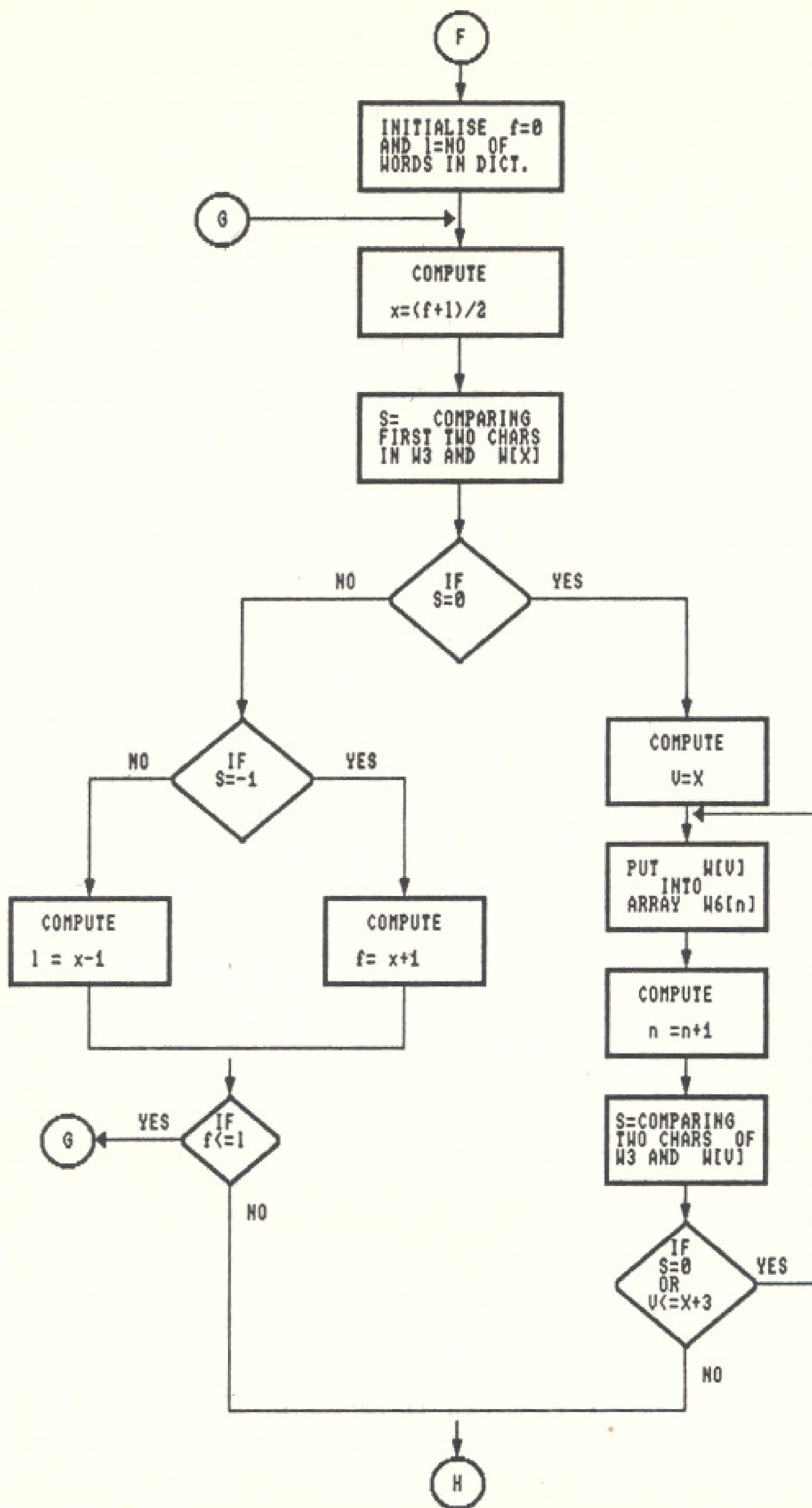
B
IF
W3[y]
IS
BLANK ' '
OR TAB '\t'
OR
RETURN
'\n'
YES
NO
W4[m] = W3[y]
m = m + 1
y = y - 1
NO
IF
Y = 0
YES
STOP

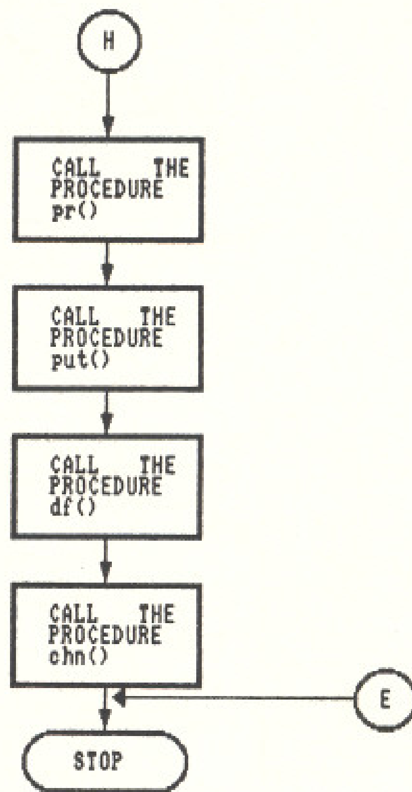
sd1( )



