USER-ORIENTED ACCESS

TO A

MULTILINGUAL DATABASE

Ruth Clarke, B.A. (Mod)
Department of Computer Science,
Trinity College, Dublin.

March 31st, 1995.

This thesis is submitted to the University of Dublin, Trinity College, in fulfilment of the requirements for the degree of M.Sc. (Computer Science).

I, the undersigned, declare that this work has not previously been submitted to this or any University, and that unless otherwise stated, it is entirely my own work.

Trinity College Library may lend or copy this thesis on request.

Signed: [redacted]

Date: 10th July 1995

ACKNOWLEDGMENTS

‘Céad Míle’ thanks to my supervisor, Professor J.G. Byrne for his infinite support and interest in the project. Among the many things I have learned under his supervision is to ‘never blame the machine’.

A special thanks to all the staff of the Computer Science Department - in particular Rosemary Welsh and Willie O’Connor who provided unlimited technical support, and the AI Lab for adopting me as one of their own.

Finally thanks to my family, friends and Stéphane for their support and encouragement throughout this thesis.

ABSTRACT

This thesis describes the development of a system that provides user-oriented access to a multilingual database. The database concerned is the 1872 Printed Catalogue of Trinity College, Dublin, which lists many rare and valuable books in over fourteen languages. Computerization of the Printed Catalogue began in 1989 and in 1993 a search and retrieval system was developed to search the catalogue.

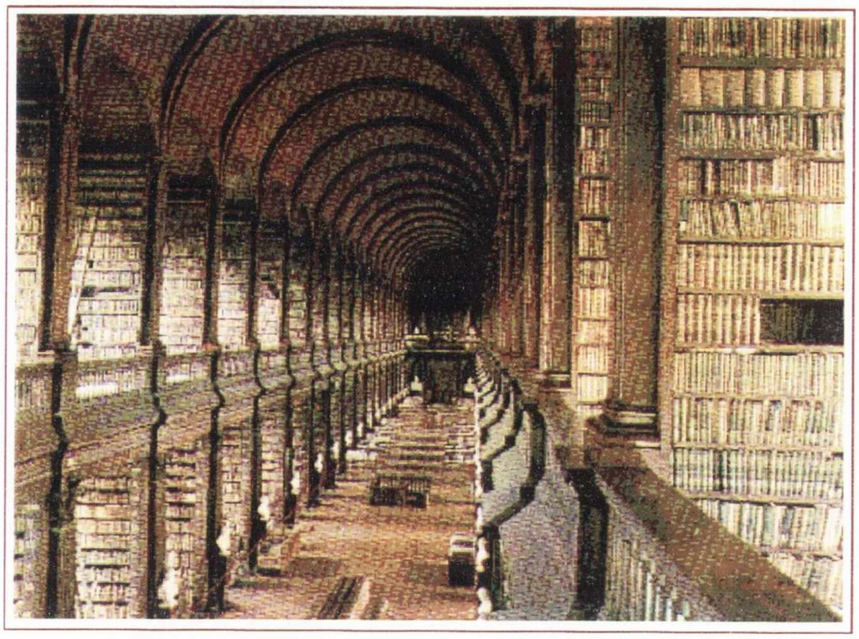
The aim of this project was to improve access to this database by supporting the information seeking strategies of the user and by providing tools to facilitate the user in his query formulation. These tools include automatic and interactive query expansion techniques - which alter the user's query to better reflect his information need and the vocabulary of the database while hiding the effects of errors created during Optical Character Recognition; term frequency information and previous search results - which determine important search terms and effective search strategies, and a comprehensive help facility to aid the user during the searching process.

Subject access to the database is achieved using 'knowledge trees' of information which are built up by expert users on their areas of specialisation. Fellow users can view the subject trees and use them to automatically formulate search statements and to enhance queries.

Multilingual access is achieved by enabling users to search the catalogue by language of title. Language recognition of the catalogue entries was required before a search by language could be developed. An author name translation tool is also described which provides multilingual access to the database by automatically translating author names, principally to and from Latin.

In addition the user interface has been redesigned to support the naive, novice and skilled users of the system. A bibliography manager is introduced which enables these users create personal files to which they can save any relevant material encountered while searching the database.

The thesis concludes with an evaluation of the new 1872 Search System to determine whether or not access to the catalogue has been improved from a user's perspective.



The Long Room, Trinity College Dublin,

where the books listed in the 1872 Catalogue are housed.

CONTENTS

FIGURES

TABLES

APPENDICES

CHAPTER 1 - INTRODUCTION

1.1 INTRODUCTION

This chapter describes the history behind the *"Catalogus Librorum impressorum qui in Bibliotheca Collegii sacrosancte et individuae Trinitatis, reginae Elizabethae, juxta Dublin, adservantus"* hereafter referred to as the 1872 Printed Catalogue. This catalogue lists all the books acquired by Trinity College Library before 1872 and contains very many rare and ancient books. However access to this catalogue is limited. There is no subject index present, and so without knowing the author, it is practically impossible to locate a book. Indeed it has happened that librarians ordered copies of books they already had as they could not locate them in the catalogue! To overcome these access limitations, computerization of the catalogue began in 1989 which eventually led to the production of the 1872 Search System. The limitations of the initial search and retrieval mechanism led to the development of a user-oriented search and retrieval system for the database - the goal of this thesis.

1.2 HISTORY OF THE 1872 PRINTED CATALOGUE

Shortly after the foundation of Trinity College Dublin in 1592, its' library came into existence and began acquiring books. Early-seventeenth-century sources provide evidence that by 1601 the nucleus of the college library was formed. Substantial purchases were made during the early years of the seventeenth century when two of the Fellows - James Ussher and Luke Challoner, went on book buying expeditions to

London, Oxford and Cambridge but these expeditions were followed by a period of slow growth in the library for the next half century. In 1656 Usher died and his personal collection consisting of some 10,000 books became the first of the library's great benefactions. This acquisition along with donations of book collections notably from William Palliser (archbishop of Cashel) in 1796, Dr. Claudius Gilbert (vice-provost of Trinity College) in 1735 who donated 13,000 volumes and Dr. John Stearne (vice-chancellor and bishop of Clogher) in 1745, marked major developments in the growth of the library.

In 1802 the size of the library was increased by nearly 50% with the purchase of the private collection of Hendrick Fagel who had been Chief Minister of the Netherlands at the time of Napoleon's invasion. This collection is a significant acquisition as it contains mainly continental works which had been poorly represented in the library up until then. The library became a copyright library in 1801 which entitled it to claim a copy of every book published in the British Isles. Since then the size of the library has grown annually. By the end of the nineteenth century, the library's collection had risen to one quarter of a million volumes. Today, the total collection numbers three million volumes and the annual intake of books and periodicals require one mile of additional shelving every year.

The initiation of the Printed Catalogue in the 1830's, came at a time of reform within the college, mirroring the reforms in society at large (Catholic Emancipation 1829, Reform Bill 1831). The catalogue is referred to as the 1872 Printed Catalogue because it lists all the books acquired by the college before 1872 - the cutoff date chosen for publication.

Bartholomew Lloyd, who was appointed provost of the college in 1831, scrutinized every aspect of college life and reorganized what he found wanting. Hence the cataloguing of the library was undertaken, a task that might not have been tackled in a more conservative period. The then part-time librarian - James Henthorn Todd embarked on this project and in 1849 the printing of the first of eight volumes was undertaken. He based the model of the 1872 Printed Catalogue on the Bodleian

Catalogue of 1843 so as may be found in the Bodleian Catalogue, latinized forms of names predominate. Also, the order of the catalogue was based on the Latin alphabet which results in the letters I/J and U/V being interfiled. Thus 'JACKSON' comes before 'IRELAND', 'NEWTON John' before 'NEWTON Isaac' and 'VOLTAIRE' before 'USSERUS'. Needless to say, this can be very confusing for the modern user and considerable experience is required to use the catalogue effectively.

Discussions were going on in cataloging circles at the time as to whether a catalogue should be a finding-list or a complete bibliographic database. When the Public University Commission described Todd's catalogue as the latter, he responded angrily by saying:

"The present catalogue aims only at giving the titles of all books under their authors...no attempt at a 'complete description' or a 'perfect arrangement' has been made."

For this reason anything other than finding a book in the catalogue is a difficult task. There is no real subject index and so without knowing the author of a book, locating its whereabouts becomes impossible. Even though some headings such as 'IRELAND' (230 entries), 'PARLIAMENT' (1107 entries) and 'BIBLIA' (825 entries) are present in the catalogue, the number of entries listed under these headings suggest that the headings are too general and a time-consuming process of sifting through them would be required before any relevant material is located.

Todd had to reorganize the process of cataloguing in 1835 as up until then, the cataloging had been carried out by scholars of the college who left as soon as their scholarship expired and hence caused great inconsistencies in the compilation of the catalogue. The catalogue slips were checked and filed in alphabetical order by Todd himself. In 1846 he claimed that the whole library had been catalogued.
In 1848 printing got under way but progress was extremely slow. It took five years to complete the printing of the A pages and over a decade for the B pages. This was due to the fact that each of the catalogue slips had to be revised before printing and the lack of assistance slowed down the process considerably. Other reasons for the delay included

financial difficulties and the appalling condition of the library roof which was in danger of collapse. During 1859 and 1860 the roof was replaced and the Long Room, which stored all the books, remodeled. In 1869 Todd died and no attempt to regalvanise the project was taken until 1872 when Henry Dix Hutton was appointed editor of the catalogue with J.H.Hessels as his assistant. These men had a stormy working relationship however which further slowed the printing process. Hutton complained to the Library Committee in 1873 that:

"A considerable portion of the slips were abridged imperfectly..... he [Hessels] omitted to mark all the slips which he verified by reference to the books, although he had agreed to do so..."

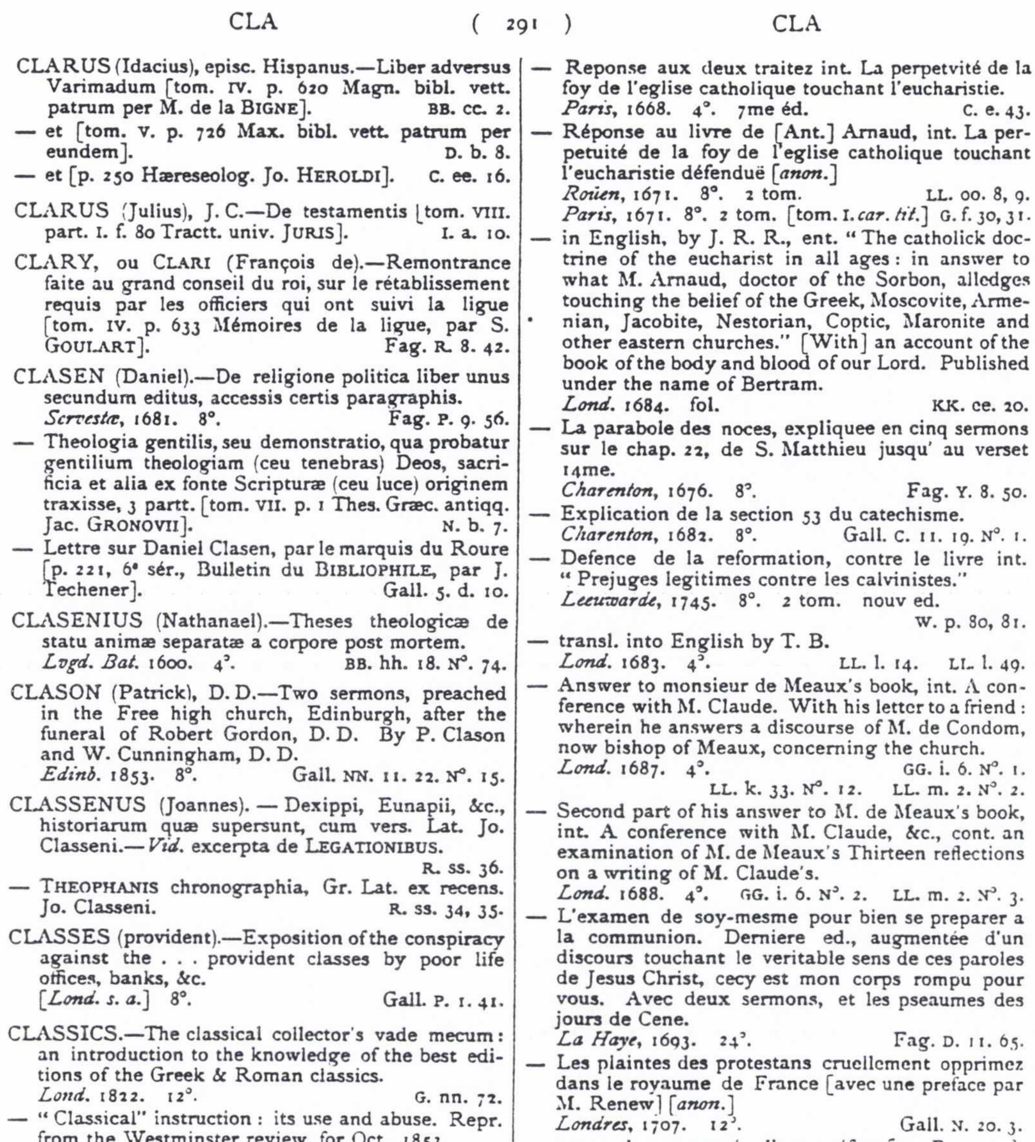
Hessels was more frank in his criticism of Hutton:

"Slips are in no condition to be printed at all... I regard [this] as the incompetency of the so-called editor who seems to be only troubled by commas and dots.....all the defects of the catalogue cannot be charged to Hutton but I wish he had some knowledge of old books, literature, which constitute the chief part of the catalogue.."

In 1878 Hessels was dismissed and T.V.Keenan was employed in his stead. Keenan was much more amenable to the editor's policy but this unfortunately did not change the rate of printing and the project staggered on slowly to completion in 1887, by which time it had taken fifty years to complete.

This brief history of the 1872 Printed Catalogue serves to illustrate that with so many different employees, various disputes between cataloguers and poor conditions in the library - the process of cataloguing was bound to incorporate an enormous amount of inconsistencies. These inconsistencies reduced users' access to the books contained in the catalogue and provided obstacles in the computerization process.

User’s access was further restricted as there are over 14 languages contained in the catalogue - some modern European languages (English, French, Italian, French, German, Dutch, Portuguese, Irish, Danish, Norwegian, Swedish) and other languages with non roman alphabets (Russian, Arabic, Hebrew and Syriac). Figure 1.1 shows one of the 1872 Catalogue pages with a number of the above languages appearing in the entries.



CLA (291) CLA

CLARUS (Idacius), episc. Hispanus.—Liber adversus Varimadum [tom. IV. p. 620 Magn. bibl. vett. patrum per M. de la BIGNE]. BB. cc. 2.

— et [tom. V. p. 726 Max. bibl. vett. patrum per eundem]. D. b. 8.

— et [p. 250 Hæreseolog. Jo. HEROLDI]. C. ee. 16.

CLARUS (Julius), J. C.—De testamentis [tom. VIII. part. I. f. 80 Tractt. univ. JURIS]. I. a. 10.

CLARY, ou CLARI (François de).—Remontrance faite au grand conseil du roi, sur le rétablissement requis par les officiers qui ont suivi la ligue [tom. IV. p. 633 Mémoires de la ligue, par S. GOULART]. Fag. R. 8. 42.

CLASEN (Daniel).—De religione politica liber unus secundum editus, accessis certis paragraphis.
Servestæ, 1681. 8°. Fag. P. 9. 56.

— Theologia gentilis, seu demonstratio, qua probatur gentilium theologiam (ceu tenebras) Deos, sacrificia et alia ex fonte Scripturæ (ceu luce) originem traxisse, 3 partt. [tom. VII. p. 1 Thes. Græc. antiqq. Jac. GRONOVII]. N. b. 7.

— Lettre sur Daniel Clasen, par le marquis du Roure [p. 221, 6e sér., Bulletin du BIBLIOPHILE, par J. Techener]. Gall. 5. d. 10.

CLASENIUS (Nathanael).—Theses theologicæ de statu animæ separatæ a corpore post mortem.
Lugd. Bat. 1600. 4°. BB. hh. 18. N°. 74.

CLASON (Patrick), D. D.—Two sermons, preached in the Free high church, Edinburgh, after the funeral of Robert Gordon, D. D. By P. Clason and W. Cunningham, D. D.
Edinb. 1853. 8°. Gall. NN. 11. 22. N°. 15.

CLASSENUS (Joannes). — Dexippi, Eunapii, &c., historiarum quæ supersunt, cum vers. Lat. Jo. Classeni.—*Vid.* excerpta de LEGATIONIBUS. R. ss. 36.

— THEOPHANIS chronographia, Gr. Lat. ex recens. Jo. Classeni. R. ss. 34, 35.

