Towards more portable COBOL

T. J. Howkins and M. T. Harandi

Department of Computer Science, University of Manchester, Manchester M13 9PL

The advent of the next version of ANSI COBOL focusses attention again on the portability of COBOL. The present ANSI standard allows more than 100,000 permissible implementations. This paper argues that COBOL needs a more rigorously defined implementation scheme which specifically allows for portability, taking cognisance of the user's operating environment and allowing many fewer implementation variants. The need for a more rational subsetting policy is also argued and how this complements the implementation scheme proposed to provide better portability is shown.

(Received March 1978)

1. Introduction
COBOL is the most widely used high level language in medium and large scale computer installations (Frazer, 1973). One of the objectives of a high level language is to achieve portability between different implementations. However, experience with COBOL has shown that the way in which the language is specified gives rise to a wide choice for the implementor which, in turn, diminishes portability. This paper examines current portability problems and proposes a solution in terms of the definition of the COBOL language.

2. Present portability problems
The COBOL language, as specified in the ANSI 1974 Standard (1974), comprises a nucleus and eleven functional modules. Each module has at least two levels and in some cases three including a null set as shown in Table 1. Full COBOL comprises all the features of the language. Minimum COBOL contains Level 1 of the nucleus, table handling and sequential I/O modules, and Subset COBOL contains any combination of the levels of the nucleus and the other modules such that the final result is less than a full COBOL implementation. As so defined, COBOL can have 104,976 official variants. This is a significant barrier to portability.

The next version of ANSI COBOL is most likely to contain a number of language enhancements which will increase the size of the language. For instance, if another functional module is added to the language, it will be seen that the number of official variants in COBOL will be multiplied by the number of levels in the new module. Even if no new module is added it is argued that the sheer size of COBOL will inhibit implementors from writing full COBOL compilers and will proliferate the number of differing but allowable versions. As a consequence the new COBOL is likely to present the user with a larger and slower compiler than at present. Indeed the user may not require all the facilities provided by the compiler in his installation.

At the other end of the scale lie the implementations for minicomputers. There has been a significant increase in the usage of minicomputers and problems are arising because no minimum standard of COBOL has been defined for these machines. Portability is therefore a problem between different manufacturers and between minis and mainframe computers.

This paper is concerned principally with the portability problems which result from a large number of allowable implementations. However it is evident that there are other causes of importability. The language definition contains ambiguities and there are areas in the language which are left explicitly implementor defined. Implementors may add extensions to COBOL to add further facilities peculiar to their environment which are not portable between different manufacturers. Lastly, the COBOL validation suite is not developed in advance of the new standard which means that compiler writers have no standard against which to measure their implementations.

In seeking an improved implementation scheme, it is considered that attention must be extended beyond the confines of the present ANSI specification to examine the roles of the user and implementor in order to assess their requirements.

3. The role of the user
An implementation policy for COBOL must have regard to the needs of users. It must provide users with features which they need and which they can use effectively and it must make allowance for the ways in which their processing needs may develop. With the increasing size of compilers it may be desirable to build compilers which allow users to generate their own subset compiler at installation level. In spite of the theoretical maximum number of implementations there is a more restricted range because there is a competitive market for hardware and software and perforce some implementations are geared to specific operational needs of users.

This suggests that the user's operational background should be one of the parameters in a revised implementation policy for COBOL and one can discern the following categories of

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Schematic of module structure of ANSI COBOL 1974</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nucleus</td>
<td>Table I/O</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
operating environment:
(a) Limited Batch which is characterised by the use of slow peripherals and serial magnetic devices
(b) Direct Access which allows serial or direct access to files
(c) Interactive and Distributed Processing Networks which have terminal networks with direct files access which can also support a batch background

The module structure of COBOL has some effect on the way in which a user can use the facilities of the language. The existing functional modules can be classified under the following headings.

1. Hardware dependent modules
   These are sequential, relative and indexed I/O, and communications. These features would determine the operating environment to which the user belongs.

2. Hardware independent modules
   These are table handling, segmentation, sort-merge, inter-program communication and library. These are dependent on backing store. Since backing storage is common to all data processing machines, these features are regarded as hardware independent.

3. Productivity aids
   This category contains the report writer and debugging modules and in essence is a subclassification of hardware independent modules. This group contains features which duplicate the essential features of the Hardware Independent Modules but which aid programmer productivity.

