SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 19(1), 1–34 (JANUARY 1989)

indx and findphrases, A System for Generating Indexes for Ditroff Documents

KRIS K. ABE[†]
Computer Science Department, University of California, Los Angeles, CA 90024, U. S. A.

AND

DANIEL M. BERRY
Computer Science Department, Technion, Haifa 32000, Israel

SUMMARY

Creating back-of-the-book indexes is a difficult task involving intelligent and clerical processes. Programs have generally not achieved the level of intelligence required to perform the intelligent process of selecting terms for the index. Semi-automatic indexing programs perform the clerical process of preparing entries once the terms have been selected. The programs do not provide assistance in term determination and most require flooding the text with indexing commands. indx differs from other semi-automatic indexing programs mainly because it does not require the insertion of indexing commands into the text to be indexed. The method by which indx assists in the creation of an index is introduced and compared with the characteristics of the other programs. This method includes the use of a program that aids the term determination process. The design, implementation, and application of indx are presented. Areas in which indx may be improved or enhanced are identified. An index of this paper created with indx is included as an example.

KEY WORDS Generating book indexes Formatting Typesetting Device independent Troff

INTRODUCTION

Anyone familiar with textbooks is also familiar with the use of the back-of-the-book indexes. For finding text dealing with a given subject, such an index is far more useful than the front-of-the-book table of contents, which just lists, in page order, the chapter, section, and subsection titles and starting page numbers. The index lists important terms occurring in the text and the numbers of the pages on which these terms actually appear or are the subject of the discussion. The index is usually alphabetized by the spelling of the terms. The index helps locate discussions of important concepts simply by knowing the keywords of these concepts. Doing the same with the table of contents requires knowing under which title a term is likely to be discussed. A good index is like an inanimate memory, in which information is stored to be recovered quickly and precisely.[1]

[†] Present affiliation: The Aerospace Corporation, El Segundo, CA 90245, U. S. A.

0038-0644/89/010001—34/$17.00

Received 1 January 1988
Revised 22 July 1988

While an index is very useful for the reader of a book, it is very difficult for the author of the book to generate. It is so difficult that many authors hire a specialist to do the task. A common use of indexes is to locate all books in a library that deal with a particular issue by looking for that issue in the indexes of all books on the shelves with the same call number. Anyone who has tried this method of search knows that many books just do not have good indexes.

Generating an index is difficult because it consists of two processes,

1. one intelligent, determining the terms that are to be indexed, and
2. one clerical, finding all places where these terms appear.

The first is difficult because it is hard to anticipate the keywords that the readers look for. If the index has too few terms, the reader will not find what he or she is looking for. If the index has too many terms, it will be too big and unwieldy. The second is difficult because it is boring and subject to clerical error, and subject to revision any time a change is made in the text. The indexer can easily fail to find some occurrences of a term.

Thoughts turn to automating the process. However, automating the process is difficult too. Programs have not achieved the level of intelligence required for doing the term determination, artificial intelligence research notwithstanding. Brute force generation of terms by simply taking all phrases that occur in the book yields an index larger than the book! Programs are very good at the clerical task of locating all occurrences of terms that actually occur as phrases in the book. However, they cannot locate a discussion about a term in which that term itself does not appear. Thus all that can be hoped for (at least for the foreseeable future) is some clerical program or perhaps some expert system aiding the term determination process and a second program doing the more mundane and error-prone task of finding all occurrences of a term that actually appears in the book.

A number of schemes exist already. However, most attack only the location of occurrences of terms, and do nothing to help determine the terms in the first place. Many also do nothing about finding discussions about the terms that do not actually contain them. Also, many require flooding the text of the book submitted to the formatter with commands to dump a term/page-number pair into a database for later sifting into an index; if not done with the aid of hideable windows, this flooding can make it hard to read the input text.

This paper describes a suite of tools developed at UCLA that assist in indexing ditroff[2] (Device Independent Typsetter RunOFF) documents. Specifically, these tools offer genuine assistance in term determination, allow finding all occurrences of and some discussions about terms, and provide a clean interface in which the text of the book to be indexed is not flooded with indexing commands.

This paper was typeset camera-ready in its entirety (including the page headers, the page numbers, and the journal-specific information in the first page's header and footer†) with the aid of ditroff, its pre- and post-processors, and the suite. In order that the page numbers in the index of the paper be correct, it was necessary to know at typesetting time on which page the article would begin. It is extremely fortunate that the paper could be scheduled to be the first paper of the present volume, for then it was trivial to know at typesetting time on which page the article would begin, namely page 1.

The rest of this section defines the terminology of indexing, describes the manual process, and briefly discusses existing programs for assisting in index generation. The second section describes the approach the suite uses and the method by which the authors exercised the suite to obtain the index. The third section describes the experience gained in using indx, and the last section contains conclusions about the indx method and program and discusses areas of further work. This paper is

†However, the journal's editors may have replaced all of the headers and footers with paste-overs.

followed by an index generated with the help of these tools. Finally the appendices show the input necessary to generate the index of the paper and give UNIX®†-style manual pages for the tools. Note that the indexing suite was not applied to the appendices! This paper is derived from a much longer thesis in which many of the details omitted to fit the size limitation may be found.[3]

Index term definitions

Most of the index entries look like the following example:

Program maintenance documentation, 11, 17, 24

The phrase ‘Program maintenance documentation’ is called a heading. As defined by professional indexer G. Norman Knight, headings are the ‘word(s) or symbol(s) selected from, or based on, an item in the text, arranged in alphabetical or other chosen order’.[4] In this paper, headings are also referred to as entry phrases. The numbers following the heading are references, which direct the reader to the location in the text where the heading is discussed. These numbers are usually page numbers, but they can also be folio, section, or paragraph numbers. The current version of indx deals only with page numbers, thus these numbers are referred to as page references. An index entry having this form is referred to in this paper as a regular entry.

At times, a regular entry may be further expanded under or may be closely related to other entries in the index. A *see also* cross-reference would be added to such terms, directing the reader to the related entries. A *see also* cross-reference follows the page references of the entry, as follows.

Program maintenance documentation, 11, 17, 24
  See also Program design language

Other types of cross-references exist to guide the reader who is unfamiliar with the terms used in the text to the relevant entries having page references. A *see* cross-reference is a direction from one heading (or subheading) to an alternative heading under which all the relevant references to an item in the text are collected. A *see under* cross-reference indicates that the subject word is used under another heading. For example, the entry ‘Taxation of costs, See under Costs’ means the term ‘taxation’ is used as a subheading under the heading ‘Costs’.[4] For *see* and *see under* cross-references, there are no page references.

As alluded to above, a subheading is the heading of an index entry that appears under another index entry. An index entry found under another entry is referred to as a subentry. At times, an index entry having subentries is referred to as a main entry. In fact, index entries not having subentries are also called main entries, as they are on the same level of the index. A main entry along with its subentries is referred to as a group entry. The group entry below has a regular entry and a *see* cross-reference as subentries.

Program design languages
  PDL, 37-40, 42
  ADA/PDL, *See* Ada

It appears that in the U.S., *see under* cross-references are not used in book indexes;[5] *see* cross-

† UNIX is a registered trademark of AT&T Bell Laboratories.

references are used instead. *See under* cross-references are used mostly in the large indexes to the periodical literature produced by commercial indexing companies. Our goal, as implementors, is to provide complete functionality with a good user interface. Since there are indexes that use *see under* cross-references, our software is obliged to provide the ability to obtain them.

Notwithstanding this observation about style, the index in this paper uses *see under* cross-references. The index in the paper and the accompanying presentation of the input to make it serve two main goals.

1. It should be comprehensive enough to allow the reader to find what is desired, in order to demonstrate the effectiveness of the term identification method and software.
2. It should include examples of all features in order to demonstrate that the all features of the indexing software work and to show the input necessary to obtain each kind of entry.

A third goal of adhering to stylistic standards is sacrificed to these two.

Formatting styles

There are several methods of formatting the index — entry-per-line, paragraph or run-in, and a combination of the two. In the entry-per-line style, each heading and subheading is placed on a separate line. The paragraph style has subheadings following the heading in a paragraph form. This latter style is common in social science texts, where the subentries, which are usually events, may be presented in an evolutionary or chronological order. Because of the natural progression of subentries as sentences in a paragraph, the subentries should be listed in numerical rather than alphabetical order. A big advantage of the paragraph style is the space it saves. It is, however, more difficult to scan this type of index, and Eleanor Harris[1] recommends that indexes having sub-subentries be presented in entry-per-line style. The combined style borrows the good points of each of the previous styles. Subentries are placed on separate lines and sub-subentries are presented in paragraph style following their parent subentries. Thus overall, entries and subentries are easily identified and some space is saved when sub-subentries are printed.

Within each style there appear to be variations in the punctuation that is used. Where some use periods, others use commas. In order to demonstrate the flexibility of our software, the first half of the index is printed in the entry-per-line style, and the second half is printed in the paragraph style. This flexibility is achieved, as explained later, by changing the definitions of the macro calls generated by the indexing software.

Manual indexing task

The indexing task is an art, as the indexer must select the terms that best convey the contents of the text. In order to get a good and well-balanced index, the indexer reads the text once rapidly and then a second time, more slowly, to get an understanding of the text. While reading, phrases are underlined and possible subjects (entry phrases) are written in the margin. After all phrases have been identified as possible entries, the indexer must decide which of these will be in the index. Several factors influence the selection process. First, the indexer must keep the needs of two types of readers in mind. There are readers who have read through the book and use the index to refer back to things already read. There are also readers who are searching through several books on the same subject, seeking specific information without having to read the entire book. Another factor influencing entry selection is the maximum index length permitted by the publisher. The terms selected for the index are written on cards or slips of paper and kept in a tray. The cards are checked by pages to prevent useless entries in the index, which are entries having a page reference that yields no information on the subject. Then they are alphabetically sorted and the index is typed from the cards.