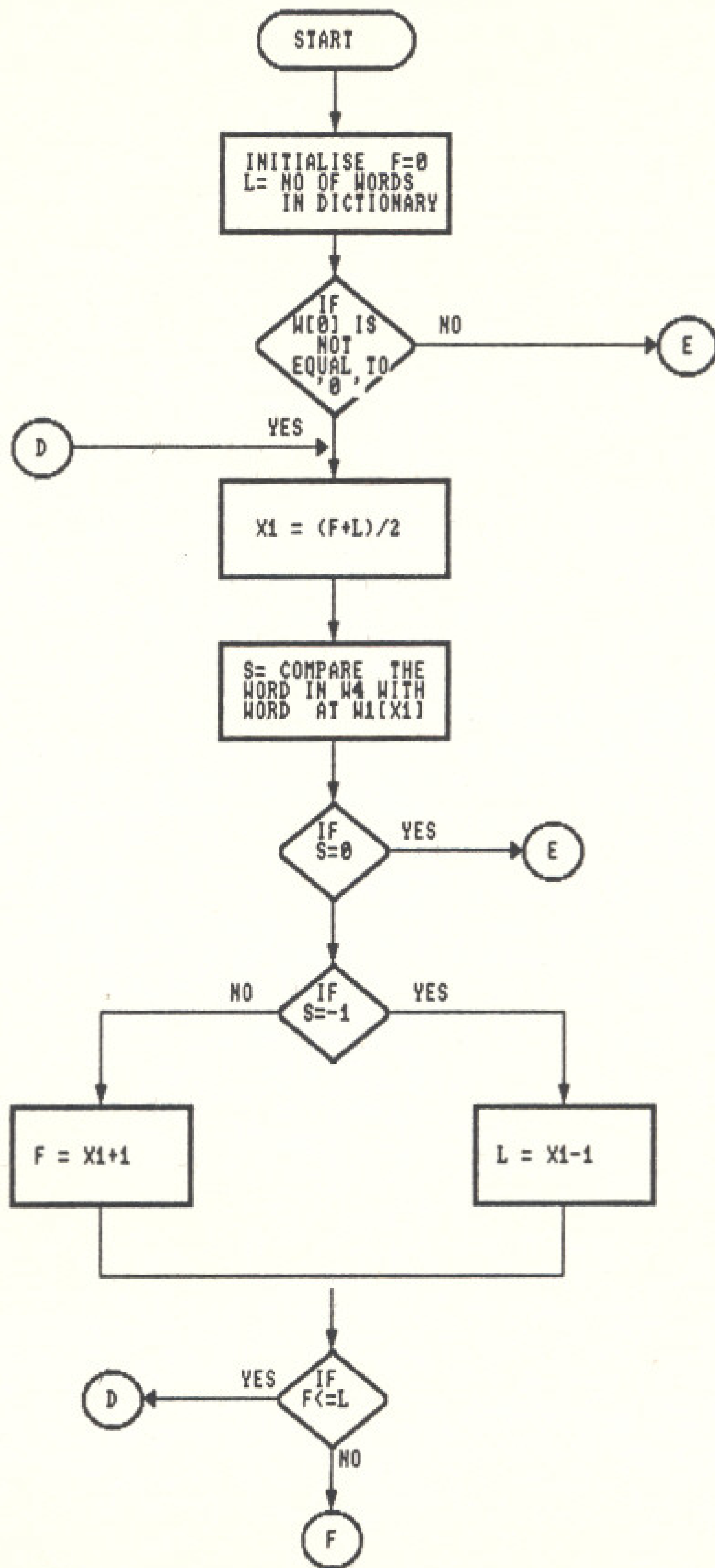
pr1()