Within the confines of his present COBOL implementation, a user's requirement of COBOL facilities may not be static. It is therefore important to consider in what directions the user can increase his use of COBOL.

The user may wish to change his configuration and change his use of COBOL accordingly. This can involve the use of additional features from the hardware dependent modules. For instance, if the user installs disc equipment, he may wish to add random access.

Within a given configuration, the user may wish to change his use of available hardware dependent or hardware independent features and/or productivity aids. These features should be ordered in a small number of levels which differentiate between users with basic or minimal facilities and more advanced users.

In increasing his use of COBOL, the user should not be forced to alter his program. He should expect some portability as he expands his use of COBOL.

4. The role of the implementor
   Users should not be the sole beneficiaries of an improved implementation plan. Implementors have problems which arise from the imprecision of the formal language specification. This forces them into taking unilateral decisions which impair portability of the language. Therefore, it is desirable to specify those areas of choice which should be left to the implementor and those which should be laid down in the next formal specification. The formal language specification should offer the implementor a set of choices which are classified into clearly defined levels. Each implementation level should contain a standard set of features. Any implementation of that level must therefore contain the complete set of standard features of that level. Furthermore, portability must be built into these implementation choices.

The implementor's choices will be dictated by the type of hardware needing support, the user demand for COBOL features and the present state of the market for COBOL compilers. The implementor must decide about extensions to COBOL, possibly to add interfaces with other equipment, software, etc.

5. Summary of requirements
   At this point it is opportune to summarise the argument so far in the form of criteria which must be satisfied in a new implementation scheme. Firstly, the revised method should recognise the user's hardware environment so as to define the features necessary to support that environment.

   Secondly, since within an operating environment there will be a variety of choices of features dictated by the user's environment, it follows that these features should be rationalised into levels. The levels should be ordered from the least inclusive or basic set of facilities to the most inclusive or full set of facilities. Furthermore, to safeguard portability, the levels should be defined so that any implementation of that level is portable to the next higher level. This means that users will have an ordered method of advancing their use of COBOL on portable lines.

   Thirdly, it has been shown that users may change their operating environments which may be reflected in their use of COBOL. It is desirable that the user should not have to change his program to run them in a changed environment and this demands that the operating environments be ordered in such a way that this desired portability can be achieved.

   Fourthly, portability must be ensured as far as possible by restricting the number of allowable implementations to a small number and by making the specification of any level, and within it any particular version of COBOL, standard to all implementations of that level.

   Finally, the new scheme should offer the implementor a defined set of choices and it should be able to accommodate change, e.g. additional operating environments, software, tools, etc.

6. A proposed implementation scheme for COBOL
   An implementation scheme is proposed in the form of a table as illustrated in Table 2 where the columns represent operating environments and the rows represent levels of implementation.

   It will be seen that the operating environments are ordered from left to right in the implementation table from the least inclusive environment (i.e. limited batch) to the most inclusive (i.e. teleprocessing). An additional property of this table is that the ordering implies that operating environments are nested. This means that any operating environment contains all the features necessary to support the operating environment(s) to its left in the table. This allows program portability between different operating environments from left to right in the implementation table.

   This nesting principle safeguards portability and tends to correlate with existing hardware usage of users. A random sample of 100 installations in the London and Manchester areas showed that all installations possessing teleprocessing equipment also possessed disc equipment. Whilst no statistics are available concerning the type of usage of this equipment, it is felt that in the majority case, teleprocessing is concerned with file updating/enquiry for which direct access is essential.

<p>| Table 2 An implementation table for COBOL |
| Operating environments |</p>
<table>
<thead>
<tr>
<th>Levels</th>
<th>Minimum</th>
<th>Intermediate</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Limited batch</td>
<td>Direct access</td>
<td>Teleprocessing</td>
</tr>
</tbody>
</table>
Each intersection of a row and a column constitutes one implementation of COBOL. The table is expandable in that a further operating environment can be added or inserted into the table and further rows may be added, thereby increasing the number of allowable implementations of COBOL. COBOL features should be allocated to this table according to the subsetting rules chosen and in accordance with the following portability rules:

1. Horizontally
Any implementation must be portable to the next implementation to the right (if any) on the same level. This allows the addition of new hardware environments while keeping the level of implementation unchanged. For instance, a user may wish to change his present operating environment to allow teleprocessing facilities and maintain existing tasks. Therefore all features of direct access COBOL must be present in teleprocessing COBOL for the portability requirements to be met.