CLASSES (provident).—Exposition of the conspiracy against the . . . provident classes by poor life offices, banks, &c.
[*Lond. s. a.*] 8°. Gall. P. 1. 41.

CLASSICS.—The classical collector's vade mecum: an introduction to the knowledge of the best editions of the Greek & Roman classics.
Lond. 1822. 12°. G. nn. 72.

— "Classical" instruction: its use and abuse. Repr. from the Westminster review, for Oct., 1853.
Lond. 1854. 8°. [Chapman's libr. for the people]. Gall. N. 4. 12. N°. 6.

— BIBLIOGRAFIA od élenco ragionato delle opere contenute nella collezione de' classici Italiani. CC. q. 19.

CLATER (Francis).—Every man his own farrier.
Lond. 1810. 8°. 21st ed. N. o. 68.

— by him and his son John Clater.
Lond. 1849. 12°. 20th ed. Gall. B. 8. 72.

— 31st ed., by E. Mayhew.
Lond. 1861. 12°. Gall. L. 19. 88.

— Every man his own cattle doctor.
Lond. 1832. 8°. 7th ed. FF. o. 66.
Lond. 1848. 12°. 10th ed. Gall. B. 8. 65.

— entirely re-written to the present date by G. Armatage.
Lond. 1870. 8°. Gall. P. 3. 26.

CLAVASIO (Angelus de).—*Vid.* Ang. CARLETUS.

CLAUBERGIUS (Johannes).—De cognitione Dei et nostri, exercitationes centum.
Harlingæ, 1685. 8°. ed. noviss. E. m. 15.

CLAUBRY (Gaultier de). — *Vid.* GAULTIER-de-Claubry.

CLAUDE (Jean).—Oeuvres posthumes.
Amst. 1688-9. 8°. 5 tom. LL. oo. 10-14. Fag. Z. 9. 1-5.

— Reponse aux deux traitez int. La perpetvité de la foy de l'eglise catholique touchant l'eucharistie.
Paris, 1668. 4°. 7me éd. C. e. 43.

— Réponse au livre de [Ant.] Arnaud, int. La perpetuité de la foy de l'eglise catholique touchant l'eucharistie défenduë [*anon.*]
Roüen, 1671. 8°. 2 tom. LL. oo. 8, 9.
Paris, 1671. 8°. 2 tom. [tom. I. *car. tit.*] G. f. 30, 31.

— in English, by J. R. R., ent. "The catholick doctrine of the eucharist in all ages: in answer to what M. Arnaud, doctor of the Sorbon, alledges touching the belief of the Greek, Moscovite, Armenian, Jacobite, Nestorian, Coptic, Maronite and other eastern churches." [With] an account of the book of the body and blood of our Lord. Published under the name of Bertram.
Lond. 1684. fol. KK. ee. 20.

— La parabole des noces, expliquee en cinq sermons sur le chap. 22, de S. Matthieu jusqu' au verset 14me.
Charenton, 1676. 8°. Fag. Y. 8. 50.

— Explication de la section 53 du catechisme.
Charenton, 1682. 8°. Gall. C. 11. 19. N°. 1.

— Defence de la reformation, contre le livre int. "Prejuges legitimes contre les calvinistes."
Leeuwarde, 1745. 8°. 2 tom. nouv ed. W. p. 80, 81.

— transl. into English by T. B.
Lond. 1683. 4°. LL. l. 14. LL. l. 49.

— Answer to monsieur de Meaux's book, int. A conference with M. Claude. With his letter to a friend: wherein he answers a discourse of M. de Condom, now bishop of Meaux, concerning the church.
Lond. 1687. 4°. GG. i. 6. N°. 1. LL. k. 33. N°. 12. LL. m. 2. N°. 2.

— Second part of his answer to M. de Meaux's book, int. A conference with M. Claude, &c., cont. an examination of M. de Meaux's Thirteen reflections on a writing of M. Claude's.
Lond. 1688. 4°. GG. i. 6. N°. 2. LL. m. 2. N°. 3.

— L'examen de soy-mesme pour bien se preparer a la communion. Derniere ed., augmentée d'un discours touchant le veritable sens de ces paroles de Jesus Christ, cecy est mon corps rompu pour vous. Avec deux sermons, et les pseaumes des jours de Cene.
La Haye, 1693. 24°. Fag. D. 11. 65.

— Les plaintes des protestans cruellement opprimez dans le royaume de France [avec une preface par M. Renew] [*anon.*]
Londres, 1707. 12°. Gall. N. 20. 3.

— nouv. ed., augmentée d'une préface [par Basnage].
Cologne, 1713. 8°. Fag. X. 9. 47.

— Account of the persecutions and oppressions of the protestants in France [*anon.*]
Dubl. 1686. 4°. LL. o. 12. N°. 4. RR. pp. 10. N°. 11.
[A great part of the original of Claude is omitted in this translation].

— His life and death [by Abel Rodolph de la Devèse]. Done out of French by G. P.
Lond. 1688. 4°. P. gg. 26. N°. 7. LL. m. 2. N°. 4.

CLAUDERUS (Gabriel).—Methodus balsamandi corpora humana, aliaque majora sine evisceratione et sectione hucusque solita.
Ienæ [1679]. 4°. OO. l. 3.

CLAUDIANUS (S.), martyr Hieropolitanus.—De SS. Severo . . . Claudiano, &c. [tom. X. Oct. p. 635 Actt. SS. BOLLANDI]. II. aa. 27.

CLAUDIANUS (Andreas).—Mavors Irlandicus sive historia de bello Hibernico biennium in Hibernia gesto; chartis consignata. Press A. 1. 38
Hafniæ, 1718. 8°. ~~Press~~ [illegible]

CLAUDIANUS (Claudius). — In Ruffinum lib. II. De bello Gildonico lib. I. Epithalamium in nuptiis Honorii et Mariæ. Panegyrici VII. In Eutropium lib. II. De bello Getico lib. I. Epigrammata quædam. De raptu Proserpinæ lib. III.
Florentiæ (hæredes Phil. Juntæ), 1510. 8°. TT. o. 10.

Figure 1.1: Sample 1872 Catalogue Page

1.3 BACKGROUND

In 1989 it was decided to convert the catalogue into machine readable form - not only to preserve the catalogue and the information contained therein but also to provide access to some of Europe's rarest books using keyword searches on all fields of the records.

- The first phase of the project consisted of converting the scanned catalogue pages into their ASCII equivalent. The fact that the pages of the catalogue were printed in a reasonably structured format with a restricted number of fonts prompted the use of Optical Character Recognition (OCR). OCR uses a variety of algorithms to recognise the characters contained in an image file and to convert them into their ASCII equivalents. Glynn Anderson [*Anderson* `92] wrote the OCR software used in this project which was written in C and used template matching to recognise the characters. Some of the catalogue pages were scanned and the OCR software was developed and tested using these pages as samples. The algorithm used had a success rate of 95.9% and was written specifically for the catalogue, using features like the dash occurring in front of each of the titles to separate the entries {Figure 1.1}.

  When this software became usable - another student, Brendan Culligan wrote both scanning and compression software for the catalogue pages. The 5155 catalogue pages were then scanned using a Microtek scanner and the new scanning software. The scanned images were 1000K in size but were reduced by 90% with the compression code which employed arithmetic coding. Once all the pages were scanned and compressed, he used the OCR software to convert the image pages to ASCII pages, modifying the OCR software where necessary in the light of the experience gained in using it.

  However the 95.9% recognition rate claimed by the OCR software meant that 4.1% of the characters were misrecognised causing spelling errors in approximately 20% of the words (since the average word contains around 5 letters). These errors were not only due to the fact that the OCR software was not perfect but due also to the age of the catalogue and the printing techniques used in the nineteenth century. Whatever the cause, in order to access the information in the catalogue, an attempt was made to correct as many of the errors as possible.

- The second phase of the project began with the correction of errors produced by the OCR software. A corpus of 50 ASCII catalogue pages was examined manually and it was observed that a small percentage of the errors were context free. For example the number '10' was often interpreted to be '1 o' by the OCR software and hence could be corrected throughout, without the interaction of a user.

Software written in the Spitbol version of Snobol4 [*Griswold* `71] to automatically correct these context free errors was run on all the pages. The remaining errors were context sensitive however and could only be corrected interactively. An example of such an error is where the letters 'fl' were often, but not always recognised as 'n' - to produce, for example, the word 'renections' instead of 'reflections'. Since the recognition was not consistent an interactive spelling corrector was written (again in Snobol4) to detect misspellings in English text resulting from misrecognised characters, and to produce candidate corrections for these misspelled words. This program was to be run on the titles of all the entries of the catalogue so each of the entries in the catalogue were first broken down into a number of fields - author, title, place of publication and date - again using software written in Snobol4.

With a view to applying this spelling corrector program to text in all of the languages of the catalogue, it was necessary to recognise the language of each entry so that an appropriate dictionary could be applied in the interactive spelling correction process. This prompted the development of a language recogniser program which used function words and character patterns to deduce the language of each entry. The language recogniser would enable the development of dictionaries in the various languages from the catalogue itself. These dictionaries would be more appropriate in the spelling correction process as the terms contained in them would vary enormously from those contained in modern electronic dictionaries.

- The final phase of the computerization project lay in making the information in the catalogue available to the modern user of the library. A free-text search and retrieval system was developed in the C++ programming language which allows the user to search on words and word stems contained in the various fields of each entry [*Culligan `93*]. A fast response time was obtained by constructing an inverted file which contains a four-letter key and information about all of the entries that contain that key. Hash coding is used to look up this file. The first four letters of the word to be searched for, provides the hash key for the inverted file. By using only the first four letters of a word for the hash key and storing the complete word in the hash location, searching both for exact matches and stem matches is possible. The ideas used for the representation of text were based on those developed by Professor F.J.Smith and his students in Queen's University, Belfast [*Smith & Devine `83*]. The searching system allows searching on the author, title, place of publication and date field and a comprehensive Boolean search is also available which allows searches on a combination of the above fields.

1.4 USER-ORIENTED SEARCH SYSTEM

Today, fifty years of manual cataloguing is available at the touch of a button. However the access to the historic works contained in the catalogue remains basic. A user enters a keyword in a certain field - if there are entries with that word contained in the catalogue then those titles are listed - if not, the search fails. The aim of this thesis is to develop the search system currently available to provide user-oriented access to the library database by guiding the user in his query formulation and encouraging him to try various queries in order to retrieve as much information as possible. It is hoped to overcome the following restrictions currently existing in the search and retrieval system.

- The entries appear in over fourteen languages and hence the loss of information incurred in a query to the database where the user assumes the use of only one language is vast. For example if the user wants to find all the books on 'mathematics' and he enters this keyword as a query then all the titles of books containing the words 'mathematique', 'Mathematik', 'mathematica' etc. will be ignored. He can overcome this restriction by using a wild card search e.g. 'mathem*' which will retrieve most of the titles containing the above keywords. However not all words in the different languages show such similarities. If for example the user wishes to find all the books on 'houses' the difficulty of the language situation is heightened. English 'House', French 'Maison', Italian 'Casa', German 'Haus' are very different in their structures. By providing some means of accessing keywords in the various languages, titles of books in languages other then the native language of the user could be retrieved.

- Due to the hash coding scheme used in the search and retrieval software, words less than four characters in length may not be used in queries. This eliminates the use of important three letter words in searching e.g. 'war' and also eliminates the use of abbreviations. Unfortunately abbreviations in the catalogue are common and occur most notably in the initials following an author's name. To improve the system, there should be a provision to access this information so that statistical analysis on authors of the catalogue could be carried out. The following are some examples of abbreviations found in the catalogue:

  CAPANUS, (Andreas), U.J.D.
  SWIFT, (Jonathan), D.D., dean of St. Patrick's, Dublin.
  CONRIUS seu CONRY (Florentius), O.S.F., archiep. Tuamensis.

- Errors remaining in the catalogue text files, produced by the OCR software should be hidden from the user. A list of the most common OCR errors was produced in the second phase of the computerization of the catalogue. e.g. 'rt' is often recognised as 'tt'. This list could be used to automatically map the keyword entered by the user to its possible misrepresentations in the text files. Should the user enter the word 'part' then this would be mapped to 'patt' before the search is implemented so that the titles containing that word would also be listed in the search results.

- A user-oriented system could perhaps provide an option for bibliography building. In this way each user could create one or more personal files, add the titles he comes across during the search process that interest him, and print this file on completion of his search. The advantage of such a facility would be the reduction of idle time each user spends in front of the search system taking down references to the books that interest him.

- To guide the user in the process of forming his queries, dictionaries and perhaps thesauri could be made available for browsing. Then certain terms could be selected from these dictionaries and added to the user's initial query to vary his search and hence his results.

- Since the catalogue is domain independent, the knowledge contained in it covers a diverse range of subjects. The average user might be competent in a number of subjects but perhaps expert in only one. If he could create a knowledge-based system containing relevant terms and information on his particular subject area, then a novice user could avail of the knowledge built by the expert in the formulation of his query. A user that is perhaps interested in 'astrology' but knows little or nothing about the subject area, could refer to the knowledge base on that subject for help regarding terms to use in his query.

The goal of this project is to improve user's access to the 1872 Catalogue. This will be achieved by improving the formulation of queries as described above so as to enhance the information retrieval obtained by the user without much extra effort on his part. Since the majority of the above facilities are interactive - the user can choose for himself whether he needs help in his query formulation or whether he can adequately express his search request. Multilingual access to the library database is another goal of this project. There are two aspects of multilingual access relevant to this database - *multilingual interfaces* where the user can choose the interface corresponding to his native language and then input a query in that language and *multilingual searches* where a monolingual user can gain access to titles pertaining to his language only. The latter facility is the only technique currently provided for this database and it requires the languages of each entry to be recognised.

1.5 THESIS OVERVIEW

Chapter 2 discusses the research area of Information Retrieval (IR) in relation to the 1872 Catalogue. The basic elements of an IR system are presented including the most common search strategies currently in use. The shift from traditional Information Retrieval to user-oriented Information Retrieval is then described. The chapter concludes with a description of the IR system developed for the 1872 Catalogue and outlines the aspects of this system that need to be altered in the development of a user-oriented IR system.

Chapter 3 introduces the concept of query expansion as a method of improving Information Retrieval. Query expansion is the process by which the user's query is altered to better reflect his information need and the vocabulary of the database. The various query expansion techniques developed specifically for the 1872 search system are presented.

Exposing users to new concepts and subjects will improve their ability to apply knowledge. Chapter 4 explores this idea by enabling users to build their own trees of domain knowledge. Once a subject tree has been built, it may be consulted by users at any stage during the search process. The knowledge contained in the subject trees may be used to create new search statements or improve previous search statements. In this way, users can create complex searches in subject areas they are unfamiliar with, thereby improving their retrieval of information.

In the development of a more user-oriented IR system, the native language of the user has not been overlooked. Chapter 5 provides a brief history of language change with specific attention to the language families present in the database, namely the Romance and Germanic language families. A multilingual aid, which employs knowledge about language change, is described. This tool was developed to help the user of any language, search for authors in the 1872 library database.

Another technique of gaining multilingual access to the library database is described in chapter 6. A language recognition algorithm is presented which uses function words and affixes to determine the language of an entry in the database. This algorithm has been run on the entire catalogue and has enabled the development of a language search. It is now possible for the user to search for a book on the language of it's entry.

Chapter 7 highlights the importance of a good user-interface in an IR system. Interface design issues and design styles are discussed, and the design of the interface for the user-oriented 1872 search system is described.

Chapter 8 concludes the thesis with an evaluation of the improved user-oriented system. Future work suggestions are presented for the further improvement and development of the 1872 Search System.

CHAPTER 2 - INFORMATION RETRIEVAL

2.1 INTRODUCTION

Most people are faced with a need for information at some stage or another, and more often than not this need is translated into a formal search perhaps in a library or information centre. To facilitate the task of the user in finding items of interest, libraries and information centres provide a variety of aids. One such aid is an Information Retrieval system. Information retrieval systems are concerned with the representation, storage and retrieval of information [*Salton `83*]. In principle, the concept of information storage and retrieval is simple.

There exists stores of documents and a person (user) formulates a question (query) to which the answer is a set of documents satisfying the information need expressed by his question. Figure 2.1 outlines the main processes carried out in Information Retrieval.

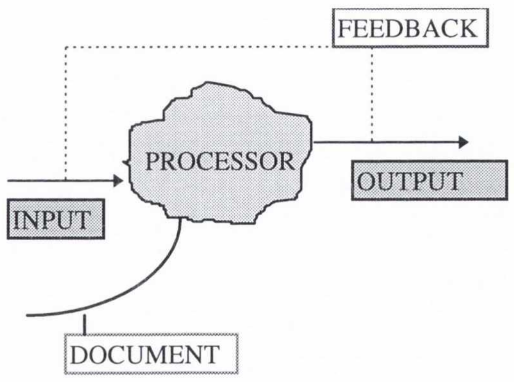


Figure 2.1: Information Retrieval System

Input: A representation for the documents and the queries is obtained by a process known as indexing. Indexing assigns terms (index terms) to documents and queries to describe their content.

