INDX, A SEMI-AUTOMATIC INDEXING PROGRAM

Kris Kiyoko Takata

June 1987
CSD-870020

UNIVERSITY OF CALIFORNIA

Los Angeles

Indx, A Semi-Automatic Indexing Program

A thesis submitted in partial satisfaction of the

requirements for the degree of Master of Science

in Computer Science

by

Kris Kiyoko Takata

1987

The thesis of Kris Kiyoko Takata is approved.

[illegible]

Joseph J. DiStefano, III

[illegible]

David G. Kay

[illegible]

Daniel M. Berry, Committee Chair

University of California, Los Angeles

1987

TABLE OF CONTENTS

page

ACKNOWLEDGEMENTS

I wish to thank the people who have helped make this thesis possible. Christine Aguilera provided the phrase finder program, allowed the inclusion of its manual page in the appendices, and helped solve some of my text-formatting problems. Jacquelyn Trang provided a productive work environment and was an invaluable source of guidance regarding the text preparation. Li-Whei Chen, Timothy Aiken, Matthew Weinstein, Michael Lyu, and Bruce Rosen helped in various ways with the program and the thesis preparation. Keith Abe and Professors David Kay and Joseph DiStefano, III contributed their time to proofread the text.

Most of all, I wish to thank Professor Dan Berry for his wonderful ideas, the helpful comments and suggestions, and for his words of encouragement. Without him, this project could not have been completed.

ABSTRACT OF THE THESIS

Indx, A Semi-Automatic Indexing Program

by

Kris Kiyoko Takata

Master of Science in Computer Science

University of California, Los Angeles, 1987

Professor Daniel M. Berry, Chair

Creating back-of-the-book indices 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 thesis created with indx is included as an example.

CHAPTER 1

INTRODUCTION

Anyone familiar with textbooks is also familiar with the use of the back-of-the book indices. For finding text dealing with a given subject, it 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 page numbers in 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 locates 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 [Harr65].

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 a task that many authors hire a specialist to do the task. Anyone who has tried to locate all books in a given library discussing a particular subject knows that many books just do not have good indices.

It is difficult because generating an index 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 at 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 generally 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 where that term does not itself 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 as well as a 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 (possibly not all) occurrences of terms, and do nothing to help determine the terms and find discussions about the terms that do not actually mention them. Also most 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; this flooding makes it hard to read the input text.

This thesis describes a suite of tools developed at UCLA; these tools offer genuine assistance in term determination, allow finding all occurrences of and some discussions about terms, and provide a cleaner interface in which the text of the book to be indexed is not flooded with indexing commands.

The rest of this chapter defines the teminology of indexing, describes the manual process, and discusses existing programs for assisting in index generation. The second chapter describes the approach the suite uses and the method by which the author exercises the suite to obtain the index. The third chapter discusses the main program of the suite, indx, and gives a modular decomposition for it. The fourth chapter describes the experience gained in using indx and the last chapter contains conclusions about the indx method and program and discusses areas of further work. This thesis is followed by an index generated with the help of these tools. Finally the appendices give UNIX®-style manual pages for the tools and show the input necessary to generate the index of the thesis. Note that the indexing suite was not applied to the appendices!

1.1 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'' [Knig70]. In this thesis, 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 thesis as a regular

UNIX is a registered trademark of AT&T.

entry.

At times, a regular entry may be further expanded under or 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 in:

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" [Knig70]. 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

1.2 Index 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 separate lines. The advantages of this style are that specific points in entries are easy to find and that the relationship of aspects to one another is more easily conveyed. This style is preferable for most texts, especially scientific and technical ones. The paragraph style has subheadings following the heading in a paragraph form. This 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 this style is the space it saves. It is, however, more difficult to scan this type of index, and Eleanor Harris in [Harr65] recommends that indices 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 the subentry. Thus overall, entries and subentries are easily identified and some space is saved when sub-subentries are printed.

1.3 The Manual Indexing Task

The indexing task is a kind of 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 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, that is 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 [Knig70] 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 [Gard82]. 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 indices, 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. For further information on the indexing process, see [Coll62, Coll69, Knig70, Harr65].

1.4 Semi-Automatic Indexing Programs

Several computer programs now exist to aid the indexer in his or her job. Since selection of the terms in the index requires human knowledge and experience, it is still left to the indexer. The programs do the rest of the work, sorting the entries and printing out the index. Some need more human intervention than others in setting up the input and further improving the output to get a nicely formatted index. All of the programs mentioned are semi-automatic because they do not automatically decide what should be in the index.

Several of the programs have been written for specific word processors or document preparation programs. Index, Documate/Plus™, and Starindex™ run with text formed using the Wordstar word processor from MicroPro, Inc. [Hard86]. With Index, a public domain program written by Tom Jennings, the words and phrases to be indexed are marked by control characters unused elsewhere in the text file. The text file contains the text and the Wordstar commands to format the text, which may include page break commands. Index uses the page break and other commands to maintain the page numbers and will count pages internally if no page commands exist in the file. If Index has been run on the text file before, the text file will contain a line beginning with "`..index`" followed by the index previously created. The creation of the index is done in a separate pass from the formatting of the text file.

Documate/Plus is a trademark of the Orthocode Corporation.

Starindex and Wordstar are trademarks of MicroPro International Corporation.

When Index is run, it removes the old index after the "..index" line, if there is such a line, and replaces it with the new index. If no index had been previously generated, the "..index" line is appended to the text file and the index is added on. Thus, the alphabetically sorted index is at the end of the text file so that when the text file is formatted, the index appears immediately after it. The "..index" command is ignored by Wordstar and is not printed. Only main entries make up the index. Case distinctions are ignored in the sort and the entries are sorted letter by letter.

In Documate/Plus, words or phrases to be indexed in the text are preceded by a "...X." Cross references are defined by using "...R" instead of "...X." These commands are ignored by Wordstar, whose commands begin with a single dot. The page numbers are calculated from the page commands on a separate pass from the formatting of the text. The indexer is able to have index commands in multiple files and to merge them to form a single index. The number of the first page of each file to be merged must be known to index the file. This may make it necessary to interrupt printing of the files to insert the page number command with the number of the first page into the next file to be printed. Documate/Plus would then be run on these files to form the index, which is written to a file separate from the text file. An entry can have four levels with this program and the last page it can refer to is page 9999.

As in Index, Starindex requires the words or phrases that are to be indexed to be marked by a control character. Major index entries are marked by a different control character, which causes this page number to be boldfaced. There are dot commands for forming a group entry. The Wordstar page length command is ignored so that when it is used, Starindex assumes 56 lines per page. Starindex is able to merge several files containing the document to be indexed and generates output files of the source text with the Starindex commands and the control characters removed, a file containing the table of contents, and a file containing the index. It is a flexible program, allowing paragraph numbering and control of the document's appearance in the output files it creates. It is not able to merge indices so the entire document must be indexed at one time.

A UNIX shell script, index, has been written by Rich Salz for indexing nroff or troff documents [Salz86]. Phrases to be indexed are given as arguments to an index macro call, `.Ix`, in the troff input text. If the phrase contains more than one word, it must be surrounded by double quotes. index takes the document text and outputs each macro call it encounters followed by a line containing the page numbers on which each entry phrase was found. This output should be saved in a file. The output will look like this:

```
.Ix "Binary search trees"
320, 341, 372
.Ix "Binary search trees, inserting a new element"
335, 336, 341
.Ix "RIGHT pointers"
208, 209, 213, 214
```

The user of the program must define the `.Ix` macro to leave blank lines between main entries, etc., and format the index with nroff or troff. Salz warns the indexer not to put any formatting commands in the arguments of the `.Ix` calls since those entries will not be sorted correctly.

A suite of awk programs to perform indexing has been developed by Jon L. Bentley and Brian W. Kernighan [Bent86]. The programs are kept separate so that the human indexer may use only those programs necessary for producing the desired index. As with index, these programs were written for troff documents, but may be altered to work with other document production systems such as $\mathrm{T_E}\chi$. The programs are set up in a pipeline fashion, with each program performing a specific task on the data running through the pipeline. The shell file make.index can be used to run all of the programs. If a subset of the programs is sufficient, they may be explicitly called on. The phrases to be indexed are arguments of `.ix` macro calls in the document text. When the document is formatted with troff, the `.ix` lines are written to troff's standard error output, `stderr`, with the format

`ix:` *the entry (up to nine words) tab-character page number*

For example, if the macro call

```
      .ix program design language
```

was on page 5 of the document, the line

```
      ix: program design language   5
```

would be written on `stderr`, which should be redirected to a file. Then make.index is run on the file of index commands, which outputs `.XX` macro calls of index terms. Then troff is used to print the final index. The index formed with this suite of programs is quite basic in its format. Only main entries can be defined and there is no cross referencing. Phrases are sorted by a given key or itself by default. Defining *see* cross references is described in [Bent86], but the human indexer must be willing to write additional awk routines. Unlike any of the other semi-automatic indexing programs, a phrase given in a macro call is also permuted and all of the permutations are added to the index. Blanks between words may be replaced by a tilde ("~") in order to control the possible permutations. When preceded by a `.tr ~` command, the tilde shows up in the index as a blank. Also unlike the other programs, the arguments of the macro calls may be more than simple phrases. Several constructs are provided to enhance the entry phrase and to allow specification of a sort key. This sort key will help the entry phrase containing formatting commands to sort correctly.

Several macros and programs have been written by users of $T_E\chi$, a document preparation system developed by Donald Knuth at Stanford. Monsanto has come out with Idx$T_E\chi$ which is written in C for the VAX/VMS system [Aurb86]. It allows three levels in the index (entry, subentry, and sub-subentry), and index entries and page references may be visually highlighted. It generates a file which may be included in the document to produce the index. A someone else [Unkn87] has written three macros

1. to put an item in the text and in the index,
2. to put an item in the text and index in italic type, which is used for items that are definitions,
3. and to put an item in the index but not in the text.

These macros output indexing macro calls for TEX, with the item and page number as their arguments. These macro calls are written to a file which can be merged with another file containing entries that have not appeared in the text, like *see* cross references. The file is sorted with the UNIX sort and run through TEX to be printed.

A third program available for TEX documents is called texindex [Corb87]. It sorts and combines entries of an index file set up by TEX and saves them in another file. TEX then must run with this file to produce the final index. Another set of macros and programs for index generation was written by Terry Winograd and Bill Paxton [Wino80]. Index terms are given as macros in the text which when run through TEX are used to create an index file. This file is then used by a set of INTERLISP programs to produce an alphabetically formatted index. TEX is then used to produce the final formatted index. More than one file may be merged to form a single index and the indexer may choose from three index styles: entry-per-line, paragraph, or combined. The indexer has control over the form of the entries, being able to specify boldfacing of pages, different types of page referencing, and any number of levels.

LaTeX, a set of TeX macros, also contains index generating macros [Lamp86]. A macro command to form an index is put in the text and other macros identify terms in the text to be indexed. When TeX is run on the document, a .idx file is created containing macros for the index. This file must be processed to create **theindex** environment, which is a list of macros defining the formats of main entries, subentries, and sub-subentries. TeX is then run on this file to get the final index, which is normally printed in two columns. A Bourne shell script, latexindex, has been written for LaTeX on UNIX systems [Hofm86]. It helps the indexer avoid manipulation of the .idx file by creating **theindex** environment. It prints the result on standard output.

Some programs have no ties to any document or word processing systems. <<ANSWER>> [Ande83], INDEX [Gard82], *INDEX [Pasa81], and INDEXIT [Fett86] are examples of them. <<ANSWER>> is an information management system in which one application creates back-of-the-book indices. Electronic cards are created using <<ANSWER>>. Fields on the cards are defined by the indexer and two cards may be shown on the screen simultaneously, which helps the indexer form cross references. The process of filling in the data cards and checking for similar information entered is analogous to the manual method of typing cards, filing them, and checking for additions and updates to them. After all the cards have been filled, another program, <<REPORT GENERATOR>>, is used to create an ASCII file of the entries. Via a word processor, the indexer may combine entries, move entries, insert cross references, etc. Then the index is printed. The program runs on an IBM PC™.

INDEX takes a file containing page numbers and the entries on each page and combines the entries, sorting them either letter-by-letter or word-by-word, depending

IBM PC is a trademark of International Business Machines Corporation.

on the indexer's preference. Other programs used with INDEX are ENTRIES, which checks entries for certain errors and changes abbreviations to long form for sorting, etc., and TEXTFORM®, which works on the output of INDEX to print or typeset the index. It also forms the table of contents.

Like INDEX, *INDEX is given a sequence of page numbers and the words or phrases selected for indexing on that page. Cross references are also entered. The program forms the entries by combining like phrases, sorting the entries, and indenting subentries. The output is a printed index having an indexer-specified line length per column, number of lines per page, and number of columns per page. The current version of *INDEX is written in SNOBOL.

INDEXIT users must enter a page number and the entry phrase on a single line. This is done for each term to be indexed and the user can see 21 of these records at a time. Only 200 entries can exist in an entry file, making it necessary for the indexer to sort and store entries in a file when the limit is reached, and then continue with record entry until there are no more entries or another limit is reached. Page references can be numbers other than page numbers, such as paragraph numbers. *See* and *see-also* cross references are defined by setting the page reference part to 0 and typing the reference. Like INDEX, INDEXIT allows the indexer to select the sort of alphabetization to be used. The indexer also specifies a list of punctuation characters to be ignored during the sorting process. The index can be four or five thousand entries long, but limited to 160,000 characters. It runs on the IBM PC and IBM XT™. The output of the index is simply a list of phrases followed by page numbers. It does not indent for subheadings and does not suppress repeated headings, so that all subentries are printed out with the form heading, subheading. The final formatting of the

IBM XT is a trademark of International Business Machines Corporation.

index is left for the indexer.

According to Linda Fetters, a freelance indexer for an indexing service, indexing programs should be judged by the following criteria: [Fett86]

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
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 in separately, the indices set up in this manner are restricted to entries occurring word-for-word in the text. When the page number and the phrases on the page that belong in the index are given as input or the phrases are preceded by index macros in the text, the phrases do not have to appear in the text. The phrases can be anything the indexer wishes them to be. Second, the ability to create cross references is desirable as noted above. 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. Third, having the capability of different ways of sorting the terms is desirable since there is no standard method. Fourth, the ability to accumulate indices 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 it has been broken into.

1.5 Motivation for the Indx Program

This paper discusses the indx program, another semi-automatic indexing program. There are several reasons behind its existence. First, it has been designed and implemented using the modern programming methods of information hiding and structured programming. A program that finds repeated phrases in arbitrary text suggests a new method of semi-automatic indexing where the phrase-finder helps the indexer select the phrases to be indexed. The feasibility of this method via this implementation is to be determined.