One of the unresolved issues in indexing is the type of alphabetization used. There is no universal standard, but two of the common ones are *word-by-word* and *letter-by-letter*. The word-by-word method is more common[4] and treats the headings and the subheadings as consisting of separate words, alphabetizing them one word at a time. For example:

Index arrays
Index generation
Indexes, efficiency and

The letter-by-letter method involves treating the headings and the subheadings as single units, alphabetizing them one letter at a time. The same three phrases above alphabetized letter-by-letter would be in the following order:

Index arrays
Indexes, efficiency and
Index generation

No clear advantage of either has been found.[6] Although alphabetizing word-by-word seems easier to perform from the human point of view, there is a tendency nowadays to use letter-by-letter or rather character-by-character in machine collated indexes, as it is simpler to generate and to explain to the reader. In the older styles, numbers would be sorted as if they were spelled out and Roman numerals would be sorted according to the values they represent, not their actual letters. However, today there appears to be a tendency to sorting numbers by their machine collating sequence. The literature abounds with information on the indexing process and should be consulted by the interested reader.[7, 8, 4, 1]

Indexing programs

Several computer programs now exist to aid the indexer in his or her job. They may be classified into two main groups, those running with the full function formatting system which is formatting the document that is being indexed and those which run as separate programs. In the first category are those that run with Wordstar™†,[9] with troff or ditroff,[10, 11] with TEX™‡,[12] and with LATEX.[13, 14] These all have the problem that they require flooding the text of the indexed document with macro calls for dumping terms paired with page numbers. These macro calls must be inserted at each place in which the term is used or discussed to force dumping of that point's page number with the term. It is the presence of these macro calls that can make it hard to read the input document. However, there are windowed, WYSIWYG formatting systems in which one can annotate arbitrary points in the text with indexing information. These include Interleaf[15] and Ventura.[16] The indexing window is invisible unless one is actually working on it; thus the input document is not obscured. Those that run without a document formatting system include <<ANSWER>>,[17] INDEX,[6] *INDEX,[18] INDEXIT,[19] and many more.[20] These stand-alone systems are intended to emulate the 3×5 card method that many indexers are wont to use. After the book has been printed, the indexer works with

†Wordstar is a trademark of MicroPro International Corporation.

‡TEX is a trademark of the American Mathematical Society.

one of these as a separate application, typing entries, including the page numbers, directly to the program. While the indexer must find and type the page numbers (perhaps using another computer or window running an editing application on the book itself), the indexer is spared the drudgery of alphabetizing, formatting, and retyping the index each time it is necessary to modify it. See the thesis[3] for more details on existing indexing software.

According to Linda Fetters, a freelance indexer, indexing programs should be judged by the following criteria:[19] (1) ease of entering index headings, (2) ability to create cross-references, (3) ease of editing index entries, (4) sorting capabilities, (5) size limitations, (6) formatting capabilities, (7) printing effects, and (8) ability to cumulate indexes.

Although the extent to which each of the above programs other than INDEXIT satisfies the criteria is unknown, several comments can be made. First, although marking words or phrases in the text is easier than entering them separately, each such term has to be marked *each* time it occurs. In such indexing systems, there are two styles of marking. In one, one puts a marking command consisting of an explicit term in the midst of each discussion about that term. The explicit term is not taken as part of the text, and it is listed in the index as having appeared at the point of the mark, even if the term really does not appear in the text. In the other style, one puts a mark at each explicit occurence of the term in the text. Indexes set up in this latter manner are restricted to terms occurring word-for-word in the text. Secondly, it is desirable to be able to create cross-references. A good index would have cross-references for readers unfamiliar with the terms used in the text, having synonymous terms in the index referencing the terms used in the text. Thirdly, having the capability of different ways of sorting the terms is desirable since there is no standard method. Fourthly, the ability to accumulate indexes would be nice to have because index entries can be set up for sections of the text in parallel and merged to form the entire index. This would save time although more space may be needed, depending on the size of the text and the number of sections into which it has been broken.

Properties of indx

The main advantage, we believe, of the indx approach, described herein, is that the text to be formatted does not have to be flooded with indexing commands. The formatter gets a cleaner input text, which is easier to subject to other analyses, and which is less likely to confuse an editor's pattern matching. In addition, by having the index terms in a separate file, in which each occurs only *once*, it is easier to add, change, and delete index terms. The old concepts of modularity and information hiding[21] come to play here. The purpose of the index terms is to generate the index; the purpose of the text is to generate the body of the book. Therefore, they should be in separate modules, each somewhat independently updateable. One can make minor cosmetic changes or even major reorganizational changes to the text without having to change the index terms, and one can adjust the detail of the index by changing the index terms without changing the text.

Not all agree that having a separate list of index terms is better. Kernighan is one of the co-authors of an indexing system in which one sprinkles the source of a document with macro calls at each occurrence of a term.[11] He points out that with a separate index terms file, one has to edit both files to keep them up to date. He, for one, would prefer to have to edit only one file.[22] In the last analysis, the users will decide which scheme is best.

The main property of ditroff that makes this separation possible is that there exists a representation of the document after formatting decisions have been made, i.e. where the page breaks are, but from which the original input's word boundaries can be extracted. By use of this word boundary information, it is possible to find the occurrences of the index terms, and by use of the page boundary information, it is possible to determine on which page each occurrence of an index term occurs.

Ditroff's so-called device-independent intermediate form fits the bill exactly. Besides the page breaks and the sequence of characters and movements or positions that are found in all such intermediate forms and that a device driver needs to print the text, there is some extra semantic information. Specifically, there are `w`s to mark ends of words and `n`*number* `0`s to mark ends of lines.

```
This is an example of a line.
```

If the above is submitted to dtroff -Tpsc, the output, folded to fit the line length, is as below:

```
H576
V96
cT
49h40i22sw51i22sw51a36nw60e36x40a36m62p40l22ew56o40fw47aw
56l22i22n40e36.n96 0
```

Note the bold faced end-of-word and end-of-line markers. The device drivers generally ignore the semantic markers, but the semantic markers permit other analyses, such as that necessary to do the index generation. Note that no `w`s are issued before hyphens generated by the formatter; they come only at the ends of input words. One can submit this intermediate form to a device driver for printing and then, together with the index terms, to the indx program to generate the index.

Without these end-of-word markers, the indexing program would be forced to guestimate the word boundaries based on relative sizes of movements. These guestimates are bound to fail in the presence of wild movements that are found in equations, tables, and pictures. Observe that this fact means that the indx approach cannot be followed on any DVI-based[23] formatter, e.g. TEX,[24] without modification of the formatter to generate more information in the DVI output (something which is politically and economically infeasible).

One of these authors participated in a project to build a bi-directional version of ditroff,[25] to be used to format text involving say, both English and Hebrew. The fact that ditroff's intermediate output has end-of-line markers allows the bi-directional formatting to be achieved by inserting a program called ffortid between ditroff and the device driver. ffortid accepts the intermediate output from ditroff; on a line-by-line basis, it reorganizes the line so that the text in each font is printed in its natural direction; and it outputs this reorganized line in the same format as produced by ditroff, so that the device driver, getting its output, would be none the wiser. The beauty of this scheme is that an unchanged ditroff can be used. Without the end-of-line markers, ffortid would have to guestimate where the end of lines are based on movements; this guestimate is again bound to fail in the presence of tables and pictures.

The lack of end-of-line markers in the DVI format prevents production of a bi-directional version of TEX using the simple scheme of reorganizing the DVI output on a line-by-line basis. The only way it can be done is to modify TEX itself as MacKay and Knuth have noted.[26]

In view of the fact that no device driver really needs the semantic markers, we thought that it was a stroke of luck, genius, or foresight that Kernighan put them in ditroff's output. Given that these markers can be used only for semantics-based processing such as indexing, it was surprising to us that Bentley and Kernighan did not follow an approach similar to ours in their indexing system.[11] When we approached Kernighan with the obvious question, he said that he had put in the end-of-word markers in order to allow someone else to build an editor for documents of the ditroff output format and had never considered using it for indexing.[22]

THE INDX METHOD

The method of producing indexes with indx allows the indexer to concentrate on obtaining the entries in the final index without initially having to know specific page references or having to insert many markers or macro calls into the input text. It is described and compared with the characteristics of some of the other existing methods below. Then the initial expectations of the method are discussed.

Description of the method

The first thing the indexer must do is select the phrases in the text that are to have page references. These phrases are put in the *phrase* file. The indexer must then set up the optional files that are described later, in order to obtain the desired index. The optional files define index terms that are to be combined with each other under one term, that are to be grouped with each other to form headings that have subheadings, that are to have *see also*, *see*, and *see under* cross-references, and that are to be given an alternative heading or subheading. The files are processed in this order and the actual steps taken to create an index are controlled by the files provided by the indexer.

For example, suppose the completed index is to contain

Alphabetic sort, *See* Binary search trees

Binary search trees, 320-372

LEFT pointers, 208-214

Pointers
    LEFT, 208-214
    RIGHT, 208-214

RIGHT pointers, 208-214

and the phrases for which page references are found are

```
Binary search trees
LEFT pointer
LEFT pointers
RIGHT pointer
RIGHT pointers
```

The optional files that would be needed are the *combine-phrase*, *group-entry*, *see*, and *alternative-index-term* files. The page references of the phrases `LEFT pointer` and `LEFT pointers` would be combined under `LEFT pointers`. The same thing would be done for `RIGHT pointer` and `RIGHT pointers`. Then the `Pointers` group entry would be built in the *group-entry* file. In it, would be the definitions for copying the entry `LEFT pointers` under the heading `Pointers` and for copying the entry `RIGHT pointers` under the heading `Pointers`. There are no *see also* and *see under* cross-references to add, so those files are not given. The *see* file contains a line mapping `Alphabetic sort` to `Binary search trees`. The *alternative-index-term* file is needed to change the subheading `LEFT pointers` under the heading `Pointers` to `LEFT` and to make a similar change for `RIGHT pointers`.

As a preview to the detailed description below, the four optional files needed to make the above index example are as follows.

combine-phrase

```
:
LEFT pointers : LEFT pointer
RIGHT pointers : RIGHT pointer
```

group-entry

```
:
Pointers : LEFT pointers :
Pointers : RIGHT pointers :
```

see

```
:
Alphabetic sort : Binary search trees
```

alternative-index-term

```
:
LEFT pointers : LEFT : Pointers
RIGHT pointers : RIGHT : Pointers
```

The input text must be prepared as described below and the indx program is run given the names of the files and the input text. The indexer should save the output to a file so that it may be examined for incorrectly sorted terms and sequences of consecutive page references that should be replaced by a spanning range of page numbers. After the ditroff macro calls are verified and corrected as necessary, they must be formatted using the appropriate macro package to get the final index.

The input text

The form of the text is much like regular text except for two things. First, all sentence punctuation must be separated from surrounding words by at least one blank, to allow for the search of phrases. Secondly, the text of page *n* must be preceded by a line beginning with the sequence `^Lp`*n*. Here `^L` is the formfeed character. If the original text is available as ditroff output, the program dedit (parse that as de-dit and not d-edit) can be used to convert the ditroff output into the text acceptable by indx. One of the differences between setting up a text file manually and running ditroff text through dedit is that all words of the dedit output will be separated from any sort of punctuation by one blank, whereas text set up manually could very well have only sentence punctuation preceded by a blank. For example, the sentence ending ‘U.S. Navy.’ should be ‘`U.S. Navy .`’ in the text file, but the ditroff equivalent of the same part of the sentence would be changed by dedit to ‘`U . S . Navy .`’ This is a relatively minor detail that can be taken care of by searching for the phrase ‘`U . S . Navy`’ and later changing the entry using the *alternative-index-term* file to ‘`U.S. Navy`’.