Processor: Concerned with the retrieval of information. By examining a search expression in response to a query, it determines which information items should be retrieved and identifies information items that are potentially relevant to the request.

Output: Set of references or documents retrieved by the processor.

Feedback: Not all of the documents retrieved by the processor will be relevant to the user. Hence the Information Retrieval system often provides a mechanism to allow a user to redefine his initial query upon inspecting its sample retrieval for relevant documents in the hope of improving the subsequent retrieval run.

Much of the research and development in Information Retrieval is aimed at improving the effectiveness and efficiency of retrieval. Efficiency is usually measured in terms of computer resources while effectiveness is measured in terms of the number of relevant documents retrieved.

2.2 SEARCH STRATEGIES IN INFORMATION RETRIEVAL

All search strategies used in Information Retrieval systems are based on comparison between the information items and the queries. The distinctions between these search strategies can sometimes be found by looking at the query language, that is the language in which the information need is expressed [*van Rijsbergen `75*]. For example, a query language which allows search statements to be expressed in terms of logical combinations of keywords dictates a Boolean search.

2.2.1 TRADITIONAL SEARCH STRATEGIES - THE BOOLEAN MODEL

For some two decades the Boolean search model has been employed in bibliographic databases as a conventional Information Retrieval model. The Boolean model is designed to retrieve all documents exhibiting the precise combination of keywords included in the query. When two keywords are related by an AND connective, both terms must be present in order to retrieve a particular record; when an OR connective is used, at least one of the query terms must be present to retrieve a particular item [*Salton, Fox, Wu `93*]. These AND and OR operators are implemented by using set intersection and set union respectively. An example follows.

Suppose a user wants to find a book in a library database on archaeology or history by a particular author, this query could be expressed using the Boolean model as follows:

author = Jones and (title = history or title = archaeology)

The order in which the Boolean operators are carried out is vital in the interpretation of the query by the system, and parentheses are usually provided to impose the standard precedence order of Boolean operators. The Boolean model is a powerful one.

- It is possible to specify phrase like constructs in an explicit manner using the AND operator. *"Wind & Instruments"*
- It is possible to specify synonyms explicitly using the OR operator. *"Implements or Tools"*
- A trained searcher can specify a query in very precise detail using these logical operators.

However this model is not without fault.

- It is difficult to formulate any but the simplest of queries using Boolean operators without a fair degree of training.

- Boolean retrieval results in a simple partition of the database - those documents that match the query and those that do not. All of the retrieved records are assumed to be of equal importance to the user. There is no way of ranking the output in order of decreasing probability of relevance.

Although Boolean searching has provided an effective way of accessing machine-readable textual data for many years, its' limitations mean that non-specialists may find great difficulty in carrying out effective searches. Hence much research in Information Retrieval is concerned with overcoming these limitations and tailoring search strategies to the non-specialist [*Willet, Ingwersen `94*]. These new search strategies may be called best-match search strategies.

2.2.2 BEST-MATCH MODEL

The best-match model functions by comparing a set of query terms with the sets of terms corresponding to each of the documents in the database, calculating a measure of similarity between the query and each document based on the terms they have in common and then sorting the documents into order of decreasing similarity with the query. The measure of similarity is calculated using:

1. Term Weighting Scheme which allocates numerical values to each of the index terms in a query or document that reflects relevant importance.
2. Similarity Coefficient which uses these weights to calculate the overall degree of similarity between a document and a query.

The output of the search is a ranked list and those documents appearing at the beginning of the list are the documents the system deems most relevant.

2.2.2.1 Limitations of Best-Match Model

This best-match model, like its Boolean counterpart, has some limitations. It is impossible to specify phrases explicitly using AND and synonyms using OR. More importantly, the query needs to contain several terms if the matching algorithm is to be able to provide a discriminating ranking of the database.

If the query contains only a few terms, the best-match search algorithm will divide the database into only a small number of groups. For a query that contains two terms - there will be just three groups identified by the system - documents that match with one query term, documents that match with both query terms and documents that match with no query term. The small number of groups identified means that the best-match search algorithms will not be able to provide a finely honed ranking for the user's inspection.

2.2.2.2 Elimination of Best-Match Model

Best-match search strategies are not feasible for use with the 1872 Search System for the following reasons:

1. A best-match search strategy takes a set of index terms comprising the query and compares them in turn with the sets of index terms that characterise each of the documents in the database. This requires descriptions of each of the documents to be constructed by choosing sets of index terms that reflect the document's content. This process of characterising documents is impractical with large databases - such as the 1872 database which contains approximately 207,000 book titles. In addition the multitude of domains and the variety of languages present would make the task of characterising the titles highly specialised.
2. As outlined in the limitations of best-match search strategies, the query needs to contain a few terms if effective ranking of documents is to be made. Likewise the terms that characterise each document need to be substantial for a matching algorithm to be employed. In the 1872 database however, the users are searching for book titles and hence tend to use few terms in their search statements. In addition the average length of a title is thirteen words with some titles containing only one word -

examples of which are *"Poems."* and *"Sermons."*. This means that ranking of the titles in order of relevance would be ineffective.

For the reasons cited above, the best-match model was deemed to be inappropriate for use with the 1872 database and so the traditional Boolean model was applied instead. In the development of the 1872 Search System, an attempt was made however to reduce the limitations of the Boolean search model by making it easy for the user to formulate Boolean queries.

2.3 THE MOVE FROM TRADITIONAL TO USER-ORIENTED INFORMATION RETRIEVAL

In 1983 Belkin states that IR research in general "appears to be moving from ad-hoc, technical, mechanism and document oriented views of problems to principled, integrated, interactive and human views of problems" [*Belkin '83*]. This move to a more user-oriented and cognitive form of IR was noted four years later by Saracevic, chief-editor of one of the core IR research journals - *Information Processing & Management.*

For him, this new approach to IR "incorporates and merges into one context the research approaches in IR within information science, on the one hand, with the research approaches on expert systems within artificial intelligence on the other." [*Saracevic `87*].

Traditional IR stresses the importance of the capability for calculation of the various techniques described above and the means to represent information in a controlled, scientific manner. The aim of traditional / classic IR then is the maximisation of retrieval performance by means of refinements of IR techniques within IR systems. User-oriented IR differs from traditional IR in its aim and focus. User-oriented IR focuses on user interaction with the IR system, and knowledge of user's information seeking behaviour. Its aim is maximisation via understanding of user behaviour and information need during retrieval.

In recent years the traditional and the user-oriented IR models are being merged into more complex and interactive solutions. The introduction of user-oriented IR brings with it knowledge about the user and user interaction which leads to the development of a more supportive user interface designed with users rather then the system in mind.

This thesis describes the development of such a user-oriented system which improves users' retrieval of information by providing support and interactive tools to aid in the formulation of search statements. Query expansion techniques, improved subject access and multilingual access are just some of the features that were added to the existing 1872 Search System in an attempt to create a user-oriented environment for the users of the 1872 catalogue. To introduce the user-oriented system, an examination of the existing search system is required.

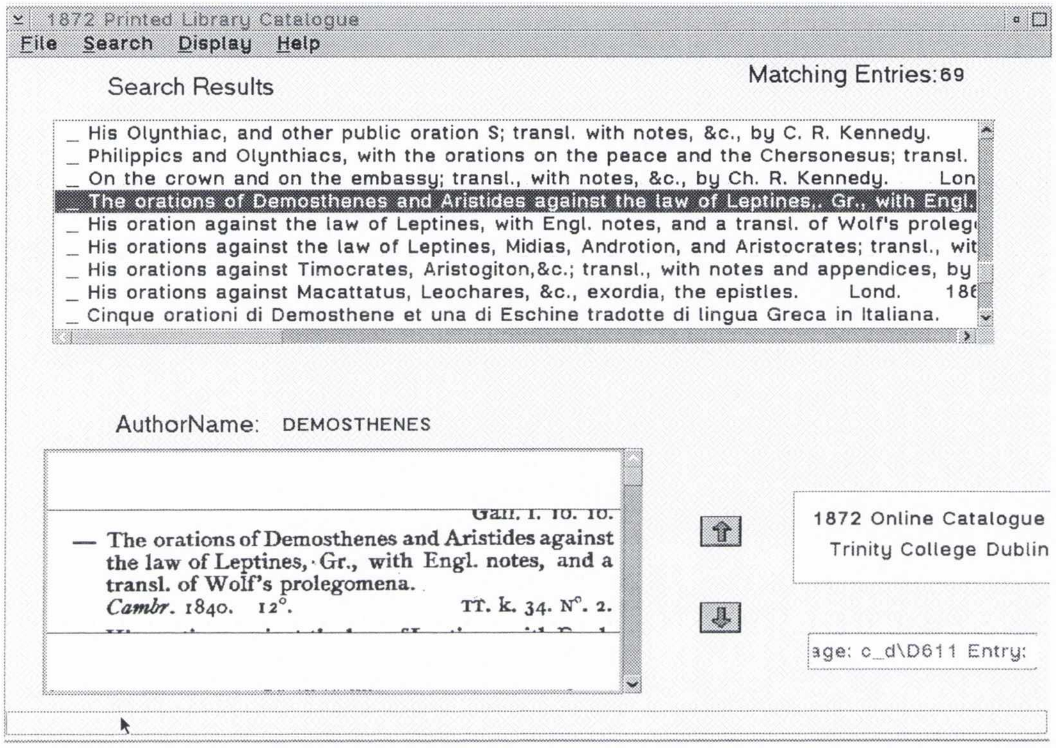
2.4 INFORMATION RETRIEVAL IN THE 1872 LIBRARY DATABASE

2.4.1 THE BOOLEAN MODEL

The 1872 Library Database makes use of the familiar Boolean Model and inherits its' strengths and weaknesses. These strengths allow the librarians of Trinity College Dublin, whose knowledge regarding the contents of the catalogue is substantial, to formulate complex queries across a number of fields - author, title, series, date and place of publication. The weaknesses make it difficult for users to formulate complex queries without considerable experience with the system. The interface designed for the 1872 Catalogue aimed at hiding some of the Boolean Model weaknesses, thereby providing easy access to the database.

2.4.2 THE USER INTERFACE

The interface developed to access the 1872 Catalogue was written in 1993 by Brendan Culligan using a powerful interface builder called GPF [*Culligan `93*]. The interface is shown in Figure 2.2.



Search Results

Matching Entries: 69

```
_ His Olynthiac, and other public oration S; transl. with notes, &c., by C. R. Kennedy.
_ Philippics and Olynthiacs, with the orations on the peace and the Chersonesus; transl.
_ On the crown and on the embassy; transl., with notes, &c., by Ch. R. Kennedy.      Lon
_ The orations of Demosthenes and Aristides against the law of Leptines,. Gr., with Engl.
_ His oration against the law of Leptines, with Engl. notes, and a transl. of Wolf's prolegı
_ His orations against the law of Leptines, Midias, Androtion, and Aristocrates; transl., wit
_ His orations against Timocrates, Aristogiton,&c.; transl., with notes and appendices, by
_ His orations against Macattatus, Leochares, &c., exordia, the epistles.      Lond.     186
_ Cinque orationi di Demosthene et una di Eschine tradotte di lingua Greca in Italiana.
```

AuthorName: DEMOSTHENES

Gall. I. 10. 10.

— The orations of Demosthenes and Aristides against the law of Leptines, · Gr., with Engl. notes, and a transl. of Wolf's prolegomena.
Cambr. 1840. 12°. TT. k. 34. N°. 2.

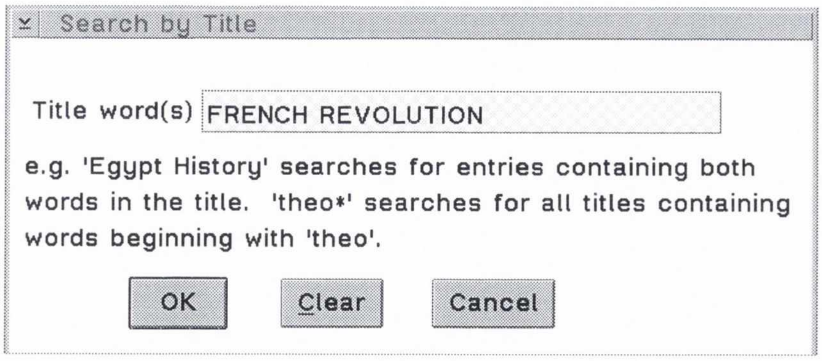
1872 Online Catalogue
Trinity College Dublin

age: c_d\D611 Entry:

Figure 2.2: The 1872 Search System User Interface

The menu at the top of the screen provides the user with the main search options. The user may search by author, title, series or using a combination of these fields. The Boolean Model of query formulation is hidden from the user except in the case of the combination search where the user may specify any number of fields to be searched. By selecting a search on author, title or series - a window appears displaying the appropriate entry field and the user's query is taken to apply to that field only. In the title search window in Figure 2.3, the user has entered two query terms, 'French' and 'Revolution'. The system then evaluates this into a Boolean query as follows:

title = 'French' and title = 'Revolution'


Search by Title

Title word(s) FRENCH REVOLUTION

e.g. 'Egypt History' searches for entries containing both words in the title. 'theo*' searches for all titles containing words beginning with 'theo'.

Figure 2.3: Title Search Window

In this way the user can simply specify which field he wishes to search under and if his query consists of two keywords an implicit AND is assumed. Wild card searches are also available. In the example given in the title search window in Figure 2.3, we see that 'theo*' searches for all words beginning with 'theo' e.g. 'theology', 'theory etc..

2.4.2.1 Interface Builder

GPF (Gui Programming Facility) is a commercial package written by Bernard Clerin to simplify the design of Graphical User Interfaces [*Clerin `91*]. With this facility, the programmer can design an interface in a manner similar to a paint package. The programmer can add any number of windows and controls[1] to the parent window, there being no practical limit to the number of windows and controls that can be added.

Actions may be associated with each of the controls so that they perform some task. To take an example, an action may be associated with a push-button and consequently if that push-button is clicked, a function, say 'print' (defined in a separate file) is called automatically. An application can be created easily in this fashion and since the interface definition is reentrant, the new GUI can be modified and improved as often as the programmer wishes.

[1] A control is a screen object that appears and behaves in a predetermined manner e.g. push-button

2.4.2.2 Interface Features

In the design of this interface, Brendan Culligan used common interface features such as menus, icons and windows. A menu is a simple selection system and is used in this interface to present the various options to the user. The menu is the first part of the interface the user interacts with. The following are the menu options provided with the interface:

FILE	SEARCH	DISPLAY	HELP
Exit	by Author	Entry Text	Index
	by Title		General
	by Series		Using Help
	Combination		Keys
	GotoPage		Product Information

Table 2.A: Menu Options in the User Interface

Windows have also been used in the design of the interface. The window on the top of the interface displays the search results i.e. the first line of all the entries that match the user's query {Figure 2.2}. By clicking on one of the rows in the search results window, the corresponding image of that entry is displayed in the bitmap window - the window at the bottom of the screen. Displaying the actual image of an entry is extremely useful. The image of an entry is perfect - there are no OCR-generated spelling errors unlike in the ASCII version. The entire entry including place of publication and shelf mark is shown and librarian's notes written on the catalogue pages themselves may be viewed. The bitmap window provides the user with complete information with regard to a book title.

Icons are also present in this interface. Icons are easy to use as the pictures usually reflect the action they carry out. For example by clicking on the upward arrow icon in Figure 2.2, the user can view the entry on the catalogue page above the one displayed in the bitmap window and by clicking on the downward arrow, the user can view the entry below the one displayed in the bitmap window.

2.4.3 SAMPLE SEARCHES

The following are some of the searches that are now possible with the 1872 Search System. Prior to the development of this system, each of the book titles retrieved using the search statements listed below, would have been inaccessible if the author name was unknown to the user.

Title = actes and Title = apotres	**(1 hit)**
Title = Irish and Title = parliament	**(33 hits)**
Title = Trinity and Title = College and Location = Dublin	**(27 hits)**
Location - Madrid and FromDate = 1750 and ToDate = 1780	**(16 hits)**
Series = singer tracts	**(756 hits)**
Title = storia and Title = italiana	**(8 hits)**
Title = oorlog and FromDate = 1700 and ToDate = 1800	**(10 hits)**
Location = Cavan	**(1 hit)**
Title = biblia and Title = sacra	**(41 hits)**
Title = coinage and Title = parthians and Location = Cork	**(1 hit)**

2.4.4 THE USER

To develop any user interface, the user type must be taken into account. The user's information need is regarded as an *incompleteness* or *gap* in their current knowledge. Taylor suggests that an information need is formed in three mentally intrinsic steps, ending up in a fourth step - the *compromised need* which is represented as a request to the Information Retrieval system [*Taylor `68*]. It is called a compromised need as the user's expectations and intent are balanced against his verbalization of the need. This results in the request not exactly mirroring the information need.

