The issues

During the latter half of the last decade a great deal of interest (and concern) was expressed about the use of computers in environments where the Roman character set is not commonly used. In some cases, unfortunately, considerable dissatisfaction arose against the computer manufacturers in the Western world because computers were oriented towards the Roman alphabet and, in fact the English language. In those countries in which the loudest concern was expressed, computers were being used by those educated in the West, in the English tongue.

There was no deliberate intent in the West to design computers in such a way as to make it difficult to process non-Roman scripts. They were built to conform to this horizon - and they conform not only in terms of processing of scripts, but also in terms of logic.

One could take the attitude that those languages which do not fit into the scheme should be modernized, and forced into it. This was the approach taken by Japan. But if a people wishes to retain their identity then it is reasonable to look for a means to help them find solutions.

Thus we are faced with the problem of trying to adapt a technology to an environment which is unlike that in which it was developed. This problem is more than just character sets. It is also a problem of language and logic. Take programming languages for example.
are not only based on the English language, they are also based on
Western logic constructs. How does an individual, whose first language
is not English, whose culture is not Western, learn to program
efficiently using our programming languages? Perhaps, as in some
South American Indian languages, there is a modal logic which permits
many different states rather than just true or false. The solution to
such a problem is more than just changing the language to use the
native character set (e.g. Arabic COBOL)! The problem of character
sets - interacting with the computer in your own language if not your
own logic, is relatively simple in comparison to the first problem.
This paper will address only the problem of character sets (the other
problem is probably worth a PhD in linguistics).

When we consider character sets, we must first consider two aspects of
caracter sets that are not even computer-based problems. The first
aspect is that of calligraphy. There must be some standard
displayable form for a character or idea. Without this
standardization, not only is computerization next to impossible, so is
every other activity that requires the printed form of the language.

Given that a standard displayable form of the language exists, the
aspect of lexicography must be considered. By lexicography, we mean
the sequence by which our symbols are "ordered" - the sequence in
which they would appear in a dictionary. If there is such a sequence,
it might become possible to use the computer to sort sets of data,
without having to write special-condition checks. (This is no simple
problem. For example, a traditional scholar of the Arabic language
may place words in a dictionary in a different order than would a
modernist; this is true of Hebrew as well. Additionally, does a lexicographical sequence exist for all languages? Does it exist for Chinese, for example?).

Given that we have overcome the problems of calligraphy and lexicography (as has been done in the Soviet Union, Israel, the Arab countries, and most recently India) we must determine whether the resulting set is conformable to ASCII/EBCDIC encoding. For it to be conformable implies that the set is easily mapped in 128/256 different codes (including any special characters). Again, the Russians, the Israelis, the Arabs, and the Indians have succeeded in deriving 7- and 8-bit representations for their languages. We therefore treat these languages as conformable languages.

Chinese, on the other hand, is not conformable, although numerous attempts have been made and are being made, to make it such. We will discuss this point later.

All languages, conformable or not, may be one of two types - left-to-right displayable or right-to-left displayable. They are broken down further into horizontal display or vertical display. For example, Chinese is non-conformable, right to left and vertical. Arabic is conformable, right to left, and horizontal. English is conformable, left to right, and horizontal.

**Conformable Sets**

Assuming current state-of-the-art technology, we will now consider conformable languages.
Conformable languages are relatively easy to support today. In fact, the problem reduces to one of input and output - and this is basically a hardware problem. Because conformable languages can be assigned bit patterns in seven or eight bits, traditional I/O devices can be used. (Even punched cards are not out of the question). All that is required is that a keyboard be designed to support this character set. This keyboard can then be used with terminals, teletypes, typewriters, even card punch machines. As an input device, OCR, reading typewritten sheets, would be excellent - especially for those languages in which characters can assume a different shape depending on this location in the string (e.g. Arabic). CRT's will probably be the most common input device. Today, many manufacturers are producing terminals which support alternate character sets. Among them are Hewlett-Packard (in the U.S.A.) and Comterm (in Canada).

With a conformable language it is always possible to design an alternate print train for an output device, or to use dot matrix to specify alternate characters. The output device also works on the basis of binary codes.

If terminals used for this work were "intelligent" and if manufacturers employed a standard means in identifying the different sets when transmitting them to the CPU, then the mixing of Roman and conformable non-Roman sets would pose no problems to existing software. However, this is not the case. There are software changes required to existing systems to handle this condition.

At IDRC, we have encountered these problems, for we are now in the process of enhancing our MINISIS software to support alternate
character sets. The following paragraphs describe difficulties which we anticipate and our solution to them.