CHAPTER 2

THE INDX METHOD

The method of producing indices 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.

2.1 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 described in Section 2.1.4 to output the index desired. 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* cross references, that are *see* cross references, that are *see-under* cross references, and that are to be given an alternate 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 *alternate-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 as defined 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* cross references to add so the file is not given. The *see* cross reference is added to the index and since there are no *see-under* cross references defined, the *alternate-index-term* file would be processed. It is needed to change the subheading "`LEFT pointers`" under the heading "`Pointers`" to "`LEFT`" and to make a similar change for "`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 useless or technically incorrect page references. A technically incorrect page reference occurs when a sequence of page numbers should be replaced by a span of page numbers. After the troff macro calls are verified, they must be printed using the appropriate macro package to get the final index. See Appendix C for a macro package example.

2.1.2 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 the search of phrases. Second, the text of page *n* must be preceded by a line having the sequence `^Lp`*n* on the leftmost end. 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. The manual page describing this program can be found in Appendix E. 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 have only sentence punctuations preceded by a blank. For example, the partial sentence ''U.S. Navy.'' could exist in the text file as ''U.S. Navy .'' 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 *alternate-index-term* file to ''U.S. Navy.''

2.1.3 Selection of phrases in the text

The phrases to be found in the text are written in a file which must always be specified. The file of phrases contains one phrase per line. These phrases must be sorted to speed up the initial building of the index. The UNIX sort command can be used with the `-f` option to sort the index ignoring case distinctions. There must be no phrases that are equivalent to each other when case distinctions are ignored. For most phrases, case distinctions are to be ignored when searching the text. Certain phrases may need to be matched exactly, so such a phrase will be flagged with 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. It must be the first character of the file, by itself on the first line. If it should become necessary to have a word of a phrase consist solely of the special character, it should be preceded with the same character. That is, if "!" is the special character and the phrase "Yippee !!" must be searched for, it should appear in the file as "Yippee !!!". This convention is used for all of the other files as well, regarding the separation character used to separate phrases on each line.

In order to come up with a list of phrases that will appear in the index, the program findphrases [Agui87] 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 phrase* 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 generally are single words.

One could keep a list of phrases that will be ignored in the majority of texts to be indexed, a general-purpose *ignored phrases* file. It would contain phrases such as "`the`", "`a`", "`of`", and "`is`." One of the options in the findphrases program is to ignore all phrases that begin with a phrase in the *ignored phrases* file. With this option in mind, a generic *ignored phrases* file has been compiled and is given in Appendix A. The *ignored phrases* file would be completed by adding phrases specific to the text that should be ignored. A few iterations may be needed to get the right maximum length of a repeated phrase and a large enough list of ignored phrases to reduce the number of repeated phrases down to a useful size, where it indicates things to be indexed. A list of specific ignored phrases for this thesis is given in Appendix A. The *phrase* files used to create the initial index and the final index are given in Appendix B. The initial index covered an earlier version of the first three chapters of this thesis. The final index follows this thesis. The boldfaced phrases in each of the *phrase* files are those that have been directly obtained from the list of repeated phrases. About 42% of the final index entries have come from the list. Many of the other phrases in each *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.

2.1.4 The optional phrase 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 phrase is 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. The page references of the second phrase are merged with the page references of the first, putting the merged list under the first term. The second term is deleted from the index. It 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 will often be the plural or some other form of the first phrase.

The *group-entry* file also contains pairs of phrases on each line. The second term will become a subentry of the first term, if the definition is valid. Only two levels are provided meaning that a main entry can have a subentry but a subentry may not have a sub-subentry. Thus, the first phrase must be a main index term. If it does not exist, a main entry will be formed for it. There are two ways to define a subentry. Either the subentry will be a copy of one of the main entries or the main entry will be moved under another main entry, thus becoming a subentry. 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 eventually get the following portion of an index (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, the *group-entry* file would contain

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

The *see-also* file contains pairs or triples of phrases on each line. The second phrase will be the *see-also* cross reference of the first phrase. If there is a third phrase, the index term to be given a *see-also* cross reference is a subentry under the heading given by the third term. Only entries having page numbers can have such cross references. For example, to eventually get the following entry (entry-per- line format):

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

the *see-also* file could contain

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

The *see* file contains pairs or triples of phrases on each line. The second phrase will be the *see* cross reference of the first, which should not already be in the index. As the *see-also* file, if there is a third phrase, the line describes a subentry to be placed under the entry given by the third phrase. This group heading must exist in the index. The *see-under* file is set up and handled in the same manner as the *see* file. 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

A *see* file entry that would yield the macros to give the index term having the cross-reference is:

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

The *alternate-index-term* file also contains pairs or triples of phrases, which define alternate phrases for main and subentries, respectively. The index term having the first phrase is taken out of the index and re-inserted in the proper position after being given the new entry phrase (the second phrase). This is provided since certain phrases found in the text are better represented in the index by other phrases.

2.1.5 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.

The index terms are printed out as a series of troff macro calls. 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 some of the order of terms as well as touching up the page references. For one thing, indx will find every page on which a phrase is found, while the indexer may not want this form 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. The 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.

2.2 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 rather tedious.

Indx provides three types of cross referencing. None of the other programs seem to support *see-under* cross referencing. Also, there is no indication that the cross references are verified as being entries in the index. Indx will not allow an index term to cross reference an entry that does not exist as a main entry in the index. Also, only entries having page references will be 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 will not be verified so it is the indexer's responsibility to use the *see-under* cross reference properly. Indx will not allow check entries or circular references, in which two terms in the index cross reference 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) will not be 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 most of the other programs, there probably will 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 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 troff macro packages provided give the user the option to select between entry-per-line style and paragraph style. The combined style is not necessary for indx since with only two levels of entries, the combined style is equivalent to the entry-per-line style. Indx is not as flexible as the Winograd and Paxton $\mathrm{T_E\chi}$ macros by which the indexer can control many parts of an individual entry. 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 italicized in the index, he or she may have "`\fIAbsolute zero\fP`" in the input text and in the *phrase* file. When troff is run on the indx output, the `\fI` and `\fP` commands will be interpreted and the "Absolute zero" entry will be italicized. An easier way to achieve this is to find the phrase "Absolute zero" and then rename it using the *alternate-index-term* file to "`\fIAbsolute zero\fP`." Entries containing such formatting commands will definitely be erroneously placed 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 and why the ability to specify a sort key in the Bentley and Kernighan indexing suite 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.

2.3 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 *alternate-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 *alternate-index-term* file should also be quite long. It is expected that most of the definitions in the *alternate-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 amount of text to be indexed, the number of phrases in the *phrase* file, or the maximum length of the phrases in the *phrase* file 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 merging that must be done.

Depending on the phrases that eventually make up the index, there may be some sorting problems. As mentioned in section 2.2, 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 ``\fIAbsolute zero\fP'' phrase mentioned earlier. The word-by-word sort would take ``\fIAbsolute'' 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 nonalphabetic characters need closer examination. These can be searched for easily.

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 amount 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 continually on consecutive pages.

CHAPTER 3

THE INDX PROGRAM

Indx is designed as a UNIX program which may be run alone or used as in the following sequence:

ditroff *textfile* | dedit | indx -p*phrasefile* | psroff -m*macropkg*.

Thus, the information needed by indx is given in files whose names are arguments to the program. The synopsis of the command is

indx -p*phrase-file* [-s*see-file*] [- a*see-also-file*] [-u*see-under-file*] [- c*combine-phrase-file*] [-g*group-entry-file*] [- n*alternate-index-term-file*] [-d*pgdelim*]

3.1 Design of the Indx Program

Indx has been designed with the goals of understandability and modifiability. The software engineering principles of abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability [Booc83] have been applied to achieve these goals. Much of the design effort has gone towards identifying the levels of abstraction of the indx program and modularizing it in such a way as to maximize the relationships among the routines of a module and minimize the relationships among the modules. This type of modularization gives modules informational strength. The design and some of its implementation are discussed in the following sections.

3.1.2 The partitioning of the program

The partitioning of indx was achieved by taking a list of the design decisions that were likely to change and designing one module for each decision in a way that would hide the decision from the other modules. Thus each module would hide some concept, data structure, or resource, which is the purpose of an informational-strength module [Myer78]. This method was suggested by D. L. Parnas in [Parn72]. Some of the obvious design decisions concern the index, the phrase-file, each of the optional files, and the input text. The rest of the design's modules concern the argument line (of the indx command), standard output, phrases (found in all of the files, the input, the output), objects read from the files (called chunks), and error handling. With the exception of the error handling module, the modules have informational strength.

The next step was deciding what functions and procedures each module would provide to allow other modules to use the hidden object, or abstract data type correctly. Package specifications, giving the type name and the operations with their interfaces (function and procedure declarations), were written for each module. Parts of the specification that were poorly designed were changed as problems in the implementation process and further program specifications identified weak points. The most frequently used modules are those related to the data from the files (including standard input) and to the index.

3.1.3 The data modules

Many parts of the program involve reading phrases from a file. Standard input can also be considered a file. Thus, one of the modules in the design contains routines defined for phrases. On a lower level, phrases are made up of words separated by blanks. Words are read from the files, but since the files may also contain other things like the end-of-file marker, the entities that are read from the files are called chunks. A chunk may be a word, an end-of-line character (`eol`) or an end-of-file character (`eof`). A word is considered anything other than `eol` or `eof`, so that single punctuations are also words. The input text, which is read from standard input, contains words just like the files mentioned above. There also is an end-of-file marker, but there are no end-of-line markers as they get masked over at a lower level. This allows the input text, which is divided into lines, to be free from the concept of lines. So, standard input is thought of as consisting of units, where a unit is a chunk that is not `eol`.

Following are the package specification parts for `chunk file`, which defines the usage of chunks while hiding the implementation of chunks, and `phrases`, which does the same thing for phrases. They are written as Ada® packages. (However, liberties are taken with Ada notation. For example, blanks are inserted in words for readability instead of using underscores.)

Module 3: `chunk file`

`open(s,f)` opens the file having the name given by `s` and returns `f` as the file descriptor.
`get next chunk(c,f)` gets the next chunk out of file `f` and puts it into `c`.
`c1 to c2(c1,c2,i)` returns one of three characters: '<', '>', or '=', reflecting the relationship of chunk `c1` to chunk `c2`. If i is true, case is ignored in the comparison.
`is eof(c)` returns true if chunk `c` is the end-of-file character and false otherwise.
`is eol(c)` returns true if chunk `c` is the end-of-line character and false otherwise.
`is potential page marker(c)` returns true if chunk `c` may be a page marker and false otherwise.
`length(c)` returns the length of chunk `c`.
`char(i,c)` returns the `i`th character of chunk `c`.
`string of(c,s)` puts the string representation of chunk `c` into string `s`.
`set page number of(c,pg)` sets the page number of chunk `c` to `pg`.
`page number(c)` returns the page number of chunk `c`.
`close(f)` closes the file given by the file descriptor `f`.

```
        with text io;
```

Ada is a registered trademark of the U. S. Department of Defense (AJPO).

```
use text io;
package chunk file is
   type chunk is private;
   type in file is private;
   procedure open(s: string; f: out in file);
   procedure get next chunk(c: out chunk;
           f: in file);
   function c1 to c2(c1, c2: chunk; i: boolean)
           return character;
   function is eof(c: chunk) return boolean;
   function is eol(c: chunk) return boolean;
   function is potential page marker(c: chunk)
             return boolean;
   function length(c: chunk) return integer;
   function char(i: integer; c: chunk)
           return character;
   procedure string of(c: chunk; s: out string);
   procedure set page number of(c: chunk;
              pg: integer);
   function page number(c: chunk) return integer;
   procedure close(f: in file);
end chunk file;
```

Module 5: phrases

clear phrase(p) initializes phrase p.
print phrase(p) prints out the chunks that make up phrase p.
add chunk to phrase(c,p) adds chunk c to the end of phrase p.
add unit to phrase(u,p) adds unit u to the end of phrase p.
freeze phrase(p) stops the changing of phrase p.
p1 to p2(p1,p2,i) returns a character reflecting the relationship of phrase p1 to p2, either '<', '=', or '>'. If i is true, case is ignored in the comparison.
chunk of(n,p,c) puts the nth chunk of phrase p into c.
length in chunks(p) returns the number of chunks in phrase p.
is empty(p) returns true if p contains no chunks and false otherwise.

```
with output file, chunk file, text file;
use output file, chunk file, text file;
package phrases is
   type phrase is private;
   procedure clear phrase(p: phrase);
   procedure print phrase(p: phrase);
   procedure add chunk to phrase(c: chunk;
           p: in out phrase);
   procedure add unit to phrase(u: unit;
           p: in out phrase);
   procedure freeze phrase(p: phrase);
   function p1 to p2(p1,p2: phrase; i: boolean)
           return character;
   procedure chunk of(n: integer; p: phrase;
           c: chunk);
   function length in chunks(p: phrase)
           return integer;
   function is empty(p: phrase) return boolean;
end phrases;
```

The chunk file routines allow operations on chunks such as comparing two chunks. The function p1 to p2 in phrases, which compares two phrases, must use the function c1 to c2 to compare the phrases chunk by chunk. Phrases consist of chunks, but the information of the makeup of chunks is kept from the routines in phrases.

Since units are nearly the same as chunks, many routines in the module for the input text are very much like those in chunk file. The module is called text file.

Module 4: text file

open opens standard input.
get next unit of text(u) puts the next unit of text into u.
make unit(c,pg) makes a unit out of a chunk by adding the page number pg of the unit to the chunk.
is page marker(u) returns true if unit u contains the page number for the following text and false otherwise. If true, the current page number is extracted from the unit.
page no(u) returns the page number that unit u is found on.
is eof(u) returns true if unit u is the end-of-file character and false otherwise.
length(u) returns the length of unit u.
char(i,u) returns the ith character of unit u.
string of(u,s) returns the string part of unit u in s.
close closes standard output.

```
with text io, chunk file;
use text io, chunk file;
package text file is
   type unit is private;
   procedure open;
   procedure get next unit of text(u: out unit);
   procedure make unit(c: in out unit; pg: integer);
   function is page marker(u: unit) return boolean;
   function page no(u: unit) return integer;
   function is eof(u: unit) return boolean;
   function length(u: unit) return boolean;
   function char(i: integer; u: unit)
            return character;
   procedure string of(u: unit; s: out string);
   procedure close;
end text file;
```

The differences between chunks and units are that units cannot be `eol` and units, being from standard input, are tied to page numbers while chunks are not. Both chunks and units represent words, which make up phrases. To avoid the difficulty of comparing two types of phrases, units and chunks are designed to be the same thing. Thus, chunks have page numbers associated with them. This does not make too much sense if the chunks are from files but the page numbers are necessary if chunks are from standard input. This is why the procedure `set page number of` and the function `page number` exist in `chunk file`.