The complexity of the information need means that retrieving information cannot succeed with research on Information Retrieval techniques alone but will require an understanding on the information need of the user and methods for aiding the user to express his request adequately.

User type also affects the information need - whether a user be an expert, a subject-specialist, an Information Retrieval specialist or a non-specialist [*Willet & Ingwersen '94*]. Some way of bridging the gap between the various user types and the Information Retrieval system is required to aid the user in developing his queries.

2.4.5 IMPROVING ACCESS TO THE 1872 CATALOGUE

Due to the hash coding scheme used in the retrieval engine of the 1872 search system, Brendan Culligan achieved a very fast response time to a query. In addition, the interface he provided also simplified users' access to the database and presented the results to the user in a structured fashion.

However many obstacles remain for the users of this system. Complex queries remain difficult to formulate for any other than expert users. No aid or guidance is provided for the user in the reformulation of a query upon an unsuccessful search. Also the interface was built with only one language in mind i.e. English and so providing interfaces in a number of languages requires major alterations to the code. This means that users who are not anglophones are forced to access the database through English.

When we consider the nature of the catalogue - the fact that it is a multilingual database and hence records may appear in languages other than that of the user plus the fact that the vocabulary used in the catalogue does not always coincide with that of a 'modern' user, the lack of support provided by the system becomes a major obstacle for the user. If the move in Information Retrieval is towards user-oriented systems, then aiding the user in his query formulation and reformulation becomes paramount.

The aim of this thesis is to improve Information Retrieval in the 1872 Library Database by moving away from the traditional IR approach towards the more recent user-oriented approach. This means that user interaction will play a strong role in the improvement of retrieval effectiveness. The following chapters describe a variety of user-oriented tools built for the database. These include automatic and interactive query expansion techniques, expert subject access and multilingual aids. Query expansion improves retrieval by adding synonyms and other relevant terms to users' queries. Users can also improve their search results interactively by 1) browsing through dictionaries and picking out terms for queries, 2) by exploring knowledge built up by users who are experts in certain domains and 3) by examining previous search strategies.

In this way, Information Retrieval can accommodate both users who feel their information need is well-defined and hence only avail of one or two of the facilities provided by the system and users who have ill-defined information needs and need to interact regularly with the system to clarify and improve upon their request to the system. After all, we should design Information Systems to conform to the needs and character of the people using them *[Belkin & Vickery `85]*.

CHAPTER 3 - QUERY EXPANSION

3.1 INTRODUCTION

Since Information Retrieval is concerned with the retrieval of documents from a database in response to a user's query, it is vital that the user's query be well formed or the search will fail. However, users of Information Retrieval systems may not always know what their information need is and consequently their query is generally not an exact representation of their information need. More specifically Information Retrieval systems that use word matching as a basis for retrieval (like the 1872 library catalogue retrieval system) require that users phrase their queries in the vocabulary of the documents present in the system. Even if their information need is clear their query terms may not match those terms present in the database - and the search will fail.

Peat & Willet have remarked that users' queries will typically consist of just a few terms germane to the topic and it is often necessary to add synonyms, variant spellings etc. of the original set of search terms to achieve an effective search [*Peat & Willet `91*]. This process of altering queries is known as query expansion. The aim of query expansion is to improve information retrieval by adding relevant terms to the user's query, deleting irrelevant terms from the user's query and altering terms in the query so that it correctly reflects the information need of the user and the vocabulary and orthography of the database. Query expansion is the topic of this chapter.

3.2 REPRESENTING QUERY EXPANSIONS

Knowledge relating to a given domain is used to expand users' queries. The knowledge used must be represented in a structured fashion so that the system can access it upon receipt of a user's query and alter the query where necessary.

The most common form of architecture used in knowledge-based systems is the production system [*Patterson `90*]. This type of system uses knowledge encoded in the form of production rules or 'if-then' rules. The rule has an antecedent or condition part and a conclusion or action part. E.g.:

IF condition

THEN action

If the condition is true, then the action is taken and the rule is said to have 'fired'.

In the 1872 Search System, query expansion takes place in a variety of domains. Production rules may be applied to those domains that require an expansion to be made to a query should a certain condition be true. These domains comprise

1. expanding queries to cater for OCR errors in the Library Database;
2. expanding queries to cater for the Latin Alphabet used in the 1872 Catalogue;
3. expanding queries to cater for author titles;
4. expanding queries to cater for synonyms used in the place of publication field;
5. expanding queries to cater for the possibility of country of publication;

and are described in the following sections.

3.3 EFFECTS OF OCR-GENERATED TEXT ON QUERY EXPANSION

Since the documents in the 1872 database are OCR-generated, we need to examine how OCR will affect the retrieval system. A number of organizations feel confident enough to use OCR directly with an Information Retrieval system regardless of the uncertain effects the noisy data may have [*Taghva et al `94*].

Research by Taghva and his colleagues showed an insignificant difference in Information Retrieval between an OCR-generated database and its corresponding 99.8% correct version. The database used in Taghva's case was scientific and the corrected version of the databases was achieved using an automatic post processing system in an attempt to correct the OCR-generated errors. Their experiment found that the percentage of documents returned from the OCR-generated database was 97.6% of those returned from the corrected database. This led to their conclusion that the effects of OCR on Information Retrieval are insignificant. However Taghva remarks in another paper that "although the average differences seem insignificant, individual queries can be greatly affected by OCR text" [*Taghva, Borsack, Condit `94*].

What of the effects of OCR-generated text on Information Retrieval in the 1872 Search System? It was felt that with this database the percentage of documents returned from the OCR-generated database would be significantly lower than 97.6%. The reason for this is that the OCR software, written especially for the 1872 database, achieved a success rate of 95.9% which appears low when compared with other OCR products (some vendors claim a 99% accuracy for their OCR products).

Yet this theory could not be tested as no 'corrected' version of the 1872 Library Database exists. Some automatic post-processing was carried out on the database to correct the OCR-generated errors but the majority of the OCR-generated errors (83.43%) were context sensitive and would therefore need to be corrected interactively [*Clarke `93*].

Dictionaries and spelling correctors for the many languages contained in the library database would need to be incorporated for this purpose. Not surprisingly, this interactive correction has not been carried out on the 1872 Library database due to the size of the database (5155 pages) and the number of dictionaries required. Instead, the OCR-generated errors have been tackled from a different angle.

3.4 HIDDEN QUERY EXPANSION

The aim of hidden query expansion is to expand the user's query so that it caters for characteristics specific to the database. In the case of the 1872 Library Database, these characteristics include the fact the database is OCR-generated and will therefore contain OCR-generated spelling errors and also the fact that a large proportion of the book titles appear in Latin and the Latin alphabet contains only 24 letters as 'i' / 'j' and 'u' / 'v' are considered equivalent.

3.4.1 OCR ERROR TYPES

In a previous work, the OCR generated errors of the 1872 Library Database were examined and were found to consist, almost exclusively, of substitution errors (where one of more letters in a word are replaced by an incorrect letter(s)) [*Clarke `93*]. The reason that substitution errors are most common is that every single character is processed during OCR so deletions and insertions of letters are rare. The recognition phase of OCR uses template matching where each character is compared with stored templates and a probability of a match is calculated for that character.

Correct	Recognised As
F	[
fl	fi
fl	fn
H	II
I	1
L	I
m	n1
m	ffi
u	n
Q	O
rt	tt
W	Vi

Table 3.A Common OCR errors

Feature extraction is also used - where features such as corners, straight lines, curves etc. are examined and again a probability of a match is calculated. This suggests why letters, which are physically similar, are misread by OCR and hence why substitution errors occur. Table 3.A lists some of the most common OCR substitution errors.

3.4.2 EXPANDING QUERIES USING OCR ERRORS

So that the user can access all the documents in the database - including those with OCR errors, the user's query is examined for letters that are frequently misrecognised during OCR and if a match is found, the query is expanded to include the OCR version of the term. Table 3.A is used to provide the substitutions. This expansion can be represented using the following production rule:

IF a term contains a string listed in the *Correct* column in Table 3.A

THEN the term is repeated with the string in the *Correct* column substituted by the corresponding string in the *Recognised As* column and an implicit OR is assumed between the query terms.

An example illustrates this process:

Title = 'various' and 'parts'

The user enters two query terms in the title search. The sequence of letters 'rt' in the word 'parts' is present in the OCR substitution table so that query term is repeated with the appropriate substitution and an implicit OR is assumed.

The query becomes:

Title = 'various' and ('parts' or 'patts')

The substitution is hidden from the user and the relevant documents are retrieved. Retrieval of information is greatly improved with this form of expansion as the following results demonstrate.

Query Term	*Before Expansion*	*Hits*	*After Expansion*	*Hits*
tartare	tartares	8	tartares OR tattares	**9**
reflections life	reflections life	0	reflections OR renections AND life	**9**
beaux arts	beaux arts	15	beaux OR beanx AND arts OR atts	**17**
culinaria	culinaria	3	culinaria OR cnlinaria	**5**

Table 3.B: Results of Query Expansion using OCR Errors

3.4.3 EXPANDING QUERIES TO CATER FOR LATIN ALPHABET

The Latin alphabet contains only 24 letters as 'i' / 'j' and 'u' / 'v' are considered equal. Hence in the 1872 library database, 'i' , 'j' and 'u', 'v' are interfiled so that in the alphabetical listing of the authors, 'VOLTAIRE' comes before 'USSERUS'. Needless to say - this can cause huge problems for the searcher of the database.

Instead of confusing the user by instructing him to substitute 'i's for 'j' and 'u's for 'v's in his query should his search fail, this was done automatically in users' queries - again using a set of production rules of the form:

IF a term in the query contains 'j'

THEN the term is repeated with 'j' replaced by 'i' and an implicit 'OR' is assumed between these terms.

In this way the query:

Author = 'Johovah'

becomes

Author = 'Johovah' or 'Iohovah'

Again, the implementation is hidden from the user and the titles retrieved increase dramatically in number.

Query Term	*Before Expansion*	*Hits*	*After Expansion*	*Hits*
evangelicos	evangelicos	15	evangelicos OR euangelicos	**20**
ejusdem	ejusdem	403	ejusdem OR eiusdem	**662**

Table 3.C: Results of Query Expansion using Latin Alphabet Substitutions

The hidden query expansion techniques described above do not mask the condition of the text from the user but they increase the user's confidence in the system by improving its' Information Retrieval effectiveness.

3.5 AUTOMATIC QUERY EXPANSION

Automatic query expansion techniques utilize the text of a user's question as input for techniques to derive a set of search terms that retrieve additional relevant documents [*Spink `94*]. Such query expansion techniques have been developed for the 1872 Search System and are described below.

3.5.1 EXPANDING AUTHOR TITLES

The hash coding scheme employed in the 1872 library search and retrieval system make use of the first four letters of words as a hash key. The complete word in then hashed to a location accessed by the four-letter key. This means that the minimum length of a word in the hash key tables is four letters and hence any word less than four letters in length cannot be used in a search statement. Unfortunately there are many words and abbreviations present in the catalogue that are less than four letters in length. 'War' is an example of an important three-letter word and 'B.C.' an example of an important abbreviation. This problem is accentuated in the author title field where author titles are often used to distinguish between authors of the same name as the following example demonstrates:

Author = 'SWIFT' retrieves

SWIFT, Daniel.
SWIFT, Deane
SWIFT, Godwin, M.A.
SWIFT, Jonathan, D.D., dean of St. Patrick's, Dublin.
SWIFT, Richard Levinge
SWIFT, Theophilus
SWIFT, Thomas

If users were able to access this author title information in their searches, then time spent browsing through irrelevant information could be reduced. The capability of searching under author titles would also enable statistical information on the authors of the catalogue to be carried out. E.g.:

- List all books written by doctors (M.D.) before 1600. (375 titles)
- How many books were written by judges (U.J.D.) (396 titles)
- List all books written by Jesuits (S.J.) on 'science' or 'mathematics'. (17 titles)

In order to be able to search on author titles, the author titles were extracted from each of the author fields in the entries and placed in a new author description field. An automatic expansion program was then run on all the author description fields to expand the abbreviations out to the appropriate descriptions. Table 3.B lists a sample of these expansions and the full table used by the program may be found in Appendix A.

Abbreviation	Expansion	Common Name
A.B.	Artium Baccalaureus	Bachelor of Arts
B.A.	Baccalaureus in Artibus	Bachelor of Arts
B.C.L.	Bachelor of Civil Law	
B.M.	Bachelor of Medicine	
C.J.	Chief Justice	
D.D.	Divinitatis Doctor	Doctor of Divinity
F.C.S.	Fellow - Chemical Society	
F.R.C.S.	Fellow - Royal College of Surgeons	
J.U.D.	Juris Utrisque Doctor	Doctor of Canon and Civil Law
LL.B.	Legum Baccalaureus	Bachelor of Laws
M.A.	Magister in Artibus	Master of Arts
M.P.	Member of Parliament	
O.S.B.	Ordinis Sancti Benedicti	Order of Saint Benedict
Ph.D.	Philosophiae Doctor	Doctor of Philosophy
S.J.	Society of Jesus	Jesuit
S.T.P.	Sanctae Theologiae Professor	Professor of Theology

Table 3.D: Author Description Expansions

With words instead of abbreviations contained in the author description fields, these fields could then be indexed in the same way the author names and titles had been indexed. Each of the words in the author description fields were extracted and hashed to a table. This table is then searched in the retrieval process. A new author search window was created to enable searches on author surname, author first name and author description or a combination of the three. Figure 3.1 shows the original author search window and Figure 3.2 the modified author search window.

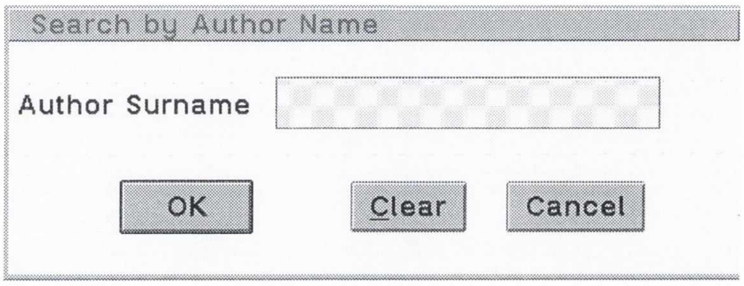


Figure 3.1: Old Author Search Window

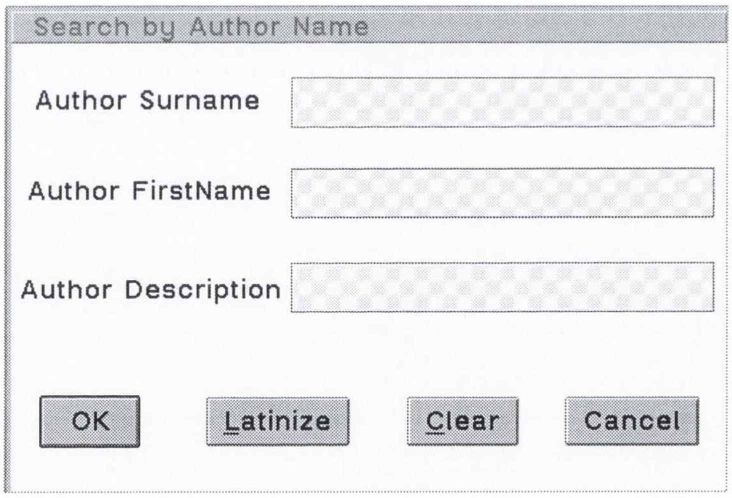


Figure 3.2: New Author Search Window

When a user enters an author description, a production rule is used to check if one of the author title abbreviations in Table 3.D has been entered.

IF term is contained in the author title abbreviation list
THEN term is replaced by corresponding expansion.

The search is then executed. The expanded author description is displayed in the entry field so that the user is aware that an automatic expansion has been carried out.

3.5.2 EXPANDING PLACES OF PUBLICATION

Another area where automatic query expansion is required in the 1872 library database is the place of publication search. The reasons for this are threefold:

- The places of publication in the database often appear in an abbreviated form. E.g.: *Dubl.* for Dublin and *Edinb.* for Edinburgh and users tend naturally to enter the full word in their search statement, thereby eliminating all entries containing the abbreviated form.

- Variant spellings of the place names are present in the database. E.g.: *Leipzig, Lipsiae, Leipsiae* and *Paris, Parrhis* and again the user, by entering in the most common or modern spelling of the word will fail to retrieve all the books published in a particular city/town.