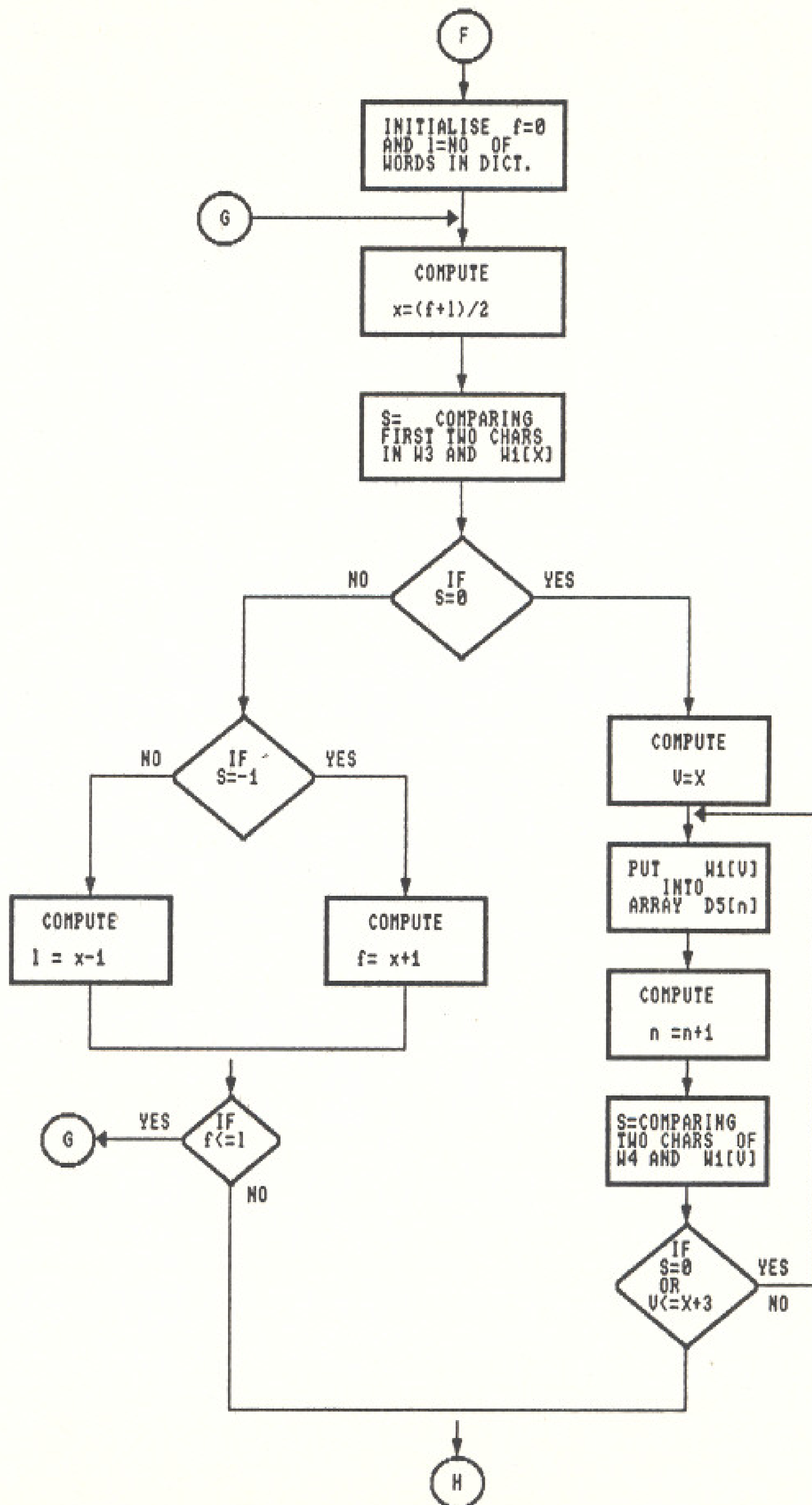


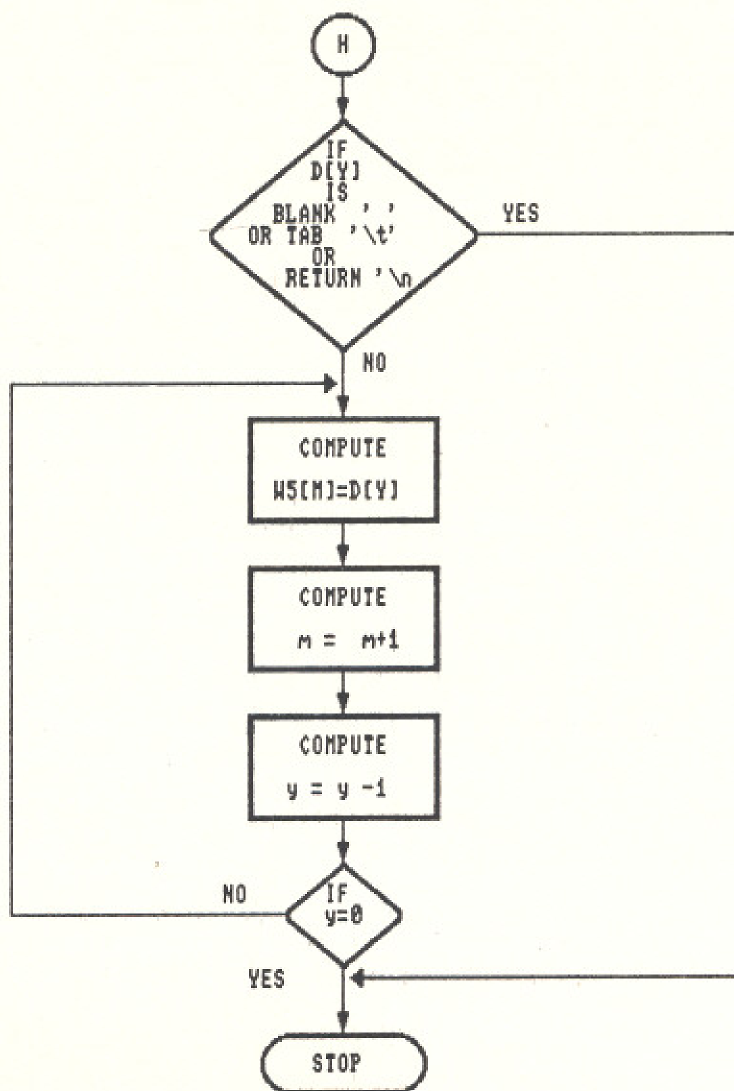

F
INITIALISE f=0 AND l=NO OF WORDS IN DICT.
G
COMPUTE x=(f+l)/2
S= COMPARING FIRST TWO CHARS IN W3 AND W[X]
IF S=0
NO
YES
IF S=-1
NO
YES
COMPUTE l = x-1
COMPUTE f= x+1
IF f<=l
YES
G
NO
COMPUTE V=X
PUT W[V] INTO ARRAY W6[n]
COMPUTE n =n+1
S=COMPARING TWO CHARS OF W3 AND W[V]
IF S=0 OR V<=X+3
YES
NO
H


H
CALL THE PROCEDURE pr()
CALL THE PROCEDURE put()
CALL THE PROCEDURE df()
CALL THE PROCEDURE chn()
E
STOP

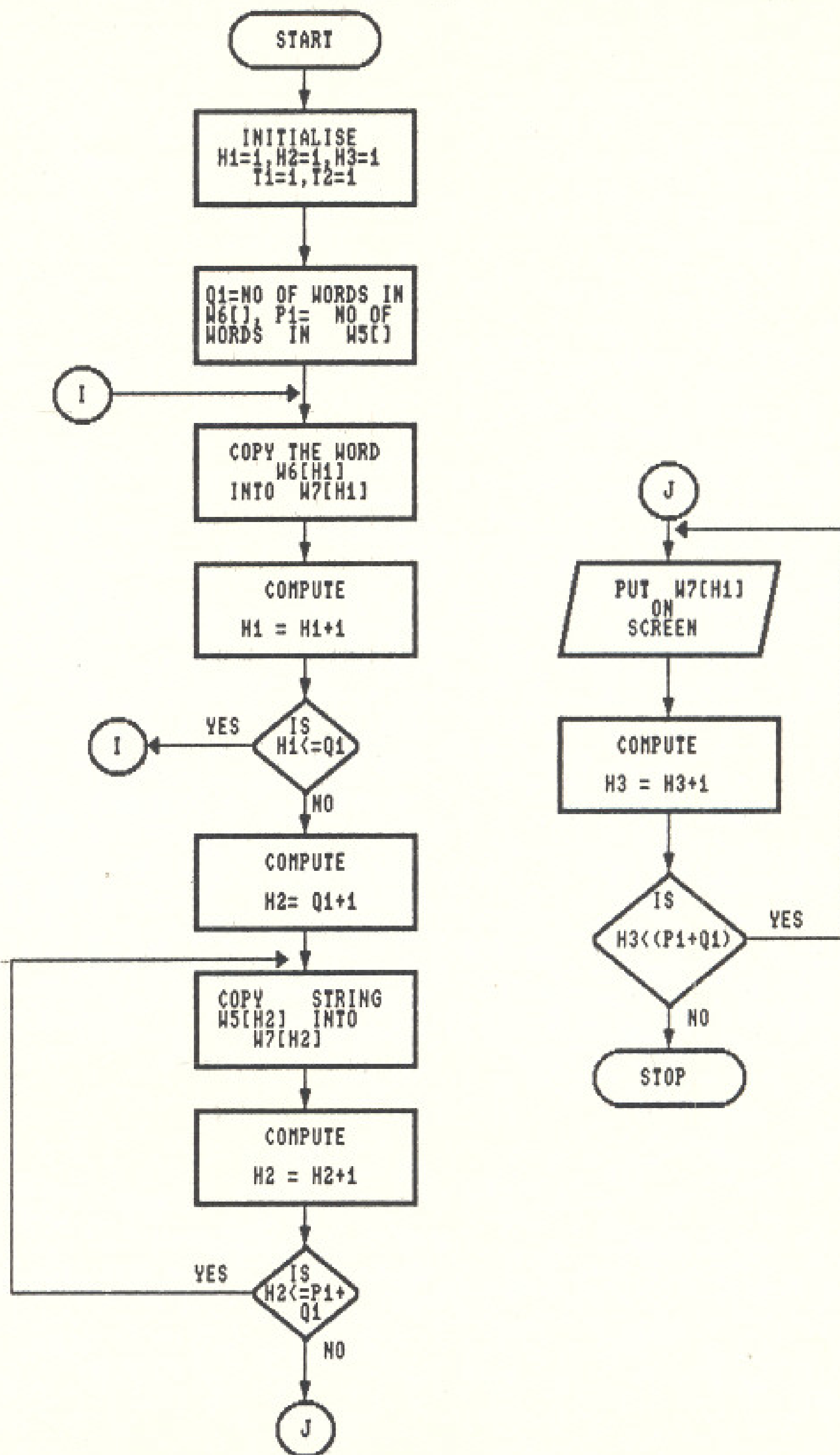
pr( )