3.1.4 The index module

The largest module of the program defines the index. It contains routines to do the following: initialize the index, set various parts of an index term, add index terms to the index as either entries or subentries, determine the contents of an index term, and print the index. There are routines for each of the operations the optional files define, including combining index terms and changing the entry phrases of index terms. The dump routines are provided for and were used to debug the code.

Module 6: `index`

`initialize index(pgdelim,delimlength)` does the necessary initializations for the index package. It is given the page delimiter which will separate pages in the page reference part and the number of characters making up the delimiter.
`add phrases beginning with this unit to index(u,m)` searches for phrases beginning with unit `u` and adds phrases that consist only of the unit to the index and saves phrases that begin with `u` but must be matched with more units to be read in from the text. `m` contains the maximum number of units that a phrase can have.
`clear index term(i)` initializes index term `i`.
`add phrase to index term(p,c,i)` puts phrase `p` in the entry part of index term `i`. If `c` is true, the case of the phrase is to be ignored.
`add page to index term(n,i)` adds page `n` to the page list of index term `i`.
`add see also to index term(p,i)` adds a see-also reference to phrase `p` to index term `i`.
`add see cross reference(p,i)` adds see cross-reference `p` to index term `i`.
`add see under cross reference(p,i)` adds see-under cross reference `p` to index term `i`.

`add sub entry to index term(i1,i2)` places index term `i2` under index term `i1`.
`freeze index term(i)` freezes index term `i` from further changes.
`add index term to index(i)` adds index term `i` to the index. `i` is a main index term, not a subentry.

`phrase of(i,p)` puts the entry part of index term `i` in phrase `p`.
`is case to be ignored for(i)` returns true if index term `i` has an entry phrase for which case distinctions are not important when searching for occurrences in the text. It returns false if exact matches of the entry phrase are to be sought.
`page list of(i,l)` puts the page list part of index term `i` in `l`.
`reference of(i,p)` puts the index term being referred to by index term `i` into `p`, valid for see, see-also, and see-under cross-references.
`is regular entry(i)` returns true if `i` is a regular index term and false otherwise. A regular index term has an entry phrase and a list of page numbers. It may be a main index term, the main term of a group term, or a subentry.
`is see cross reference(i)` returns true if `i` is a see cross-reference and false otherwise.
`has see also cross reference(i)` returns true if `i` has a see-also cross-reference and false otherwise.
`is see under cross reference(i)` returns true if `i` is a see-under cross-reference and false otherwise.
`is main entry of group entry(i)` returns true if `i` is the main entry of a group entry and false otherwise.
`is sub entry of group entry(i)` returns true if `i` is a subentry under a group entry and false otherwise.

`prepare index for further definitions` does the preparations necessary for processing the optional files (see, see-also, etc.).
`get main index term(p,i)` returns true if a term in the index has phrase `p` in the entry part and false otherwise. If an index term having that phrase has been found, it will be in `i`. Otherwise, `i` will be undefined.
`get sub entry(p,g,i)` returns true if a sub-entry, having phrase `p`, of the group entry whose phrase is `g` exists in the index and false otherwise. If a sub-entry has been found, it will be in `i`. Otherwise, `i` will be undefined.
`combine index terms(i1,i2)` merges the page list of the index term `i2` with the page list of the index term `i1`. It is meant to be used to combine main index terms only, not sub-entries.
`delete main index term(i)` deletes main index term `i` from the index.
`delete sub entry from index term(g,s)` deletes subentry `s` from main index term `g`.
`copy entry under group entry(i1,i2)` forms a group relationship between index terms `i1` and `i2` in which a copy of `i2` is placed under `i1`. Thus the entry `i2` is found both under a group and as a main entry in the index.
`move entry under group entry(i1,i2)` forms a group relationship between index terms `i1` and `i2` in which `i2` is placed under `i1`, removing it from the main entries of the index.
`change main index term entry(i,p)` replaces the entry phrase of the main index term `i` with `p`.
`change sub entry of index term(i,g,p)` replaces the entry phrase of the sub-entry `i` under the group entry `g` with `p`.
`freeze index` freezes the index from further changes.

`print index(alt)` prints the macro calls to form the index. if `alt` is true, the alternate-index-term file was given and alternate entries for cross-references are searched for when an index term having a cross-reference is printed.
`dump index term(i)` prints out the contents of index term `i`. It is provided for debugging purposes.
`dump all index terms` prints out the contents of all index terms in the index. It also is provided for debugging purposes.

```
with output file, chunk file, phrases, text file,
  alternate index term file;
use output file, chunk file, phrases, text file,
  alternate index term file;
package index is
   type page list is private;
   type index term is private;
   procedure initialize index(pgdelim: string;
                delimlength: integer);
   procedure
     add phrases beginning with this unit to index
           (u: unit; m: integer);
   procedure clear index term(i: index term);
   procedure add phrase to index term(p: phrase;
    c: boolean; i: in out index term);
   procedure add page to index term(n: integer;
               i: in out index term);
   procedure add see also to index term(p: phrase;
          i: in out index term);
   procedure add see cross reference(p: phrase;
          i: in out index term);
   procedure add see under cross reference
      (p: phrase; i: in out index term);
   procedure add sub entry to index term
        (i1,i2: index term);
   procedure freeze index term(i: index term);
   procedure add index term to index(i: index term);

   procedure phrase of(i: index term;
              p: out phrase);
   function is case to be ignored for(i: index term)
              return boolean;
   procedure page list of(i: index term;
          l: out page list);
   procedure reference of(i: index term;
              p: out phrase);
   function is regular entry(i: index term)
              return boolean;
   function is see cross reference(i: index term)
              return boolean;
   function has see also cross reference
           (i: index term) return boolean;
   function is see under cross reference
         (i: index term) return boolean;
   function is main entry of group entry
         (i: index term) return boolean;
   function is sub entry of group entry
         (i: index term) return boolean;

   procedure prepare index for further definitions;
   function get main index term(p: phrase;
      i: out index term) return boolean;
```

```
        function get sub entry(p: phrase;
             g,i: index term) return boolean;
        procedure combine index terms
                  (i1,i2: index term);
        procedure delete main index term(i: index term);
        procedure delete sub entry from index term
                (g,s: index term);
        procedure copy entry under group entry
              (i1,12: index term);
        procedure move entry under group entry
              (i1,i2: index term);
        procedure change main index term entry
            (i: index term; p: phrase);
        procedure change sub entry of index term
                  (i,g: index term; p: phrase);
        procedure freeze index;

        procedure print index(alt: boolean);
        procedure dump index term(i: index term);
                -- for debugging
        procedure dump all index terms;-- for debugging
    end index;
```

3.1.5 The algorithm

A rough outline of the algorithm is the following.

```
assign default optional file flags, page delimiter
get the arguments (* phrase-file name, etc *)
InitializeIndex(pgdelim,pgdelimlength)
if ReadInPhrases(phrasefilename,MaxUnitsInPhrase) then begin
   UnitOpen      (* open standard input *)
   OutOpen       (* open standard output*)

   GetNextUnitOfText(theunit)
   if not UnitIsEOF(theunit) then
      if not IsPageMarker(theunit) then begin
         PrintErrorMsg(NoPageMark,nolinenum)
         (* close standard input & output and *)
         (* end program *)
      end
      else begin
         GetNextUnitOfText(theunit)
         while not UnitIsEOF(theunit) do begin
            if not IsPageMarker(theunit) then
               AddPhrasesBeginningWithThisUnitToIndex
                  (theunit,MaxUnitsInPhrase)
            GetNextUnitOfText(theunit)
         end
      end

   PrepareIndexForFurtherDefinitions
   if gotcombine then
      FormCombinedEntriesWhileValidatingTerms
         (combinefilename)

   if gotgroup then
```

```
         FormGroupEntriesWhileValidatingTerms
            (groupfilename)

      if gotseealso then
         FormSeeAlsoCrossRefsWhileValidatingTerms
            (seealsofilename)

      if gotsee then
         FormSeeCrossRefsWhileValidatingTerms
            (seefilename)

      if gotseeunder then
         FormSeeUnderCrossRefsWhileValidatingTerms
            (seeunderfilename)

      if gotalternate then
         FormAlternateEntriesWhileValidatingTerms
            (alternatefilename)

      PrintIndex(gotalternate)

      UnitClose      (* close standard input *)
      OutClose       (* close standard output *)
   end
   else PrintErrorMsg(NoCaseChar,errlinenum)
```

3.1.5.1 Finding page references

Page references for the phrases in the *phrase* file are found during the `AddPhrasesBeginningWithThisUnit` procedure. When the phrases are read in from the file, they are placed in the index. Then as the units of the text are read in, the index is searched for phrases beginning with the unit. All phrases that may be in the index are kept in a table of phrases. As each unit is read, it is appended to the phrases in the table and these phrases are then searched for in the index. If a phrase is found in the index, the page number of the first unit of the phrase is added to the list of page references of the index term. The page number of the first unit refers to the page that the phrase begins on. Whether or not an index term is found, the phrase remains in the table until it has reached the maximum length of the phrases in the phrase-file. This allows the finding of embedded phrases. The index is then checked for entries that begin with this unit read. If there are such entries, the phrase consisting of this unit is searched for in the index. If found, its page reference is added to the index term's page list. The phrase is added to the table of phrases since it may be part of more index phrases.

3.1.5.2 Processing the optional files

After all page references have been found, the indexer's indication of whether phrases should be searched for with or without case distinctions is no longer needed. In order to process the optional files, phrases in those files must match those in the index exactly. Thus, it is necessary to set all of the index terms to not have case distinctions ignored. This is done in `PrepareIndexForFurtherDefinitions`.

The contents of the optional files have been mentioned in section 2.1.4 and are described further on the manual page in Appendix E. The first lines of the files define a separation character of the files which is mainly used to separate phrases. Because the indexer is free to choose this character, the situation of having a word of a phrase consist only of the file's separation character is rare. If it does occur though, the indexer should immediately precede the word by the separation character.

The files chosen are based on the most common types of index terms (group entries to two levels; *see*, *see-also*, and *see-under* cross references) and the fact that many index terms are forms of the phrases used in the text (a combination of different phrases; an alternative way of stating a phrase). The processing order chosen has been partially determined by the decision to search the index when validating the lines of each file instead of searching the given files for erroneous combinations of phrases. An example of an erroneous combination is to have a phrase in the *phrase* file as a first term in the *see* file. This is illegal since index terms having *see* cross references do not have page references. The order is chosen to get by with as little error checking as possible.

After the text has been read, the index consists of main entries having page references. Since the ability to combine phrases is provided to get all of the page references to a concept under one phrase and to simulate the occurrence of a phrase on a page where the concept is discussed but the actual phrase does not occur, it makes sense to combine terms that are known to have only page references. Thus combining phrases is done first.

After phrases are combined, the index still consists of main entries having page references. Group entries are then formed. Some of the entries will be removed from the index and placed under another entry, thus the resulting set of main entries will usually be the same as before (if all subentries were formed by duplicating main entries) or smaller. It is possible for the amount of main entries to increase if new main entries are defined during this stage. It is better to form group entries before the cross references since cross referenced terms are main entries and all main index terms after the *group-entry* file is processed will remain main entries.

Once the groups are set up, the cross references can be added. Among the three types, terms containing *see-also* cross references must have page numbers. The other two must not. So while all terms in the index are known to have page references, the *see-also* file is processed. There is one exception to the rule of terms having page numbers. In the *group-entry* file, if a main index term is not found in the index, one is created. This is useful for setting up a heading which categorizes the subentries without having to be searched for in the text. It would be acceptable in this case for a reader to refer to the subentries and be referred to another entry for more information.

The *see* and then the *see-under* cross references are added. The *see* file is processed first to make the check for chain references easier. It is impossible to define two entries that refer to each other since the cross referenced index term must exist in the index before other terms can reference it. It is also impossible to define terms that circularly reference themselves, as

Indexing, *See* Storage and retrieval

Retrieval of information, *See* Indexing

Storage and retrieval, *See* Retrieval of information

for example, since the term to be added must not already exist in the index and the cross referenced term must exist. Without checking whether the cross referenced term itself has a *see* cross reference, a chain reference would be possible, for example:

Document preparation systems, 10-12, 14, 16-25

Indexing programs, *See* Semi-automatic indexing

Semi-automatic indexing, *See* Document preparation systems

It would be more direct to have "Indexing programs" cross reference "Document preparation systems." Chain references are impossible to define as the check mentioned above is made. The see under index terms are created if the term to be added does not already exist in the index and the cross referenced term is the main entry of a group entry. This is the only criteria and it is left to the indexer to make sure that the term having the see under cross reference is used in a subentry of the group.

The last step in creating the index is to change the entry phrases as defined in the *alternate-index-term* file. Terms (main or sub) are removed from the index, given the new phrase, and reinserted in the index. Main entries changed are saved in a table so that as the index is printed, those terms having cross references to entries that have been changed can be altered to cross reference the correct term.

For all of the optional files, lines defining some invalid operation cause an error message to be printed. The message contains a description of the error, the type of file, and the line number of the file. The building of the index continues until all files have been processed.

3.1.5.3 Printing the index

The index is then printed as a list of nroff/troff macro calls. A macro call has the following form:

`.IX` *l type "entry-phrase" "pg refs" "cross ref"*

where *l* is `0` for main entries and `1` for subentries, and *type* is either `reg`, `also`, `see`, or `under`. A `regular` type macro will have a null *cross reference* string and `see` and `under` type macros will have a null *pg refs* string.

3.1.6 Error handling

Error checking is done in the main program and in the modules of the input files. Two types of errors cause no index to be printed. If there is no case distinction character in the *phrase* file, no phrases are read in. If the first unit of the input text is not a page marker, the rest of the input text is not read. All definitions in the optional files are processed if they describe legal operations, as mentioned before. Lines having errors are flagged. The index will be printed even if these errors occur, but it will be incorrect. It may help the indexer determine the error with the optional files.

There is one error message routine. It contains all of the error messages of the program, which are accessed by a message number. If the line number of the file is given, it will also be printed.

3.2 Implementation of the Indx Program

Indx is written with Berkeley Pascal version 2.0. The features of this version of Pascal helped the implementation be as close to the design (written as Ada packages) as possible.