Selection of phrases in the text

The file of phrases to be searched for in the text must always be provided. The phrase file contains one phrase per line. In order to speed up the initial building of the index, the phrases of the phrase file must be sorted, and each phrase may occur no more than once. Normally, indx ignores case distinctions when searching for phrases in the text, thus normally two phrases differing only by the case of one or more letters will not both be in the phrase file. The UNIX sort command can be used with the `−f` option to sort the index ignoring case distinctions. In the situation in which the case of one or more letters in a phrase matters, the phrase is to be matched in the text only exactly, and the phrase may otherwise even be the same as another phrase in the phrase file, the phrase to be matched exactly will be flagged with a word consisting of a single occurrence of a special character at the end of the line. The special character ideally should be one that is not used in any of the phrases. The special character for the file is announced by its being the first character of the file and the sole occupant of the first line. If it should become necessary to have a non-special use of the special character in a phrase, then it should appear doubled. That is, if '!' is the special character and the phrase 'Yippee !!' must be searched for exactly, it should appear in the file as `Yippee !!!!`. This convention of doubling non-special uses of the special character is used for all other files as well; in the other files, the special character introduced as the first and only character of the first line is used to separate elements of pairs and triples.

In order to determine the list of phrases that will appear in the index, the program findphrases[27] may be used. findphrases scans the input text, finding all repeated phrases up to a user-specified number of words in length, ignoring phrases given in an *ignored-phrases* file. By forming a large enough file of ignored phrases, one can end up with a meaningful list of repeated phrases. This list will contain many, if not all, of the items either that should appear in the index or that suggest other phrases that should appear in the index. This list is just a preliminary one since there will possibly be items to be indexed that occur on only one page, and not on the list of repeated phrases. Some of these unrepeated items may be found by using the `–t` option of findphrases to obtain a list of the tokens, which are single words or single items of punctuation.

One can keep a list of phrases that will be ignored in the majority of documents written in one natural language, i.e. a general-purpose *ignored-phrases* file. For English, it contains phrases such as `the`, `a`, `of`, and `is`. For a particular document, the general-purpose *ignored-phrases* file would be complemented by adding document-specific phrases that should be ignored. A few iterations may be needed to get the right maximum phrase length and a large enough list of ignored phrases to reduce the number of repeated phrases to a useful size, so that it really indicates phrases to be indexed. Normally phrases that start with a phrase from the *ignored-phrases* file but which are not themselves listed in the *ignored-phrases* file are not ignored. However, sometimes one finds too many phrases that start with other phrases in the output. Therefore, one of the options in the findphrases program is to ignore all phrases that begin with a phrase in the *ignored-phrases* file. In any case, with the help of several successive runs of findphrases on the text of a document, the indexer creates the *phrase* file for indx, consisting of all of the phrases whose page numbers are to be gathered

The *phrase* file and the optional files used to create the index of this paper are given in the Appendix. The boldfaced phrases in the *phrase* file are those that have been directly obtained from the list of repeated phrases. About 42 per cent of the final index entries of the thesis came from the list. Many of the other phrases in the *phrase* file have been derived from the list of repeated phrases. That is, the list directed attention to parts of the text that should be indexed and often suggested part of a phrase to be used in the *phrase* file. Merely being directed to a part of the text helped in selecting a phrase to obtain the page reference.

The various optional files

The phrases for which page references are to be combined, the phrases which should be grouped to form group entries, the phrases which should have *see also* cross-references, *see* cross-references, *see under* cross-references, and the phrases whose entry phrases are to be changed are stored in separate files. Depending on the index being formed, any combination of these files may be given by the indexer.

The *combine-phrase* file contains pairs of phrases on each line. Both phrases must be in the *phrase* file. The page references of the second phrase are merged with the page references of the first, putting the merged list with the first term. The second term is deleted from the index. This file is provided to allow concepts that are expressed by more than one phrase to be indexed as one entry and to simulate the appearance of having a reference to a page discussing but not actually containing a term. The second phrase of a pair will often be the plural or some other variation of the first phrase.

The *group-entry* file also contains pairs of phrases on each line. At least the second phrase of each pair must be in the phrase file. The second term will become a subentry of the first term, if the definition is valid. Only two levels are provided; a main entry can have a subentry but a subentry may not have a sub-subentry. Thus, the first phrase will be a main index term. If it is not in the *phrase* file, a main entry will be formed for it, but it will not have page numbers. There are two ways to define a subentry. Either the subentry will be a copy of one of the main entries, or the subentry will appear only under a main entry. Lines ending with the separation character of the file will have the subentry as a copy of the main entry given by the second phrase. For example, to get the following portion of an index, shown in paragraph style:

programming language, 3: C, 4, 7; Pascal, 4, 7, 9

the *group-entry* file might contain

```
!
programming language ! Pascal
programming language ! C
```

If ‘Pascal’ should also be in the index as a main entry, the *group-entry* file would contain

```
!
programming language ! Pascal !
programming language ! C
```

The three *see...* files, the *see*, the *see-also*, and the *see-under* files, share a common basic purpose and format. The files consist of lines containing two or three phrases. In all cases, the index entry for the first phrase will refer to the second phrase with the appropriate ‘see...’, i.e. ‘see’, ‘see also’, or ‘see under’. If there is a third phrase, then the referring entry, described above, will be put as a subentry under the occurrence of the third phrase as a main entry. For each line, the second phrase must be in the *phrase* file. For any line with three phrases, if the third phrase is not already in the *phrase* file or is not already defined as a main entry in a group, then a main entry for that phrase is made. Now the discussion turns to the differences between the three kinds of *see...* files.

In the *see-also* file, all first phrases must be in the *phrase* file. The result of a line is that the entry for the first phrase will have following its page numbers, a *see also* cross-reference to the second phrase. For example, to get the following entry, shown in entry-per-line format:

Program documentation, 11, 17, 24
    See also program design language

the *see-also* file should contain

```
:
Program documentation : program design language
```

In the *see* and *see-under* files, first phrases of pairs or triples are *not* in the *phrase* file. The result of a line is that the entry for the first phrase, which necessarily has no page numbers, will have a *see* or a *see under* cross-reference to the second phrase. The *see under* cross-reference is used when the second phrase has a main entry with subentries and it is desired to refer one or more subentries, and the *see* cross-reference is used when the second phrase has only a regular entry with no subentries under it. Suppose the following portion of the index is desired, shown in entry-per-line format:

PDL; *See* program design language
.
.
.
program design language, 17, 25, 27

The *see* file should contain

```
:
PDL : program design language
```

The *alternative-index-term* file also contains pairs or triples of phrases, which define alternative spellings for main and subentries, respectively. The effect of a line is that the index entry, either a main entry or a subentry, for the first phrase is changed to have the second phrase as its term, and it is printed in the position dictated by the spelling of the new phrase. This file is provided, because certain phrases found in the text may be better represented in the index by other phrases.

Description of the output

As indx reads the files, the lines describing valid transactions are processed and error messages for any invalid lines in the optional files are printed out. Whatever is in the index at the end of the processing, even if there are illegal descriptions, is printed out after all optional files have been processed. With both the error messages and the generated, partial index, it proves quite easy to track down the sources of the errors and to fix them.

The index terms are printed out as a series of ditroff macro calls. At least two macro packages are available, one for the entry-per-line style and one for the paragraph style. The terms are alphabetized word-by-word, with the case of the letters ignored. Non-alphabetic characters in entries are included in the sorting process, which may cause terms to be incorrectly sorted. Strictly ASCII comparisons are made so that numbers will appear before entries beginning with ‘A’.

The indexer may have to correct the order of terms as well as touch up the page references. For one thing, indx will find every page on which a phrase is found, whereas the indexer may not want all of them in the index. In addition to possibly removing some page numbers, the indexer may wish to replace a sequence of consecutive numbers with a range of page numbers, for example replace ‘3,4,5,6’ by ‘3–6’. These two forms are not identical because individually listed numbers means the subject is discussed intermittently on each page, whereas a range of numbers means the subject is discussed continuously on these pages. A program is incapable of making this distinction. So, the human indexer must do it.

It would be useful to have a browser program to find entries with page number ranges. This program would search the indx output for sequences of consecutive numbers, display the entry, and ask for and make changes desired by the human indexer. However, a multi-window workstation works nearly as well.

Comparison with other automated tools

indx, just like the other indexing programs, must be given the phrases of the index. However, the actual phrases for which page entries are to be found are stored in a file instead of being marked in the text or given as macro arguments. Unlike the other programs, indx searches the text for occurrences of the phrases to get the page references. The definition of index entries via the *phrase* file and the optional files is unique to indx. One of the disadvantages of having multiple files defining index entries is that a change to a phrase in the index may have to be made not only to the *phrase* file but to any of the optional files the phrase is in. This is not difficult but can be rather tedious. However, the number of changes to be made is always quite limited, in most cases, one line per file.

indx provides three types of cross-referencing. None of the other programs seems to support *see under* cross-referencing. This is probably because *see under* cross-references are not used in books in the U.S. Also, in most, cross-reference targets are not verified as being entries in the index. indx does not allow an index term to cross-reference an entry that does not exist as a main entry in the index. In addition, only entries having page references are allowed to have *see also* cross-references. It is verified that all entries referred to by a *see under* cross-reference be main headings of group entries. The usage of the actual phrase in a subheading of a group referred to by a *see under* cross-reference is not verified, so that it is the indexer's responsibility to use the *see under* cross-reference properly. indx does not allow check entries or circular *see* references, in which two page-less terms in the index cross-reference only each other. If such a pair of entries is desired for the purpose of detecting and proving copyright violations, it must be manually added to the macro calls for the index. More generally, having a *see* cross-reference to an index term that has a *see* cross-reference to any term is not allowed by indx.

The sorting capabilities of indx are average when compared with those of the other programs. The indx program implementation inherently supports word-by-word alphabetization without ignoring punctuation. As in many of the other programs, there will probably be phrases in the wrong order, making the human indexer responsible for correctness.