2. Vertically
Any implementation must be portable to the next higher level of the same hardware dependent environment. This makes expansibility possible when new features are added to an unchanged hardware environment so that portability works from the minimum implementation through to the maximum implementation in any column of the table. For example, a user may proceed from minimum direct access COBOL to maximum direct access COBOL and by doing so would gain some additional hardware dependent features (for the direct access environment only!) and some software tools. For instance, portability would require that the maximum COBOL contained all features of intermediate and minimum levels of the same hardware environment.

3. Diagonally
As a consequence of the above two rules it can be seen that there is diagonal portability. In other words any implementation is portable to a higher level of a more advanced operating environment. For instance, a user should be able to progress from intermediate direct access to maximum teleprocessing without loss of portability.

7. The impact of change
A necessary requirement of this implementation plan is that it can absorb change without major restructuring. Such changes could arise from the need to alter features within an operating environment, addition of new hardware dependent modules and the addition of new software tools and productivity aids. Any such change will require corresponding changes throughout the table to maintain portability. As an example, the addition of data base facilities could be accommodated in the full level of the leftmost operating environment in which data bases could be implemented. Alternatively it could be considered as an operating environment in its own right and could be added to the right of the table or inserted within the table. Either method would require such changes to be reflected throughout the table, that is, below and to the right of the point at which the change is made.

8. Subsetting
An implementation policy for COBOL and the question of subsetting the language are related but distinct problems. It is felt necessary to consider the topic of subsetting in that context for whatever subsetting policy is chosen, it must conform to the implementation rules laid down so far in this paper. Perhaps the first question to answer is whether the COBOL language needs subsetting. No subsetting at all would be useful for enhancing portability by reducing the number of allowable COBOL implementations to one. It would also provide users with the full facilities of the language. However there is no guarantee that implementors would implement all of the language and a number of hybrid versions may appear. This may be partially (although not wholly) offset by subsetting. All features of the language are not necessarily applicable to all installations, particularly small/medium size organisations which would be faced with redundant features. There may be unwanted overheads in compiling time and storage when using a full version of the language whereas smaller compilers may permit better optimisation. A small number of official variants is more likely to tempt the implementor to choose one version rather than inventing his own hybrid. All this suggests that some form of subsetting is desirable.

9. Objectives of a subsetting policy
The original objectives of the COBOL language should be preserved in any subsetting policy. These are:
(a) readability/documentation
(b) maintainability
(c) productivity
(d) portability
(e) efficiency
(f) ease of use and learning.

Table 3 A verb classification

<table>
<thead>
<tr>
<th>Hardware dependent</th>
<th>Data manipulation</th>
<th>Arithmetic</th>
<th>Program control</th>
<th>Productivity aids</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>INITIALISE</td>
<td>ADD</td>
<td>ALTER</td>
<td>COPY</td>
</tr>
<tr>
<td>DELETE</td>
<td>INSPECT</td>
<td>COMPUTE</td>
<td>CALL</td>
<td>GENERATE</td>
</tr>
<tr>
<td>CLOSE</td>
<td>MOVE</td>
<td>DIVIDE</td>
<td>CANCEL</td>
<td>INITIATE</td>
</tr>
<tr>
<td>DISABLE</td>
<td>SEARCH</td>
<td>MULTIPLY</td>
<td>ENTER</td>
<td>MERGE</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>SET</td>
<td>SUBTRACT</td>
<td>EXIT (PROGRAM)</td>
<td>SORT</td>
</tr>
<tr>
<td>ENABLE</td>
<td>STRING</td>
<td></td>
<td>GO TO</td>
<td>SUPPRESS</td>
</tr>
<tr>
<td>OPEN</td>
<td>UNSTRING</td>
<td></td>
<td>IF</td>
<td>TERMINATE</td>
</tr>
<tr>
<td>READ</td>
<td></td>
<td></td>
<td>PERFORM</td>
<td>USE</td>
</tr>
<tr>
<td>RECEIVE</td>
<td></td>
<td></td>
<td>STOP</td>
<td>USE (Debugging)</td>
</tr>
<tr>
<td>RELEASE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REWRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Computer Journal Volume 22 Number 4
### Table 4 An elementary subset of hardware independent verbs