3.2.1 Faking packages in Pascal

In order to implement the set of packages given in the design, the bodies of the routines of a package were placed in one file. Berkeley Pascal has a feature that allows procedure and function declarations to exist in separate files from their bodies. The declarations are placed in a file having the suffix ".h" and each of the routines are declared `external`. The file containing the bodies and any other file that calls these routines must have a line near the top saying "`#include` *filename*.h." For example, the routines of the `chunk file` module are declared in `chunkfile.h` and defined in `chunkfile.p`. Parts of these files are given below.

************ chunkfile.h ******************

```
(*  package specification for CHUNK FILE  *)

procedure ChnkOpen(filename:string; var f:infile);
   external;
   (* opens the file having the name given by      *)
   (* filename and returns f as the file descriptor*)

procedure GetNextChunk(var c:chunk; var f:infile);
   external;
   (* gets the next chunk out of file f and puts it*)
   (* into c.  a chunk is a word, end of file, or  *)
   (* end of line             *)

function isEOF(c:chunk): boolean;  external;
   (* returns true if chunk c is end of file *)
   (* character and false otherwise        *)
```

************ chunkfile.p ******************

```
(* package CHUNK FILE *)

#include "globals.h"
#include "stringtype.h"
```

```
#include "chunktype.h" (* chunk type declaration *)
#include "filetype.h"

#include "chunkfile.h"

var charsaved: boolean; (* global vars of package *)
    savech: char;

procedure ChnkOpen;
  (* opens the file having the name given by       *)
  (* filename and returns f as the file descriptor*)
  (* it also initializes the global variables of  *)
  (* this package             *)

  begin
    if filename <> 'standardinput' then
      reset(f,filename);
    savech:= ' ';
  end;

function isEOF;
  (* returns true if chunk c is end of file *)
  (* character and false otherwise      *)
  begin
    isEOF:= c.body[1] = chr(1)
  end;
```

****

Any of the other modules which use the chunk file routines must contain "#include chunkfile.h" in order to access the routines.

Note that chunkfile.p could contain fully declared functions and procedures. That is, they would not be externally declared in chunkfile.h. These routines would be internal, or known only by the chunk file routines. None of the other files may use them since they are not declared in the .h file. Routines of this sort are plentiful in the index module, and they are not in the .h file because they are implementation dependent. The routines in .h files (in the decomposition) must be implementation independent for a well-designed module.

Because Pascal does not support the concept of packages and cannot restrict abstract data types to specific packages, the restriction of accessing the data types only through their routines provided must be voluntary. For example, the phrases module must use the chunk file module so that phrases.p must include `chunkfile.h` and `chunktype.h`, which contains the type declaration for chunks (and units). The implementation of a chunk then, is available to the routines in `phrases`, so that these routines may directly refer to parts of a chunk instead of using the `chunk file` routines. However, it defeats the purpose of the design to do this.

3.2.2 Implementation of the index

The index is implemented as a doubly-linked list of index terms in order to have the ability of deleting an index term without having to sequentially traverse the index. Deletions are done by combining index terms, by changing the entry phrase, and possibly by forming group entries. The terms are in sorted order so that the print index procedure begins at the head of the list and sequentially traverses it to print the index. The `index term` type is a pointer to a node as depicted in Figure 3.1.

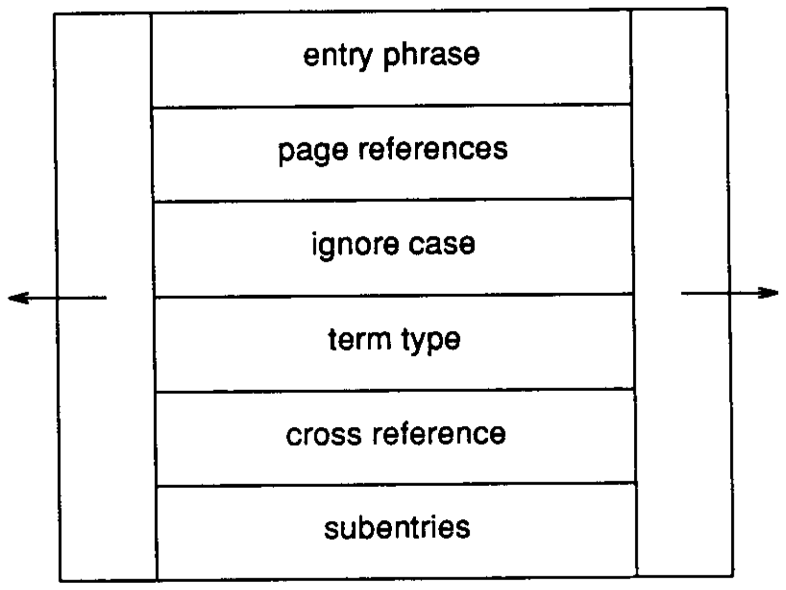

Fig. 3.1 An index term node

The `entry phrase` and `cross reference` parts are `phrases` and the item labeled `subentries` is an `index term`. The `page references` is a linked list of `integers`. The `ignore case` field indicates whether case is to be ignored when searching for the index term. The `term type` field indicates the type of index term; a regular entry, a main entry of a group entry, or a subentry. It also describes the type of cross reference it contains, if there is one. The `next` and `previous` fields, which are not labeled but are pictured on the right and left sides, respectively, point to the index terms that succeed and precede the index term.

Figure 3.2 is a representation of an index. The particular index shown is of the first three index terms in which the second index term has two subentries. The darkly shaded areas indicate nil pointers. The diagonally shaded rectangle represents the `subentries` part of the index term, which being an `index term` itself has its `next` field pointing to its first subentry. The `previous` field, not pictured, is nil.

In order to find an index term quickly, the phrases are hashed to give a location in the `search table`. The `search table` is an array of buckets that contain a binary tree of index terms which get hashed to that location. Because the subentries of an index term are linked directly to their main entries and the number of subentries is usually small, the `search table` contains only main entries. Thus to find a subentry, the main entry must be found and then its subentry list sequentially searched for the desired subentry.

The hash function used depends only on the first word of the phrase. The integer represented by the binary representations of the second through the fifth characters is divided modulo 1793 (the number of search buckets) and stored in the `search table` location one greater than this result. Should the word contain less than five letters, blanks are used in place of the nonexistent letters. Index terms get

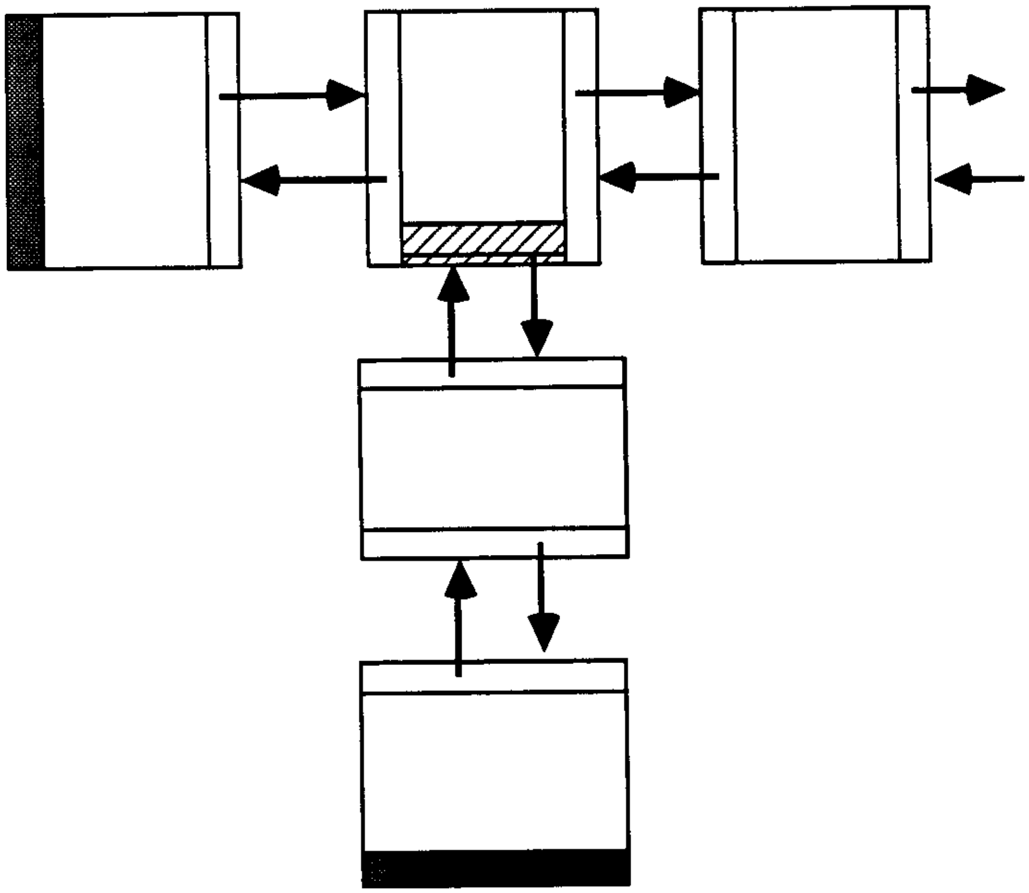

Fig. 3.2 Part of an index in which the second index term has two subentries.

distributed among buckets more evenly than if they were alphabetically partitioned (as they will appear in the final index) but this hash function may not be very good if many phrases begin with the same word. The function is simple, though, and also allows the program to determine easily whether a phrase in the index begins with a certain word.

The decision of using binary search trees in the hash buckets as opposed to using a linked list structure was influenced by the need to rapidly search the index. The bulk of searching the index comes not from the processing of the optional files, but rather from searching for phrases from the text that are in the index. During the processing of the optional *combine-phrase*, *group-entry*, and *alternate-index-term* files, deletion of main entries in the index means deletion of a node from a binary search tree (in the `search table`) as well as deletion from a doubly-linked list (the `index`). Binary tree deletion is a more complicated process than linked list deletion is, however, it was anticipated that the number of searches would be far greater than the number of deletions.

Insertions and retrievals of main entries in a binary search tree are done by comparing the headings of the main entries to be inserted or retrieved to the heading of a node in the tree, ignoring case distinctions. The main entries cannot be stored and retrieved in tree locations determined by case distinct comparisons since the result of having no match between the heading being inserted or sought and a heading of an entry in the tree could mean one of two things. It could mean that the phrases are not the same even if case distinctions were to be ignored or it could mean that the phrases are equivalent if case distinctions are ignored. Thus the process of searching for a heading involves two steps. First, a search is made for the case-ignored equivalent of the given phrase. Then if such a heading is found, the result of the search depends on whether the heading of the main entry is to be matched exactly or with case distinctions ignored. If case distinctions are to be ignored, then the main entry has been found. If case distinctions are to be made, the given phrase must be compared again with the heading of the main entry, this time without ignoring case distinctions, to determine whether or not the given phrase has indeed been found.

Because main entries are stored in binary search trees disregarding the case distinctions of their headings, no phrases in the *phrase* file can be equivalent when case distinctions are ignored. Even if some scheme is devised to store and retrieve main entries having such equivalent headings, correct retrieval of a main entry having such a heading cannot be guaranteed, since main entries can be added via the *see* and *see-under* files, which may mess up the scheme.

3.2.3 The implementation of phrases, chunks, and units

Phrases are implemented as linked lists of chunks to avoid having a limitation on the number of words a phrase can contain. Chunks and units are the same thing, as mentioned earlier in section 3.1.3. They are records consisting of four fields: `body`, `allupper`, `length`, and `page`. The body contains the characters of the chunk, stored as a `string` (a packed array of char). `Allupper` also contains the characters of the chunk with all lower case characters converted to upper case. `Length` is the number of characters of a chunk and `page` the page number on which the chunk is found. As stated before, it makes sense to refer to the page number only for chunks (units) read in from standard input. For chunks read in from files, `page` is set to zero. The `allupper` field is added to support the comparison of chunks, ignoring case distinctions. Since both kinds of comparisons could be made for a phrase (ignore case when sorting, do not ignore case when seeking an index term in the index), it is easiest to store both the original phrase and an all upper case version.

CHAPTER 4

THE INDXING EXPERIENCE

The first application of indx was to form an index of the first three chapters of this thesis. This task was split into four parts. The first and most difficult entailed selecting the entries for the index, which will be referred to as entry selection, and selecting the actual phrases from the text that represented them, which will be 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 the troff `mI` macro package.

The index was then updated after the completion of this thesis. It was checked for incorrect page references and printed to give the index at the end of this thesis.

4.1 The Selection of Entries and Phrases

Being a novice indexer, 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. To further aid phrase selection, findphrases could be altered to allow specification of significant phrases which must also be sought. These significant phrases could be nonrepeated phrases. They could also begin with a phrase in the *ignored phrases* file. An option to print all phrases containing a significant phrase would also be helpful. To minimize the amount of output, only such phrases of maximum length could be printed. 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. This difficulty was due to lack of indexing experience.

In order to use findphrases, a file of ignored phrases was needed. As shown in Appendix A, 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 first three chapters, 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 run on parts of Chapters 1 through 3 with a maximum repeated phrase length of five. These chapters were run through the necessary troff preprocessors and piped through dedit into a file. The file had to be partitioned since it was too long to be processed at once. Also to reduce execution time, the –u option, which would have suppressed phrases that are wholely and everywhere contained in another phrase as explained further on the manual page in Appendix E, was not used. It turned out that it was better not to use this option. This will be explained later in this section. 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 will occur less frequently than their respective main entries.

An alternative of using troff and dedit for preparing the findphrases input file is to use deroff. Deroff will output the source text with the troff 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 troff, 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.

Although the list of repeated phrases aided the entry selection and phrase selection processes, the selections were basically done a chapter at a time since the entire text was not scanned at once. It took a 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. Due 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.

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 maximum length of a phrase in the *phrase* file, the longer the process of searching the text will take. As mentioned earlier, it was advantageous that the –u option was not used with findphrases. Although the repeated phrase list 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. Second, 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`. This is a consideration only because of the particular hash function chosen.

4.2 Setting Up the Indx Files

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

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 on the left side were placed in the *phrase* file, which was then sorted using the UNIX `sort`. The optional files were created using the phrases on the left side. The only files in which an entry from the right side was used were the *group-entry* file, when group headings were not searched for in the text, and the *alternate-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 '
```

alternate-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
```

The files used for the first index are given in Appendix B. There are 96 phrases in the *phrase* file and the following table gives the number of definitions in each of the optional files.

File	# of definitions
combine-phrase	29
group-entry	43
see-also	4
see	2
see-under	7
alternate-index-term	48

As expected, the *alternate-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 *alternate-index-term* file were for renaming subentries, also as anticipated. Out of the 48 definitions, 37 of them were for subentries.