**MINISIS approach**

MINISIS is a generalized database management system which is readily usable as an information system because it is equipped with a set of end-user processors. Because MINISIS is a turnkey information system then the problems, which we have already discussed, are more than just the problems of storing and processing the mixed character sets; we have also to face the problems of user dialogue. Before discussing these problems we must make the following assumptions about the input/output device:

1) the device should be able to handle alternate character sets and should employ a standard means of identifying the different characters sets;

2) the CPU should be able to select different character sets for the output device for printing;

3) character sets will not exceed 128 characters (including any special characters).

These problems can be divided into four categories: data structure, input/output, sorting, and user dialogue.

**Data structure**

A database is a collection of records which consists of a set of fields. A field may be composed with one or more character sets.
Assuming that records are constructed with several character sets the problem of identifying different character sets arises and we have to establish a method of identifying these sets. There are two approaches to solving this problem. 

1) The first approach is to assign unique codes to the characters of different character sets. But this approach only permits limited character sets to be used in the database because most computer manufacturers use 2 to the seventh or 2 to the eighth bits to represent characters internally, and 2 to the eighth bits only permits us to define 2 sets of 128 unique characters. 

2) The second approach is to prefix the character string with a character set indicator. Because a unique indicator is assigned to different character sets this approach allows us to mix more than two character sets within a record. The second approach is employed by MINISIS so the MINISIS users can use as many character sets as they need to construct their databases.

Mixing character sets within fields is the other problem which we have encountered; its problems are:

1) Complicated processing is needed to recognize different character sets during input and output of data.

2) It is impossible to establish a filing sequence between different character sets when sorting mixed character sets.

3) Difficulties could be encountered when the reading/writing direction of different character sets is different. (i.e. English is read from left to right and Arabic is read from right to left).
4) It is also impossible to form access points (inverted terms) using mixed character sets because the list of access points is maintained in sorting order and we are unable to order the list in desired sequence.

Thus, MINISIS will not allow users to mix character sets within fields but the user can overcome this problem by putting different character sets in a repeatable field or separate fields and printing them together. Characters are represented in codes (a set of bit patterns) inside the computer. These codes affect much of computer processing, such as sorting etc. Because of the importance of coding schemes several international coding schemes such as ASCII, EBCDIC, ISCII, etc, have been established for various languages. But if there is no coding scheme existing for an alternate character set, and a user wishes to establish his own coding scheme, then the following should be taken into consideration in designing the coding scheme:

1) several characters are reserved as control characters for input/output devices. These control characters should be retained in the character sets. The codes assigned to these control characters should use industrial standard codes such as ASCII, EBCDIC, and should be the same across different character sets.

2) Besides control characters, the remaining internal codes are assigned to numbers, the alphabet and punctuation marks. The codes assigned to these characters should be in accordance with the sorting sequence in which these characters would appear in a dictionary.
3) Alphabetic characters may exist in upper and lower case in some languages. The internal codes of the alphabet should be divided into two groups, one group for lower case and the other group for upper case. Separating upper case and lower case characters into two groups will make the upshifting operation much easier because we can easily convert lower case characters to upper case characters by adding a factor (the difference value between the lower case character and the upper case character) to the internal code of lower case character.

**Input/output**

Because the interface between the input/output device and the MINISIS software is done by the operating system, in general MINISIS does not care about the type of input/output devices. For example the user can use a terminal made by either Hewlett-Packard or Digital Equipment Company...etc. Because most of the developmental work in computer hardware has been done in Western world a standard coding scheme for the Roman alphabet has been established and all computer manufacturers employ this standard coding scheme in their hardware. But some manufacturers may employ different coding schemes when building the non-Roman script terminal. Thus problems will occur because different type of input/output devices generate different codes for the same language and difficulties will be encountered when processing the character sets. Our solution to this problem is to provide an interface routine to each type of input/output device. The interface routine performs the following functions:
1) Depending on the type of terminal, the interface routine will map the external codes into the standard codes, which are needed by MINISIS software, or vice versa.

2) According to the application's request, the upshifting operation on the character string should be done in the interface routine because different alphabets have their own rules for upshifting the characters.

3) This interface routine also handles the printing problems of those alphabets whose reading/writing direction is from right to left. This function may not be required for some types of input/output device because such a function may be built into the input/output device (i.e. intelligent terminal).

This interface routine can be placed either inside the MINISIS software or in the terminal. The functions of this interface routine are very time-consuming and ideally, should be placed in the terminal. MINISIS will implement these functions in the software at the initial phase of this enhancement.