The maximum size of an index in indx is unknown, as it is limited by the memory size of the machine. For any set of files, though, the total number of entries and subentries can be approximated by summing the number of lines in the *phrase*, *see*, and *see-under* files, adding the number of lines in the *group-entry* file which define the subentry as a duplicate of a main entry, and then subtracting the number of lines in the *combine-phrase* file. The *phrase* file gives the maximum number of main entries having page numbers and the *see* file and *see-under* file together give the minimum number of entries (main or sub) not having page numbers. One entry is created for every subentry formed by duplicating a main entry, and one entry is destroyed for every line in the *combine-phrase* file. The first reason why this sum is only an approximation is that when the *group-entry* file is processed, if no main entry exists in the index for a subentry to be placed under, it is created and added to the index. Entries formed in this manner will not have page references or cross-references. The second reason is that using the number of lines in each of the files includes the first line which has no phrases. To calculate the exact number of macro calls in the output, the formula described above should be used with one less than the number of lines in each file. This subtotal should then be increased by the number of unique group headings which were not searched for in the index.

The two ditroff macro packages provided with the software give the user the option to choose between entry-per-line style and paragraph style. The combined style is not necessary for indx, because with only two levels of entries, the combined style is equivalent to the entry-per-line style. These macros are not as flexible as the Winograd and Paxton TEX macros by which the indexer can control many parts of an individual entry.[28] indx formats a basic index, which should be sufficient for most indexers. However, since the macros are published, they can be modified to be as fancy as desired.

Although main page references cannot automatically be boldfaced, the indexer may be able to print entry phrases that have special characters or characteristics. For example, if the indexer desires the phrase ‘Absolute zero’ to be boldfaced in the index, he or she may have `\fBAbsolute zero\fP` in the input text and in the *phrase* file. When ditroff is run on the indx output, the `\fB` and `\fP` commands will be interpreted and the ‘Absolute zero’ entry will be bold faced. An easier way to achieve this is to find the phrase ‘Absolute zero’ and then rename it using the *alternative-index-term* file to `\fBAbsolute zero\fP`. Entries containing such formatting commands will definitely be missorted in the index and their positions must be corrected by the human indexer. This is why such commands are advised against in Salz’s index[10] and why the ability to specify a sort key in the Bentley and Kernighan indexing suite[11] is a good idea.

The entire index must be built at once. One of the problems in building the index with sections of the text is that entries being cross-referenced to may not exist in the portion of the index being built. The program would be unable to verify cross-references without having the entire index to search.

Initial expectations of the method

It should be fairly easy to set up the files for the index terms once the indexer identifies the terms to be indexed and their relationship to each other. The optional files should be straightforward if the indexer uses each file correctly. For example, the second terms of the *combine-phrase* file will be deleted from the index after its page numbers have been added to the other term so no other optional files should contain that phrase. The only valid way such a phrase could exist in another file is if another phrase is renamed to this deleted phrase in the *alternative-index-term* file.

Because a subject heading is most likely to be referred to using more than one phrase, it is expected that the *combine-phrase* file will be quite long. Also, because a subject heading is most likely to be a synopsis of the phrase actually used in the text, the *alternative-index-term* file should also be quite long. It is expected that most of the definitions in the *alternative-index-term* file will be for subentries. Parts of the phrase used for obtaining the page references of a subentry will probably appear redundant when the entire entry is printed, as the group heading identifies the subject of the entry. Thus, most subheadings would be shortened or summarized.

The amount of time needed to create the index depends on several things. Increasing either the length of the text to be indexed, the number of phrases in the *phrase* file, or the lengths of these phrases will increase the execution time. The number of phrases to be combined will also have a great influence on the time it takes to generate an index because of the page-number merging that must be done.

Depending on the phrases that eventually make up the index, there may be some sorting problems. As mentioned above, alphabetization is done word-by-word. Phrases containing punctuation characters may end up slightly misplaced since there is no way to ignore the punctuation in the sort. Those misplaced phrases will be either inverted, since they contain commas, or contain printing sequences. An example of this second type is the `\fBAbsolute zero\fP` phrase mentioned earlier. The word-by-word sort would take `\fBAbsolute` as the first word of the phrase instead of `Absolute`. It is the human indexer’s responsibility to check that the index terms are sorted correctly. Only those terms containing non-alphabetic characters need closer examination. These can be found easily by any editor.

The standard output should be redirected to a file so that the indexer can check the order and the page references of the entries. It is expected that many of the page numbers for a term may be useless, since every single occurrence of the term gets listed. The number of useless terms should decrease as increasingly specific phrases are selected. Most of the indexer's time spent on preparing the indx output for printing will be ensuring that page references are meaningful and replacing a page sequence with a range if the subject is discussed continuously across consecutive pages.

THE INDXING EXPERIENCE

The first application of indx was to form an index for the thesis[3] on which this paper is based. This task was split into four parts. The first and most difficult entailed selecting the entries for the index, referred to as entry selection, and selecting the actual phrases from the text that represented them, referred to as phrase selection. The second part involved setting up the necessary files and running indx. The third part involved checking the output for incorrectly sorted entries and incorrect page references. After the macro calls were checked, the final index was printed using an ml macro package developed for ditroff.

The selection of entries and phrases

Because the authors were novice indexers, the phrase selection task was very difficult despite having the findphrases output. The list of repeated phrases aided in determining much of the entries that should be in the index. In fact, out of the 96 phrases in the *phrase* file, 63 were taken directly from the list and six were derived from the list. Some of the phrases from the list would have been found without the help of findphrases. These phrases, though, are obvious since they are names of things, such as of the optional files, of the sorting schemes, of the formatting styles, and of the other programs mentioned. These are almost always referred to using the same phrase. It would have been difficult to find phrases for general discussions whose topics should be indexed, such as the manual indexing task and phrase selection. However, it was still necessary to read through the text to identify other entries that should be included but were either not repeated or were described in such a way that their significant terms were too common to be searched for. Much of the difficulty lay in how the entries were arranged — whether partially related entries should be kept separate or placed together under one heading. It seems that this difficulty was due to lack of indexing experience.

In order to use findphrases, a file of ignored phrases was needed. The *ignored-phrases* file was a combination of generic and text-specific ignored phrases. The text-specific ignored phrases should include not only phrases not meant to be indexed but common phrases that alone will not be an entry in the index. For example, since 'index' is such a common word in the thesis, it made sense to ignore it, as all of the occurrences of the phrases would not be referring to the same concept anyway. Ignoring this word, however, caused the references to the programs Index, index, and INDEX to be omitted from the list of repeated phrases. Thus phrases for these three programs had to be determined manually. The ignored phrases used were determined with the intention of using the –b option of findphrases, which causes any phrase beginning with a phrase in the *ignored-phrases* file to also be ignored. Care must be taken in determining text-specific ignored phrases when the –b option is used, so that important phrases will not be ignored unintentionally.

findphrases was to be run on the thesis with a maximum phrase length of five. The thesis was run through the necessary ditroff preprocessors, through ditroff, and then through dedit into a single file. This file had to be partitioned since it was too long to be processed at once by findphrases. The main problem with not being able to submit the whole text at once to findphrases and having to submit parts separately is that some repeated phrases may simply not be found. If a particular phrase appears only once in each part, then it will not show up in any part's listing, whereas it would show up if the whole text were submitted at once. Thus, manual care had to be taken not to lose any of these phrases. This care involved manually inspecting the the complete list of tokens generated for each part. It was also little more work to check back and forth between the different lists for entries to be indexed. A problem was encountered when there were too many repetitions of phrases, causing findphrases to stop. The highly repetitive phrases were added to to file of ignored phrases and the program was restarted. Owing to lack of time, previous runs were not repeated with the expanded *ignored-phrases* file. Thus most of the lists were obtained with different *ignored-phrases* files.

Normally, it would be useful to suppress counting phrases which are everywhere wholly contained in the same other counted phrase; i.e. if the words 'espresso' and 'coffee' each appear ten times and so does the phrase 'espresso coffee', not much information is conveyed by listing the individual words and their appearance just clutters the list of repeated phrases. One can use the −u option of findphrases to suppress the listing of these contained phrases. However to reduce execution time, this option was not used. Also, as will be explained later in this section it turned out that it was better not to use this option.

findphrases printed the input text file with the lines numbered and the list of repeated phrases, first sorted by frequency and then sorted alphabetically. Since the frequency with which phrases occur is not too important for the index, the alphabetically sorted list was mainly used. The list was scanned for possible entries and it was easy to determine the context in which the phrases occurred by referring to the lines listed. Because the input text included the indx page markers, it was also easy to determine whether a phrase would give the page references desired for an entry. The frequency listing, although it was not used at all to create the first index, may be helpful in suggesting subentries, as subentries are more specific than main entries and thus should occur less frequently than their respective main entries.

The lists were also scanned for terms that may have been missed when reading through the text. It seemed that skimming through the text first to jot down possible entries and phrases before looking at the repeated phrases list was easier than beginning with the list. The entries seemed to be more balanced this way.

An alternative of using ditroff and dedit for preparing the findphrases input file is to use deroff. Deroff will output the source text with the ditroff commands removed. This output should be saved in a file and is in the form accepted by findphrases. The file is obtained faster than with ditroff, but a major drawback of this method is that page markers are not present to aid phrase selection. It is also possible to use the source file as input to findphrases. This would allow searching for phrases containing commands such as font changes. Any formatting commands not contained in phrases should be added to the *ignored-phrases* file. As with deroff, the input file will not contain any page markers.

As the entries and phrases were selected, one of the foremost priorities was to reduce the amount of processing done by indx. This priority affected phrase selection in several ways. First of all, the maximum number of words in a phrase was kept to a minimum. Remember, the longer the lengths of the phrases in the *phrase* file, the longer searching the text will take. As mentioned earlier, it was advantageous that the −u option was not used with findphrases. Although the list of repeated phrases was much longer than if the option had been used, it enabled identification of the shortest possible phrase that would yield the same page references as the phrase for an entry in the index. For example, in Chapter 1, ‘semi-automatic indexing’ was defined on the very pages that the word ‘semi’ is found on. Thus, instead of using the entire phrase consisting of four words, the word `semi` was used in the *phrase* file. Secondly, the number of phrases beginning with the same word was kept to a minimum. More specifically, having phrases which were identical in the first five letters of the first word was avoided as much as possible. This was done to have the phrases distributed evenly in the search table of the hash function.