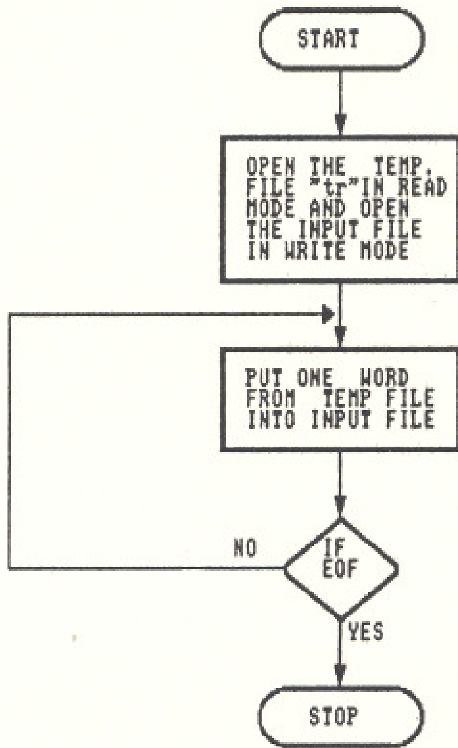

F
INITIALISE f=0 AND l=NO OF WORDS IN DICT.
G
COMPUTE x=(f+l)/2
S= COMPARING FIRST TWO CHARS IN W3 AND W1[X]
IF S=0
NO
YES
IF S=-1
NO
YES
COMPUTE l = x-1
COMPUTE f= x+1
IF f<=l
YES
G
NO
COMPUTE V=X
PUT W1[V] INTO ARRAY D5[n]
COMPUTE n =n+1
S=COMPARING TWO CHARS OF W4 AND W1[V]
IF S=0 OR V<=X+3
YES
NO
H


H
IF
D[Y]
IS
BLANK ' '
OR TAB '\t'
OR
RETURN '\n
YES
NO
COMPUTE
W5[M]=D[Y]
COMPUTE
m = m+1
COMPUTE
y = y -1
NO
IF
y=0
YES
STOP

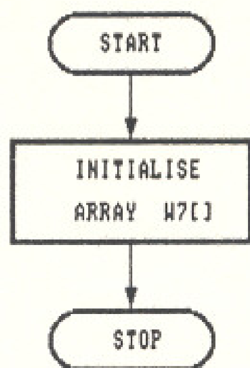
put( )



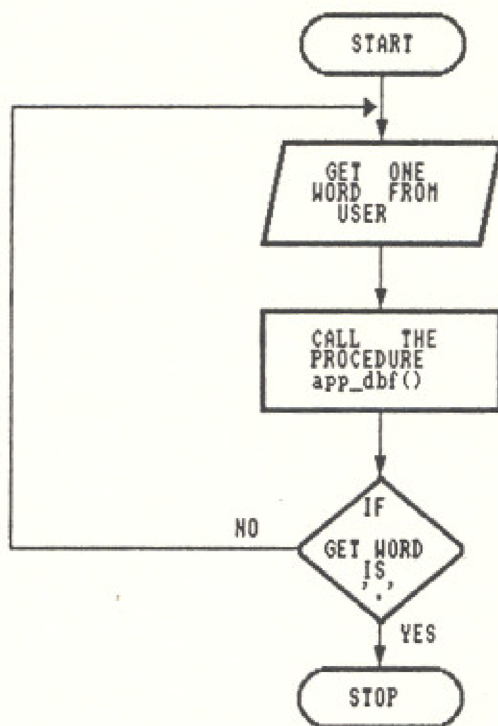
rep( )



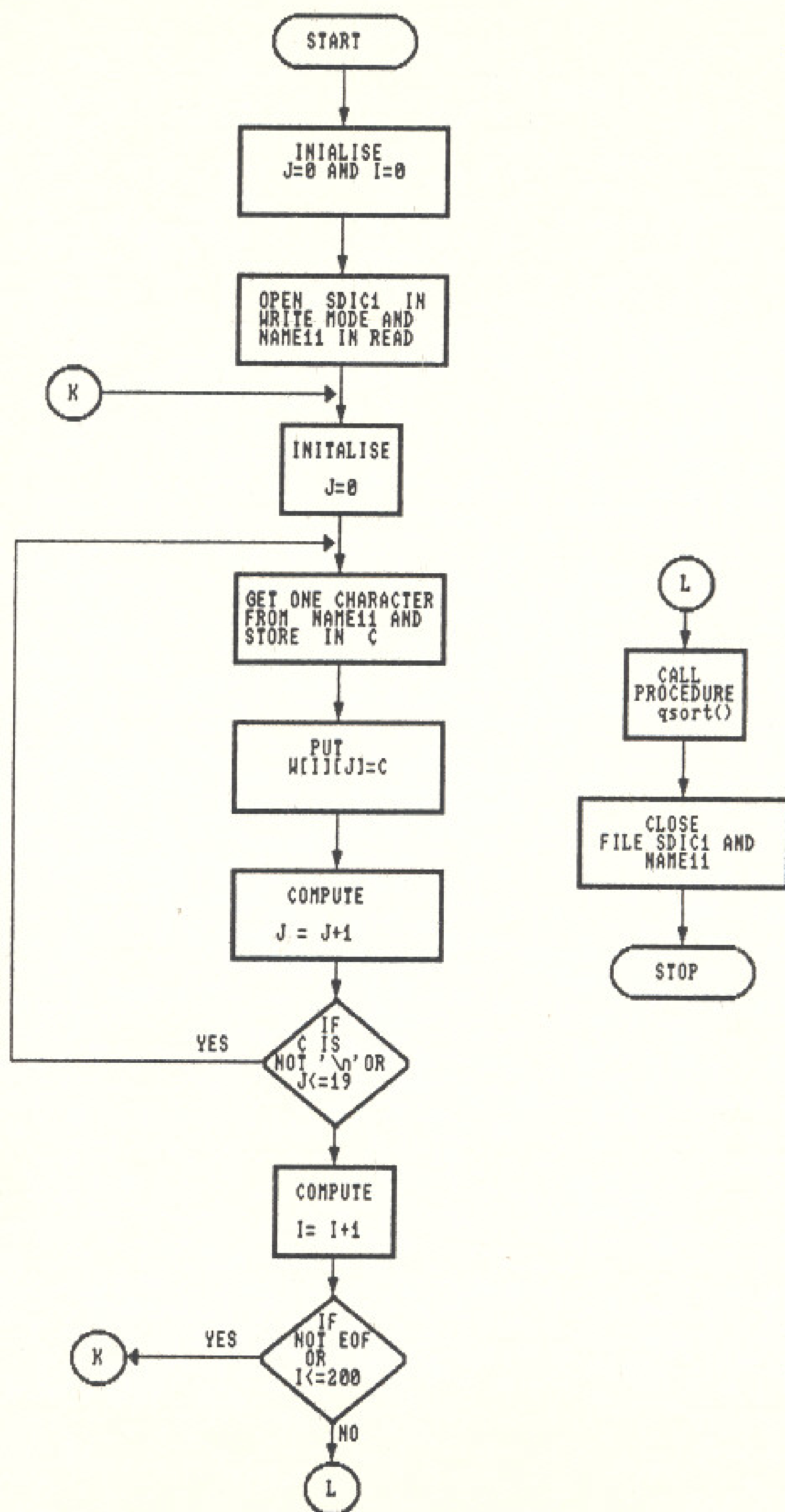
chn( )



append( )