The individual files used as input to findphrases were concatenated and given as standard input to indx and the output collected in another file.

4.3 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 amount 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 chapter.

4.4 Updating the Index

The index described so far has not been included in the appendices since it was of the first three chapters of an earlier version of this thesis. The general method described here was used to update the first index to obtain the index found at the end of this thesis. For some yet unknown reason, this chapter was never successfully processed by findphrases. Thus phrase selection for the entries of this chapter were determined without a list of repeated phrases. The last chapter was run through findphrases with the same options as before. A slight deviation from the method described above was also tried. As each chapter was scanned, desired and tentative entries were written down, again along one side of the sheet. Phrases from the text were written opposite these entries as a first guess. Then the list of repeated phrases was scanned to finalize the phrases selected for each of the entries. Lines containing useless entries for a given phrase were identified on the list to help the page reference verification process done later. Phrases already selected to build the first index were searched for on each of the new lists and if they were found, those occurrences were checked for useless page references as well. The opposite was also done. Phrases selected to index this part of the thesis were searched for on the other lists. If any were found, the occurrences were checked to see whether or not they were useless. 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.

After selecting the phrases for the fourth and fifth chapters, indx was rerun on the entire text. Several results of the method were discovered while checking the page references. Updating the text often caused indexed discussions originally on one page to be split among 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 book has been suggested by Daniel M. Berry [Berr87]. The Revision Control System [Tich82] 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. A shell script that may be used to accomplish this revision process is given in Appendix D. This shell script was not used since the entire thesis was too long to be run through findphrases at one time. The method of using diff to compare old and new phrase lists for a given part of the thesis 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.

4.5 Printing the Final Index

The output file was then run through troff with the mI macro package defined for the entry-per-line format as provided in Appendix C. The first index contained 87 macro calls altogether, which equaled the number calculated as described in Section 2.2. That is, 96 (*phrase*) + 2 (*see*) + 7 (*see-under*) + 0 (*group-entry*,duplicating terms) - 29 (*combine-phrase*) + 11 (*group-entry*, group headings added) = 87. The current index contains 115 subentries and entries.

The *phrase* file and the optional files used to generate the index are given in Appendix B. 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 and the macro calls used to generate the current index.

CHAPTER 5

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 may be modified to improve its performance and to provide additional features.

5.1 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 `search table`. The page merging routine implemented 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 probably is 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 will be 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 chunks 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 would provide 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 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.

Despite these advantages, the hash function does not distribute the index terms evenly among the buckets of the `search table` when many groups of index terms begin with the same five characters, or more specifically, 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 indices 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.

5.2 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 to form the first draft of the index for Chapters 1 through 3. Additional features may include: allowing any number of cross references, having more levels of entries available, being able to choose between the word-by-word or the letter-by-letter sorting schemes, having a separate sort key if desired, being able to merge indices, and allowing the second term in a *combine-phrase* file definition to remain in the index.

5.2.1 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. The changes necessary would be confined to the `index` module, the cross reference modules, the `error` module, and the macro used to print the final index. In the `index` module, the `cross reference` field of an `index term` must be changed to accommodate more than one phrase. The procedure bodies of `add see also to index term`, `add see cross reference`, `add see under cross reference`, `clear index term`, and `print index` must also be changed. In the cross reference modules, having a second cross reference for an index term will no longer be viewed as an error. The `.IX` macro definition must be altered appropriately. Note that the changes needed lie entirely within the bodies of the procedures, so that the procedure and function interfaces of all of the affected modules remains unchanged.

5.2.2 Increasing the index levels

The vast majority of indices 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 indices. The current `index term` data type is capable of handling sub-subentries since the `subentries` field of the `index term` is an `index term`. However, many changes must be made to the design. More routines may be added to the `index` module, such as `add sub subentry to subentry` and `change sub subentry`, or existing routines redesigned to accommodate sub-subentries. The format of the *group-entry* file must be changed as well as the formats of the *cross reference* files and the *alternate-index-term* file. The amount of error checking needed in each step of the index-building process will increase. The routines to print the index and the `.IX` macro definition must also be changed to allow for a third level of entries.

5.2.3 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. A letter-by-letter sort is quite involved given the current implementation of `phrases`. Perhaps the `phrase` type should be enhanced to facilitate a letter-by-letter sort, just as a `chunk` is implemented to support comparisons with or without case distinctions equally well. Another argument, perhaps `-o`*x* for ''order'' where *x* is either `w` or `l`, would be used to indicate the sorting scheme preferred. Function `p1 to p2` would have to be modified for this improvement.

5.2.4 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. Incorporating this feature involves expanding the `index term` data type to contain the sort key of the entry. Sorting would then be carried out on this field. The sort key would have to be defined somehow in the *phrase* file, separated from the phrase by the special character of the file.

5.2.5 Merging indices

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 indices on a chapter-by-chapter basis, whereby indices of each chapter are merged to form the index. The merging would be performed by another program to preserve the function of indx, however indx would have to be modified in order to allow for merging. One of the modifications needed would be to remove the verification that the cross referenced entries exist in the index. This verification would have to be dropped since it is conceivable that the part of the index being generated is not self-contained, that is an entry may cross reference another entry which will be defined in another part of the index. This is a very important check to prevent errors in an index. Perhaps the merge program could be accommodated by adding another argument to indx by which the indexer may indicate that cross references should not be verified. The verification may later be carried out during the merging. In addition, verification that chain references or circular references do not exist should also be done at this time.

5.2.6 Keeping a combined phrase in the index

While using indx to build the initial index of this thesis, 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 a situation existed, another phrase must be chosen to yield the same page numbers. This puts an additional burden on the human indexer and causes the existence of more index terms than necessary.

The decomposition of the `index` module will need no changes since `combine index terms` only combines the page references of the given terms, leaving the second term intact. It does not delete the second index term. `Form combined entries while validating terms` first combines the index terms and then deletes the second term for each definition given in the *combine-phrase* file. So this procedure would have to be changed to either combine or combine and delete. The *combine-phrase* file could be set up like the *group-entry* file, with the lines ending with the separation character of the file describing a combination only, not a combination and deletion.

5.3 Conclusions

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

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 automatically done by the human indexer when inserting indexing macros in a text must be explicitly defined in the *combine-phrase* and *alternate-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, the various places in the text where the macro is given must be searched for and changed, whereas with indx, the phrase must be changed in a few short files. In many cases, the only phrase that would need changing would be in the *alternate-index-term* file. Second, it seems that it would be easier to track down discrepancies in the index when all definitions are organized in specific files. Third, it is easier to remove index terms from the index by deleting the appropriate lines of the files involved. Fourth, as discussed earlier, one can calculate the total number of entries by the number of definitions in the files. Also, it would 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 to be altered 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 indices to the extent that cross references in the index will be valid 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 also relieves the indexer of having to know the page references for a given entry as the phrases are searched for in the text. This characteristic however, means that it takes longer for indx to get the macro calls comprising the index than the other programs which simply extract embedded commands and merge like entries, or which are given the page numbers and the entries on each page. It is also 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.

In regards to the design method and the implementation, indx is an example of a program modularized in such a way as to confine each design decision to one module. The modularization chosen and the Pascal implementation of the Ada packages enhanced the readability and understandability of the code. The design aided the coding process as many procedure bodies consisted mainly of using the routines exported by the other modules. The design should reduce the changes that would be necessary to implement any of the enhancements discussed earlier in this chapter. However, as with all modifications proposed, the positive and negative impacts of adding each feature must be examined in detail before any decision to add a feature is made.

The process of creating indices using indx is enhanced with other tools. Some of these tools exist, 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. The shell script in Appendix D can be adapted and used for updating indices.

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. Other options are to list all significant phrases and to list all phrases containing significant phrases. The significant phrases would be provided in a file similar to the *ignored phrases* file. 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, along with its existing tools provide much help to the person creating an index, but the indexing task remains a highly intellectual one.

REFERENCES

[Agui87] Christine Aguilera, *personal communication regarding findphrases*, 1987.

[Ande83] Charles Anderson, ''<<ANSWER>>: an 'off-the-shelf' program for computer-aided indexing,'' *The Indexer,* Vol. 13, No. 4, October 1983, pp. 236-238.

[Aurb86] Richard L. Aurbach, ''re: Idx$T_E\chi$,'' *TUGboat,* Vol. 7, No. 3, 1986, p. 187.

[Bent86] Jon L. Bentley and Brian W. Kernighan, ''Tools for Printing Indexes,'' *Computing Science Technical Report,* No. 128, AT&T Bell Laboratories, October 1986.

[Berr87] Daniel M. Berry, *personal communication*, 1987.

[Booc83] Grady Booch, *Software Engineering with Ada*, Menlo Park, CA: The Benjamin/Cummings Publishing Co., 1983.

[Coll62] Robert Collison, *Indexing Books*, New York, NY: John De Graff, Inc., 1962.

[Coll69] Robert L. Collison, *Indexes and Indexing*, Tuckahoe, NY: John De Graff, Inc., 1969.

[Corb87] Jonathan Corbet, *electronic mail message re: texindex program,* Jan 29, 1987.

[Fett86] Linda K. Fetters, ''INDEXIT: An Economical but Limited Indexing Program,'' *DATABASE,* Vol. 9, No. 5, October 1986, pp. 54-56.

[Gard82] Ron Gardner and Eve Gardner, ''Computer-aided indexing with SPITBOL and TEXTFORM,'' *The Indexer,* Vol. 13, No. 2, October 1982, pp. 115-119.

[Hard86] Paul Hardy, ''Computer-aided indexing of technical manuals,'' *The Indexer,* Vol. 15, No. 1, April 1986, pp. 22-24.

[Harr65] Eleanor T. Harris, *A Guide for the Preparation of Indexes,* Santa Monica, CA: The Rand Corporation, 1965.

[Hofm86] Thomas Hofmann, ''re: latexindex ,'' *TUGboat,* Vol. 7, No. 3, 1986, p. 186.

[Knig70] G. Norman Knight, in *Training In Indexing,* G. Norman Knight, Ed. Cambridge, MA: The M. I. T. Press, 1970.

[Lamp86] Leslie Lamport, LATEX *User's Guide & Reference Manual,* Reading, MA: Addison-Wesley, 1986.

[Myer78] Glenford J. Myers, *Composite/Structured Design,* New York, NY: Van Nostrand Reinhold Co., 1978.

[Oste87] Leon Osterweil, ''Software Processes are Software Too,'' *9th International Conference on Software Engineering,* Computer Society Press of the IEEE, 1987, pp. 2-13.

[Parn72] D. L. Parnas, ''On the Criteria to be Used in Decomposing Systems into Modules,'' *Communications of the ACM,* Vol. 15, No. 12, December 1972, pp. 1053-1058.

[Pasa81] Jay M. Pasachoff and Nancy P. Kutner, ''Computer assistance in indexing with *INDEX,'' *The Indexer,* Vol. 12, No. 4, October 1981, pp. 173-174.

[Salz86] Rich Salz, ''INDEX,'' in *Mirror Systems,* 1986.

[Tich82] Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' *Proceedings of the 6th International Conference on Software Engineering,* IEEE, September 1982.

[Unkn87] Unknown, *electronic mail message re:* TEX *indexing macros,* Jan 29, 1987.

[Wino80] Terry Winograd and Bill Paxton, *An Indexing Facility for* TEX, July 17, 1980 .

INDEX

APPENDIX A

Ignored Phrases

This appendix contains the lists of generic and text-specific ignored phrases used for the findphrases program run on sections of this thesis. The *ignored-phrases* file used is a concatenation of both lists. Neither list is comprehensive as there were phrases that should also have been ignored appearing on the list of repeated phrases. The list of generic ignored phrases may be expanded as desired.

List of Generic Ignored Phrases
for findphrases –b option

a	give	rest
ability	given	same
able	gone	say
about	good	several
above	has	she
added	have	should
adding	having	shows
addition	he	since
advantage	her	size
after	here	so
all	his	some
allow	how	specific
also	however	still
an	identified	such
and	if	suppose
another	important	supposed
any	in	taken
appear	including	than

are	individual	that
as	into	the
asking	is	their
at	it	them
available	its	then
be	itself	there
because	kind	therefore
become	know	these
been	large	they
beginning	like	things
being	make	this
both	many	thought
but	may	through
by	maybe	thus
can	means	times
certain	mentioned	to
come	more	too
consist	most	two
contain	must	up
containing	necessary	us
contains	need	use
define	needed	used
definition	new	uses
describes	no	using
describing	not	various
desired	now	was
determine	of	way
discussed	often	we
do	on	well
does	one	what
each	ones	whatever
easily	only	when
either	or	where
end	other	whether
enough	others	which
example	out	whose
exhibiting	pair	will
exist	paper	with
existing	placed	without
exists	possibly	would
finally	precede	write
finds	preceded	written
first	prepared	
following	presented	

for	processed
found	processing
from	provided
get	provides

List of Thesis-specific Ignored Phrases
for findphrases –b option

"	declared	navy
'	delete	open
(	discussed	p
)	dump	pairs
*	end	phrase
-	entry	phrases
/	example	prepare
3	exist	procedure
6	file	quite long
a	form	reference
absolute zero	found	retrieval of information
add	fourth	s
added	freeze	section
b	getnextchunk	sense
blank	i	set
book	index	storage and retrieval
c	indexer	string
calls	indexes	support
cards	indices	term
change	initialize	texts
char	iseof	triples
character	items	u
chnkopen	length	unit
chunk	line	yippee
clear	list	zero
close	majority	[
contains	make	]
copy	move	^
costs	n	`
declaration		

Files for the First Index

The *Phrase* File

:
* INDEX :
< < answer
actual phrase
algorithm
allows finding
alphabetized
alternate -
ASCII
automating
called chunks
Chain references
chunkfile
chunkfile . p
Circular references
combine -
combined
combined style
create cross
criteria
dedit
ditroff
documate
doubly
else PrintErrorMsg
end index
entry -
Entry phrase
entry phrases
error
file of phrases
find a subentry
findphrases
finds repeated
four fields
generate it
generate the index
group -
Group entry
Module 3
Module 4
Module 5
Module 6
nonalphabetic
Page reference
page references
paragraph ,
paragraph or
paragraph style
phrases (found
phrases . p
phrases are implemented
punctuation
range
Regular entry
related
retrievals of
Roman numerals
Salz
Search Table :
Search Table contains
see - also cross
see - also file
see - under cross
see - under file
see and
see cross
see file
select the phrases
select the terms
semi
separation character
sorted
sorting
special character
Starindex
subentries
Subentry