Setting up the optional files for indx

As the entries and the phrases to give the page references were selected, they were written on a sheet of paper with the phrases on one side and the entries on the other. For example:

Phrases	**Entries**
select the phrases + select the terms	selection of phrases
allows finding + finds repeated	findphrases program
+ findphrases	
alphabetized	Alphabetization schemes
letter ’	letter-by-letter
word ’	word-by-word

This example shows the phrases associated with three of the entries of the index. Two are main entries, one is a group entry. The phrases shown added together will be combined to give the page references, and the italicized phrase shows the phrase under which all of the page references end up. All of the phrases were placed in the *phrase* file, which was then sorted using the UNIX sort. The optional files were created using these same phrases. The only files in which an entry was used were the *group-entry* file, when group headings were not searched for in the text, and the *alternative-index-term* file. It was very easy to set up the files once the phrases and entry phrases were outlined as shown. One of the phrases was missing from the *phrase* file, which caused several error messages to print out on the first run of indx, but the mistake was easy to track and correct. For the fragment of the index given above, the optional files and their contents are:

combine-phrase file:

```
:
select the phrases : select the terms
allows finding : finds repeated
allows finding : findphrases
```

group-entry file:

```
:
alphabetized : letter ’
alphabetized : word ’
```

alternative-index-term file:

```
:
select the phrases : Selection of phrases
allows finding : findphrases program
letter ’ : letter-by-letter : alphabetization
```

```
word ’ : word-by-word : alphabetization
alphabetization : Alphabetization schemes
```

In the run of indx on the thesis, there were 96 phrases in the *phrase* file, and the following table gives the number of definitions in each of the optional files.

File	No. of definitions
combine-phrase	29
group-entry	43
see-also	4
see	2
see-under	7
alternative-index-term	48

As expected, the *alternative-index-term* and *combine-phrase* files contained many more definitions than the other files. However, the *group-entry* file also contained many definitions since many group relationships were formed for this index. The majority of the definitions in the *alternative-index-term* file were for renaming subentries, also as anticipated. Out of the 48 definitions, 37 of them were for subentries.

Checking the indx output

The file was checked for correct sorting of phrases and correct page references. None of the entries were sorted in the wrong order since none contained any formatting commands. The page reference check was time-consuming since all of the references had to be looked up to verify their usefulness. Setting up editors in multiple shell windows would ease the checking process as the human indexer can simultaneously view the macro call being checked, the indx input text, and any of the indx files. If there was a subsequent page reference to the next page, a check was also made to see if the sequence should be replaced by a range. Keeping a record of the phrases and the pages on which they were inspected and chosen for indexing would have made the page reference check considerably faster, especially for the more common phrases which may supply extraneous page references. There were several-page-long runs which were replaced by ranges, but most of the corrections stemmed from removing useless page references. This is partially due to the method of finding all occurrences of a phrase, but with a better selection of phrases, the number of useless page references should decrease. One disadvantage of having separate lists of repeated phrases is that a phrase selected from one of the lists may be good for one part of the text, but will bring in other useless page references when the entire text is used. For example, ‘semi’ as mentioned earlier was sufficient to give the page references for the definition of semi-automatic indexing found in Chapter 1 of the thesis. Later in Chapter 3, ‘semi’ appears as part of an example, which resulted in an extraneous page reference. In fact, more extraneous page references have been generated for that entry in this section.

Updating the index

The general method described here was used to update the first index to obtain the index found at the end of this paper. Updating the text often caused indexed discussions originally on one page to be split over two pages. If the phrase selected to obtain the original page reference does not appear twice in such a location that both page references will be obtained, another phrase from the missing page must be selected for that entry. This is not very desirable but cannot be avoided at the moment. The opposite situation may occur as well. Discussions once split over two pages may end up being on one page. If more than one phrase is used to obtain the page reference and one of the phrases does not add in other page references for the entry, it may be deleted from the *phrase* and the *combine-phrase* files as it will no longer be contributing any new line numbers for the entry. Having extra phrases will prolong the execution of indx, but having several extra phrases is better than having too few. It was also found that phrases once specific enough to obtain page references for a given entry could become too general if the textual additions contained many trivial occurrences of the phrase.

The ability to run the entire text at once through findphrases would have helped make new phrase choices if they were needed. The UNIX program grep was also used to help determine whether a phrase would provide the correct page references. It was especially helpful for indexing Chapter 4 since there was no list of repeated phrases for the chapter. It was very helpful for determining whether a phrase which begins with a phrase that was ignored by findphrases should be used in the *phrase* file.

A method of updating the index after revising the text has been suggested. The Revision Control System[29] is used to keep old versions of the repeated phrases list which may be compared with a newly generated list using diff. The difference identified will indicate whether changes need to be made to the *phrase* and the optional files to update the index. Should the difference in lists be mainly composed of the same phrases found on different lines, this method will have saved a lot of time over redoing the index from scratch. The method of using diff to compare old and new phrase lists for a given part of the paper would have been used had the changes been extensive, but since most of the changes were stylistic ones, findphrases was not rerun on these parts. Hindsight revealed that given even minor changes, it may be very useful to rerun findphrases as the sectioning of the text into pages will be changed.

Indexing this paper

The same *phrase* file used for indexing the thesis was used initially to index this paper since no new terms are introduced in this paper. During the scan of the `.IX` lines, the terms having no occurrences were checked. If the term belonged to a discussion of the thesis but was omitted for this paper, it was edited out of the `.IX` macros. If the phrase chosen to represent a discussion was removed or the phrase was no longer sufficient to provide the page references, another phrase to represent the discussion was selected. One phrase which was part of a further expanded discussion was added to the *phrase* file in order for the page number of the new discussion to show up under its respective index term. In two cases, large group entries were compressed into main entries since much of the detailed discussion had been omitted. For these, the phrases in the *phrase* file were kept and the *combine-phrase*, *group-entry*, and *alternative-index-term* files were changed.

The output file was then run through ditroff partially with the ml.line.per.entry and partially with the ml.paragraph macro packages defined for the entry-per-line and paragraph formats.

The *phrase* file and the optional files used to generate the index are given in the Appendix. The boldfaced phrases in the *phrase* file are those that were taken directly from the lists of repeated phrases. Also given in that appendix are the macro calls output from indx from which the macro calls used to generate the index of this paper were obtained and the ml.entry.per.line macro package showing one possible definition for the `.IX` macro.

AREAS OF FURTHER WORK AND CONCLUSIONS

The indx program was designed to provide the indexer with enough options for creating a good index. As is true of most programs, there are several areas in which indx should be modified to improve its performance and to provide additional features.

Improving the performance of indx

The two major areas where improvement directly affects the performance of indx are the page-merging routine used in combining phrases and the hash function used to distribute the index terms among the binary search tree buckets of the hash function search table. The implemented page-merging routine is not the usual kind of merge routine in which several lists are merged to give a new list. The merged list replaces the original page list of the first term on a *combine-phrase* file line. More specifically, the page references of the second list that are not already present in the first list are inserted into the first list. The page lists are scanned from the beginning and should the end of the first list be reached before the end of the second list during the merging process, the remainder of the second list is appended to the first list. Note that in this case, the remainder of the list is copied and added to the first list instead of being moved over. This way, the page list of the second term remains intact, although, by the definition of combining phrases, the second term will be deleted from the index after the page lists are merged.

There is probably a more clever and faster way of merging the two lists, which once implemented would decrease the execution time for combining phrases. An efficient routine is desirable here since combining phrases to form entries is one of the most frequent operations, even for a small index.

The hash function used is very simple and easy to calculate. It was chosen mainly because it avoided having to retrieve all of the tokens of a phrase. It is particularly advantageous to have the function depend on the first word only since the existence of a heading that begins with a certain word can be easily determined with its help. The hash function provides access to the only binary tree that might contain an index heading beginning with the given word. Were the hash function dependent on the entire phrase, this fast lookup would not be possible without either searching the index directly, thus bypassing the search table, or adding some data structure by which this information would be obtained.

The hash function has some drawbacks, as it does not distribute the index terms evenly among the buckets of the search table when many groups of index terms have the same second through fifth characters. It is not unreasonable to expect that several headings of an index will begin with the same word. As the number of headings that get hashed to the same location increases, the time to delete one of those headings will also increase, whether it be through its combination with another entry, its relocation under another entry, or its removal in the process of renaming the entry. Thus, as larger indexes are produced, the advantages of the current hash function would be outweighed by the disadvantage of managing large binary search trees. A slightly more complicated, but better distributing hash function should be used despite having to add a table structure for determining whether any index entry begins with a given word.

Additional features of indx

Several features that would be nice to have available became apparent after examining other semi-automatic indexing programs and after using indx.

1. **Increasing the number of cross-references**. Although having one entry to cross-reference is usually sufficient, there are times when a second or even a third heading should also be

cross-referenced.

2. **Increasing the index levels**. The vast majority of indexes desired could be built by indx if the number of index levels possible were increased to three. Having more than three levels would probably not be worth the effort to implement since four or more levels is quite uncommon in indexes.
3. **Sorting scheme options**. indx would also be more complete if it provided the indexer with the choice of sorting entries word-by-word or letter-by-letter.
4. **An explicit sort key**. As used in the Bentley and Kernighan indexing programs, having the ability to explicitly define a sort key guarantees that the index terms will be sorted correctly even though they may contain formatting commands, punctuation characters, or digits.
5. **Merging indexes**. indx may be very useful if parts of an index could be created and merged to form a complete index. This would allow the creation of indexes on a chapter-by-chapter basis, whereupon indexes of each chapter are merged to form the index.
6. **Keeping a combined phrase in the index**. While using indx to build the initial index of this paper, the option of keeping the second term of a *combine-phrase* file definition rather than having it automatically deleted became desirable. Just as the *group-entry* file allowed an entry to become a subentry while remaining a main entry, the ability to use a phrase to supply page references to more than one phrase was found to be handy. Currently, if such entries were desired, another phrase would have to be chosen to yield the same page numbers. This would put an additional burden on the human indexer and would cause the use of more index terms than necessary.

These changes could be incorporated by changing the program or by using the Bentley and Kernighan approach of adding awk scripts in a pipe line.

Conclusions

The indx program has been successfully used to create the index of this paper. The method as a whole is simple to use once the difficult task of selecting phrases and entries is done. This selection task must be done for all semi-automatic indexing programs. However, instead of entering them in the source text, they are entered in the *phrase* file, thus avoiding messing up the source text.