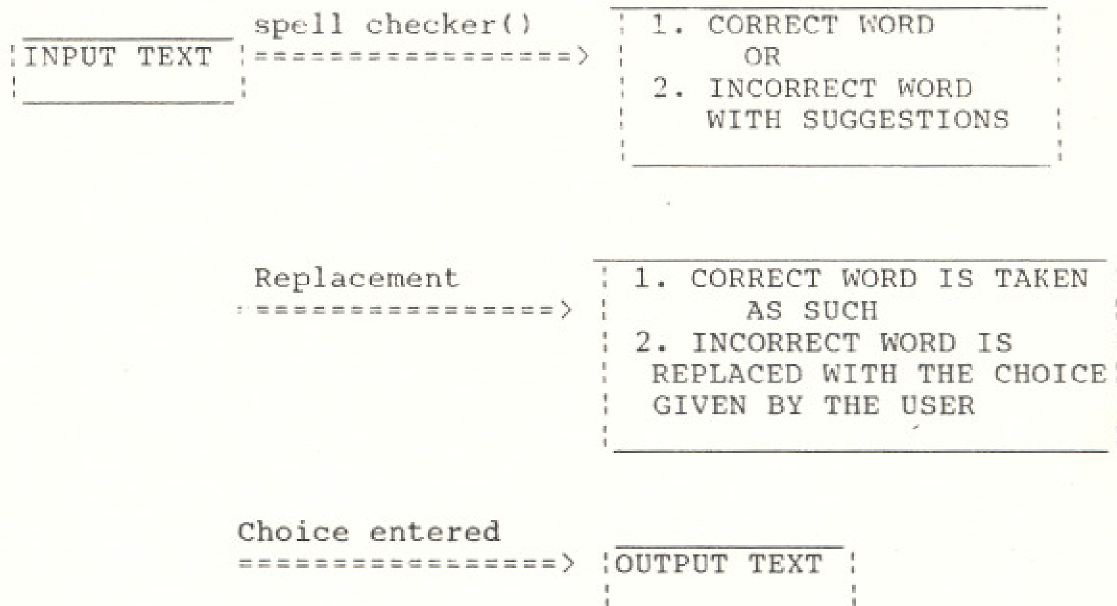


sort( )



IMPLEMENTATION

LOGICAL IMPLEMENTATION:



For testing the spell checker, we took a text file 'text' in which punjabi words were entered for checking their spellings. The words in this file were

ਉਗੀ ਉਪਦੇਸ਼ ਉਰਨ ਉਪਰਾਲਾ ਔਕੜਾ ਅੱਖਾਂ ਉਠਣਾ
ਉਪਜੌਣਰ ਅਪਘਾਤ ਅਨਾਜ ਉੱਟ ਉਜਾੜਾ

The spell checker starts working after pressing <return>. Then it asks the user "ENTER THE NAME OF FILE FOR CHECKING SPELLINGS" . We entered the name 'text' for checking the spellings.

The spell checker works for each word of the file 'text' and checks the spellings by comparing each word with the dictionary available to it.First of all, "ਛੁਟੀ" was checked and it was found correct .Then the word is displayed on the screen and the message is displayed "WORD IS CORRECT" . Similarly, "ਉਪਦੇਸ਼" was found correct. Then, "ਉਦਸ਼" was found incorrect .The word and its mirror image was displayed and message "WORD IS INCORRECT". The choices were also displayed on the screen

1. ਆਦਸ਼
2. ਉਠਣਾ
3. ਉਤਣਾ

we gave the choice 1 and "ਉਦਸ਼" is replaced with "ਆਦਸ਼" in temporary file 'tr' . Hence, all the words were checked by spell checker. Next incorrect word was "ਉਪਿਯੋਗਰ" The choices for this incorrect word were

1. ਉਪਯੋਗਤਾ
2. ਉਪਦੇਸ਼
3. ਉਪਰਾਮ
4. ਉਪਰਾਲਾ

we entered the choice 1 and "ਉਪਿਯੋਗਰ" is replaced with "ਉਪਯੋਗਤਾ" in file 'tr'. Then, "ਅਖਿਬਾਰ", "ਆਦਸ਼", "ਉਟ", are found correct and "ਉਕਾਸ਼" is found

incorrect. The word and its mirror image is displayed on the screen and the message that "WORD IS INCORRECT" . The choices displayed were as follows:

1. ਉਧਾਸ਼ਾ
2. ਉਪਗਾਸ਼ਾ
3. ਉਧਾਤ੍ਰਣਾ

We entered the choice 1 and "ਉਧਾਸ਼ਾ" word is replaced with "ਉਮਾਸ਼ਾ" word, but in the temporary file 'tr'.

After checking all the words, the spell checker asks the user "WHETHER YOU WANT TO SAVE CHANGES". We entered the 'yes' for our input file 'text' and all the changes were saved i.e. the original file is replaced with the new selected words from the choices. The spell checker asks "WHETHER YOU WANT TO APPEND IN DICTIONARY". We entered the choice 'no' for this and exit to the spell checker. Now. the input file 'text' is free from all spelling mistakes.

TOP LEVEL PSEUDOCODE:

```
Open files 'sdic' and 'sdic1' in read mode

Open file 'tr' in write mode

do scr

Open the text file in read mode which is to be checked for
spellings

do sd

do sd1

DO WHILE end of input file (these words are in input file)

           do ab

           do pr1

           write the word into temporary file 'tr'
ENDDO

close input file and temporary file 'tr'

IF you want to save changes
           THEN
                do rep
           ELSE
                goto next statement
ENDIF

IF you want to append in dictionary
           THEN
                do append
           ELSE
                goto next statement
ENDIF
```

PSEUDOCODE FOR scr()

```
DO WHILE character get from keyboard is not '\n'

            ask for '\n' character from keyboard

ENDDO

ask for the name of the input file from user which is to be
checked for spellings

get filename
```

PSEUDOCODE FOR sd()

```
Initialise 'i' equal to zero

DO WHILE  (either end of file 'sdic' or 'i' is greater than
           2000)

           initialise 'j' equal to zero

           DO WHILE (either get character is return or 'j'
                     is greater than 19)

                    get next character from file 'sdic'
                    store it in an array w[i][j]

                    increment 'j' by one
           ENDDO

           initialise array w[i][j] equal to zero

           increment 'i' by one
ENDDO
```

PSEUDOCODE FOR sd1()

```
Initialise 'i' equal to zero

DO WHILE  (either end of file 'sdic1' or 'i' is greater than
           2000)

           initialise 'j' equal to zero

           DO WHILE (either get character is return or 'j'
                     is greater than 19)

                    get next character from file 'sdic1'

                    store it in an array w1[i][j]

                    increment 'j' by one
           ENDDO

           initialise array w1[i][j] equal to zero

           increment 'i' by one
ENDDO
```