- No countries appear as places of publication in the catalogue. Instead the important cities and towns of each country are contained in the place of publication field. This means that users who do not know the correct place of publication of the book they are searching for will be forced either to make various attempts at the search using the various cities of the country they are interested in or will be unable to conduct a successful search. The following example highlights the problem. Let's say a user wishes to find a book published in Ireland in 1830. The user is unsure as to where exactly the book was published. He might first conduct a search with the place of publication as 'Dublin' but if this fails what is to be tried next? If he knew that the following Irish cities and towns are present in the database - Dublin, Limerick, Waterford, Cork, Castlebar, Westport & Tralee he could repeatedly reformulate his search until all searches have been exhausted or the book is found. But this is a very time consuming process. If a search on country of publication was enabled which expanded the country into the various cities and towns present in the database, then this problem could be overcome.

By automatically expanding queries to deal with the above considerations, users' search results would improve dramatically with little effort, along with their confidence in the system.

3.5.2.1. Synonym Lists

The abbreviations and variant spellings of the place names are expanded in the same way - using synonym lists. The user enters a place of publication as a query term and this query term is compared with a synonym list of abbreviations and a synonym list of variant spellings using production rules. If the query term is found to have a corresponding list of synonyms then these synonyms are implicitly OR'd to the query term and the search is executed. Again this expansion results in a higher percentage of search results {Table 3.E}.

Query	*Before Expansion*	*Hits*	*After Expansion*	*Hits*
geneva	geneva	8	geneva OR geneve OR genevae	**371**
hambourg	hambourg	9	hambourg OR hamburg OR hamb.	**77**
cambridge	cambridge	111	cambridge OR cambr.	**1249**

Table 3.E: Results of Query Expansion using Synonyms

3.5.2.2. Mappings

To enable a search on country of publication, a mapping between countries and their cities was first constructed. Table 3.F displays a sample of the more common mappings that were built through repeated searches of the database.

Ireland	Dublin, Limerick, Waterford, Castlebar, Tralee, Westport, Cavan, Cork.
England	London, Bath, Birmingham, Bridgenorth, Cambridge, Chester, Oxford, Durham, Eton, Exeter, Liverpool, Brighton, Bristol, Manchester, Loughborough, Southampton.
France	Paris, Nantes, Rouen, Lille, Bordeaux, Rennes.
Scotland	Aberdeen, Edinburgh, Dundee, Glasgow.

Table 3.F: Country of Publication Expansions

When the user enters a place of publication, it is first compared with the listed countries using production rules. If a match is found, a search statement is automatically constructed with the corresponding cities OR'd together. This search statement then replaces the initial query and the search is executed. If the place of publication entered by the user is not contained in the countries list, then it is tested for abbreviations and variant spellings as described previously.

Query	*Before Expansion*	*Hits*	*After Expansion*	*Hits*
France	France	0	Paris OR Nantes OR Rouen OR Lille OR Bordeaux OR Rennes	**5362**
Italy	Italy	0	Bologna OR Firenze OR Genoa OR Napoli	**105**
Scotland	Scotland	0	Aberdeen OR Edinburgh OR Dundee OR Glasgow	**3743**

Table 3.G: Results of Query Expansion using Country of Publication

3.6 INTERACTIVE QUERY EXPANSION

Interactive Information Retrieval is an interactive communication process that occurs during the retrieval of information by involving all the major participants in Information Retrieval - i.e. the user, the intermediary (interface) and the Information Retrieval system [*Ingwersen* `92]. It attempts to provide a supportive approach to information retrieval by allowing the user to interact with the system in order to improve search results. Interactive query expansion is one means of providing such support and differs from automatic query expansion in that the user can choose whether or not to avail of the expansion techniques.

The following are the interactive query expansion techniques, common to many Information Retrieval systems that were developed for the 1872 Search System.

3.6.1 PROXIMITY SEARCHING

If the user enters a query consisting of more than one query term AND'd to another for example:

a) *Title = 'French' and 'revolution'*

b) *Title = 'Irish' and 'parliament'*

the system will retrieve all the documents containing both words. However these words may have no relation to each other in the documents retrieved. The following titles are taken from the results of both of the above queries:

a) "*....the revolution of the Roman government...written originally in French.*"

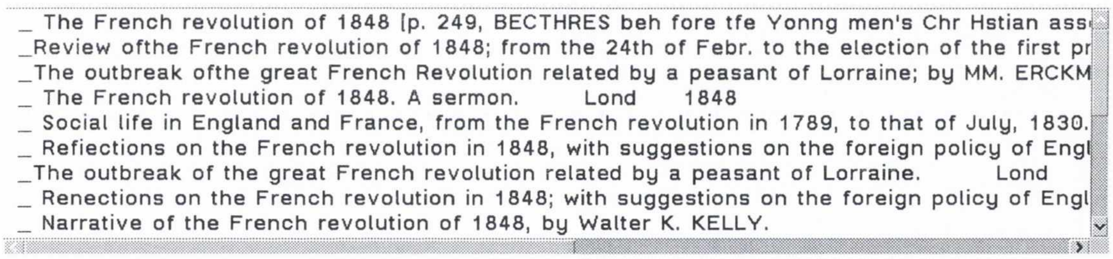
b) *"A letter from an English gentleman to a member of parliament; shewing the hardships....with which the Irish nation has been treated."*

In both the cases the query terms used in the search statement are unrelated in the retrieved documents. These documents are called **false drops**.

The aim of proximity searching is to avoid false drops, thereby decreasing the number of irrelevant documents retrieved. The user can inform the system that he wishes to use proximity searching by placing an indicator (usually '&' or '+') between the query terms of his query. The search statement becomes:

Title = 'French' + 'revolution'.

The system then conducts the search as before by retrieving all documents in the database that contain both query terms. There are 68 titles in the 1872 Catalogue that contain the words 'French' and 'revolution'. However, before the documents are displayed on the screen, a post-processor selects those documents whose query terms occur adjacently. In this case - 19 of the titles contain the word 'French' next to the word 'revolution'. Only these documents are listed as search results. The results of this proximity search are given in Figure 3.3.



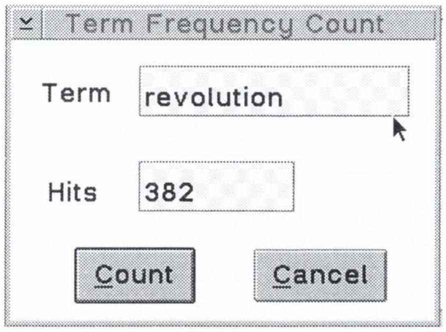
```
_ The French revolution of 1848 [p. 249, BECTHRES beh fore tfe Yonng men's Chr Hstian ass
_Review ofthe French revolution of 1848; from the 24th of Febr. to the election of the first pr
_The outbreak ofthe great French Revolution related by a peasant of Lorraine; by MM. ERCKM
_ The French revolution of 1848. A sermon.        Lond      1848
_ Social life in England and France, from the French revolution in 1789, to that of July, 1830.
_ Refiections on the French revolution in 1848, with suggestions on the foreign policy of Engl
_The outbreak of the great French revolution related by a peasant of Lorraine.         Lond
_ Renections on the French revolution in 1848; with suggestions on the foreign policy of Engl
_ Narrative of the French revolution of 1848, by Walter K. KELLY.
```

Figure 3.3: Proximity Search Results

3.6.2 TERM FREQUENCY INFORMATION

Salton suggests that the frequency of use of a given term may correlate with some indication of the importance of that term in the given subject area [*Salton `83*]. A term frequency tool is provided with the 1872 Search System to enable the user to check the frequency of occurrence of their search terms before executing a search. A low term frequency will indicate to the user that the search will provide only a small number of documents to browse through and that perhaps an expansion of his query is required.

While a high term frequency (anything from 100-10000) will force the user to refine his search if he does not wish to browse through a very large number of titles.



revolution

382

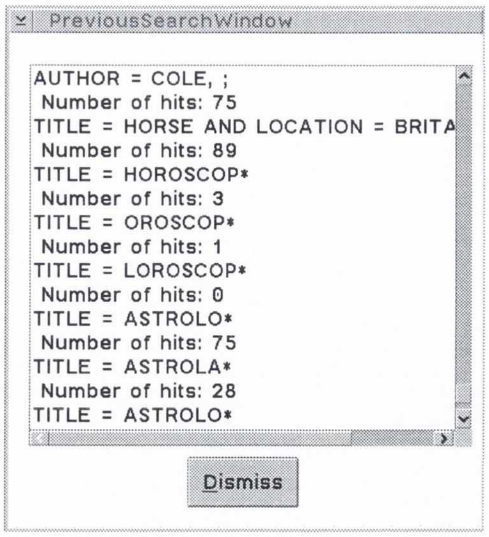
Figure 3.4: Term Frequency Count Window

Lancaster & Fayer claim that the success of the searcher depends on his ability to think of alternative approaches to Information Retrieval [*Lancaster & Fayer `73*]. By providing interactive access to term frequency, the system prompts the user to alter his search statement where required.

3.6.3 PREVIOUS SEARCH RESULTS

Each search carried out by the system is logged in a file along with its' search results. This file can be opened by the user at any stage during the search process to help him with his query formulation. If the user has no search statement in mind, he can just browse through the previous searches to perhaps get ideas for queries or just to get a feel for the contents of the database and the means of retrieving information. If, on the other hand, the user has a query in mind, then he may extrapolate much information from the previous searches executed by the system, such as:

1. the various strategies of formulating searches with his query terms.
2. the fields used in the searches e.g. author, title, series and more specifically the combinations of fields used with his query terms.
3. the different search terms used in conjunction with his search terms.



```
AUTHOR = COLE, ;
 Number of hits: 75
TITLE = HORSE AND LOCATION = BRITA
 Number of hits: 89
TITLE = HOROSCOP*
 Number of hits: 3
TITLE = OROSCOP*
 Number of hits: 1
TITLE = LOROSCOP*
 Number of hits: 0
TITLE = ASTROLO*
 Number of hits: 75
TITLE = ASTROLA*
 Number of hits: 28
TITLE = ASTROLO*
```

Figure 3.5: Previous Search Results Window

Previous search results give some idea of the specificity of search terms but also the search strategies and various knowledge domains used. Such information leads to the construction of effective query statements as users can interactively expand their search by selecting relevant terms from the previous searches [*Salton `83*].

3.7 INFORMATION RETRIEVAL INTERACTION

In an attempt to move towards the user-oriented Information Retrieval approach where interaction with the system is paramount, other interactive features were added to the 1872 Search System to improve retrieval effectiveness. Chapter 4 describes expert subject access to the database where users are encouraged to encode information in a domain that they are familiar with, while browsing through domain knowledge that they are less familiar with. This interactive tool is very important since user's seeking behaviour seems to depend on background knowledge, the subject domain in question and the extent to which their need or underlying problem is developed [*Ingwersen `92*].

Their seeking behaviour and hence success in searching will progress with interactive access to the various domains contained in the catalogue. In this way - interactive Information Retrieval may be seen as a vital and *supportive* process in problem solving and decision making.

CHAPTER 4 -EXPERT SUBJECT ACCESS

4.1 INTRODUCTION

Recent studies have found that subject searches form a large proportion of on-line catalogue use and that users have difficulty doing subject searches [*Khoo & Poo `94*]. Users were found to have problems -

1. matching their terms with those indexed in the on-line catalogue;
2. identifying terms broader or narrower than their topic of interest;
3. increasing search results when too little or nothing is retrieved;
4. reducing search results when too much is retrieved.

The Council on Library Resources study carried out by Mathews [*Mathews et al* `88] found that the reason for this was that the user's query represented an inadequate or incoherent state of knowledge, while the database represented a coherent state of knowledge. So the users' lack of knowledge concerning the contents of the database and the search strategies to use, greatly affect their performance in searching the system. Knowledge becomes the key, therefore, to intelligent subject access of the database.

Knowledge is continuously increased as a result of exposure to new perceptions, facts and situations imprinted in the mind and manipulated by the reasoning ability [*Beerel `87*]. Hence the greater the exposure to new concepts and new areas of knowledge, the better the reasoning ability and the more astute the ability to accumulate and apply knowledge. Therefore by exposing the user of the database to new concepts and areas of knowledge and by facilitating the formulation of a search, their expertise in accessing information in the database will be improved.

In this chapter, *knowledge trees* are used to expose the user to knowledge in various areas of specialization, thereby enhancing their ability to acquire and use knowledge across diverse domains. Because the domains of interest in a library catalogue are so numerous and often completely unrelated it was decided to allow users who specialize in specific domains to build up their own expert trees of knowledge. A domain independent knowledge tree construction tool is developed to allow users to achieve this by structuring their knowledge in a predefined way. Subsequent users may then be able to access 'expert' information in order to improve their search performance if required.

4.2 EXPERT SYSTEMS

The knowledge tree construction tool enables experts to represent their knowledge in a structured way and provides easy access to this knowledge for users of the 1872 Catalogue. As with expert systems, knowledge acquisition and representation become the most vital aspects of the knowledge trees and contribute to acceptance of the knowledge tree tool by the user. Manipulating knowledge is not a trivial task however, and the obstacles encountered in doing so are similar to those found in expert systems. The following sections describe the process of knowledge acquisition and knowledge representation in expert systems which may be applied directly to the knowledge tree construction tool. First we will look at expert systems and their advantages over humans.

An expert system is a computer based system that can perform some task that requires expertise [*Vadera `89*]. Knowledge obtained from experts is entered into the system in a coded form. This knowledge is kept separate from the rest of the system and is called the 'knowledge base'. The knowledge in the knowledge base is represented with symbols employing techniques such as frames, logic and semantic networks. These are natural forms of representation and are hence easy to modify. Expert systems aim to emulate human experts' problem solving techniques in a narrow domain by using the knowledge contained in their knowledge base to offer advice on request.

4.2.1 HUMAN EXPERTS VERSUS EXPERT SYSTEMS

Extracting information from an expert is a very time consuming task and it is common practice to employ a knowledge engineer or intermediary to elicit the knowledge from the expert. Weiss & Kulikowski suggest that this is the case because experts often lack the skills necessary to organize and structure their knowledge [*Weiss & Kulikowski `84*].

Unfortunately experts are expensive, scarce, inconsistent, busy, mortal and are usually expert in only one specific domain [*Beerel `87*]. Expert systems on the other hand can overcome some of these problems.

1. They can preserve valuable knowledge over an infinite amount of time.
2. They can be made available 24 hours a day.
3. They can be applied to various different applications without changing the software.
4. They are consistent in their knowledge representation.
5. They can improve the performance of non-experts.

The above properties apply also to the knowledge tree construction tool. However, as with expert systems, the knowledge trees cannot be built without experts to provide the knowledge and so some kind of compromise must be reached.

4.2.2 THE EXPERT SYSTEM BOTTLENECK

It is said that expert system technology is limited by the expert system bottleneck of knowledge acquisition [*Tuhrim et al `88*]. A knowledge engineer is required to extract the expert's knowledge and transfer it into appropriate data structures in the knowledge base so that it can be processed by the system {Figure 4.1 (A)}. To encode an expert's knowledge, the knowledge engineer needs to know:

- what concepts exist in the expert's domain.
- how important these concepts are and the relationships between them.
- facts and heuristics about the expert's domain.
- classificatory knowledge which enables the expert to distinguish between similar items.

The techniques the knowledge engineer uses in the knowledge acquisition process include

- Interviewing (which is particularly useful for acquiring basic knowledge about the domain),
- Protocol Analysis (recording expert's step by step information processing by asking him to think aloud).
- Observation (of the expert at work)
- Multidimensional Techniques (where the expert attempts to form a map of his specialized domain. This is often achieved by sorting cards of 'concepts' into groups according to certain criteria. These criteria are then noted by the knowledge engineer).

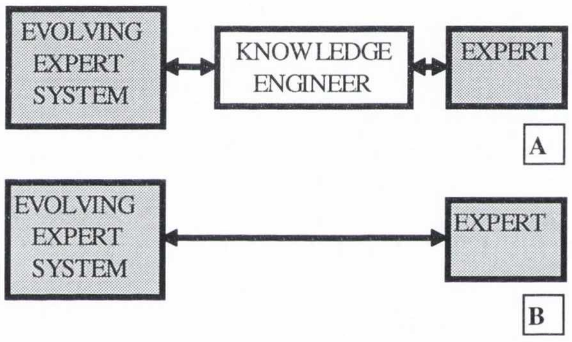


Figure 4.1: Knowledge Acquisition

With all this work in mind - it is not surprising that Sowizral claims that one of the most difficult tasks facing expert system developers is knowledge acquisition [*Sowizral `85*]. Yet there are ways of overcoming the vast workload of the knowledge engineer in the knowledge acquisition phase of the expert system development. One such method would be to allow the expert become the knowledge engineer {Figure 4.1 (B)}. Recent efforts have been made to develop tools facilitating direct construction of the expert system by the non-computer scientist.

The knowledge tree construction tool follows this trend by enabling expert users to build knowledge trees themselves without the involvement of a knowledge engineer. Some form of representing the knowledge in a structured format must be found so that expert users can express their knowledge without difficulty and so the system can process and manipulate the knowledge effectively and consistently in response to a user's request.