**Sorting**

Sorting plays an important role in computer processing because most computer processing involves the sorting of data in some kind of sequence. Sorting the Roman alphabet is quite straightforward but problems will occur when sorting mixed character sets. Interfiling mixed character sets is impossible because no filing relationship exists between different character sets. Our solution to this problem is that the data field is assumed to be entered in one character set.
However some data fields in different records may be entered in other character sets, and the data fields, which are written in the same character sets, are grouped together and are sorted within the group in accordance with the sequence in which the data fields would appear in the dictionary. The final product of sorting is the concatenation of several groups of sorted records. The output priority of different groups of sorted records is defined by the user. It means that English-speaking countries may want the Roman alphabet preceding the Arabic, but an Arab country may want the Arabic alphabet preceding the Roman.

User dialogue

Even the user dialogue can be carried out in various languages while accessing MINISIS processors. However we still face the problems of interpreting the user responses (commands). The problems are that 1) the commands are not constructed in the user's own logic 2) it is very difficult to extract components from the command because the delimiter of the component is varied from character set to character set. For example we have to test a character 26 times to see whether the character is an alphabet or not. But it may not be the same in other character sets so that we cannot establish a rule to define the component. The syntax of MINISIS commands is driven by a set of internal tables which can be modified. Our solution to this problem is to make internal tables become external tables which are user definable. Before using the external table the following procedures have to be completed by the MINISIS application programmer:
1) Define all valid commands used in the application. For example, you may define ADD, DELETE, CHANGE, ... etc. in the MODIFY processor.

2) After defining valid commands you have to identify the key components of command. Take the example below:

```
ADD ID1, ID2, ID3
```

where ADD, ID1, ID2 and ID3 are the key components but the punctuation marks are not.

3) This information is saved in the mapping table.

Having defined the mapping table MINISIS now runs a new processor which will generate the external table based on mapping table and the user input. The user input can be the structure of command, the keywords used in the command and the definition of key components.

**Non-conformable sets**

Non-conformable character sets present an even more interesting problem. As an example of a non-conformable set, we will consider the Chinese character set. It is a well-known fact that it is impossible to represent all Chinese characters in 256 combinations of bit patterns. Over the past few years, a great deal of effort has been expended to devise a means of representing the set; in fact, there even exists a standard Chinese character code for information exchange which is based on various ISO 7-bit standards. All efforts share the common assumption that any solution must fit within the current 7- or 8-bit standard, since this is the way in which computers are currently built.
We believe that another solution exists. This solution is more elegant, and requires fewer arbitrary contortions in mapping any character set. It involves the use of a variable number of bits (up to 24, say, which would give you 2 to the 24 possibilities - more than enough to handle the Chinese set). If one used n bits per character then an encoding scheme (such as the Huffman code) could be used to assign patterns of small n to frequently occurring characters. These techniques would encompass the Roman character set as well.

This solution requires that a different approach be used in the design of internal character-handling circuitry, or some extension to the operating system of the machine. Given the complexity of the task of attempting to fit the Chinese set to current conditions, we would not be surprised if such an approach were adopted in the longer term.

If we assume that computers can be extended to support variable patterns for character representation, then we can assume that I/O devices can be developed to be driven in the same mode. Output devices would not present any great problem. Input devices are another question altogether.

Again, consider what has been done to date. Most efforts have been directed towards developing elaborate keyboards for inputting Chinese characters. There are large keyboards with several levels of escape sequences which guide a user in the construction, stroke by stroke, of a full character; there are keyboards with several levels which, if a user selects one combination, the most probable next sequences are presented to the user - from which a choice must be made, and so forth. All these approaches require a great deal of time on the part
of the user – also a great deal of dexterity and familiarity with a complex keyboard.

Are there any alternatives? Perhaps an intelligent graphics pad could be used for input. This would probably not require much more time for input than the keyboard approach. What about OCR units, or even spoken-word recognition systems? For phonetic languages, the latter approach might indeed prove the most practical.

For the Chinese language, spoken-word recognition systems might not work because the ideographs (written form) don't match the phonems (spoken form). To go from the encoding of phonems of the spoken language to the actual script form can be as complex a task as translating from one language to another! Perhaps we would have to consider output in spoken form as well!

In any case, all these suggested devices would be equipped with microprocessors which could do some editing and then verify with the user that the result is correct before it is transmitted.

Although we do not see any immediate practical solution to this problem (and practical does not mean forcing a solution within the existing system), we do believe that more emphasis should be given to investigating radical approaches (such as those outlined in this paper). At the same time, it might be possible to accommodate different modes of logic as well. It should not be beyond the bounds of our ingenuity to meet this new challenge.