<table>
<thead>
<tr>
<th>Data manipulation</th>
<th>Arithmetic</th>
<th>Program control</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE all formats except CORRESPONDING</td>
<td>ADD, SUBTRACT, MULTIPLY, DIVIDE</td>
<td>All formats except CORRESPONDING</td>
</tr>
</tbody>
</table>

### Table 5 Proposed minimum COBOL subset for hardware independent verbs

<table>
<thead>
<tr>
<th>Data manipulation</th>
<th>Arithmetic</th>
<th>Program control</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE all formats except CORRESPONDING</td>
<td>ADD, SUBTRACT, MULTIPLY, DIVIDE</td>
<td>All formats except CORRESPONDING</td>
</tr>
<tr>
<td>INSPECT all formats limited to single characters</td>
<td>MULTIPLY, DIVIDE</td>
<td>All formats</td>
</tr>
<tr>
<td>STRING</td>
<td>UNSTRING</td>
<td></td>
</tr>
</tbody>
</table>

In addition there are some other relevant objectives. The size of the compiler should be minimized. It should be relatively easy to map COBOL features onto hardware. Language complexity should be restricted as far as possible. Any subsetting policy for any hardware environment should meet all or most of these objectives.

**10. Application of subsetting principles to ANS 1974 COBOL**

A COBOL program is likely to contain a selection of hardware dependent features, hardware independent features and productivity aids together with the language concepts and data declarations which they support. An analysis of verbs applicable to these categories is shown in Table 3.

The immediate problem is to define the least inclusive subset. It was stated earlier that productivity aids duplicate more essential features of the language and therefore, they should be excluded from the least inclusive subset of the language.

Firstly consider hardware independent features. An elementary set of verbs can be proposed which provides the basic set of facilities for routine data processing. These are shown in Table 4. All other hardware independent verbs can be simulated by those shown in Table 4. ALTER has been omitted from the discussion since it is being deleted from the language. Array accessing is by means of subscripting.

It will be seen that this elementary subset does not satisfy fully the subsetting objectives. Certain functions that are not represented in the elementary subset can be simulated at the cost of additional coding effort and with lower efficiency. For instance, string handling and looping are not dealt with efficiently. A further objection to this subset is that it does not admit the 'macro' type COBOL statements such as COMPUTE, INSPECT, PERFORM, which increase programmer productivity by automating some essential features of the elementary subset.

Obviously this elementary version of COBOL is unsatisfactory and must be expanded to meet the desired objectives. A subset which can be called minimum is shown in Table 5.

With regard to data manipulation, it will be seen that STRING/UNSTRING and INSPECT (single character) are included to provide comprehensive string handling facilities. All versions of MOVE are admitted except the CORRESPONDING option which it is felt is unnecessary in minimum COBOL because it can be simulated easily with no loss of efficiency and with better readability. Note also that SEARCH and SET are excluded in favour of subscripting. COMPUTE is provided to aid efficiency and conciseness. Of the program control verbs, PERFORM is included in three formats omitting the TIMES and VARYING options. The PERFORM UNTIL format is an essential loop structuring facility which can support structured programming.

Secondly, the verbs which can be allocated to the hardware dependent minimum subset are shown in Table 6. This allocation is made on the same principles as the hardware independent verbs. Note that READ . . . INTO, WRITE . . . FROM and REWRITE . . . FROM are included since they provide additional productivity advantages. Each hardware environment contains sufficient features for the user to make use of that environment effectively.

Having examined the subsetting problems of the minimum subset, it is appropriate to consider the most inclusive version of the language. This will contain all the features of the language including productivity aids and software tools. Essentially this implies a two-tiered implementation structure for COBOL but there is a possible intermediate tier which could be made up of facilities which are not software tools or productivity aids in the definition of those terms but which are extensions of facilities in the minimum subset. These include items such as:

- (a) CORRESPONDING options of ADD, MOVE, SUBTRACT
- (b) CLOSE . . . NO REWIND/LOCK/REMOVAL
- (c) PERFORM . . . VARYING/TIMES
- (d) SET & SEARCH

There are two alternative strategies for this class of feature. Either they are added to the minimum subset to provide an intermediate subset or they are confined to the most inclusive version of the language.