4.3 KNOWLEDGE REPRESENTATION

Knowledge representation is concerned with how the knowledge is organized and represented in the knowledge base. Handling the knowledge is one of the most intricate parts of system development. The power and the success of the system depends on how this is carried out [*Beerel `87*]. The most common methods of representing knowledge are logic, production rules, frames and semantic networks. Choosing which technique to use for knowledge representation is not a trivial task. A technique is needed that will be suitable for communicating with experts in specialized fields and will offer a format that can be applied universally. In short we need a representation that is right for the domain and right for the task [*Shashtri `88*].

4.3.1 CHOOSING A KNOWLEDGE REPRESENTATION TECHNIQUE

What criteria need to be taken into account in the selection of an appropriate representation tool?

1. The tool should have metaphysical adequacy - there should be no contradiction between the facts we wish to represent and our representation of them.
2. It must have epistemic adequacy - we should have the ability to express the facts we wish to express with the representation tool.
3. The tool must be heuristically adequate - so that it can express the reasoning behind its problem solving.
4. It must be uniform - all different types of knowledge must be represented in the same way, and finally
5. The tool must have computational tractability - it must be able to be easily manipulated within the system.

4.3.2 KNOWLEDGE REPRESENTATION TECHNIQUES

There are three major ways of representing knowledge in an expert system - predicate calculus, production rules and structured objects which include frames and semantic networks.

Predicate Calculus:

Predicate calculus comes from logic and is a powerful means of transforming natural language into a formal representation. Stylized patterns are developed called 'predicates' which reduce the number of ways to say the same thing. A predicate asserts a fact about one or more objects but has to be unambiguous. *Human(George)* is an example. Often more than one predicate is required to fully express an English sentence. Search statements can also be expressed in this way:

author = 'Swift' and title = 'travels' and location = 'Dublin'

translates to:

Author(Swift) & Title(travels) & Location(Dublin)

Predicate calculus, like all logic languages, depends on the ability to draw inferences in order to derive new facts from those that are given [*Alberico & Micco `90*]. However the power of predicate calculus is not required in the representation of knowledge in the subject trees as no inferences about the facts presented in the knowledge trees are called upon during the search process.

Production Rules:

Production rules represent knowledge in English like conditional statements. The rules describe condition-dependent actions of the form

IF condition, THEN Action.

The rule is applicable if the condition or antecedent is true. This is the most common form of knowledge representation because the rules are easy to express and to understand. This form of knowledge representation has already been used in the 1872 search System to represent query expansion rules, an example being:

IF author = S.J.

THEN author = Jesuit

However the knowledge in the subject trees cannot be represented in this fashion as there is no action to take if some condition is true. Even if production rules could be used, a new rule would have to be created for each new subject added to the tree. This does not provide the flexibility required for the requisite addition and deletion of subjects. Instead a representation is required which represents simple facts about particular subject areas and which is easily manipulated.

Frames:

Frames are predefined data structures that contain objects or situations broken down into their constituent parts {Figure 4.2}.

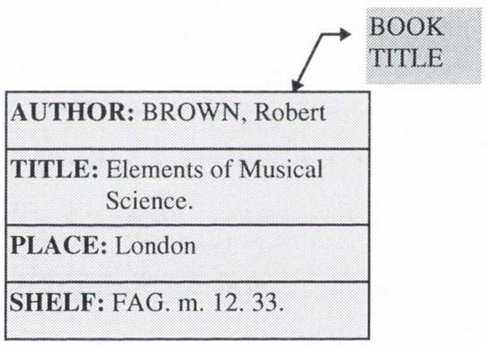


Figure 4.2: Example of a Frame

The parts, sometimes referred to as attributes, are held in 'slots' within each frame. Slots may contain a variety of information such as default values, rules and pointers to other frames. The pointers mean that frames can be linked together to form a hierarchy. The reasoning process for frames involves trying to fill in the slots and selecting the most likely frames that will result in a conclusion.

Frames could be used to represent knowledge in the subject trees in the 1872 Search System as the pointers would enable a hierarchy of subjects to be built while the slots could be used to store the subject name and its' corresponding number of hits {Figure 4.3}.

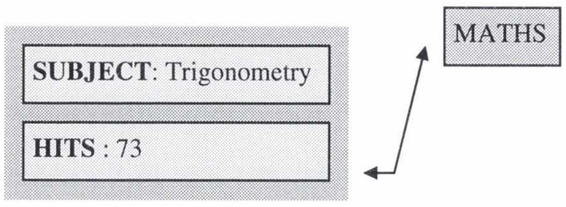


Figure 4.3: Example of a Frame Representing a Subject

<u>Semantic Networks:</u>

Semantic Networks are graphical representations of concepts and relationships in a particular domain. The nodes in the network correspond to concepts and the arcs or links to relationships between these concepts. The concepts or nodes in the network are atomic in the sense that they are never subdivided. Many different types of relationships can be represented in a semantic network - 'is_a' (which means something is an instance of something else), part-of (something is part of something else) and many others. There is a semantic sense called 'ordinality' belonging to the relationships meaning that they can only be applied in a particular direction.

E.g. *Leg (part-of) Human ≠ Human (part-of) Leg*

Algebra (element of) Maths ≠ Maths (element of) Algebra

The relationships between concepts may also be used to create inheritance hierarchies. In these cases concepts or objects can inherit properties from other concepts.

Semantic networks like frames enable the representation of a hierarchical structure like the subject trees. The semantic links would also make it possible to represent relationships between subjects. Adding new nodes to the network and deleting nodes from the network is a simple process which makes semantic networks an ideal candidate for the representation of knowledge in the subject trees.

4.4 THE 1872 CATALOGUE SUBJECT TREES

The 1872 Catalogue subject tree construction tool aims at improving subject access by exposing users to new and related information. This tool is developed to provide better support for the conceptual moves made by the users. They may use the system to scout the subject area they are interested in and can then modify their search strategy to retrieve better results by narrowing a search that is too broad and broadening one that is too narrow. They can also use the system to examine the vocabulary that exists in the database that they would otherwise have missed. This is an important option as users feel that on-line aids for finding, browsing and selecting controlled vocabulary are imperative [*Markey & Vizine_Goetz `88*]. They are especially unfamiliar with the controlled vocabulary used in this database - largely because the vocabulary employed from the fourteenth to the nineteenth century often differs greatly from today's vocabulary.

4.4.1 KNOWLEDGE ACQUISITION

In an attempt to avoid the knowledge acquisition bottleneck described earlier, the subject tree construction tool employed in the 1872 catalogue database enables experts to author their own knowledge trees. In this way knowledge on a number of diverse fields of interest can be made available to the user as an aid to query formulation.
To facilitate the development of this kind of system the subjects will need to be classified using a tree type structure {Figure 4.4}. Moving up the tree will mean broadening the search and moving down the tree - refining the search.

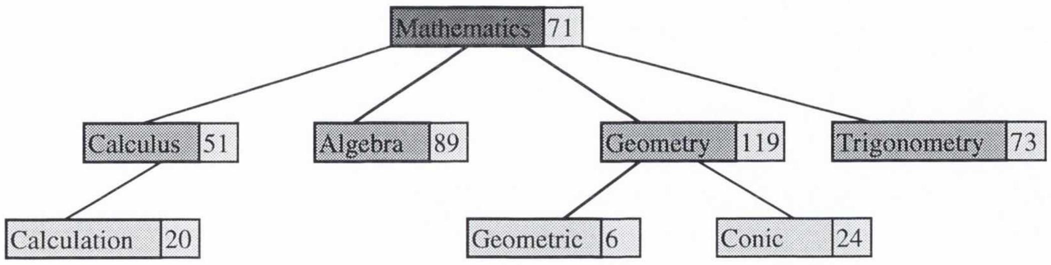
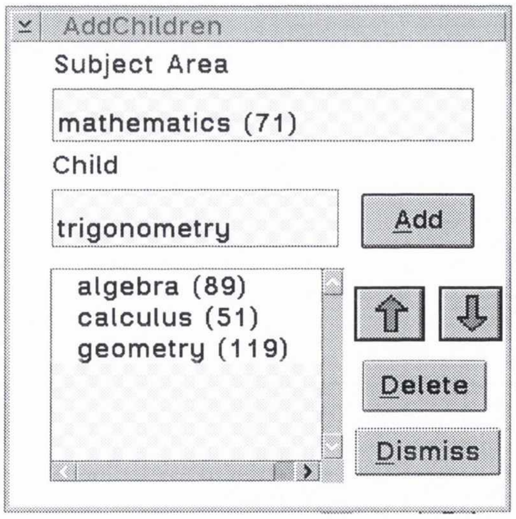


Figure 4.4: Tree Structure of Subjects

This is a simple form of representation and was chosen because the tree structure is a common one and hence thought easy to use for the experts building the subject information. The expert must first structure his knowledge in a hierarchical form. Since the knowledge he builds is aimed at assisting users searching for books in that particular area, he can enter any information he deems relevant - subject headings, truncated words (for wild card searches), relevant authors, common places of publication for that particular subject area etc.. He may then 'create' a subject area on the system and may fill his knowledge tree by adding children to the main subject heading (root node). As each child subject is added to the tree, the system automatically determines how many titles in the catalogue contains that subject and the resulting number of hits is appended to the subject in the tree. Figure 4.5 shows the window designed for subject tree development. In this window we see that the expert's subject heading is 'Mathematics' and that he has added a number of children to this heading. Children may be added to these children by selecting one of the existing children, for example 'Geometry', and clicking on the downward arrow. The main subject heading is then changed from 'Mathematics' to 'Geometry' and children may then be added to 'Geometry'.



mathematics (71)

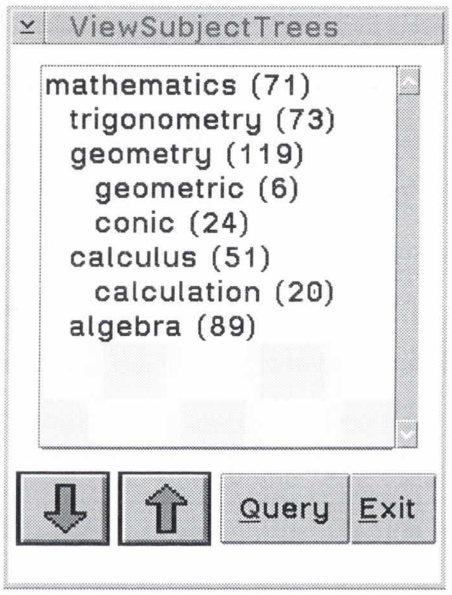
trigonometry

algebra (89)
calculus (51)
geometry (119)

Figure 4.5: Building a Subject Tree

At present there is no limit on the number of levels the expert can add to his tree. The knowledge tree can be edited at a later stage using the same window by adding relevant sub-headings where required and by deleting irrelevant headings.

Upon completion the tree may be viewed in the form shown in Figure 4.6. Initially only the subject headings of the experts' trees are displayed in this window but by selecting one of the headings and clicking on the downward arrow, a subject tree appears. To return to the main subject heading, the user must click on the upward arrow.



```
mathematics (71)
  trigonometry (73)
  geometry (119)
    geometric (6)
    conic (24)
  calculus (51)
    calculation (20)
  algebra (89)
```

Figure 4.6: Viewing the Subject Tree

4.4.2 KNOWLEDGE REPRESENTATION

The existence of semantic networks is implied wherever information is conveyed in node-arc graphical form, where those nodes and arcs are assigned meanings and the topology of the graph is significant to those meanings [*Griffith `82*]. It is not surprising that this technique of knowledge representation was chosen for the subject tree construction tool of the 1872 catalogue, where the subject trees adhere strictly to this form. The subject headings and sub-headings are the nodes with the very general relationship HAS_MEMBER between them.

Mathematics	HAS_MEMBER	Algebra
Geometry	HAS_MEMBER	Geometric

The nodes themselves contain not only the subject heading but also the number of titles in the database that contain that word. Each node therefore is represented using a frame with two slots - one containing the name and the other containing the hits.

4.4.2.1 Implementation

The semantic network of knowledge is implemented using a powerful programming tool called the *Linked List.* Knuth claims that the introduction of links to other elements of data is an extremely important idea in computer programming, and provides the key to the representation of complex structures [*Knuth `81*]. In a linked list - each node/concept points to at least one other node, whether that node be at the same level or at an inferior level in the network. The power of linked lists lie in the easy addition and deletion of nodes. Because the subject trees in the 1872 Catalogue may contain *n* levels, each node in the network will point to the next node in that level and a node in the next level (if one exists). Figure 4.7 shows how the linked list is used in the implementation of the mathematics tree described above.

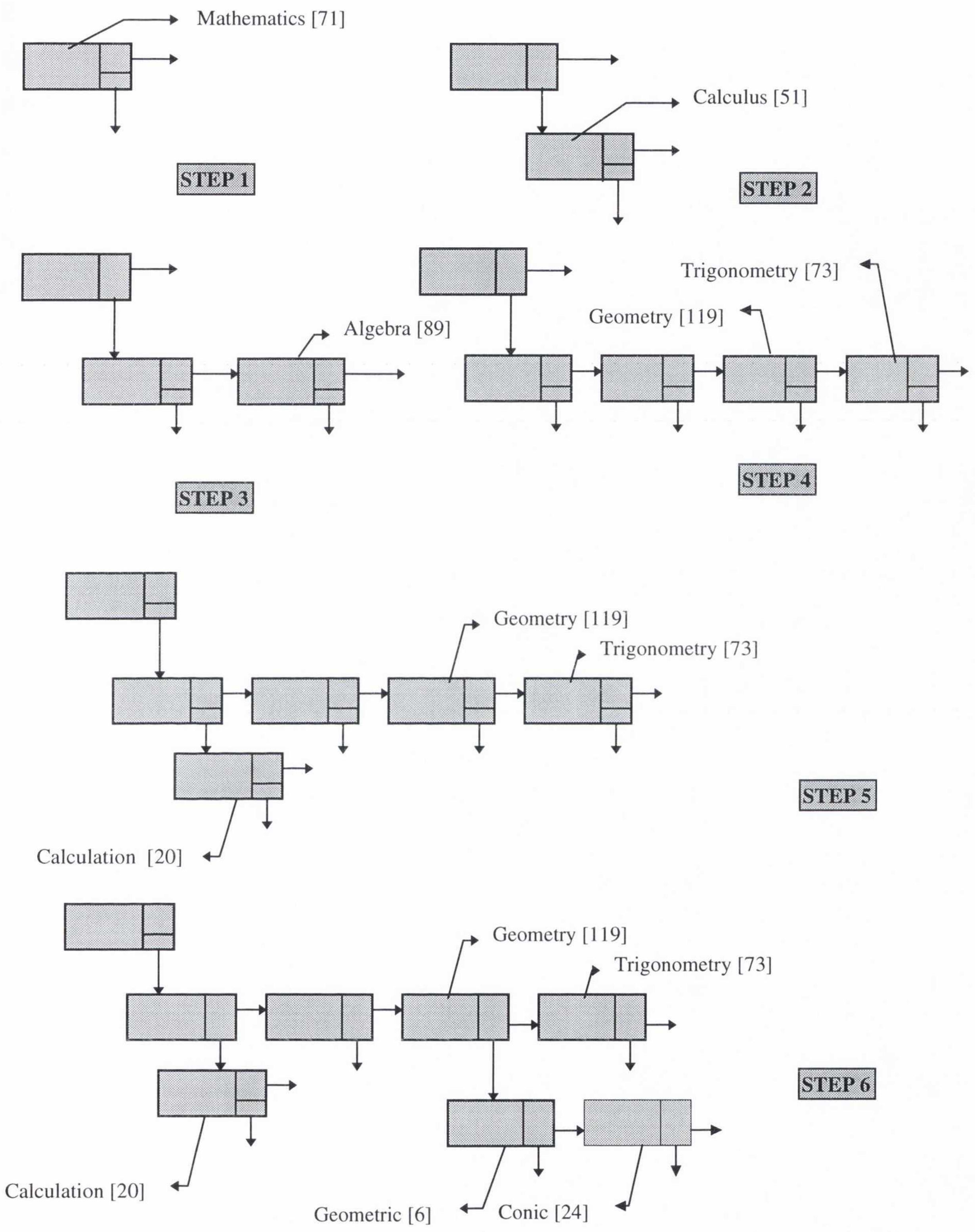


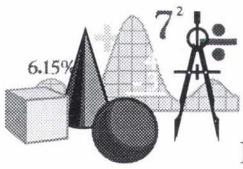
Figure 4.7: Implementation of Semantic Network

When the expert creates a subject area - the root node ('mathematics') of the linked list is created. Each node has two pointers - one sibling pointer, pointing to the next node and one child pointer, pointing to the next node at the next level. In *Step 1* both of these pointers are null. When the expert adds a subheading ('calculus') to the list, a new node is created and the child pointer of the root node points to it (*Step 2*). As the remaining sub-headings are added, new nodes are created in turn. The child pointer of the root node is set to point to the new node ('algebra') and the new node's sibling pointer set to point to the previous node ('calculus'). *Step 3 & Step 4* illustrate this process, with the child pointers of the new node being null.