Heading
headings and
incorrectly sorted
Index :
INDEX takes :
INDEXIT
LATE\(*x
letter '
main entries
Main entry
Table of contents
TE\(*x
the input text
the phrase file
troff macro calls
troff macro packages
unfamiliar
units are
word '

The Optional Files Used for the First Index

combine-phrases file:

```
:
algorithm : else PrintErrorMsg
allows finding : findphrases
allows finding : finds repeated
called chunks : chunkfile
called chunks : Module 3
combined : combined style
create cross : related
create cross : unfamiliar
end index : Module 6
Entry phrase : entry phrases
four fields : chunkfile . p
generate it : generate the index
Heading : headings and
incorrectly sorted : nonalphabetic
incorrectly sorted : punctuation
Main entry : main entries
Module 5 : phrases ( found
Page reference : page references
paragraph style : paragraph ,
paragraph style : paragraph or
phrases are implemented : phrases . p
retrievals of : Search Table contains
see cross : see and
select the phrases : select the terms
sorting : sorted
Subentry : subentries
the phrase file : file of phrases
units are : Module 4
units are : the input text
```

group-entry file:

```
:
Alphabetization problems : ASCII
Alphabetization problems : incorrectly sorted
Alphabetization problems : Roman numerals
```

```
alphabetized : letter '
alphabetized : word '
Checking indx output for : actual phrase
Checking indx output for : range
Chunks : called chunks
Chunks : four fields
create cross : see - also cross
create cross : see - under cross
create cross : see cross
Formatting styles : combined
Formatting styles : entry -
Formatting styles : paragraph style
Optional files : alternate -
Optional files : combine -
Optional files : group -
Optional files : see - also file
Optional files : see - under file
Optional files : see file
Optional files : separation character
Phrases : Module 5
Phrases : phrases are implemented
Semi-automatic indexing : automating
Semi-automatic indexing : criteria
Semi-automatic indexing : semi
Semi-automatic indexing programs : * INDEX
Semi-automatic indexing programs : < < answer
Semi-automatic indexing programs : documate
Semi-automatic indexing programs : Index
Semi-automatic indexing programs : INDEX takes
Semi-automatic indexing programs : INDEXIT
Semi-automatic indexing programs : LATE\(*x
Semi-automatic indexing programs : Salz
Semi-automatic indexing programs : Starindex
Semi-automatic indexing programs : TE\(*x
Storage and retrieval : find a subentry
Storage and retrieval : retrievals of
The index : doubly
The index : end index
the phrase file : special character
Units : units are
```

see-also file:

```
:
doubly : Search Table : The index
Entry phrase : Heading
sorting : alphabetized
troff macro calls : troff macro packages
```

see file:

```
:
alphabetic order : Alphabetization problems : Checking indx output for
Check entries : Circular references
```

see-under file:

```
:
Alternate-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
```

alternate-index-term file:

```
:
sorting : Sorting of entries
select the phrases : Selection of phrases
allows finding : findphrases program
generate it : Manual indexing task
letter ' : letter-by-letter : alphabetized
word ' : word-by-word : alphabetized
alphabetized : Alphabetization schemes
semi : definition of : Semi-automatic indexing
automating : motivation for : Semi-automatic indexing
criteria : criteria for programs : Semi-automatic indexing
* INDEX : *INDEX : Semi-automatic indexing programs
< < answer : <<ANSWER>> : Semi-automatic indexing programs
```

```
documate : Documate/Plus : Semi-automatic indexing programs
Salz : index (with troff) : Semi-automatic indexing programs
TE\(*x : with TE\(*x : Semi-automatic indexing programs
LATE\(*x : with LATE\(*x : Semi-automatic indexing programs
INDEX takes : INDEX : Semi-automatic indexing programs
entry - : entry-per-line : Formatting styles
paragraph style : paragraph (run-in) : Formatting styles
see cross : see : create cross
see - also cross : see also : create cross
see - under cross : see under : create cross
create cross : Cross references
the phrase file : Phrase file
alternate - : alternate-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
ASCII : ASCII order : Alphabetization problems
Roman numerals : numerals : Alphabetization problems
incorrectly sorted : special characters : Alphabetization problems
range : page references : Checking indx output for
actual phrase : see-under cross references : Checking indx output for
algorithm : Algorithm of indx
error : Error handling
troff macro calls : Output of indx
troff macro packages : Formatting final index
called chunks : design : Chunks
four fields : implementation : Chunks
Module 5 : design : Phrases
phrases are implemented : implementation : Phrases
end index : design : The index
doubly : implementation : The index
units are : design : Units
retrievals of : of main entries : Storage and retrieval
find a subentry : of subentries : Storage and retrieval
```

Files for the Final Index
The *Phrase* File

:
* INDEX :
< < answer
a minimum
abstraction
actual phrase
algorithm
allow for merging
allows finding
allupper
alphabetized
alternate -
approximated
are word -
ASCII comparisons
automatically deleted
automating
awk
browser program
calculate
called chunks
Chain references
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
documate
LATE\(*x
letter method
main entries
Main entry
merging
mI :
minor changes
modifications proposed
Module 5
Module 6
nonalphabetic
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

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
generate the index
grep
grep was
group -
Group entry
hash function
Heading
headings and
ignored phrases
incorrectly sorted
increased to three
Index :
INDEX takes :
INDEXIT
individual files
indx helps
involves expanding
see - also cross
see - also file
see - under cross
see - under file
see and
see cross
see file
select the phrases
select the terms
semi
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
TE\(*x
the input text
the method
the phrase file
troff macro calls
troff macro packages
unfamiliar
units are
updating the index
used in a subentry

The Optional Files Used for the Index

combine-phrases file:

```
:
actual phrase : used in a subentry
algorithm : else PrintErrorMsg
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
end index : Module 6
Entry phrase : entry phrases
findphrases needs : significant phrases
from the list : a minimum
generate it : generate the index
Heading : headings and
incorrectly sorted : erroneously placed
incorrectly sorted : nonalphabetic
incorrectly sorted : punctuation
Main entry : main entries
Module 5 : phrases ( found
novel idea : indx helps
obtain the index found : changes been extensive
obtain the index found : grep
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
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
troff macro packages : mI
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 method
alphabetized : are word -
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
Enhancing the features of indx : allow for merging
Enhancing the features of indx : automatically deleted
Enhancing the features of indx : increased to three
Enhancing the features of indx : involves expanding
Enhancing the features of indx : scheme
Enhancing the features of indx : second 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 : alternate -
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
Semi-automatic indexing programs : * INDEX
Semi-automatic indexing programs : < < answer
Semi-automatic indexing programs : awk
Semi-automatic indexing programs : documate
Semi-automatic indexing programs : Index
Semi-automatic indexing programs : INDEX takes
Semi-automatic indexing programs : INDEXIT
Semi-automatic indexing programs : LATE\(*x
Semi-automatic indexing programs : Salz
Semi-automatic indexing programs : Starindex
Semi-automatic indexing programs : TE\(*x
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:

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

see file:

```
:
alphabetic order : Alphabetization problems : Checking indx output for
Check entries : Circular references
findphrases : allows finding : indx tools
Searching the index : search table
```

see-under file:

```
:
aid for : allows finding : phrase selection
Alternate-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
```

alternate-index-term file:

```
:
```

```
algorithm : Algorithm of indx
ASCII comparisons : ASCII order : Alphabetization problems
Roman numerals : numerals : Alphabetization problems
incorrectly sorted : special characters : Alphabetization \
problems
letter method : letter-by-letter : alphabetized
are word - : word-by-word : alphabetized
alphabetized : Alphabetization schemes
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 thesis 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
second cross : more cross references : Enhancing the features of indx
increased to three : more entry levels : Enhancing the features of indx
scheme : sorting scheme option : Enhancing the features of indx
involves expanding : sort key identification : Enhancing the features \
of indx
automatically deleted : redefine combined phrases : Enhancing the \
features of indx
allow for merging : merging indices : 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
troff macro packages : Formatting final index
entry - : entry-per-line : Formatting styles
paragraph style : paragraph (run-in) : Formatting styles
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
alternate - : alternate-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
troff macro calls : Output of indx
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
awk : awk index tools : Semi-automatic indexing programs
* INDEX : *INDEX : Semi-automatic indexing programs
< < answer : <<ANSWER>> : Semi-automatic indexing programs
documate : Documate/Plus : Semi-automatic indexing programs
Salz : index (with troff) : Semi-automatic indexing programs
TE\(*x : with \*(TX : Semi-automatic indexing programs
LATE\(*x : with \*(LT : Semi-automatic indexing programs
INDEX takes : INDEX : Semi-automatic indexing programs
sorting : Sorting of entries
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

```
.IX 0 reg "Algorithm of indx" "37, 38" ""
.IX 0 reg "Alphabetization problems" "" ""
.IX 1 reg "ASCII order" "23" ""
.IX 1 reg "numerals" "6" ""
.IX 1 reg "special characters" "13, 17, 18, 23, 24, 26, 27, 50, 64" ""
.IX 0 reg "Alphabetization schemes" "1, 6, 23, 55" ""
.IX 1 reg "letter-by-letter" "6" ""
.IX 1 reg "word-by-word" "6" ""
.IX 0 under "Alternate-index-term file" "" "Optional files"
.IX 0 reg "Chain references" "41, 64" ""
.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" "23, 28, 56, 57, 68" ""
.IX 1 reg "see-under cross references" "24, 40, 41" ""
.IX 0 reg "Chunks" "" ""
.IX 1 reg "design" "30, 31, 32, 33, 34, 36, 43, 44, 45" ""
.IX 1 reg "implementation" "43, 44, 49" ""
.IX 0 also "Circular references" "24, 64" "Chain references"
.IX 0 under "Combine-phrase file" "" "Optional files"
.IX 0 reg "Creating the thesis index" "57, 58, 59" ""
.IX 0 reg "Cross references" "4, 14, 30, 51" ""
.IX 1 reg "see" "4, 10, 11, 13, 16, 17, 20, 22, 24, 34, 35, 36, 39, 41, \
42, 49, 62" ""
.IX 1 reg "see also" "4, 13, 16, 17, 20, 21, 24, 35, 40" ""
.IX 1 reg "see under" "4, 16, 17, 20, 24, 34, 35, 39, 41" ""
.IX 0 reg "dedit" "18, 29, 52, 67" ""
.IX 0 reg "Determining size of the index" "25, 61, 66" ""
.IX 0 reg "ditroff" "18, 29, 67" ""
.IX 0 reg "Enhancing the features of indx" "" ""
.IX 1 reg "merging indices" "64" ""
.IX 1 reg "more cross references" "62" ""
.IX 1 reg "more entry levels" "63" ""
.IX 1 reg "redefine combined phrases" "65" ""
.IX 1 reg "sort key identification" "64" ""
.IX 1 reg "sorting scheme option" "49, 63" ""
.IX 0 also "Entry phrase" "3, 5, 9, 10, 13, 20, 22, 26, 34, 35, 41, 45, \
46, 54" "Heading"
.IX 0 reg "Error handling" "2, 10, 22, 30, 39, 42, 54, 62, 63" ""
.IX 0 reg "findphrases program" "15, 19, 50, 51, 52, 53, 54, 55, 57, 58, \
59, 67, 68" ""
.IX 1 reg "aid in phrase selection" "50, 53" ""
.IX 1 reg "ignored phrases file" "19, 20, 51, 53, 68" ""
.IX 1 reg "improvements suggested" "51, 67, 68" ""
.IX 1 reg "preparing input file" "52, 53" ""
.IX 0 reg "Formatting final index" "25, 50, 59" ""
.IX 0 reg "Formatting styles" "" ""
.IX 1 reg "combined" "5, 11, 16, 17, 20, 25, 27, 40, 54, 65" ""
.IX 1 reg "entry-per-line" "5, 11, 21, 22, 23, 25, 29, 42, 59" ""
.IX 1 reg "paragraph (run-in)" "5, 11, 21, 23, 25" ""
.IX 0 reg "Group entry" "4, 8, 17, 35, 36, 37, 41, 46, 54, 66" ""
.IX 0 under "Group-entry file" "" "Optional files"
.IX 0 reg "Heading" "3, 4, 5, 6, 13, 16, 17, 21, 22, 27, 40, 48, 49, \
51, 61, 62, 66" ""
.IX 0 reg "Improving indx's performance" "" ""
.IX 1 also "hash function" "46, 47, 54, 60, 61" "Search Table"
.IX 1 reg "page merging routine" "27, 60, 61, 64" ""
.IX 0 reg "indx design" "29, 30, 67" ""
.IX 1 reg "modifiability of" "67" ""
.IX 0 reg "indx method" "3, 16, 17, 18, 26, 28, 50, 56, 57, 59, 65" ""
.IX 1 reg "advantages of" "66" ""
.IX 1 reg "disadvantages of" "66, 67" ""
.IX 0 reg "indx tools" "" ""
.IX 1 reg "browser program" "23, 68" ""
.IX 1 reg "dedit" "18, 29, 52, 67" ""
.IX 1 see "findphrases" "" "findphrases program"
.IX 1 reg "grep" "58" ""
.IX 1 reg "updating shell script" "9, 12, 58, 59, 67" ""
.IX 0 reg "Main entry" "4, 8, 9, 10, 12, 21, 24, 25, 35, 36, 40, 41, \
42, 46, 48, 49, 52, 54, 65, 66" ""
.IX 0 reg "Manual indexing task" "3, 59" ""
.IX 0 reg "Optional files" "" ""
.IX 1 reg "alternate-index-term" "17, 18, 22, 26, 27, 36, 41, 48, 54, \
55, 63, 66" ""
.IX 1 reg "combine-phrase" "17, 20, 25, 26, 27, 48, 54, 55, 58, 59, 60, \
62, 65, 66" ""
.IX 1 reg "group-entry" "17, 21, 25, 40, 48, 54, 55, 59, 63, 65" ""
.IX 1 reg "see" "22, 25, 39, 41" ""
.IX 1 reg "see-also" "21, 22, 40" ""
.IX 1 reg "see-under" "22, 25" ""
.IX 1 reg "separation character" "19, 21, 39, 65" ""
.IX 0 also "Output of indx" "18, 23, 42" "Formatting final index"
.IX 0 reg "Page reference" "3, 4, 6, 11, 13, 16, 17, 20, 23, 24, 25, \
26, 27, 28, 34, 38, 39, 40, 45, 46, 50, 52, 53, 54, 56, 57, 58, 60, 65, \
66, 67, 68" ""
.IX 0 reg "Phrase file" "16, 18, 20, 24, 25, 26, 27, 38, 39, 42, 49, \
50, 53, 54, 55, 58, 59, 64, 65" ""
.IX 1 reg "special character" "19, 64" ""
.IX 0 reg "Phrases" "" ""
.IX 1 reg "design" "30, 32" ""
.IX 1 reg "implementation" "44, 49" ""
.IX 0 reg "Preparing input files" "9, 11, 14, 16, 18, 21, 22, 26, 40, \
54, 55, 65" ""
.IX 0 reg "Preparing input text" "2, 16, 17, 18, 19, 26, 30, 31, 33, \
42, 52" ""
.IX 0 reg "Regular entry" "4, 35, 36, 46" ""
.IX 0 reg "Search Table" "46, 48, 54, 60, 61" ""
.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" "5, 15, 16, 50, 51, 52, 53, 54, 55, \
57, 67" ""
.IX 1 under "aid for" "" "findphrases program"
.IX 1 reg "difficulties in" "51, 53" ""
.IX 1 reg "for indx" "20, 50, 51, 53" ""
.IX 0 reg "Semi-automatic indexing" "" ""
.IX 1 reg "criteria for programs" "14, 41" ""
.IX 1 reg "definition of" "7, 10, 15, 41, 53, 56, 62, 65" ""
.IX 1 reg "motivation for" "2" ""
.IX 0 reg "Semi-automatic indexing programs" "" ""
.IX 1 reg "*INDEX" "12, 13" ""
.IX 1 reg "<<ANSWER>>" "12" ""
.IX 1 reg "awk index tools" "9, 10" ""
.IX 1 reg "Documate/Plus" "7, 8" ""
.IX 1 reg "INDEX" "12" ""
.IX 1 reg "Index" "3, 5, 6, 7, 8, 11, 46, 51, 57, 59" ""
.IX 1 reg "index (with troff)" "9, 26" ""
.IX 1 reg "INDEXIT" "12, 13, 14" ""
.IX 1 reg "Starindex" "7, 8" ""
.IX 1 reg "with \*(LT" "12" ""
.IX 1 reg "with \*(TX" "9, 10, 11, 12, 25" ""
.IX 0 also "Sorting of entries" "6, 7, 8, 9, 10, 11, 12, 13, 14, 17, \
18, 23, 24, 27, 45, 49, 50, 52, 54, 56, 62, 63, 64" "Alphabetization schemes"
.IX 0 reg "Storage and retrieval" "" ""
.IX 1 reg "of main entries" "46, 48" ""
.IX 1 reg "of subentries" "46" ""
.IX 0 reg "Subentry" "4, 5, 11, 12, 13, 21, 22, 25, 27, 34, 35, 40, 41, \
42, 45, 46, 52, 55, 59, 63, 65" ""
.IX 0 reg "Table of contents" "1, 9, 13" ""
.IX 0 reg "The index" "" ""
.IX 1 reg "design" "34, 37" ""
.IX 1 also "implementation" "45, 48" "Search Table"
.IX 0 reg "Units" "" ""
.IX 1 reg "design" "30, 31, 33, 34, 49" ""
.IX 1 under "implementation" "" "Chunks"
.IX 0 reg "Updating an index" "57, 58, 59" ""
```