The final subsetting question concerns language concepts and data declarations. The allocation of some of these features is obvious and directly related to the allocation of verbs with
which they are associated. For instance, cd-name and Communication Section must be included in the minimum tele-
processing subset. All features associated with Report Writer, Library, Debugging and Sort-Merge have their place in the
maximum level of subsets. However, there are features associated with hardware independent verbs, limited batch and
direct access which need further rationalisation as with the
CORRESPONDING option mentioned above. Following the
same policy applied to define the minimum limited batch
subset, all but the following can be allocated to that subset:
(a) Condition names, RENAMES clause, Level numbers 66 and
88
(b) Index-name, indexing, USAGE INDEX, INDEXED BY,
OCCURS (Format 2)
(c) LINAGE COUNTER, LINAGE clause
(d) Data name qualifications
(e) SELECT . . . OPTIONAL, MULTIPLE FILE TAPE
(f) BLOCK CONTAIN integer-1 TO integer-2 CHAR
RECORDS
(g) OCCURS . . . DEPENDING ON . . .
(h) Nested REDEFINES
Consideration would have to be given to including these in an
intermediate tier or leaving them in the maximum version of
the language.

It can be seen that any subset of COBOL comprises hardware
independent, hardware dependent and language concepts/data
declarations. At the most inclusive level, productivity aids and
software tools are also incorporated.

11. Subsetting and the implementation scheme
The objective of this section is to discuss the subsets deter-

Table 6 Minimum subset for hardware dependent verbs

<table>
<thead>
<tr>
<th>Limited batch</th>
<th>Direct access</th>
<th>Teleprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ . . . AT END INT INTO</td>
<td>READ . . . INVALID KEY NEXT RECORD</td>
<td>SEND</td>
</tr>
<tr>
<td>WRITE . . . AFTER/BEFORE ADVANCING FROM</td>
<td>START</td>
<td>RECEIVE</td>
</tr>
<tr>
<td>ACCEPT \ single</td>
<td>REWRITE FROM</td>
<td>ENABLE</td>
</tr>
<tr>
<td>DISPLAY \ message</td>
<td>DELETE</td>
<td>DISABLE</td>
</tr>
<tr>
<td>OPEN . . . INPUT OUTPUT EXTEND</td>
<td>OPEN I-O</td>
<td>ACCEPT MESSAGE</td>
</tr>
</tbody>
</table>

Table 7 Unified structure for a portable COBOL language

<table>
<thead>
<tr>
<th>Hardware Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited batch</td>
</tr>
<tr>
<td>Minimum limited batch</td>
</tr>
<tr>
<td>Minimum hardware independent features</td>
</tr>
<tr>
<td>Intermediate limited batch</td>
</tr>
<tr>
<td>Minimum limited batch</td>
</tr>
<tr>
<td>Intermediate hardware dependent features</td>
</tr>
<tr>
<td>Minimum limited batch</td>
</tr>
<tr>
<td>Maximum limited batch</td>
</tr>
<tr>
<td>Maximum limited batch</td>
</tr>
</tbody>
</table>

problems. The proposed scheme specifically allows for the user's
operating environment. It also restricts the number of allow-
able versions of COBOL whilst ensuring that implementations
are portable in an ordered progression. For instance, with three
levels in the table, there would be only nine official COBOL
implementations.

An assumption has been made that operating environments
are nested by use. This may not apply in all cases but it has been
adopted because it is representative of current operating
practice and because portability may be achieved only if these
environments are so treated. The approach is still valid even
where some parts of an implementation dictated by nesting of
such environments are not needed by the user. In this case the
unwanted features are an overhead but this is justified by the safeguarding of portability and the consequent benefit to the majority of users who follow the nesting concept.

The proposed scheme helps the implementor in his choice of version. It is flexible and can accommodate change.

A subsetting policy has been chosen which contributes to the objectives of the language and which satisfies the portability rules of the implementation scheme.

References


---

**Calls for papers**

**Data bases**

A conference on data bases will be held on 2-4 July 1980 at the University of Aberdeen, Aberdeen, UK. Research papers are invited on all aspects of data bases, particularly in relation to implementation and concerning

- Distributed data bases
- Data Dictionary
- Conceptual schema
- End user facilities
- Restructure and reorganisation
- Performance optimisation
- Integrity and privacy

Please send a title and a half-page abstract of the paper by 17 December 1979 to

Dr S. M. Deen
Department of Computing Science
University of Aberdeen
Aberdeen AB9 2UB
Scotland

Tel: 0224-40241 ext 6421 or 6417 (office)

The Completed papers (not more than 15 A4 pages) (5 copies) must be submitted by 15 January 1980. Notice of acceptance or rejection will be mailed to authors at the beginning of April.