In order to move down a level, the expert may choose one of the sub-headings. This sub-heading ('calculus') then becomes the root node and the above procedure may be repeated. *Step 5* shows the concept 'calculation' being added to its parent node 'calculus'. *Step 6* shows the addition of new nodes to the concept 'geometry'.

The expert may view his work upon completion as illustrated in Figure 4.6.

4.4.3 USING THE KNOWLEDGE TREES TO SEARCH THE CATALOGUE



If the user is looking for books on a particular subject, he may search the available subject headings for a heading related to his subject of interest. So if he is looking for books on 'geometry' he may browse through the subject areas listed (e.g. 'history', 'geography', 'mathematics', 'medicine') and may position himself in the MATHEMATICS knowledge base (if one exists), and then be given access to a subset of subjects - calculus, algebra, geometry, etc..

By selecting an appropriate subset, geometry in this case, he can again find a set of subsets or a set of terms related to his area of interest that occur in the catalogue and their frequency of occurrence. E.g. 'Geometric' 6 hits, 'Conic' 24 hits. Not only will this allow for a better understanding of the contents of the catalogue, but it will guide the user through information related to their subject of interest. In her design model for subject access to an on-line catalogue, Bates emphasises the importance of user orientation claiming that it enables the user to get a feel both for interacting with the system and for the intellectual world of the system through exploration of vocabulary and relationships between terms [*Bates `86*].

Should the user be a little unsure of the subject heading to search with regard to his area of interest - he may use the 'FIND' option in the system which, when you enter your area of interest, will return the subject heading under which your subject appears. The user is informed if his subject area is not present in any of the knowledge trees built by the experts. Figure 4.8 illustrates the use of the 'find' option. Here the user wishes to find the subject heading for 'geometry'.

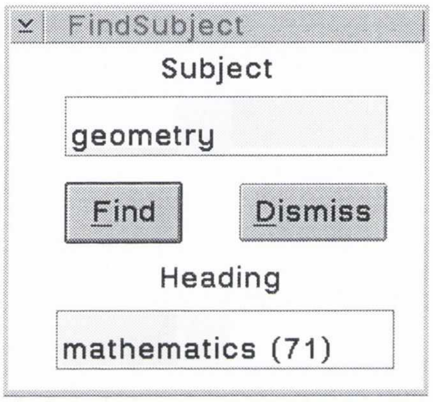


Figure 4.8: Find Subject Window

4.4.3.1 Automatic Search Formulation

Hildreth among others noted that Boolean logic appears to be one of the most difficult aspects of information retrieval [*Hildreth `83*]. For this reason, an automatic search formulation option was developed which constructs queries automatically from the subject trees the user accesses. This option is best illustrated with an example.

If the user is viewing the 'mathematics' subject tree, as in Figure 4.6, and decides to search the database, he has only to select an element in the tree and press the 'query' button in the window. This forces the system to automatically produce a query containing the element selected by the user and the children of that element (if it has any children).

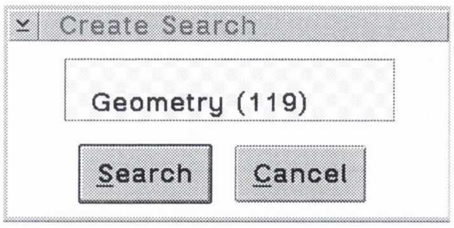


Figure 4.9: Automatic Search Formulation

Another window appears to receive confirmation from the user as to whether to execute the search or not {Figure 4.9}. If, to take an example, the user selects the element 'geometry' in the 'mathematics' tree, then a search for all the books containing the word 'geometry' or 'geometric' or 'conic' is executed by the system and 145 entries are output to the screen. In this way, not only is the obstacle of creating Boolean searches avoided by the user but users may search on a number of topics with just one touch of a button.

The subject tree construction tool described in this chapter has provided a unique way of improving user's subject access to the 1872 Catalogue. Users are encouraged to share their knowledge in a way that will benefit both other users and the system itself. It is hoped that over time, a comprehensive subject index will be built using these trees so that users may improve their retrieval of information by referring to the subject index present in the 1872 Search System.

CHAPTER 5 MULTILINGUAL ACCESS

5.1 INTRODUCTION

Multilingual access is a vital step in the improvement of users' retrieval of information in the 1872 Search System. Due to the nature of the catalogue - it being itself multilingual, users searching the system will lose information if they are unable to retrieve book titles in languages other than their own.
There are two aspects of multilingual access that can be addressed for use with the 1872 Search System. The first concerns the provision of multilingual interfaces which would allow users to query the system through menus and screens in their own language. The current system provides an English interface only. By translating the words in the menus, information messages and error messages, a number of multilingual interfaces could be developed with ease for the system. However the nature of the interface builder requires that the system be developed from the interface. Toggling between a number of languages in a uniform interface is not an option provided with GPF. The entire system would have to be replicated for each new interface designed. For this reason, this aspect of multilingual access was not pursued.

The second and more challenging aspect of multilingual access lies in the provision of a system which would allow users to access titles in a number of different languages. Since computers and computer systems are being used by people all over the world, in developing software we cannot assume that users will be native speakers of any one language [Yazdani `93]. There are different levels of this problem but the most basic lies in the lexical level which is concerned with the words used to interact with the system. The problem of multilingual access arises in the context of the 1872 Catalogue where A) author names are listed in languages other than that of the author, causing confusion among users, and B) where entries can appear in any one of at least fourteen languages. It is often the case that a book written in one language is listed in a different language and since the author names are predominantly Latin - it often occurs that an Italian or English author is listed under the Latin translation of his / her name. These aspects of the catalogue make the need for multilingual access essential if the user is to be satisfied with his retrieval of information. This chapter reviews language change throughout the ages in order to capture the differences and similarities between the most common languages of the 1872 Catalogue - i.e. those in the Romance and Germanic language families. Understanding these differences and similarities leads to the provision of effective multilingual tools to aid the user search the catalogue.

Since the search system is keyword based - the user can search for words in the language(s) he is familiar with but will lose information contained in entries written in other languages. By providing a search by language of title, the user could search for entries in any of the languages contained in the 1872 Catalogue thereby reducing the loss of information incurred in a monolingual search. This form of multilingual access is described in the next chapter.

5.2 THE ORIGIN OF LANGUAGE

For as far back as we can trace in history, man has always spoken many different languages. If, at one time, there was a single language from which all other languages subsequently descended, there is no hard linguistic evidence to prove such a fact [*Katzner `90*]. However, inquiries into the existence of one proto-language began as far back as the seventh century when an Egyptian pharaoh named *Psamtik* arranged for two newborn babies to be reared in isolation until their first words could be recorded. Their first utterance was 'bekos' which turned out to be the word for bread in Phyrgian (a language of Antolia).

This led to the pharaoh's conclusion that Phyrgian was the original language of the earth [*Renfrew `94*]. Investigations into the origin of language continued up until the nineteenth century when scholars exhaustively examined many contemporary languages in the hope that some common elements would point to a primeval source. Today's linguists realize that a clear picture of what happened perhaps a million years ago cannot be obtained and that the study of language will be confined to the more recent historical period.

5.3 LANGUAGE CHANGE

Ferdinand de Saussure - the famous Swiss linguist - has made the claim that "time changes all things: there is no reason why language should escape the universal law" [*Saussure `59*], and this claim is not unfounded. Languages have been changing and developing since the beginning of time. Research in historical linguistics is concerned with the factors that cause this. Anyone who attempts to study historical linguistics must be aware of the multiplicity of factors involved [*Aitchison `91*]. Language change is triggered by a number of sociological and linguistic factors but may also be affected by demographic movements such as:

1. **Initial Migration:** Humans spread from Africa to the rest of the world beginning about 100,000 years ago.
2. **Farming Dispersal:** Populations expanded as farming developed in several places. The original farmer's language spread and differentiated to form language families.
3. **Dominance:** Incoming minorities conquered populations and imposed their language upon them.

Sociological language change occurs when speakers alter the way they speak in an attempt to imitate what they perceive to be a more prestigious variety of speech. The history of English provides numerous examples of this - one being the loss of postvocalic 'r' resulting in the word 'far' being pronounced as [fa]. Need is another form of sociolinguistic change. Languages can alter as the need of it's users alter. Unneeded words are dropped from a language and new technical terms are added to a language. These sociological changes in languages turn out not to be the 'real' causes of language change however, but simply accelerating agents which utilize and encourage

trends already existing in languages [*Aitchison `91*]. The 'real' changes are found within the languages themselves.
Linguistic changes imply changes within the language itself. Phonetic, morphological, lexical and semantic changes have been noted in a number of languages.

5.3.1 PHONETIC CHANGES are sound changes that occur in languages. Articulation is often made easier by modifying a sound so that it is more like or unlike its neighbouring sounds. *Assimilation* is the modification of a particular segment to make it more like a neighbouring sound and is one of the most frequent sources of sound change.

Old Italian	**Italian**		**-** *Assimilation*
/okto/	/otto/	'eight'	/kt/ → /tt/
/nokto/	/notto/	'night'	

In Italian the 'k' was converted to 't' under the influence of the neighbouring /t/, making the above Italian words easier to articulate.

Other phonetically conditioned changes include *dissimilation* which is the modification of a segment to make it less like its neighbours.

Latin	**Spanish**		**-** *Dissimilation*
/anma/	/alma/	'soul'	/nm/ → /lm/

The /nm/ in Latin is changed to /lm/ in Spanish to avoid consecutive nasal consonants.

Epenthesis or segment addition serves to break up a sequence of sounds that would otherwise be difficult to pronounce.

Old English	**English**		- *Epenthesis*
/breml/	/brembl/	'bramble'	/ml/ →/mbl/

Another phonetic change - *lenition* or weakening, weakens a consonant under the influence of a vowel.

Latin	**Portuguese**		*- Lenition*
cippum	cepo	'stump'	/pp/ → /p/

5.3.2 MORPHOLOGICAL CHANGES are changes which affect the structure of words in a language. The most widespread morphological changes involve the *loss and addition of affixes*. English has borrowed its most widely used suffixes from French. Many French words containing the suffix '**ment**' made their way into the language e.g. 'commencement'. This suffix then became established in the English language and was used with words that were not of French origin e.g. 'merriment'.

Analogy is another morphological change. It involves the inference that if two elements are alike in some respects, that they should be alike in others as well. The development of the plural ending '**s**' in English was influenced by analogy. The following examples show the old English plural form of stone - 'stanas' changing from the 'as' ending to the more familiar 's' ending.

Old English	**English**	*- Analogy*
stan (sg.) stanas (pl.)	stone (sg.) stones (pl.)	
hund (sg.) hundas (pl.)	hound (sg.) hounds (pl.)	

5.3.3 LEXICAL CHANGES occur in the lexicon or words of a language. New words are often formed in response to the need for new lexical items. The word 'smog' was developed in English as a blending of two words - 'smoke' and 'fog'. Besides the formation of new words, borrowing words from different languages is a frequent lexical change.

The almost universal use of the American idiom 'O.K.' is an example of borrowing. Languages can borrow words from non-dominant languages present in the same area for example the word 'moose' in English is borrowed from the Amerindian languages. Historical events can also lead to the acquisition of foreign words. The Norman conquest of England in 1066 is an example of this. The French-speaking conquerors gradually learned the English language but retained many of their French terms. Table 5.A. shows some of the French loan words in English which pertain to areas of officialdom [*O'Grady et al '89*].

Government	tax, revenue, government, parliament, duke, authority
Religion	prayer, sermon, religion, chaplain
Judiciary	judge, defendant, jury, evidence, jail, crime
Science	medicine, physician, science
Culture	art, sculpture, satin, ruby
Warfare	army, navy, battle, soldier, enemy, captain

Table 5.A: French Borrowed Words in English

5.3.4 SEMANTIC CHANGES Changes in word meaning occur continually in all languages however word meanings do not jump from one meaning to another completely unrelated one. *Semantic Broadening* is the process where meanings of words become broader while *Semantic Narrowing* has the opposite effect. Another semantic change where a word loses its former meaning and takes on a new but related meaning is called a *Semantic Shift.* Examples of these semantic changes are given below.

Word	**Old Meaning**	**New Meaning**	
bird	small fowl	any winged creature	- *Broadening*
hound	any dog	a hunting breed	- *Narrowing*
immoral	not customary	unethical	- *Shift*

5.4 LANGUAGE FAMILIES

For more than 200 years, linguists have recognised that some languages have such similarities in vocabulary, grammar, the formation of words and the use of sounds that they must stem from a common ancestor. These ancestral alliances are termed language families [*Renfrew `94*].

FAMILY	*SUBGROUP*	*BRANCH*	*MAJOR LANGUAGES*	*MINOR LANGUAGES*
Indo-European	Germanic	Western	English, German, Flemish, Dutch, Afrikaans, Yiddish	Luxembourgian Frisian
		Northern (Scandinavian)	Swedish, Danish, Norwegian, Icelandic	Faroese
	Italic		Latin	
	Romance		Italian, French, Portuguese, Spanish Rumanian	Catalan, Provençal, Sardinian, Moldavian Rhaeto -Romanic
	Celtic	Brythonic	Welsh, Breton	
		Goidelic	Irish (Gaelic) Scottish (Gaelic)	
	Hellenic		Greek	

Table 5.B: Extract from Katzner's Classification of Indo-European Languages

The Indo-European family is the largest family of languages whose speakers embrace approximately one half of the word's population. This language family was first classified by *William Jones,* a British judge and scholar living in India. In his 1786 address to the Royal Asiatic Society, he notes:

"The Sanscrit language, whatever be its antiquity, is of a wonderful structure; more perfect than the Greek, more copious than the Latin, and more exquisitely refined than either, yet bearing to both of them a stronger affinity, both in the roots of the verbs and in the forms of the grammar, than could possibly have been produced by accident; so strong indeed, that no philosopher could have examined them all three, without believing them to have sprung from some common source, which, perhaps, no longer exists: there is similar reason ... for supposing that both the Gothick and the Celtick.... had the same origin with the Sanscrit;"

The general consensus is that original Indo-European civilization developed somewhere in eastern Europe about 3000 B.C.. About 2500 B.C. it broke up and people migrated to many different sections - some headed north toward Russia others west through Europe to Italy, France, Germany and the British Isles.

Wherever they settled they seem to have overcome the existing population and imposed their language upon them. Table 5.B shows a section of the Indo-European family and its subgroups, including the two great classical languages of antiquity - Latin and Greek. The two most common subgroups of the Indo-European family are Romance and Germanic.

5.4.1 ROMANCE

Romance languages are the modern descendants of the language of the Roman empire - Latin. As the armies of Rome extended the boundaries of the empire into much of Europe, Latin was introduced everywhere as the new language of administration. As the empire began to crumble and the Roman administration began to disappear, the Latin of each region began to develop in an individual way (except in the domains of science and mathematics where it remained the principal language until the seventeenth century). Each developed individual characteristics as they were influenced by the speech of the surrounding peoples.

The evolution of Romance languages continued into modern times. Many words still exhibit remarkable uniformity. The following example illustrates this uniformity for the words 'bread' and 'three':

Latin	Italian	Spanish	French	
panis	pane	pan	pain	‘bread’
tres	tre	tres	trois	‘three’

5.4.2 GERMANIC

By the first century B.C., Germanic peoples speaking a fairly uniform language were living on both sides of the Baltic Seas. In time, East, West and North Germanic dialects were developed. In the fifth century A.D. three West-Germanic tribes - the Angles, the Saxons and the Jutes crossed the North Sea into Britain, bringing with them a language that would be later known as English. The Germanic languages, like other family sub-groups, show remarkable similarities:

English	Dutch	German	Danish	
man	man	man	man	‘man’
mother	moeder	Mutter	moder	‘mother’

5.5 AUTHOR NAMES IN THE 1872 CATALOGUE

As mentioned earlier, the author names in the 1872 database are listed predominantly in Latin. However sometimes the translation of an author name in other languages such as English, Italian French etc. appears alongside its Latin equivalent. Some examples include:

CARTESIUS, seu DESCARTES (Renatus)
CLERICUS, seu CLARKE (Samuel), M.A.
FREDRICK seu FREDERICUS
ROSSUS,ROUS,ROUSE,seu ROWS (Johannes),
Interiorus Templi socius.

In addition some variant spellings for author names appear:

> GRAY, or GREY (Zachary), LL.D.
> SPENSER, or SPENCER (Edmund)
> SMITH or SMYTH (Edward), bp. of Down and Connor.
> CHAMILLARD, ou CHAMILLART (Stephanus), S.J.