The novel idea of using different files to create the various parts of the index may at first seem more involved than the other programs especially since what is easily done by the human indexer when inserting indexing macros in a text must be explicitly defined in the *combine-phrase* and *alternative-index-term* files. However, there are several reasons for storing all the information about the index in several files than having parts of the index scattered among the text. First, should the human indexer wish to change the heading of a term, all the various places in the text where the macro is given would have to be searched for and changed, whereas with indx, the phrase would have to be changed only in a few short files. In many cases, the only phrase that would need changing would be in the *alternative-index-term* file. Secondly, it seems easier to track down discrepancies in the index when all definitions are organized in specific files. Thirdly, it is easier to remove index terms from the index by deleting the appropriate lines of the files involved. Fourthly, as discussed earlier, one can calculate the total number of entries by the number of definitions in the files. Also, it should be easier to restructure the index. For example, pulling several main entries under one group or breaking up a group entry into several main entries are accomplished by changing one line in the optional files for each entry involved instead of changing each occurrence of the indexing commands for those entries in the source text. In general, changes may be made to the index without having to touch the text file.

indx helps the human indexer create good indexes to the extent that cross-references in the index will be checked and the annoying chain reference in which the reader is directed from one entry to another to yet another without having any page references to look up is disallowed. It is necessary for the human to spend some time eliminating useless page references. Using specific phrases that are as short as possible will reduce the amount of checking needed.

The process of creating indexes using indx is made easier by use of other tools, such as dedit and findphrases. Dedit prepares source text in ditroff output format for use by indx. The dedit output has also been shown to be useful as input to findphrases. findphrases has been shown to aid in the phrase selection process. It was found to be especially helpful in identifying phrases that would locate discussions of concepts not having specific phrases to represent them. In order to improve its usefulness, findphrases needs to be able to search through the entire text at once so that separate lists will not have to be cross checked manually.

Other tools do not exist yet. Additional options to findphrases would make it more useful for indexing. An option to list all phrases may help the indexer in identifying phrases for index entries, but there may be too much output for the human indexer to sift through. The browser program suggested earlier would aid the page reference check specifically in the conversion of a sequence of page numbers into a range of numbers.

indx performs the clerical process of the indexing task, finding the page references of entries and arranging the entries in the desired order, without cluttering the text being indexed with indexing commands. This program and its related tools provide much help to the person creating an index. However, the indexing task remains a highly intellectual and human intensive one.

One of the referees for the paper appears, from the thoroughness of his or her comments, to be a professional indexer who has used other indexing software. The referee legitimately complained about all the gyrations that indexer must perform to produce an index with the suite of programs described herein and was glad that he or she does not have to use this present suite.

We are forced to agree that there are a lot of gyrations. However, do notice that almost all of the gyrations were in the term selection process, which by its very intelligence requiring nature is the hardest to automate. findphrases is offered only as a tool to be used by an intelligent indexer to help locate the phrases that are talked about in the document being indexed. Ultimately the indexer has to know the document well enough to pick the terms him or herself. The alternative is to use no tool and do it completely by hand, as with the other systems mentioned in the introduction section.

That findphrases is the weakest part of the suite is demonstrated by the fact that this referee found some indexing terms that we had failed to identify. These were added to the appropriate files for generation of the current index of this paper. It is hoped that the difficulties with this weakest link do not detract from satisfaction with the rest of the suite. If we would be satisfied with unverified page references and either

1. lists of only individual page numbers and no ranges or
2. uniform replacement of each sublist of consecutive page numbers by a range regardless of whether or not the range represents a single discussion,

then the production of the formatted document and formatted index can be completely automatic once the terms are identified. Indeed, in preparing drafts, we skipped the page number verification and range identification, and we have a makefile, shown in the Appendix, that prepares a complete printed copy of the paper while we go out to lunch. Only just before sending a version of the paper to the journal did we manually intervene in this process to verify page numbers and identify ranges.

ACKNOWLEDGEMENTS

The authors thank Christine Aguilera for her willingness to modify her findphrases program to meet the needs of indexing. They thank David Kay and the anonymous referees for their comments on an earlier draft. Finally they thank Brian Kernighan for his useful discussions over the network and for providing a set of ditroff macros for typesetting this paper in *SP&E* 's format.

REFERENCES

1. E. T. Harris, 'A guide for the preparation of indexes', *Technical Report*, The Rand Corporation, Santa Monica, CA, 1965.
2. B. W. Kernighan, 'A typesetter-independent TROFF', *Computing Science Technical Report No. 97,* Bell Laboratories, Murray Hill, NJ 07974, March 1982.
3. K. K. Takata (n.k.a. K. K. Abe), 'indx, a semi-automatic indexing program', *M.S. Thesis*, Computer Science Department, UCLA, Los Angeles, CA, 1987.
4. G. N. Knight, *Training In Indexing,* The M. I. T. Press, Cambridge, MA, 1970.
5. *Chicago Manual of Style,* Thirteenth Edition, University of Chicago Press, Chicago, 1982.
6. R. Gardner and E. Gardner, 'Computer-aided indexing with SPITBOL and TEXTFORM', *The Indexer*, **13**, 115–119, 1982.
7. R. L. Collison, *Indexing Books,* John De Graff, Inc., New York, NY, 1962.
8. R. L. Collison, *Indexes and Indexing,* John De Graff, Inc., Tuckahoe, NY, 1969.
9. P. Hardy, 'Computer-aided indexing of technical manuals', *The Indexer*, **15**, 22–24, 1986.
10. R. Salz, 'INDEX', *Technical Memo*, Mirror Systems, Inc., 1986.
11. J. L. Bentley and B. W. Kernighan, 'Tools for printing indexes', *Electronic Publishing*, **1**, 3–17, 1988.
12. R. L. Aurbach, 're: IdxTEX', *TUGboat*, **7**, 187, 1986.
13. L. Lamport, LATEX *User's Guide & Reference Manual,* Addison-Wesley, Reading, MA, 1986.
14. T. Hofmann, 're: latexindex ', *TUGboat*, **7**, 186, 1986.
15. *Interleaf Workstation Publishing Software User's Guide,* Interleaf, Inc., 1986.
16. R. Kerstetter, *Illustrated Ventura,* Wordware Publishing, Inc., Plano, TX, 1988.
17. C. Anderson, '<<ANSWER: an "off-the-shelf" program for computer-aided indexing', *The Indexer*, **13**, 236–238, 1983.
18. J. M. Pasachoff and N. P. Kutner, 'Computer assistance in indexing with *INDEX', *The Indexer*, **12**, 173–174, 1981.
19. L. K. Fetters, 'INDEXIT: an economical but limited indexing program', *DATABASE*, **9**, 54–56, 1986.
20. L. K. Fetters, 'Indexing software abounds', *Small Press*, **4**, 50–55, 1986.
21. D. L. Parnas, 'On the criteria to be used in decomposing systems into modules', *Communications of the ACM*, **15**, 1053–1058, 1972.
22. B. W. Kernighan, *Private communication*, 1987.
23. D. Fuchs, 'Device-independent file format', *TUGboat*, **3**, 14–19, 1982.
24. D. E. Knuth, *The* TEX*book,* Addison-Wesley Publishing Co., Reading, MA, 1984.
25. C. Buchman, D. M. Berry, and J. Gonczarowski, '*DITROFF/FFORTID,* an adaptation of the UNIX *DITROFF* for formattingbi-directional text', *ACM Transactions on Office Information Systems,* **3**, 1985.
26. D. E. Knuth and P. MacKay, 'Mixing right-to-left texts with left-to-right texts', *TUGboat*, **8**, 14–25, 1987.
27. C.S. Aguilera, 'Finding abstractions in problem descriptions using findphrases', *M.S. Thesis*, Computer Science Department, UCLA, Los Angeles, CA, 1987.
28. T. Winograd and B. Paxton, 'An indexing facility for TEX', *TUGboat*, **1**, Appendix A, 1980.
29. W. F. Tichy, 'Design, implementation, and evaluation of a revision control system,' *Proceedings of the 6th International Conference on Software Engineering,* Computer Society Press of the IEEE, Washington, D.C., September 1982.

INDEX

Blind references. *See* Chain references
Chain references 22
  See also Circular references
Check entries. *See* Circular references
Checking indx output for
  alphabetic order. *See* Alphabetization problems
  page references 9, 12, 15, 18, 22
  see-under cross-references 13
Circular references
  See also Chain references
Combine-phrase file. *See under* Optional files
Creating the paper index 2, 3, 18, 19
Cross references 3, 6, 22
  see 3, 10, 12, 13
  see also 3, 10–11, 13
  see under 3–4, 8, 10, 12, 13
dedit 9, 16, 22
ditroff 1–2, 5, 6–7, 9, 12, 14, 15–16, 19, 22, 23
Enhancing the features of indx 6, 7, 21
Entry phrase 3–4, 10, 14, 17
  See also Heading
Error handling 2, 12, 17
findphrases program 1, 10, 15–16, 19, 22, 23
  aid in phrase selection 15–16
  ignored phrases file 10, 15–16
  improvements suggested 22
  preparing input file 16
Formatting final index 14, 15, 19
Formatting styles
  combined 4, 8, 10, 14, 17, 21
  entry-per-line 4, 11, 12, 14, 19
  paragraph (run-in) 4, 11, 12, 14
Group entry 3, 8, 17, 21
Group-entry file. *See under* Optional files
Hash Function 17, 20
Heading 3–5, 8–9, 14–15, 20–21
INDEX 5, 23
INDEXIT 5–6, 23
indx method 1–2, 8–9, 14–15, 18–19, 21
  advantages of 6, 21–22
  disadvantages of 22

Now switch to paragraph style

indx tools: browser program, 13, 22; dedit, 9, 16, 22; findphrases, *See* findphrases program; grep, 19; updating shell script, 19, 23
Main entry, 3, 11–12, 13, 17, 19, 21
Manual indexing task, 4, 15
Numerals, alphabetization of, 5
Optional files: alternative-index-term, 8–9, 12, 14, 17–19, 21; combine-phrase, 8–9, 11, 13, 14, 17–19, 20–21; group-entry, 8, 11, 13, 17–19, 21; see, 8, 12, 13; see-also, 11; see-under, 13; separation character, 11
Output of indx, 9, 12. *See also* Formatting final index
Page merging routine, 14, 20, 21
Page reference, 3, 4, 8–9, 10–12, 13–17, 18–19, 20–22
Performance, improving indx's: hash function, 17, 20. (*See also* Search Table); page merging routine, 14,
20–21 Phrase file, 8–9, 10–12, 13–17, 18–19, 21: special character, 10
Preparing input files, 6, 8–9, 14, 17
Preparing input text, 2, 8–9, 10, 14, 16
Regular entry, 3, 12
Search Table, 17, 20
Searching the index. *See* Search Table
See file. *See under* Optional files
See-also file. *See under* Optional files
See-under file. *See under* Optional files
Selection of phrases, 4, 8, 15–16, 17, 22: aid for, *See under* findphrases program; difficulties in, 16; for indx, 10, 15–17
Semi-automatic indexing: criteria for programs, 6, 23; definition of, 1, 17, 18, 20, 21, 23; motivation for, 2
Semi-automatic indexing programs, 5, 7, 14, 21, 23: *INDEX, 5, 23; <<ANSWER>>, 5, 23; INDEX, 5, 23; INDEXIT, 5–6, 23
Size of the index, determining, 13, 20, 21
Sorting of entries, 4–5, 6, 9, 12, 13–14, 21. *See also* Alphabetization schemes
Special characters, alphabetization of, 9, 12, 13–14, 15, 21
Subentry, 3, 4, 11, 12, 13, 14, 16, 18, 21
Table of contents, 1
Updating an index, 18–19

APPENDIX: INDX FILES AND OUTPUT

This appendix contains the files used by indx to form the index of this paper. The boldfaced phrases in the *phrase* file have been taken from the lists of repeated phrases from running findphrases. To save space, the *phrase* file is printed in two columns. Next are the optional files in order of processing by indx: *combine-phrase*, *group-entry*, *see-also*, *see*, *see-under*, and *alternative-index-term*.