We intend to publish the proceedings of the conference. This conference is being sponsored by The British Computer Society. A few selected papers will be published in *The Computer Journal*.

**Special issue of Artificial intelligence on computer vision**

Papers are invited for inclusion in a special issue of the journal *Artificial Intelligence* which is intended to represent the current state of the art in computer vision. It is expected that the special issue will consist of a limited number of high quality papers, each of which will be closely refereed. To be timely, it is intended that the special issue shall be shipped to the printer by Spring 1980.

**Purpose**

The pace of developments in computer vision over the past five years has been considerable. Significantly, there has been a shift away from toy worlds and knowledge directed processing, and a growing concern with low to intermediate level issues such as edge detection, stereopsis, the computation of surface orientation, motion perception and computations involving texture. Much recent work has drawn explicit inspiration from, and has made substantial contributions to, perceptual psychology and neurophysiology. At the same time, other influential studies have supported impressive engineering applications, from remote sensing robotics to the processing of satellite images. Overall, a number of sophisticated mathematical analyses of relatively narrowly defined problems have been developed, and it is expected that the papers comprising the special issue will reflect this trend. While growing numbers of researchers, in disciplines traditionally outside artificial intelligence, are being excited by their recent exposure to the subject, workers in others areas of AI increasingly feel out of touch. It is intended that the special issue will serve the needs of both groups, by acting as a statement of the current state of the art, providing an introduction to the work of the current field leaders, and serving as a route into the literature.

**Intended coverage**

Contributions are solicited from all areas of computer vision. The following list of topics is meant to be suggestive but not definitive: edge extraction, lateral inhibition, motion perception, visual tracking, colour, transparency, highlights, computation of occlusion, depth and shading, surface orientation, texture, stereopsis, representation of visual information, representation of space, cognitive maps, recognition, text perception.

Studies of image enhancement, picture coding and the transmission of images, will be excluded from the scope of the special issue, but pattern recognition studies are not excluded. However, it seems that the main thrust of the recent developments in computer vision has been elsewhere, and so they will be accorded lower priority.

A paper which is included in the special issue will typically

(a) report on work which has already reached a state of completion
(b) minimise conjecture, supposition, and extensive discussion of future intentions and aspirations
(c) precisely define the intended goal of the computation, and make clear the extent to which it has been achieved.

A contribution may take the form of an extended survey including the author's previously published work.

**Deadline** Complete manuscripts (three copies) including figures and tables should be received by 15 January 1980. Manuscripts should be in English and should follow the guidelines of *Artificial Intelligence*. Each paper will be quickly, though closely, refereed. Papers deemed to be outside the scope of the special issue will be sent to the Editor of the AIJ as possible contributions to a regular issue of the journal.

Contributions should be mailed to the guest editor for the special issue

Dr J. M. Brady
Department of Computer Science, Essex University
Wivenhoe Park, Colchester CO4 3SQ
Essex.

Enquiries (but not complete manuscripts) may also be mailed via the ARPA net to BRADY @ MIT-MC.

**Computational linguistics**

The eighth international conference on 'Computational linguistics', (COLING 80) to be held in Tokyo, Japan on 29 September—4 October 1980, will be devoted to practical results and theoretical underpinnings from artificial intelligence and theoretical linguistics. Papers are solicited in the areas of discourse, syntax, semantics, morphology, history of linguistics, lexicography and literary studies, artificial intelligence, dialect variations, natural language input and output, data base manipulation, machine translation, voice synthesis and recognition, and problems of ideographic character recognition and output.

The deadline for abstracts is 15 January 1980, and they should be sent to the Programme Chairman, Professor D. Hays, 5048 Lake Shore Road, Hamburg, New York, 14075, USA.

Further information (including exact details for the submission of papers, accommodation, registration and charter details, plus the programme when fixed) will be sent to all members of the *UK Committee for COLING 80*. You may join this committee simply by writing to Professor Yorick Wilks, Department of Language and Linguistics, University of Essex, Colchester, Essex, UK.

The Computer Journal Volume 22 Number 4