The troff Macro Calls to Form the Index
After Removing Useless Page References and Replacing Page Sequences

```
.IX 0 reg "Algorithm of indx" "37-38" ""
.IX 0 reg "Alphabetization problems" "" ""
.IX 1 reg "ASCII order" "23" ""
.IX 1 reg "numerals" "6" ""
.IX 1 reg "special characters" "13, 17, 23, 24, 26, 27, 64" ""
.IX 0 reg "Alphabetization schemes" "1, 6, 23" ""
.IX 1 reg "letter-by-letter" "6" ""
.IX 1 reg "word-by-word" "6" ""
.IX 0 under "Alternate-index-term file" "" "Optional files"
.IX 0 reg "Chain references" "41, 64" ""
.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" "23, 28, 56, 57, 68" ""
.IX 1 reg "see-under cross references" "24, 41" ""
.IX 0 reg "Chunks" "" ""
.IX 1 reg "design" "30-32, 33, 34" ""
.IX 1 reg "implementation" "43-44, 49" ""
.IX 0 also "Circular references" "24, 64" "Chain references"
.IX 0 under "Combine-phrase file" "" "Optional files"
.IX 0 reg "Creating the thesis index" "57-59" ""
.IX 0 reg "Cross references" "4, 14" ""
.IX 1 reg "see" "4, 10, 11, 13, 22, 24, 34, 35, 36, 39, 41" ""
.IX 1 reg "see also" "4, 13, 16, 21, 24, 35, 40" ""
.IX 1 reg "see under" "4, 16, 24, 34, 35, 39, 41" ""
.IX 0 reg "dedit" "18, 29, 52, 67" ""
.IX 0 reg "Determining size of the index" "25" ""
.IX 0 reg "ditroff" "18, 29" ""
.IX 0 reg "Enhancing the features of indx" "" ""
.IX 1 reg "merging indices" "64" ""
.IX 1 reg "more cross references" "62" ""
.IX 1 reg "more entry levels" "63" ""
.IX 1 reg "redefine combined phrases" "65" ""
.IX 1 reg "sort key identification" "64" ""
.IX 1 reg "sorting scheme option" "63" ""
.IX 0 also "Entry phrase" "3, 5, 22, 26, 34, 35, 46" "Heading"
.IX 0 reg "Error handling" "22, 42" ""
.IX 0 reg "findphrases program" "15, 19, 50-52, 67" ""
.IX 1 reg "aid in phrase selection" "50, 53" ""
```

```
.IX 1 reg "ignored phrases file" "19, 20, 51, 53" ""
.IX 1 reg "improvements suggested" "51, 67-68" ""
.IX 1 reg "preparing input file" "52-53" ""
.IX 0 reg "Formatting final index" "25, 59" ""
.IX 0 reg "Formatting styles" "" ""
.IX 1 reg "combined" "5, 11, 25, 65" ""
.IX 1 reg "entry-per-line" "5, 11, 23" ""
.IX 1 reg "paragraph (run-in)" "5, 11, 21, 23, 25" ""
.IX 0 reg "Group entry" "4, 8, 17, 35, 36, 37, 66" ""
.IX 0 under "Group-entry file" "" "Optional files"
.IX 0 reg "Heading" "3-4, 6, 13, 16, 17, 21, 22, 27, 40, 48-49, 66" ""
.IX 0 reg "Improving indx's performance" "" ""
.IX 1 also "hash function" "46-47, 60, 61" "Search Table"
.IX 1 reg "page merging routine" "27, 60-61" ""
.IX 0 reg "indx design" "29-30, 67" ""
.IX 1 reg "modifiability of" "67" ""
.IX 0 reg "indx method" "16-18, 26-28, 50, 57, 59, 65" ""
.IX 1 reg "advantages of" "66" ""
.IX 1 reg "disadvantages of" "66-67" ""
.IX 0 reg "indx tools" "" ""
.IX 1 reg "browser program" "23, 68" ""
.IX 1 reg "dedit" "18, 29, 52, 67" ""
.IX 1 see "findphrases" "" "findphrases program"
.IX 1 reg "grep" "58" ""
.IX 1 reg "updating shell script" "58-59, 67" ""
.IX 0 reg "Main entry" "4, 21, 24, 35, 36, 40, 41, 42, 46, 48-49, 52" ""
.IX 0 reg "Manual indexing task" "3" ""
.IX 0 reg "Optional files" "" ""
.IX 1 reg "alternate-index-term" "17, 18, 22, 26, 27, 36, 41, 54-55, \
66" ""
.IX 1 reg "combine-phrase" "17, 20, 26-27, 58, 60, 65" ""
.IX 1 reg "group-entry" "17, 21, 40, 54" ""
.IX 1 reg "see" "22, 25, 39, 41" ""
.IX 1 reg "see-also" "21, 22, 40" ""
.IX 1 reg "see-under" "22, 25" ""
.IX 1 reg "separation character" "19, 21, 39, 65" ""
.IX 0 also "Output of indx" "18, 23, 42" "Formatting final index"
.IX 0 reg "Page reference" "3, 4, 6, 16, 17, 20, 23, 26, 28, 34, 38-39, \
46, 56-58" ""
.IX 0 reg "Phrase file" "16, 18-20, 24, 27, 38, 42, 49, 50, 53, 58, 59, \
65" ""
.IX 1 reg "special character" "19" ""
.IX 0 reg "Phrases" "" ""
.IX 1 reg "design" "30, 32" ""
```

```
.IX 1 reg "implementation" "44, 49" ""
.IX 0 reg "Preparing input files" "18-22, 26, 54-55, 65" ""
.IX 0 reg "Preparing input text" "18, 42" ""
.IX 0 reg "Regular entry" "4, 35, 36" ""
.IX 0 reg "Search Table" "46-48, 54, 61" ""
.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" "5, 15, 16, 50-53, 57, 67" ""
.IX 1 under "aid for" "" "findphrases program"
.IX 1 reg "difficulties in" "51, 53" ""
.IX 1 reg "for indx" "20, 50-53" ""
.IX 0 reg "Semi-automatic indexing" "" ""
.IX 1 reg "criteria for programs" "14" ""
.IX 1 reg "definition of" "7, 15" ""
.IX 1 reg "motivation for" "2" ""
.IX 0 reg "Semi-automatic indexing programs" "" ""
.IX 1 reg "*INDEX" "12, 13" ""
.IX 1 reg "<<ANSWER>>" "12" ""
.IX 1 reg "awk index tools" "9-10" ""
.IX 1 reg "Documate/Plus" "7, 8" ""
.IX 1 reg "INDEX" "12" ""
.IX 1 reg "Index" "7-8" ""
.IX 1 reg "index (with troff)" "9, 26" ""
.IX 1 reg "INDEXIT" "12, 13-14" ""
.IX 1 reg "Starindex" "7, 8" ""
.IX 1 reg "with \*(LT" "12" ""
.IX 1 reg "with \*(TX" "9, 10-12, 25" ""
.IX 0 also "Sorting of entries" "6-7, 14, 23, 24, 27, 63, 64" \
"Alphabetization schemes"
.IX 0 reg "Storage and retrieval" "" ""
.IX 1 reg "of main entries" "46-48" ""
.IX 1 reg "of subentries" "46" ""
.IX 0 reg "Subentry" "4, 5, 21, 25, 27, 34, 35, 40, 41, 42, 46, 52, 63" \
""
.IX 0 reg "Table of contents" "1" ""
.IX 0 reg "The index" "" ""
.IX 1 reg "design" "34-37" ""
.IX 1 also "implementation" "45-48" "Search Table"
.IX 0 reg "Units" "" ""
.IX 1 reg "design" "30, 31, 33, 34" ""
.IX 1 under "implementation" "" "Chunks"
.IX 0 reg "Updating an index" "57-59" ""
```

APPENDIX C

Troff Macro Definitions

This appendix contains troff macro definitions for printing out an index in the entry–per–line style and the paragraph style. The entry–per–line macro definition was used to format the index. A header file and possibly a trailer file should be defined to set up other quantities such as point size and line length.

Entry–per–line Style definition

```
.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
..
```

The following lines should be placed in a header file:

```
.nr PL 0   \" initialize level number of previous index term
.nr TP 0   \" initialize type of previous index term to regular
.ds CR ''  \" initialize cross reference of previous index term
.in +3n    \" a trailer file must undo this indentation
```

The following is the paragraph style definition:

```
.de IX
.if \\$1=1 \{\
.       ie \\n(PL=0 :
.       el ;
.       ie '\\$2'see' \\$3, \\fISee\\fP \\$5\c
.       el \{\
.               ie '\\$2'under' \\$3, \\fISee under\\fP \\$5\c
.               el \{\
\\$3, \\$4\c
.                       if '\\$2'also' \&.(\\fISee also\\fP \\$5)\c\}\}\}
.if \\$1=0 \{\
.
.       ie \\n(PL=0 \{\
.               if \\n(TP \{\
.
.                       ie \\n(TP=1 \&. \\fISee also\\fP \\*(CR\c
.                       el \{\
.                               ie \\n(TP=2 \&. \\fISee\\fP\\ \\*(CR\c
.                               el \&. \\fISee under\\fP \\*(CR\c\}\}\}
.       el \" do nothing
.\"
.       ti -3n
.               \" move further left for main entry
.       ie '\\$2'reg' \{\
.               nr TP 0
.                       \" TP = 0 for regular entry
.               ie '\\$4'' \\$3\c
.               el \\$3, \\$4\c\}
.       el \{\
.               ie '\\$2'also' \{\
.                       nr TP 1
.                               \" TP = 1 for see-also entry
.                       ie '\\$4'' \\$3\c
.                       el \\$3, \\$4\c\}
.               el \{\
.                       ie '\\$2'see' \{\
.                               nr TP 2
.                                        \" TP = 2 for see entry
\\$3\c\}
.                      el \{\
.                                  nr TP 3
.                                        \" TP = 3 for see-under entry
\\$3\c\}\}
.            ds CR \\$5\}\}
.                                  \" save cross reference for later printing
.nr PL \\$1
.            \" save level number
..
```

APPENDIX D

Shell Script for Updating Indices

This appendix contains a shell script which may be used or adapted to update already existing indices formed with indx and findphrases. The script below is in effect a process program [Oste87].

```
EDITOR= your favorite editor
FPHARGS= options for findphrases
BOOK= list of files making up book in order
PREPASSES= pipe of prepasses
MACROS= name of macro package, e.g., X for -mX
INDXARGS= options for indx
DEVICE.DRIVER= ditroff device driver
INDEXMACROS= macros for formatting index
if ! -e RCS/phrase.list,v then
       findphrases $(FPHARGS) $(BOOK) > phrase.list
       ci -l phrase.list
       cp phrase.list term.list
       @echo Dear Author: the term.list file is now a copy of phrase.list
       @echo edit it to a real list of terms!
       $(EDITOR) term.list
else
       findphrases $(FPHARGS) $(BOOK) > phrase.list
       rcsdiff phrase.list >& changes
       ci -l phrase.list
       @echo Dear Author: the changes file shows the changes to the
       @echo phrase.list
       @echo edit term.list to reflect the contents of changes
       $(EDITOR) term.list
fi
/bin/cat $(BOOK) | $(PREPASSES) | dtroff -m$(MACROS) > book.dt
dedit book.dt| indx $(INDXARGS) > index
$(DEVICE.DRIVER) book.dt
troff -m$(INDEXMACROS) index
```

APPENDIX E

Manual Pages of Indx, Dedit, and Findphrases

This appendix contains the manual pages for the indx, dedit, and findphrases programs discussed in the thesis. They explain the usage of each program, identifying all of the options available.