Following the files is a printout of the output obtained directly from indx as a result of using these files. The input text is the main body of the paper itself. Following this output are an entry-per-line version of the definition of the `.IX` macro, that is the ml.entry.per.line macro package, and the makefile that is used to typeset this paper.

The *phrase* file

:
* INDEX :
< < answer
a minimum
abstraction
actual phrase
algorithm
allow for merging
allows finding
allupper
alphabetized
alternative -
approximated
ASCII comparisons
automatically deleted
automating
awk
browser program
calculate
called chunks
Chain reference
changes been extensive
chapters were run
check back
checking the page
chunk file
chunkfile . p
Circular references
combine -
combined
combined style
create cross
criteria
dedit
deroff
design decisions
difficulty laid
ditroff
ditroff macro calls
ditroff macro packages
documate
doubly
else PrintErrorMsg
enabled identification
end index
entry -
Entry phrase
entry phrases
erroneously placed
error
eventually get the following
expected that many
extraneous
file of phrases
find a subentry
find an index
findphrases
findphrases needs
finds repeated
from the list
generate it
grep
grep was
group -
Group entry
hash function
Heading
headings and
ignored phrases
incorrectly sorted
increased to three
INDEX :
INDEXIT
index for the paper
index of the paper
individual files
indx helps
involves expanding
LATEX
letter - by
main advantage
main entries
Main entry
manual indexing
merging
mI :
minor changes
modifications proposed
Module 5
Module 6
non-alphabetic
novel idea
obtain the index found
on one side
Page reference
page reference check
page references
paragraph ,
paragraph or
paragraph style
partitioned since
phrase selection
phrases (found
phrases . p
phrases are implemented

printed using
program modularized
punctuation
range
Regular entry
related
repeated phrases aided
retrievals of
revision control
Roman numerals
Salz
scheme
search table :
searching for a heading
second cross
see - also file
see - under file
see also cross
see and
see cross
see file
see under cross
select the phrases
select the terms
semi
semiautomatic
separation character
set up
shell script
should save
significant phrases
sorted
sorting
special character
spend some time
standard input can
standard input is
Starindex
subentries
Subentry
Table of contents
takes longer
TEX
the input text
the method
the phrase file
unfamiliar
units are
updating the index
used in a subentry
word - by

The optional files

combine-phrases **file:**

```
:
actual phrase : used in a subentry
algorithm : else PrintErrorMsg
allow for merging : automatically deleted
allow for merging : increased to three
allow for merging : involves expanding
allow for merging : scheme
allow for merging : second cross
allows finding : findphrases
allows finding : finds repeated
allupper : chunkfile . p
calculate : approximated
called chunks : chunk file
chapters were run : the input text
combined : combined style
create cross : related
create cross : unfamiliar
deroff : partitioned since
difficulty laid : check back
ditroff macro packages : mI
end index : Module 6
Entry phrase : entry phrases
findphrases needs : significant phrases
```

```
from the list : a minimum
generate it : manual indexing
Heading : headings and
incorrectly sorted : erroneously placed
incorrectly sorted : non-alphabetic
incorrectly sorted : punctuation
Main entry : main entries
Module 5 : phrases ( found
novel idea : indx helps
novel idea : main advantage
obtain the index found : changes been extensive
obtain the index found : grep
obtain the index found : index for the paper
obtain the index found : index of the paper
Page reference : page references
paragraph style : paragraph ,
paragraph style : paragraph or
phrase selection : select the phrases
phrase selection : select the terms
phrases are implemented : phrases . p
program modularized : abstraction
program modularized : design decisions
range : checking the page
range : extraneous
range : page reference check
repeated phrases aided : enabled identification
retrievals of : find an index
retrievals of : searching for a heading
see cross : see and
semiautomatic : awk
semiautomatic : documate
semiautomatic : LATEX
semiautomatic : Salz
semiautomatic : Starindex
semiautomatic : TEX
set up : eventually get the following
set up : individual files
set up : on one side
shell script : revision control
sorting : sorted
Subentry : subentries
takes longer : spend some time
the method : expected that many
the method : printed using
the method : should save
the phrase file : file of phrases
units are : standard input can
units are : standard input is
updating the index : minor changes
```

group-entry **file:**

```
:
allows finding : deroff
allows finding : findphrases needs
allows finding : ignored phrases
allows finding : repeated phrases aided
Alphabetization problems : ASCII comparisons :
Alphabetization problems : incorrectly sorted :
Alphabetization problems : Roman numerals :
alphabetized : letter - by
alphabetized : word - by
Checking indx output for : actual phrase
Checking indx output for : range
Chunks : allupper
Chunks : called chunks
create cross : see also cross
create cross : see under cross
create cross : see cross
Formatting styles : combined
Formatting styles : entry -
Formatting styles : paragraph style
Improving indx’s performance : hash function :
Improving indx’s performance : merging :
indx tools : browser program
indx tools : dedit :
indx tools : grep was
indx tools : shell script
Optional files : alternative -
Optional files : combine -
Optional files : group -
Optional files : see - also file
Optional files : see - under file
Optional files : see file
Optional files : separation character
phrase selection : difficulty laid
phrase selection : from the list
Phrases : Module 5
Phrases : phrases are implemented
program modularized : modifications proposed
Semi-automatic indexing : automating
Semi-automatic indexing : criteria
Semi-automatic indexing : semi
semiautomatic : * INDEX :
semiautomatic : < < answer :
semiautomatic : INDEX :
semiautomatic : INDEXIT :
Storage and retrieval : find a subentry
Storage and retrieval : retrievals of
The index : doubly
The index : end index
the method : novel idea
the method : takes longer
the phrase file : special character
Units : units are
```

see-also **file:**

```
:
alphabetized : sorting
```

```
Chain reference : Circular references
Circular references : Chain reference
ditroff macro calls : ditroff macro packages
doubly : search table : The index
Entry phrase : Heading
hash function : search table : Improving indx’s performance
sorting : alphabetized
```

see **file:**

```
:
alphabetic order : Alphabetization problems : Checking indx output for
Automatic indexing : Semi-automatic indexing
Blind references : Chain reference
Check entries : Circular references
findphrases : allows finding : indx tools
Searching the index : search table
```

see-under **file:**

```
:
aid for : allows finding : phrase selection
Alternative-index-term file : Optional files
Combine-phrase file : Optional files
Group-entry file : Optional files
implementation : Chunks : Units
See file : Optional files
See-also file : Optional files
See-under file : Optional files
```

alternative-index-term **file:**

```
:
< < answer : <<ANSWER>>
* INDEX : *INDEX
algorithm : Algorithm of indx
ASCII comparisons : ASCII order : Alphabetization problems
Roman numerals : numerals : Alphabetization problems
incorrectly sorted : special characters : Alphabetization problems
letter - by : letter-by-letter : alphabetized
word - by : word-by-word : alphabetized
alphabetized : Alphabetization schemes
Chain reference : Chain references
range : page references : Checking indx output for
actual phrase : see-under cross-references : Checking indx output for
called chunks : design : Chunks
allupper : implementation : Chunks
obtain the index found : Creating the paper index
see cross : see : create cross
see also cross : see also : create cross
see under cross : see under : create cross
create cross : Cross references
calculate : Determining size of the index
allow for merging : Enhancing the features of indx
error : Error handling
ignored phrases : ignored phrases file : allows finding
deroff : preparing input file : allows finding
findphrases needs : improvements suggested : allows finding
repeated phrases aided : aid in phrase selection : allows finding
allows finding : findphrases program
ditroff macro packages : Formatting final index
entry - : entry-per-line : Formatting styles
paragraph style : paragraph (run-in) : Formatting styles
hash function : Hash Function
merging : page merging routine : Improving indx’s performance
modifications proposed : modifiability of : program modularized
program modularized : indx design
novel idea : advantages of : the method
takes longer : disadvantages of : the method
the method : indx method
shell script : updating shell script : indx tools
grep was : grep : indx tools
generate it : Manual indexing task
Roman numerals : Numerals, alphabetization of
alternative - : alternative-index-term : Optional files
combine - : combine-phrase : Optional files
group - : group-entry : Optional files
see file : see : Optional files
see - also file : see-also : Optional files
see - under file : see-under : Optional files
ditroff macro calls : Output of indx
merging : Page merging routine
Improving indx’s performance : Performance, improving indx’s
the phrase file : Phrase file
Module 5 : design : Phrases
phrases are implemented : implementation : Phrases
set up : Preparing input files
chapters were run : Preparing input text
search table : Search Table
difficulty laid : difficulties in : phrase selection
from the list : for indx : phrase selection
phrase selection : Selection of phrases
semi : definition of : Semi-automatic indexing
automating : motivation for : Semi-automatic indexing
criteria : criteria for programs : Semi-automatic indexing
< < answer : <<ANSWER>> : semiautomatic
* INDEX : *INDEX : semiautomatic
semiautomatic : Semi-automatic indexing programs
sorting : Sorting of entries
Determining size of the index : Size of the index, determining
incorrectly sorted : Special characters, alphabetization of
retrievals of : of main entries : Storage and retrieval
find a subentry : of subentries : Storage and retrieval
end index : design : The index
doubly : implementation : The index
units are : design : Units
updating the index : Updating an index
```