PSEUDOCODE FOR ab()

```
Get a single character from input text  file

IF it is end of file or blank or return or tab
           THEN
                put this  character into file 'tr'
           ELSE
                initialise 'a' equal  to zero

           DO WHILE (get character is not end of file or blank
                     or tab)

                        store the character in array w3[a]

                        increment 'a' by one

                        get next  character from input text file
           ENDDO
ENDIF

initialise 'y' equal to one less than length of string
contained in array w3

DO WHILE  'y' is not equal to zero
           IF   w3[y] is blank or tab or return
                THEN
                     goto next statement
                ELSE
                     store the contents of w3[y] in w4[m]

                     increment 'm' by one

                     decrement 'y' by one
           ENDIF
ENDDO
```

PSEUDOCODE FOR pr1()

```
Initialise f=0,l=no of words in dictionary - 1

DO WHILE  (f<=l)

          compute  x1=(f+l)/2

          compare the strings  in  w3[]  and  w[x1]

          store the value   returned  after  comparing in a
          variable    named s

          /* if two strings are equal  value of s will be '0'
          if w3  is greater than w[x1] value will be 1
          otherwise  the  value will be  -1              */

          IF   s=0
               THEN
                    return to  main
               ELSE
                    IF  s=-1
                        THEN
                               compute  l=x1-1
                        ELSE
                               compute  f=x1+1
                  ENDIF
          ENDIF
ENDDO

Initialise f=0,l=no of words in dictionary

DO WHILE  (f<=l)

          compute  x1=(f+l)/2

          compare only first two  characters of    the
          strings  in w3[] and w[x1]

          store the value   returned  after  comparing in a
          variable    named s

          /* if two strings are equal  value of s will be '0'
          if w3  is greater than w[x1] value will be 1
          otherwise  the  value will be  -1              */
```

```
          IF  s=0.
               THEN
                    compute v=x1.

                    DO WHILE  (s=0 and  v<=(x1+3))

                         put string w[v] into w6[n].

                         increment 'n'   by 1.

                         increment 'v'   by 1.

                         compare  strings  in w3  and  w[v]
                         and store the value in s.
                    ENDDO
               ELSE
                    IF  s=-1
                         THEN
                              compute  l=x1-1.
                         ELSE
                              compute  f=x1+1.
          ENDIF
ENDDO

do  pr().

do put().

do df().

do chn().
```

PSEUDOCODE FOR pr()

```
initialise f=0,l=no of words in dictionary - 1.

DO WHILE   (f<=l)

           compute   x=(f+l)/2

           compare the strings  in  w4[] and  w1[x]

           store the value   returned  after  comparing in a
           variable    named s.

           /* if two strings are equal  value of s will be '0'
           if w4  is greater than w1[x] value will be 1
           otherwise  the  value will be  -1.           */

           IF   s=0
                THEN
                     return to  main.
                ELSE
                     IF  s=-1
                         THEN
                                compute  l=x-1.
                         ELSE
                                compute  f=x+1.
                    ENDIF
           ENDIF
ENDDO

Initialise f=0,l=no of words in dictionary .

DO WHILE  (f<=l)

           compute  x1=(f+l)/2 .

           compare only first two  characters of    the
           strings    in w4[] and w1[x]    .

           store the value   returned  after  comparing in a
           variable    named s.

           /* if two strings are equal  value of s will be '0'
           if w4  is greater than w1[x] value will be 1
           otherwise  the  value will be  -1.           */
```

```
          IF  s=0.
               THEN
                    compute v=x.

                    DO WHILE  (s=0 and  v<=(x+3))

                         put string w1[v] into w5[n].

                         increment 'n'   by 1.

                         increment 'v'   by 1.

                         compare  strings  in w4  and  w1[v]
                         and store the value in s.
                    ENDDO
               ELSE
                    IF  s=-1
                         THEN
                              compute  l=x-1.
                         ELSE
                              compute  f=x+1.
          ENDIF
ENDDO

initialise  k=1.

DO WHILE   k<=n

          initialise   e=0.

          initialise   l1= length  of the  string  in  d[k].

          DO WHILE     l1>=0.

               IF  d[k][l1]  is not( blank or return or tab)
                    THEN
                         compute  w5[k][e] = d[k][l1].
               ENDIF

               compute w5[k][e]=0.

               increment e by 1.

               decrement l1 by 1.
          ENDDO

          increment k by 1.
ENDDO.
```

```
          IF  s=0.
               THEN
                    compute v=x.

                    DO WHILE  (s=0 and  v<=(x+3))

                         put string w1[v] into w5[n].

                         increment 'n'   by 1.

                         increment 'v'   by 1.

                         compare  strings  in w4  and  w1[v]
                         and store the value in s.
                    ENDDO
               ELSE
                    IF  s=-1
                         THEN
                              compute  l=x-1.
                         ELSE
                              compute  f=x+1.
          ENDIF
ENDDO

initialise  k=1.

DO WHILE   k<=n

          initialise   e=0.

          initialise   l1= length  of the  string  in  d[k].

          DO WHILE     l1>=0.

               IF  d[k][l1]  is not( blank or return or tab)
                    THEN
                         compute  w5[k][e] = d[k][l1].
               ENDIF

               compute w5[k][e]=0.

               increment e by 1.

               decrement l1 by 1.
          ENDDO

          increment k by 1.
ENDDO.
```

PSEUDOCODE FOR df()

```
initialise  k2= no of  options available(words  in array  w7)

DO WHILE  q2 > k2

           ask  user  to enter  his  choice to  replace with
           the incorrect  word  with .
ENDDO

IF  q2=0
           THEN
                return  to calling procedure.
           ELSE
                replace  string  in w3  with  w7[q2].
ENDIF
```

PSEUDOCODE FOR put()

```
Initialise h1,h2,h3,t1,t2 equal to one

initialise q1 equal to the number of words in array w6

initialise p1, the number of words in array w5

DO WHILE  h1 is less than q1

          copy the word from w6[h1] into w7[h1]

          increment h1 by one
ENDDO

initialise h2 equal to (q1+1)

DO WHILE  h2 is less than (p1+q1)

          copy the word w5[h2] into w7[h2]

          increment h2 by one
ENDDO

DO WHILE  h3 is less than (p1+q1)

          put w7[h3] to the screen

          increment h3 by one
ENDDO
```

PSEUDOCODE FOR chn()

Initialise array w7

PSEUDOCODE FOR rep()

```
open the temporary file "tr" in read mode.
open the input text file in write mode.
DO WHILE not end of "tr" file
            put one word from "tr" file into input file.
ENDDO.
```

PSEUDOCODE FOR append()

```
DO WHILE    get  character  from  key-board  is  not   '.'
            ask  the  user  for  next  word  to append.
            do app_dbf
ENDDO
```