NAME

indx – create an index for an arbitrary text

SYNOPSIS

indx **–p***phrase-file* [**–s***see-file*] [**–a***see-also-file*] [**–u***see-under-file*] [**–c***combine-phrase-file*] [**–g***group-entry-file*] [**–n***alternate-index-term-file*] [**–d***pgdelim*]

DESCRIPTION

Indx generates calls to troff macros to create an index containing the terms specified by the user, of the standard input, which is written to standard output. The input must be in a general format where all words and sentence punctuations are separated by blanks and virtual page numbers are provided on a separate line preceding the text that belongs on the page. This page number must be immediately preceded by a formfeed character (^L) followed by a lowercase p and starts on the leftmost end of the line. If a file to be indexed is in ditroff output format, the program *dedit*(1) can be used to convert the file into the format expected by *indx.*

The output contains calls to troff macros of the form

.IX *l type "entry-phrase" "pg refs" "cross ref"*

where *l* is zero for main entries and one for subentries, and *type* is either reg, also, see, or under. A reg type macro will have a null *cross ref* string and see and under type macros will have a null *pg refs* string. These types of index terms are described below. The page references will be separated by the *pgdelim* string given in the **-d** option. If this page delimiter includes blanks, it must be enclosed in quotes with each blank of the delimiter preceded by a backslash. Should a backslash be desired as part of the delimiter, it must also be preceded by a backslash. If it is not specified, the default delimiter ", " will be used. Two .IX macros exist to print the index in one of two styles. In the entry-per-line style, each entry (main or sub-) is placed on a separate line. In paragraph style, sub-entries are listed after the main entry and separated by semi-colons. Several special types of index terms may be specified in addition to the regular index term, which consists of a phrase as it appears in the *phrase-file* followed by the list of page numbers on which the phrase appears. These special types are generated by the optional files, which are discussed later.

The *phrase-file* contains all phrases for which page numbers are to be found. These phrases are searched for as word-for-word occurrences in the text input. There is one phrase per line and the phrases must be alphabetically sorted. Any phrase whose page numbers will be listed in the index should be in the *phrase-file*, whether the page numbers will be under that specific phrase, under another phrase, or both. The first line of the *phrase-file* should have one character, which will be used to flag phrases in which case distinctions are to be made when searching for occurrences. All lines ending with this character will have phrases that are case-sensitive. Should this special character be needed as a word of a phrase, the character should be doubled. More generally, if any of the words consist only of this special character, it should be immediately preceded by the special character. The same property applies to the separation characters of the optional files. If a phrase does not exist in any of the other files, it will be a regular index term having an entry phrase exactly the same as the phrase in the *phrase-file*. If a phrase does exist in one or more of the other files, it may or may not be a regular index term, depending on which of the other files it appears in.

The remaining files mentioned in the synopsis have the following general form: the first line consists of a character referred to as the separator character and the remaining lines contain phrases separated by the separator character. There must be at least two phrases on each line for each file. The first phrase will be referred to as the first term and the second phrase as the second term. If there is a third phrase, it will define the main entry of a group entry. Thus, all lines in these files having three terms correspond to index terms that are sub-entries. As in the *phrase-file*, phrases must be given exactly as they are to appear in the index.

The *see-file* contains *See* cross-references. The first and second terms appear in the index, but the *first term* will not have any page numbers listed. Instead, a reference to the *second term* will be made:
See *second term.*
If there is a third term, the index term containing the *see* cross-reference will be placed under the group given by the third term. An index term may have only one cross-reference.

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

A *see-file* entry that would yield the macros to give the index term having the cross-reference is:

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

The *see-also-file* contains *See also* cross-references. The first and second terms appear in the index and have their page references listed. The page numbers for the first term are followed by a reference to the second term:
See also *second term.*
If there is a third term, the index term that will contain the *see-also* cross-reference must be a sub-entry of the group given by the third term. An index term may have only one cross-reference.

For example, to eventually get the following entry (entry-per- line format):

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

the *see-also-file* could contain

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

The *see-under-file* contains *See under* cross-references. The first and second terms appear in the index, but the *first term* will not have any page numbers listed. Instead, a reference to the *second term*, which is the main entry for a group of entries, will be made:
See under *second term.*
This is to be used when the *first term* is actually used under a different heading. There can be a third term on a line, as described for the *see-file*. As with the other types of cross-references,

an index term can have only one cross-reference.

For example, a *see-under file* containing:

```
:
program maintenance documentation : documentation
```

would yield a macro corresponding to the index entry

program maintenance documentation, See under documentation

The *combine-phrase-file* contains pairs of terms for which the first term will list, in addition to its own page references, the page references of the second term. The second term will not appear in the index. This may be used to index plurals along with their singular references, for example, to list the page references of **program design languages** under the index term **program design language.** There can be only two phrases per line, since phrases will be combined before group-entries are created.

The *group-entry-file* groups sub-terms with terms. The second term will be the sub-term of the first term. A maximum of two levels for a group term can be defined, thus the first term cannot be a sub-term of a group. There can be only two phrases per line in this file. If the main term is not found in the index, an index term for it will be inserted and the group relationship established. When the second term is grouped under the first term, it will be no longer be a main term. If it is desired that the second term remain as a main term in the index, the second phrase must be followed by the separator character of the file.

For example, to eventually get the following portion of an index (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, the *group-entry-file* would contain

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

The *alternate-index-term-file* pairs terms that appear in the text with their corresponding index term, which may or may not appear in the text. This option is provided to list entries defined in the *phrase-file* under a different term, which may, for example, be inverted: **INDX, manual page of** instead of **manual page of INDX**. The first term is the term in the text and the second term is the index term to be used. Lines in this file may have three phrases, defining an entry change for a sub-entry.

Indx will build all regular index terms defined by the *phrase-file* and then will process the optional files in the following order: *combine-phrase-file*, *group-entry-file*, *see-also-file*, *see-file*, *see-under-file*, and *alternate-index-term-file*.

As an example, suppose the completed index is to contain

```
Alphabetic sort, See Binary search trees

Binary search trees, 320-372
  inserting a new element, 335-341

Binary trees, 203-207

Insertion
  binary trees, 335-341

LEFT pointers, 208-214

Pointers
  LEFT, 208-214
  RIGHT, 208-214

RIGHT pointers, 208-214
```

The following files would be needed:

phrase-file

```
:
Binary trees
Binary search trees
insert the element z
LEFT pointer :
LEFT pointers :
RIGHT pointer :
RIGHT pointers :
z into a binary tree
z into an existing binary tree
```

combine-phrase-file

```
:
LEFT pointers : LEFT pointer
RIGHT pointers : RIGHT pointer
z into a binary tree : insert the element z
z into a binary tree : z into an existing binary tree
```

group-entry-file

```
:
Binary search trees : z into a binary tree :
Insertion: z into a binary tree
Pointers : LEFT pointers :
Pointers : RIGHT pointers :
```

alternate-index-term-file

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

```
                    RIGHT pointers : RIGHT : Pointers
                    z into a binary search tree : inserting a new element : Binary \
                    search trees
                    z into a binary search tree: binary trees : Insertion
     see-file
                    :
                    Alphabetic sort : Binary search trees
```

Something like the following would be generated:

```
.IX 0 see "Alphabetic sort" "" "Binary search trees"
.IX 0 reg "Binary search trees" "320, 321, 333, 335, 348, 367" ""
.IX 1 reg "inserting a new element" "335, 337, 338, 339, 341" ""
.IX 0 reg "Binary trees" "203, 204, 205, 206, 207" ""
.IX 0 reg "Insertion" "" ""
.IX 1 reg "binary trees" "335, 337, 338, 339, 341" ""
.IX 0 reg "LEFT pointers" "208, 210, 211, 214"
.IX 0 reg "Pointers" "" ""
.IX 1 reg "LEFT" "208, 210, 211, 214" ""
.IX 1 reg "RIGHT" "208, 209, 211, 213, 214" ""
.IX 0 reg "RIGHT pointers" "208, 209, 211, 213, 214" ""
```

Note that the page references from the program are pages on which the exact phrase occurs. It may be the case that the subject is continuously discussed from one page occurrence to the next, even past the last page of the reference list. It is the indexer's responsibility then to replace these lists with the ranges. Thus, the completed index would be as given above.

SEE ALSO

findphrases(1) may be used to obtain a list of phrases that would probably appear in the index. It creates a table of phrases that are repeated in an arbitrary text. Using it should decrease the time and effort required to obtain a list of index terms.

dedit(1) may be used to convert ditroff output to the input text format.

troff(1) may be used to format the text and the index. The **mI** macro package contains the macros for indexing.

NAME

dedit – extract pure text and page numbers from ditroff output

SYNOPSIS

dedit [–o*list*] [*file*]

DESCRIPTION

Dedit extracts pure text and page numbers from *ditroff*(1) output, paying attention to the end-of-word and end-of-line markers and ignoring movements. If a file name is not given, the standard input is used as input. Otherwise, the contents of the file is used as the input. The output is written to standard output.

The -**o** option is used to print only the pages enumerated in *list*. The list consists of pages and page ranges, eg. 7-11, separated by commas. The range n- goes from n to the end; the range -n goes from the beginning to and including page n.

Dedit is intended to prepare text in ditroff output format for use in the *indx*(1) program, which will create an index for the text.

SEE ALSO

dtroff(1), *indx*(1)

NAME

findphrases – find repeated phrases in an arbitrary text

SYNOPSIS

findphrases [**–n***number*] **–p***punctuation-keyword-file* [**–x***ignored-phrases-file*] [**–m***multi-tokens-file*] [**–u**] [**–b**] [**–s**] [**–t**] [**–v**] [**–c**]

DESCRIPTION

All files mentioned in the synopsis provide their data in what is referred to as free format subject to particular restrictions to be described for each case. In free format, the items of the file may be entered zero or several per line with a mixture of blanks and tabs before, in between, and after the items. Obviously, no item can include a blank, a tab, or a newline.

The **–n** argument is optional and if present provides a number *number* serving as the maximum length phrase (to be described later) to be tallied. If this argument is not present, if it does not supply a number, or if the supplied number is outside the reasonable range of greater than zero and less than or equal to 50, then *number* is taken as 10.

The *punctuation-keyword-file* contains in free format a list of those character strings to be taken as punctuation/keywords (see below). The optional *ignored-phrases-file* contains one-per-line a list of phrases to be ignored in the tallying (see below). In each line, the tokens (see below) are in free format. The optional *multi-tokens-file* contains in free format a list of those character strings consisting of more than one symbolcharacter (see below) which are to be taken as multi-tokens (see below).

No assumptions are made about the standard input, thus it may be an arbitrary text. The program parses the text into words and symbolcharacters. These in turn are formed and classified into tokens and punctuation/keywords based on the information provided by the *punctuation-keyword-file* and, when the **–m** option is present, the *multi-tokens-file*.

First some definitions are necessary:

> *Whitespace: blank, tab, newline, beginning-of-file, end-of-file*
>
> *Wordcharacter: letter, digit,* _
>
> *Symbolcharacter:* any printable character which is neither a wordcharacter nor a blank
>
> *Word:* any sequence of wordcharacters delimited on each side by whitespace or a symbolcharacter
>
> *Punctuation/Keyword:* whatever is in the *punctuation-keyword-file*; the symbolcharacter strings are called punctuation and the wordcharacter strings are called keywords
>
> *Multi-token:* whatever is in the *multi-tokens-file*
>
> *Token:* any word, symbolcharacter, or multi-token which is not listed in the *punctuation-keyword-file*
>
> *Sentence:* list of tokens delimited on each side by punctuation/keyword
>
> *Phrase:* one or more consecutive tokens occurring within one sentence

The main job of this program is to tally the occurrence of all phrases in all sentences. The maximum length phrase that has to be considered is that of *number* tokens. If the *ignored-phrases-file* is provided, then the phrases given in the file are to be ignored in the tallying. If the **–b** option is used along with the *ignored-phrases-file*, then phrases which begin with an ignored phrase are also ignored in the tallying.

The standard output consists of:

> a copy of the input as is, with the lines numbered and the punctuation/keywords overstruck two times (i.e., printed three times in place) so that they can be spotted easily,
>
> a frequency ranked table of the repeated phrases. i.e., those appearing more than once among the sentences; that is the entries of the table are given in order of decreasing frequency, and
>
> an alphabetically ordered table of the repeated phrases.

In the two tables, the entry for a repeated phrase consists of:

> a sequence of asterisks indicating the phrase's frequency as a percentage of the maximum frequency; in this one asterisk represents 10%,
>
> the actual number of occurrences of the repeated phrase,
>
> the repeated phrase itself, and
>
> a list of the numbers of all lines containing the beginning of the repeated phrase.

In printing the repeated phrase itself in a table entry, the underscores, i.e., ''_'', are printed as blanks. This means that an underscore can be used immediately preceeding or following a word that looks like a keyword to prevent it from being considered a keyword.

Note that the definition of ''phrase'' is independent of the number of times it occurs in the sentences. An *ignored phrase* is simply one to be ignored in the tallying but not in seaching for phrases. A phrase which contains an ignored phrase which itself is not ignored is to be tallied. When the –**b** option is present, a phrase which begins with an ignored phrase is not to be tallied. A *repeated phrase* is one whose final tally is greater than one. Only the repeated phrases show up in the tables of the output.

Typically, the *ignored-phrases-file* will contain so-called noise phrases such as ''a'', ''an'', ''the'', ''of'', ''of the'', etc. plus any useless phrases found in previous runs of the program.

One particular configuration of the files is as follows:

> *Punctuation-keyword-file:* ; [ ] **abort accept access all and array at begin body case constant declare delta digits do else elsif end entry exception exit for function generic goto if in is limited loop mod new not null of or others out package pragma private procedure raise range record rem renames return reverse select separate subtype task terminate then type use when while with xor**
>
> *Multi-tokens-file:* ** := <= >= /= .. <> << >>

This configuration is suited for finding repeated phrases in Ada™ (Ada is a trademark of the U. S. Department of Defense.) or in an Ada-based program design language.

If the –**u** option is present, then only the unique phrases that are not wholely and everywhere contained in another phrase are listed in the tables of the output. In addition to the already specified output, if the –**s** option is present, then all the sentences are listed; if the –**t** option is present, then all the tokens are listed; if the –**v** option is present, then the output is verbose with the punctuation/keywords listed, and when the –**m**, and respectively the –**x**, option is present, the multi-tokens, and respectively the ignored phrases, are listed. If the –c option is present, then upper and lower case distinctions are to be applied in determining whether a phrase is in a sentence. The default is to ignore case distinction in the comparisons.