The untouched output of indx for the index (folded to fit the line length)

```
.IX 0 reg "*INDEX" "5, 23" ""
.IX 0 reg "<<ANSWER>>" "5, 23" ""
.IX 0 reg "Algorithm of indx" "" ""
.IX 0 reg "Alphabetization problems" "" ""
.IX 1 reg "ASCII order" "12" ""
.IX 1 reg "numerals" "5" ""
.IX 1 reg "special characters" "4, 9, 10, 12, 13, 14, 15, 21" ""
.IX 0 also "Alphabetization schemes" "1, 5, 12, 17" "Sorting of
entries"
.IX 1 reg "letter-by-letter" "5, 17, 21" ""
.IX 1 reg "word-by-word" "5, 12, 13, 14, 17, 18, 21" ""
.IX 0 under "Alternative-index-term file" "" "Optional files"
.IX 0 reg "ASCII comparisons" "12" ""
.IX 0 see "Automatic indexing" "" "Semi-automatic indexing"
.IX 0 see "Blind references" "" "Chain references"
.IX 0 also "Chain references" "22" "Circular references"
.IX 0 see "Check entries" "" "Circular references"
.IX 0 reg "Checking indx output for" "" ""
.IX 1 see "alphabetic order" "" "Alphabetization problems"
.IX 1 reg "page references" "9, 12, 15, 18, 22" ""
.IX 1 reg "see-under cross-references" "13" ""
.IX 0 reg "Chunks" "" ""
.IX 1 reg "design" "" ""
.IX 1 reg "implementation" "" ""
.IX 0 also "Circular references" "" "Chain references"
.IX 0 under "Combine-phrase file" "" "Optional files"
.IX 0 reg "Creating the paper index" "2, 3, 18, 19" ""
.IX 0 reg "Cross references" "3, 6, 15, 22" ""
.IX 1 reg "see" "3, 10, 12, 13" ""
.IX 1 reg "see also" "3, 10, 11, 13" ""
.IX 1 reg "see under" "3, 4, 8, 10, 12, 13" ""
.IX 0 reg "dedit" "9, 16, 22" ""
.IX 0 reg "ditroff" "1, 2, 3, 5, 6, 7, 9, 11, 12, 13, 14, 15, 16, 17,
19, 21, 22, 23" ""
.IX 0 reg "Enhancing the features of indx" "6, 7, 21" ""
.IX 0 also "Entry phrase" "3, 4, 10, 14, 17" "Heading"
.IX 0 reg "Error handling" "2, 12, 17" ""
.IX 0 reg "findphrases program" "1, 10, 15, 16, 17, 19, 22, 23" ""
.IX 1 reg "aid in phrase selection" "15, 16" ""
.IX 1 reg "ignored phrases file" "10, 15, 16" ""
.IX 1 reg "improvements suggested" "22" ""
.IX 1 reg "preparing input file" "16" ""
.IX 0 reg "Formatting final index" "14, 15, 19" ""
.IX 0 reg "Formatting styles" "" ""
.IX 1 reg "combined" "4, 8, 10, 14, 17, 21" ""
.IX 1 reg "entry-per-line" "4, 11, 12, 14, 19" ""
.IX 1 reg "paragraph (run-in)" "4, 11, 12, 14" ""
.IX 0 reg "Group entry" "3, 8, 17, 21" ""
.IX 0 under "Group-entry file" "" "Optional files"
.IX 0 reg "Hash Function" "17, 20" ""
.IX 0 reg "Heading" "3, 4, 5, 8, 9, 14, 15, 20, 21" ""
.IX 0 reg "INDEX" "5, 15, 23" ""
.IX 0 reg "INDEXIT" "5, 6, 23" ""
.IX 0 reg "indx design" "" ""
.IX 1 reg "modifiability of" "" ""
```

```
.IX 0 reg "indx method" "1, 2, 8, 9, 14, 15, 18, 19, 21" ""
.IX 1 reg "advantages of" "6, 21, 22" ""
.IX 1 reg "disadvantages of" "22" ""
.IX 0 reg "indx tools" "" ""
.IX 1 reg "browser program" "13, 22" ""
.IX 1 reg "dedit" "9, 16, 22" ""
.IX 1 see "findphrases" "" "findphrases program"
.IX 1 reg "grep" "19" ""
.IX 1 reg "updating shell script" "19, 23" ""
.IX 0 reg "Main entry" "3, 11, 12, 13, 16, 17, 19, 21" ""
.IX 0 reg "Manual indexing task" "4, 15" ""
.IX 0 reg "Numerals, alphabetization of" "5" ""
.IX 0 reg "Optional files" "" ""
.IX 1 reg "alternative-index-term" "8, 9, 12, 14, 17, 18, 19, 21" ""
.IX 1 reg "combine-phrase" "8, 9, 11, 13, 14, 17, 18, 19, 20, 21" ""
.IX 1 reg "group-entry" "8, 11, 13, 17, 18, 19, 21" ""
.IX 1 reg "see" "8, 12, 13" ""
.IX 1 reg "see-also" "11" ""
.IX 1 reg "see-under" "13" ""
.IX 1 reg "separation character" "11" ""
.IX 0 also "Output of indx" "9, 12" "Formatting final index"
.IX 0 reg "Page merging routine" "14, 20, 21" ""
.IX 0 reg "Page reference" "3, 4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22" ""
.IX 0 reg "Performance, improving indx’s" "" ""
.IX 1 also "hash function" "17, 20" "Search Table"
.IX 1 reg "page merging routine" "14, 20, 21" ""
.IX 0 reg "Phrase file" "8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
21" ""
.IX 1 reg "special character" "10" ""
.IX 0 reg "Phrases" "" ""
.IX 1 reg "design" "" ""
.IX 1 reg "implementation" "" ""
.IX 0 reg "Preparing input files" "6, 8, 9, 14, 17" ""
.IX 0 reg "Preparing input text" "2, 8, 9, 10, 14, 16" ""
.IX 0 reg "Regular entry" "3, 12" ""
.IX 0 reg "Search Table" "17, 20" ""
.IX 0 see "Searching the index" "" "Search Table"
.IX 0 under "See file" "" "Optional files"
.IX 0 under "See-also file" "" "Optional files"
.IX 0 under "See-under file" "" "Optional files"
.IX 0 reg "Selection of phrases" "4, 8, 15, 16, 17, 22" ""
.IX 1 under "aid for" "" "findphrases program"
.IX 1 reg "difficulties in" "16" ""
.IX 1 reg "for indx" "10, 15, 16, 17" ""
.IX 0 reg "Semi-automatic indexing" "" ""
.IX 1 reg "criteria for programs" "6, 23" ""
.IX 1 reg "definition of" "1, 17, 18, 20, 21, 23" ""
.IX 1 reg "motivation for" "2" ""
.IX 0 reg "Semi-automatic indexing programs" "5, 7, 14, 21, 23" ""
.IX 1 reg "*INDEX" "5, 23" ""
.IX 1 reg "<<ANSWER>>" "5, 23" ""
.IX 1 reg "INDEX" "5, 15, 23" ""
.IX 1 reg "INDEXIT" "5, 6, 23" ""
.IX 0 reg "Size of the index, determining" "13, 20, 21" ""
.IX 0 also "Sorting of entries" "4, 5, 6, 9, 12, 13, 14, 15, 16, 17,
18, 21" "Alphabetization schemes"
.IX 0 reg "Special characters, alphabetization of" "4, 9, 10, 12, 13,
14, 15, 21" ""
.IX 0 reg "Storage and retrieval" "" ""
.IX 1 reg "of main entries" "" ""
.IX 1 reg "of subentries" "" ""
.IX 0 reg "Subentry" "3, 4, 11, 12, 13, 14, 16, 18, 21" ""
.IX 0 reg "Table of contents" "1" ""
.IX 0 reg "The index" "" ""
.IX 1 reg "design" "" ""
.IX 1 also "implementation" "" "Search Table"
.IX 0 reg "Units" "" ""
.IX 1 reg "design" "" ""
.IX 1 under "implementation" "" "Chunks"
.IX 0 reg "Updating an index" "18, 19" ""
```

The actual index given in this paper is obtained by removing useless page references and replacing sequences of consecutive page numbers by ranges of page numbers. Both must be done manually, as they involve inspection of the referenced page numbers to determine if the reference is useful and if, in fact, the sequence represents a single continuous discussion. If one does not mind arbitrarily replacing all sequences with ranges despite the semantics of the referenced text, it is straightforward to write a program to do this replacement.

An entry-per-line version of mI macro package

```
.de IX
.if \\$1>0 .in +(3n*\\$1u)u
.       \" indent a little for every sub-level
.in +3n
.       \" left margin in case line is too long and must be continued
.       \" on the next line
.ti -3n
.       \" left margin for the entry
.ie ’\\$2’see’ \\$3.\ \ \\fISee\\fP\ \\$5
.el \{\
.       ie ’\\$2’under’ \\$3.\ \ \\fISee under\\fP\ \\$5
.       el \{\
\\$3\ \ \\$4
.               \" entry phrase & page refs
.               if ’\\$2’also’ \{\
.               br
\\fISee also\\fP\ \ \\$5\}\}\}
.in -3n
.       \" move margin back to line up with starting of the entry
.if \\$1>0 .in -(3n*\\$1u)u
.       \" get back to original margin of main entries
..
```

Makefile used to format this paper:

```
# printpaper contains troff macro definitions and register settings
#       and the title and abstract
# the sec?’s contain sections of the paper
# refs contains the refer database
# header.index contains troff macro definitions and register settings
#     for printing the index
# tail.index resets the macros and registers for normal text after the
#     index
# app contains the appendix
#     app includes (via .so’s) optional files from run of indx; phrases
#     file must be copied manually into app because some of its words
#     are boldfaced
# phrases comb grp see also under alt are the required and optional
#     files for indx

F=paper
B=printpaper sec1 sec2 sec3 sec4
PAGES=
refs.ia: refs
       mkey refs | inv -n refs

$F.ref: $B refs.ia
       refer -e -n -p refs $B > $F.ref

$F.ddt: $F.ref
       /bin/cat $F.ref | tbl | dtroff $(PAGES) -mXP | dedit > $F.ddt

$F.raw.indx: $F.ddt phrases comb grp see also under alt
       indx -pphrases -ccomb -ggrp -ssee -aalso -uunder -nalt < $F.ddt \
       > $F.raw.indx

$F.raw.indx.trf: $F.raw.indx
# prepare a formattable copy of the raw index for inclusion in appendix A
#     by folding too-long lines, inserting \& in front of each macro call
#     and doubling escapes
       fmt $F.raw.indx |alg.trf >$F.raw.indx.trf

$F.indx: $F.raw.indx
       cp $F.raw.indx $F.indx
# when preparing a non-final draft, comment out the next two lines
       echo now edit the $F.indx by collecting sequences of pages
       vi $F.indx

# to print the paper
psroff: $F.indx $F.raw.indx.trf app
       /bin/cat $F.ref header.index $F.indx tail.index app | tbl \
       | psroff $(PAGES) -mXP
```