PSEUDOCODE FOR sort()

```
Open  file  sdic1  in write  mode
Open  file  name11 in read  mode
Initialise  i=0
DO WHILE  not  end of file name11 or  i<=200
          compute j=0
          DO WHILE  get character is not return '\n' or j<=19
               compute  w[i][j]=c
               increment i by 1
               get  next  character from file  name11
          ENDDO
ENDDO
do qsort()
close  files
close  files  sdic1 and name11
```

PSEUDOCODE FOR reverse()

Initialise t=0

Open file dic.txt in read mode having pointer 'fp'

Open file namell in write mode having pointer 'fp1'

DO WHILE t=0

do rev(fp,fp1)

ENDDO

PSEUDOCODE FOR rev()

Get one character from input file having file pointer 'p' and store in 'c'

IF c is end of file
THEN
compute t=1
ENDIF

IF c is not equal to return '\n' or eof or blank ' '
THEN

do rev(p,p1)
ENDIF

Put one character to the output file having pointer 'p1'

ENVIRONMENT

For the development of any good computerised system, it is essential to have a good blend of hardware and equally good system software available. NIC being the pioneering national computer organisation has all the latest technology available at its state headquarters.

Following is a brief description of hardware and software used during the course of project development :

HARDWARE

The whole work is done on HCL PC SUPER-AT and L32QI Dot Matrix Printer whose system configuration is as follows :

HCL Super Chip PC/AT 386

It is a multiuser system with six dumb terminals attached to it.

MAKE	: HCL Super Chip PC/AT 386
MAIN MEMORY	: Base 640 KB Extended 7168 KB
SECONDARY MEMORY	: 1 Hard Disk of 300 MB
MICROPROCESSOR	: INTEL 80386 running at 20 MHz
MATH COPROCESSOR	: INTEL 80387 running at 20 MHz

TERMINALS	: 6 HCL make dumb terminals
FLOPPY DRIVE	: 5-1/4" High Density Drive
CARTRIDGE TAPE DRIVE	: Standard Cartridge Drive supporting 60 MB or higher

GIST TERMINALS:

Since our project is to deal with Punjabi, so for doing work in Punjabi we were provided with the **GIST** (Graphics and Intelligence based Script Technology) terminals. GIST makes it possible to interact with the computer in most Indian languages and a few foreign languages. It adapts available English software to provide a universal script processing environment.

Compatability of GIST Terminals

The GIST terminal provides emulation for

(i) IBM ANSI terminal

(ii) DEC VT-52 terminals

(iii) DEC VT-100 terminals

(iv) DEC VT-220 terminals

(v) DEC VT-320 terminals

So computers using any of these terminal types can be upgraded for multi-lingual script interaction by replacing it with a GIST terminal.

Since the terminals attached to the HCL-386 system are having the type DEC VT-100, so the system has been upgraded to have interaction with the GIST terminals.

PRINTER

The L32QI Printer is a parallel interface dot matrix printer which provides a range of types of high quality printing while maintaining a high standard of performance and reliability. The following features are provided :

- 132 Column printer at 10 cpi
- Bidirectional printing using logic seeking to maximize printing efficiency.
- Standard IBM PC compatible character sets plus eight software selectable alternate international character sets (EPSON FX80 Compatible). Printable draft mode characters may be redefined using FX-80 compatible down loading.
- Correspondence quality printing selectable by push button,

microswitch or software command.

- Printing of underline, double width, emphasized, double strike, subscript or superscript characters.

- High resolution Bit Image Graphics. IBM compatible line and Mosaic Graphics in both Draft and CQ modes.

- Switch and software selectable form lengths.

- Various vertical and horizontal tab settings.

- Rear and bottom paper insertion path for fanfold paper, front insertion for cut sheets. Sensor switch detects when paper required.

- Operational control panel with POWER ON and ONLINE indicators and ONLINE, LINE FEED/FORM FEED and Correspondence Quality (CQ) push buttons.

- Integral anti-noise cover with paper tear bar.

- A serial interface option can be implemented by the user at site.

SYSTEM SOFTWARE

Operating System (XENIX) :

XENIX is the most popular implementation of the UNIX operating system at the micro level. This system is at the heart of more working multi-user business solutions than all other UNIX system versions combined. Today, the SCO XENIX System V offers the most productive, powerful and flexible operating environment available for PCs.

A powerful multi-user, multi-tasking operating system, SCO XENIX System V allows file sharing between independent terminal operators or between application programmers under the control of one or more users. Originally developed by Microsoft Corporation, XENIX has become the standard UNIX System implementation for personal computers today, installed on more than 85% of all Microcomputers running any version of the UNIX System world-wide.

SCO XENIX System V consists of three modules :

(a) Operating System

(b) Development System

(c) Text Processing System

(a) Operating System :

It contains the full set of XENIX utilities required to run business applications, administer the system, edit files and communicate with others. , csh(advanced c shell), vsh(visual shell), vi(visual editor), electronic mail, uucp and over 100 other UNIX System commands.

The hard disk can be shared between seperate XENIX and DOS partitions, allowing the user to alternate between operating systems. Utilities are provided for moving files between the XENIX and DOS environments.

(b) Development System :

It supplies all the tools needed to write 'C' and assembly language programmes as well as providing a powerful DOS cross development environment to create programs for both DOS and UNIX.

(c) Text Processing System :

It contains tools for the preparation of large or complex documents. Included are the nroff, document formatter, macro packages and special formatting tools.

Our system has been developed in 'C' language because it is a powerful tool for system programming.

'C' Language Characteristics:

C is a language of functions,data types, assignments and flow control. To program in 'C', a function must be called and most functions return values. The value returned from a function, the value of a data variable , or the value of a constant can be used in an assignment statement to change the value of another variable. With the addition of flow control- if, while, for, do, switch the 'C' language takes on the structure of a high level language, enabling and promoting good programming style.

In 'C', pointer variable can be declared that points to any data type. The address arithmetic of 'C' is sensitive to the properties of the pointer being adjusted. Pointers to functions are also supported.

'C' functions are recursive by default. A function can be coded that does not work in a recursive operation, but the language tends to naturally support recursion and requires little recursion programming effort.

'C' allows to develop a program in multiple source files that are independently compiled. The relocatable object modules of individual source files are linked into a single executable program.

The 'C' language has no input/ output operations. The compiler compiles a language of functions, and all input and output is done with functions. Because of this feature, a standard library of functions has evolved, and this standard is what gives C its most endearing quality.C language code can be portable.

CONCLUSION & REMARKS

CONCLUSION AND REMARKS

The work done during the System Development Project is discussed in detail in the chapter 'Design'. The spell checker developed checks and corrects the spellings of Punjabi and other Indian languages text. Although it works well, yet there are some **enhancements** for the efficient working of the spell checker which are as follows:

1.Linking to an Editor:

The spell checker can be linked to an editor and thus it will work more efficiently. It can become more user friendly. The user can work with the spell checker on-line . Then its another advantage will be that it will check the spellings in between the file. The user can see the incorrect words and correct words on the screen where file exists. Thus, linking to an editor is important enhancement for spell checker.

2.Compression of dictionary:

The dictionary of words can be compressed / compacted to save the memory space as there are thousands of words and they are sorted. So compression is easy. There are standard 100 compression algorithms existing at this stage. Any suitable method can be applied for compression.

We know dictionary is sorted. So generally, adjacent words have some characters common. So instead of writing same sequence again and again, a code can replace that and memory space can be saved. e.g.

1. ਉਮਤਗ
2. ਉਮ
3. ਉਮਤਤ

"ਉਮ" is common in these words. So it can be coded.

Compression algorithm can be implemented in one of three ways:

(a)Software:

Implementation of this algorithm takes 30 microseconds to compress a character. Thus, it is not very efficient method of compression.

(b)Microprogramming:

Implementation of this algorithm takes 2 microseconds to compress a character. When there in no hardware facility available, then this is the best method of compression.

(c)Special Hardware:

Implementation of this algorithm takes fraction of microsecond to compress a character. Thus, this is the best and most efficient method of compression but for implementing this method, hardware facility should be available.

3.Organisation of Dictionary:

We know that all words of a language have different frequency of their usage in text.e.g. I, is, am, are etc. words have highest frequency in a English text. Some words have very low frequency of their occurrence in text. e.g. name of any city or person. So we can divide the dictionary in two parts.

One part that contains a pool of words which are commonly used and these words are chosen by experience. These words are kept in RAM always because to search these words, it will not take so much time. The words of Punjabi can be

ਉਹ ਤੇ ਇਹ ਵਿਚ ਹਨ ਵਿੱਚੋਂ ਵਿਚੋਂ

Second part of the dictionary is kept on the peripheral device i.e. on the hard disk. This part will contain the words like

ਉਸਤਤ ਉਪਦੇਸ਼ ਉਪਰਾਮ [illegible]

which are not commonly used.

ANNEXURES

ANNEXURE - I

ASCII CODES FOR PUNJABI LETTERS:

ਉ	424	ਢ	448	ਔ	433
ਅ	420	ਣ	449	ਐ	429
ਇ	422	ਤ	450	ਆ	421
ਈ	423	ਥ	451	ਊ	425
ਸ	471	ਦ	452	◌ੇ	485
ਹ	472	ਧ	453	ਏ	427,428
ਕ	435	ਨ	454,455	◌ੇ	480,481
ਖ	436	ਪ	456	◌੍ਹ	488
ਗ	437	ਫ	457	ਿ◌	475
ਘ	438	ਬ	458	◌ੁ	477
ਙ	439	ਭ	459	◌ੌ	417
ਚ	440	ਮ	460	◌ੌ	418
ਛ	441	ਯ	461,462	◌ੈ	486
ਜ	442	ਰ	463,464	◌ੈ	482
ਝ	443	ਲ	465,466,467	◌ਾ	474
ਞ	444	ਵ	468	◌ੀ	476
ਟ	445	ੜ	447+489	◌ੂ	478
ਠ	446	ਸ਼	469,470		
ਡ	447	ਓ	431,432		

ANNEXURE - II

ਪੰਜਾਬੀ ਸ਼ਬਦ ਕੋਸ਼

ਉਈ

ਉਸ

ਉਸਲਾ

ਉਹਨੂੰ

ਉਖੜਨਾ

ਉੱਗਣਾ

ਉਗਲ

ਉਗਾਹੀ

ਉੱਘ

ਉਚਾਈ

ਉਚਿਤ

ਉੱਚਾ

ਉਚਾਰਨ

ਉਜਲਾ

ਉਜਾੜਨਾ

ਉਠਣਾ

ਉਡਣਾ

ਉਡੀਕ

ਉਤਰਨ

ਉਤਸ਼ਾਹ

ਉਤਸੁਕ

ਉਤਪਾਦਨ

ਉੱਤਮ

ਉੱਤਰ

ਉਤਰਾਧਿਕਾਰੀ

ਉਤਸ਼ਾਹ

ਉਥੋਂ

ਉਥੇ

ਉਦਘਾਟਨ

ਉਦਯੋਗ

ਉਦਾਸ

ਉਦਾਹਰਣ

ਉਦੇਸ਼

ਉਦੋਂ

ਉਧਰ

ਉਧਾਰ

ਉਨੱਤ

ਉਪਹਾਸ

ਉਪਦੇਸ਼

ਉਪਯੋਗਤਾ

ਉਪਰਾਲਾ

ਉਪਰੋਂ

ਉਭਰਨਾ

ਉਮਰ

ਉਲਟ

ਉਲਝਣ

ਉਲੰਘ

ਉਵੇਂ

ਊਂਘਣਾ

ਊਂਘਲਾ

ਊਣਾ

ਊਲਜਲੂਲ

ਓਟ

ਓਦੋਂ

ਅਸਚਰਜ

ਅਸੰਜਮ

ਅਸਤੇ

ਅਸਥੀਰ

ਅਸ[illegible]

ਅਸ[illegible]

ਅਸ[illegible]

ਅਸ[illegible]

ਅਸਮਤੁਲਨ

ਅਸਮਾਨ

ਅਸਲਾ

ਅਸਲੀਅਤ

ਅਸਲੀਲਤਾ

ਅਸਲੋਂ

ਅਸਾਰ

ਅਸਾਧਾਰਨ

ਅਸਾਨ

ਅਸੁਰੱਖਿਅਤ

ਅਧੂਲ

ਅਹਿਸਾਨ

ਅਹਿੰਸਾ

ਅਧਿਮਾਯ

ਅਧਿਧਾਰ

ਅਨੁਭਵ

ਅਨੁਖਣ

ਅਫ਼ਸੋਸ

ਅਨਾਜ

ਅਫ਼ਸਰੀ

ਅੱਸੀ

ਅਧਿਕਾਰਤ

ਅਸਥਾਨ

ਅੱਖਰ

ਅਫ਼ਸਰ

BIBLIOGRAPHY

BIBLIOGRAPHY

1.Jean-Paul Tremblay, Paul G. Sorenson, "An Introduction to Data Structures With Applications", (Tata McGraw-Hill Publishing Company Limited).

2.Ellis Horowitz, Sartaj Sahni, "Fundamentals of Data Structures in Pascal".

3.Whitten,Bently,Barlow, "System Analysis and Design Methods", (galgotia publication pvt. Ltd.)

4.Stan Kelly-Bootle, "Mastering Turbo C", (BPB publications).

5.Steven Holzner,Peter Norton,"C Programming:The Accessible Guide top Professional Programming",(Prentice Hall of India Limited).

6.Brian W. Kernighan, Dennis M Ritchie,"The C Programming Language",(Prentice Hall of India Limited).

7.Manuals:

(a) SCO Lyrix Release 6.1.0
(b) User Manual of "DCM GIST Terminal"