Since searching by author name is a popular search with users of this system and since users tend to express their query in their own language, numerous author names will not retrieve documents even if the author name exists in the catalogue. Reasons for this include the fact that the author name could be listed in Latin (and not the users native language) and the fact that the name could be listed under a variant spelling. By providing some means of mapping names to their variant spellings and Latin equivalents, multilingual access to the author names of the 1872 Catalogue can be achieved. How then can the knowledge of language change discussed above be used to produce such a multilingual tool?

5.5.1 SOUNDEX CODE

Phonetic changes, which describe sound changes in a language, have prompted the use of a technique called the Soundex Code. This technique is used to locate all similar-sounding names by mapping them to names that are phonetically alike. While there are some minor variations, the following algorithm describes the process.

5.5.1.1 Soundex Algorithm

All nonalphabteic characters (',-blank) are eliminated.
All lowercase letters are set to uppercase.
The first letter is moved to the phonetic code.
The vowels (A,E,I,O,U ,H ,Y and W) are removed.
The following replacements are made:

Labials:	B,F,P,V	→	1
Gutterals, sibilants:	C,G,J,K,Q,S,X,Z	→	2
Dentals:	D,T	→	3
Long Liquid:	L	→	4
Nasals:	M, N	→	5
Short Liquid:	R	→	6

Two or more adjacent identical digits are combined. Thus, LL → 4, SC → 2, MN → 5.
The first three digits are concatenated to the phonetic code.

Name	SoundexCode	Name	SoundexCode
McCloud	M253	Fredricus	F636
MacCloud	M253	Fredrick	F363
McLoud	M253		
McLeod	M253	Clericus	C462
M'Cloud	M253	Clarke	C462
Hasse	H2	Ross	R2
Howse	H2	Rousseau	R2
Hesch	H2	Rossi	R2
Heisch	H2	Roscoe	R2

Table 5.C: Soundex Code Examples

5.5.1.2 Using the Soundex Code

The surnames and their corresponding soundex codes shown in Table 5.C illustrate how many name variations are merged using the Soundex Code.

However this method of mapping similar names is unsuitable for use with the 1872 Search System for the following reasons. In order to be able to search for similar author names while maintaining the consistency of the search and retrieval engine, each author name would be assigned a code and then those author names having the same code would be hashed to the same location in a hash table. For each search by author name therefore, both the author name hash table and the soundex code hash table would have to be searched. Also, the soundex code for a name does not always yield the expected results as seen in Table 5.C where 'H2' and 'R2' yield names that are not related. This could lead to inconsistencies in the search results. For this reason, an alternative approach to author name access was considered.

5.5.2 COGNATES

If we were unaware of the existence of the Indo-European family, we could still determine that the Romance languages descended from a common source by systematically comparing their vocabulary. It is obvious by examining the word for 'bread' in the Romance languages (Section 5.4.1) and the word for 'mother' in the Germanic languages (Section 5.4.2) that the words are regularly derivable from one another. The existence of similarities in the form, meaning and sound of words among languages, point to the presence of a common ancestor. The name given to words in languages that have descended from a common ancestor is a **cognate.**

5.5.2.1 Author Name Cognates

We can see from groups of author names like 'Clericus' / 'Clarke' and 'Fredrick' / 'Fredericus' that these names stem from a common ancestor. These groups of author names can then be termed 'cognates'. Like all cognates it is easy to see the relations between author name cognates. Often the stem of the author name remains the same and changes occur only at the beginning and the end of names.

Because of the very strong similarities between the author names of languages, it was deemed feasible to extract translation rules from the author name cognates and use them during the search process to translate author names - especially those not listed with their cognates. Examples include:

Name	Listed Under:
CARDANO	CARDANUS (Joannes Baptista)
KEPLER	KEPPLERUS seu KEPLERUS (Joannes)

5.5.2.2 Translation Rules

The list of author names and their cognates were examined and for each author name / cognate pair, rules depicting the differences between the names were automatically extracted and recorded in a table along with their frequency of occurrence.

IUS#	US#	201
US#	#	139
IUS#	#	101
US#	E#	90
US#	I#	77
E	I	72
Y	I	71
I	E	66
A	E	58
E	A	56
#C	#CH	55
INUS#	US#	53
CK	K	49
AE	E	48
O	U	42
C	K	41
IUS#	E#	41
ST	T	41
US#	O#	41
L	LL	37
IO	O	35
C	CH	34

Table 5.D: List of Frequently Occurring Translation Rules

So, for the following author name / cognate pairs:

DESCARTES → CARTESIUS GREGORY → GREGORIUS
SMITH → SMYTHAEUS

the rules derived are as follows (where '#' is taken to be the null string):

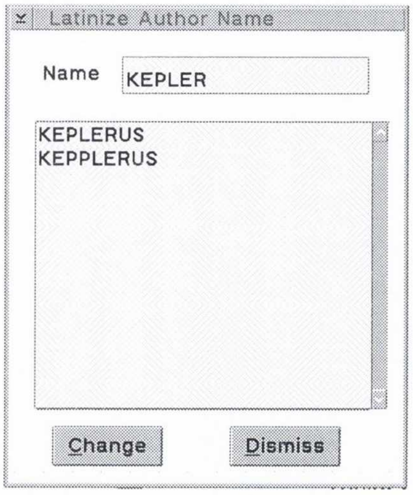
DES→#	I→Y	Y→IUS
ES#→ESIUS	#→AEUS	

These rules were then sorted in descending order of frequency, to be used by the system to translate author names {Table 5.D}.

The program used to extract the rules automatically was written in Snobol4 and may be found in Appendix B along with its results.

5.5.3 MULTILINGUAL AUTHOR TRANSLATION TOOL

In the author search window, the user is given the option to translate his author name into other languages with the 'latinize' tool. The reason the tool is called such is that most of the translation that occurs involves translation either to or from Latin. By selecting this option, the user forces the system to examine the author name for substrings present in the translation rules. If a one or more such substrings are found in the author name, then the corresponding rules may be applied to produce new author names. Each of the new author names produced by the translation process are checked for validity. This is achieved by automatically searching the catalogue for these new names - those that are present in the catalogue are considered to be valid - those that are not present are taken to be invalid and are discarded.
The valid author names produced by the translator are then displayed in a window in decreasing order of probability (those containing more frequently occurring translation rules are taken to be more probable).



KEPLERUS
KEPPLERUS

Figure 5.1: Results from 'Latinizing' the Name 'Kepler'

Figure 5.1. gives the translations that the system found for the author name 'Kepler'. The Latin translation of 'Kepler' is listed ('Keplerus') along with a spelling variation of that translation ('Kepplerus'). The user can then search for one of the new author names by selecting it and clicking on the 'change' button in the translation window. This button changes the author name in the author search window to be the author name translation selected by the user and the search may then be executed.

This author translation tool is the first step towards multilingual access to this multilingual database. It is particularly useful in overcoming the confusion arising from author names appearing in Latin while providing easy access to variant spellings of author names.

CHAPTER 6 - LANGUAGE RECOGNITION

6.1 INTRODUCTION

In the previous chapter, a multilingual tool for accessing author names was described. In this chapter a more universal form of multilingual access - searching the database by language of entry is explored. Before this type of multilingual access could be addressed in the 1872 Search System however, the languages of the catalogue entries had to be recognised. Language recognition for the entries in the library database began in a previous work where an attempt to correct spelling errors resulting from OCR misrecognitions necessitated the recognition of the languages of the entries so that the correct dictionaries could be incorporated in the interactive correction of spelling errors [*Clarke `93*]. The language recognition technique used claimed a success rate of 70.38% and is described below. Improvement on this success rate was deemed necessary if multilingual access was to be attempted successfully. The language recognition algorithm was hence altered and a success rate of 92.56% achieved.

6.2. LANGUAGE RECOGNITION

Language recognition involves determining the language of a text. It uses knowledge about the structure and components of a language and requires intuitive data to differentiate between a number of languages. Knowing how humans recognise languages can aid in the development of a language recognition algorithm.

6.2.1 HUMAN RECOGNITION

To deduce to which language a word belongs, a human delves into diverse fields of knowledge - pragmatic knowledge (knowledge about the world around us), morphological knowledge (how the word is structured), phonetic and acoustic knowledge (how the word sounds in a human's pronunciation), contextual knowledge (to which language the surrounding words could belong) and general perception. The human's recognition process is generally accomplished using one of three techniques [*Glantz `57*].

1. The data pattern will be the exact equivalent of a pattern stored in the memory system. As a human gains more experience and education, his knowledge bank increases in size. In the case of language recognition, when an anglophone sees the word 'the' for example, he knows immediately that the word is an English word because it is contained in his knowledge bank.
2. Associated or collateral information is used to recognise the information contained in the subject data. In language recognition terms, this may be interpreted in the following way. If a human sees the word 'tellement' and has not studied the French language, then he could still recognise the language as being French both by comparison with languages already studied and by deduction. Deduction involves general knowledge about the world (pragmatic knowledge) such as whether a word could belong to a romance language or whether the word is written in an alphabet other than that of the human's native language.
3. Further information is sought if the first two methods have failed. In language recognition, the human who is unable to recognise the given language, seeks further information as to what the language could possibly be. This could involve simply looking up a book or perhaps further research and study.

Machine recognition has, to date, been concerned with the first of these techniques - exact equivalence, while modeling the human's search for further information remains a futuristic dream.

6.2.2 MACHINE RECOGNITION ALGORITHMS

The aim of a machine recognition algorithm is to simulate the human's recognition processes in determining the language of a text. Prior to the development of a language recognition algorithm for use with the 1872 Catalogue, a variety of machine recognition algorithms were examined to test their suitability for use with the catalogue. The machine recognition algorithms reviewed here, use exact equivalence to recognise languages.

6.2.2.1 Ingle's Method (1976)

Ingle compiled a list of one or two letters in alphabetical order, quoting after them numbers indicating the various languages in which they occur.

'at'	9,13,15,16	9 = Danish	15 = Norwegian
		13 = Turkish	16 = English

To determine the language of a text, he extracted one or two letter words from the text, noted in which language they occurred and eliminated languages not applicable to all the one or two letter words [*Ingle `76*].

This method was tested on the 50 page corpus of the 1872 Library Database and the success rate of 30.45% achieved proves how unsuitable this method of language determination is for this database. Catalogue entries often consist of less than five words meaning that there are few one or two letter words available in an entry to use in the elimination of incorrect languages. The one or two letter words that are present are more than often found in a number of other languages making the elimination of incorrect languages impossible.

6.2.2.2 Cavner & Trenkle's Method (1994)

Cavner & Trenkle used n-grams of characters to determine the language of a text [*Cavner & Trenkle `94*]. They used the 100 most frequent n-grams of length 1-5 and achieved a success rate of 97% on documents longer than 300 characters. This method would be unsuitable for the 1872 Printed Catalogue because the n-gram tables built up from modern electronic dictionaries would not adequately represent the vocabulary used in the catalogue which spans five centuries. Orthography too has also changed over the centuries and many old French, English and German spellings may be found in the catalogue which would also cause inconsistencies between the n-grams contained in the catalogue and the modern n-gram tables.

6.2.2.3 Kulikowski's Method (1991)

Kulikowski used frequent 2-3 character words in a semi-automatic recognition of nine languages [*Kulikowski `91*]. He tested his method on a line of text (40-80 characters) and claimed a success rate ranging from 63% to 85% across languages.

6.2.2.4 Clarke's Method (1993)

The machine recognition algorithms examined were found to be unsuitable for use with the 1872 Catalogue. More intuitive knowledge about languages is required in the determination of languages if machine recognition algorithms are to model the human recognition process. In an attempt to achieve this, a small table of intuitive data containing frequently occurring function words and morphemes was built up for seven of the languages in the catalogue (Latin, English, German, Dutch, Italian, French, Spanish). Machine recognition was then carried out on the entries of the catalogue using this data [*Clarke `93*].

6.2.3 LANGUAGE RECOGNITION IN THE 1872 DATABASE

The language recognition algorithms described below avoid the traditional dictionary look-up technique, which would not only have involved importing several dictionaries (which are not readily available) but also time consuming exhaustiveness in the matching process. Instead they use exact equivalence in conjunction with morphological knowledge about languages to distinguish one language from the other.

6.2.3.1 Function Words

Function words include articles, prepositions, pronouns, quantifiers and demonstratives and are employed as a recognition technique because nearly every sentence in the seven languages contains at least one of those words. Examples of function words taken from the seven languages may be seen in Table 6.A.

English	German	Dutch	Spanish	French	Italian	Latin
the	der	ik	el	le	la	seu
that	ein	een	nos	que	del	per
of	von	van	ese	une	quel	de
which	auf	af	un	de	i	quo
a	mit	der	che	qui	nella	ejus

Table 6.A: Sample Function Words

However adding every function word to the language table for each of the seven languages would have defeated the purpose of avoiding the classical look-up technique. Instead, only the high frequency function words are recorded in the table. Determining these high frequency function words involved analysis of text in each of the languages (or more precisely those languages that the author was not familiar with i.e. Spanish, Italian, Dutch and Latin). Those function words that appeared to be occurring frequently in the texts studied, were compared against a word frequency count of the catalogue. If the frequency count in the catalogue also displayed a high count for those words, then the words were entered into the appropriate language tables. The function words that were duplicated in more than one language were then removed from the language tables to facilitate the differentiation between languages. The following are examples of function words occurring in more than one language:

la	occurs in French, Italian, Spanish.
il	occurs in French, Italian.
in	occurs in Dutch, English, Italian, German.
de	occurs in Latin, French.
of	occurs in Dutch, English.
est	occurs in Latin, French.

Yet function words alone were not sufficient for language recognition as the following example illustrates:

La fille est pratique

Since duplicate function words were omitted from the language tables, 'La' and 'est' cannot be used to recognise the language of this sentence. So how can this sentence be recognised as being French? Adding nouns and verbs to the language tables would be equivalent to creating full dictionaries. Hence the idea of morphology or word structure was addressed.

6.2.3.2 Morphology

According to Chomsky, certain restrictions on the laws which a language must in some sense conform to, are genetically inherited as part of the normal make-up of the human mind [*Chomsky `90*]. There are restrictions with regard to the sound and structure of languages which guide and limit a person's acquisition of language and enable him to differentiate one language from another.

The aim of studying morphology is to elucidate certain principles that apply to the structure of words in all languages. A *morpheme* is the minimal indivisible or primitive unit of a word [*Matthews `91*]. The word 'being' may be divided into two morphemes -

'be' and 'ing'. An inflection may consist of plurals, adverbials, past tenses, nominals and affixes, to name but a few. The above word contains only one inflection 'ing' ('be' is the root). Hence the morpheme 'ing' may be described as an inflectional morpheme.

Insight into what language text is written in, may be gained from the study of these inflectional morphemes. In most languages there is a degree of consistency found in inflectional morphemes. For example inflectional morphemes in English occur at the end of words while those in the Athbaskan language occur at the beginning of words (e.g. Navaho). There is a reason for this. If inflectional morphemes are always marked in the same place, they are more easily perceived or more easily recognised by people learning the language. Inflectional morphemes are hence motivated. Similar arguments may be applied in accounting for the ways in which languages differ.
Capturing inflectional morphemes for each of the languages and adding them to the tables of function words already compiled, should enable the language recogniser to perform more efficient distinctions between languages.

In order to achieve this, morphological research on the structure of the words in each of the languages was required. The consistency of the inflectional morphemes (occurring at the end of words in all the languages studied) was confirmed, and the more common morphemes identified for each of the languages. Examples include:

'elle', 'ique'	occurring at the end of a French word.
'ing', 'hood'	occurring at the end of a English word.
'mus', 'arum'	occurring at the end of a Latin word.
'ista', 'encia'	occurring at the end of a Spanish word.
'chi', 'iamo'	occurring at the end of a Italian word.
'keit', 'ung'	occurring at the end of a German word.
'inge', 'je'	occurring at the end of a Dutch word.

A manual comparison of the resulting morphemes eliminated any possibility of duplication and the morphemes were then tested for inflection to determine whether they occurred at the beginning or end of a word. The inflectional morphemes of the languages studied occurred at the end of words. This is one of the reasons why suffixes and not prefixes were included in the language tables. The other reason stems from the fact that languages in a language family (e.g. Romance, Germanic) have common prefixes since they descend from a common source. For example the prefix 'con' is common to Italian, Spanish, Portuguese and French since these languages stem from a common source (Latin). Adding prefixes to the language tables would make differentiation between languages more difficult.

A hyphen preceding the morphemes in the table was used to differentiate them from the function words. The morphemes were than added to the language tables {Table 6.B}.

English	No.	German	No.	Dutch	No.
the	153605	des	8138	van	5519
to	34320	der	3369	den	2312
with	19332	die	1065	het	1314
from	14762	zu	397	voor	262
of	409	mit	297	hem	198
-ing	33162	nach	101	zijn	117
-ry	28476	bei	31	heeft	79
-ly	111000	seit	16	-ing	33162
-ful	504	-ung	1355	-inge	626
-hood	301	-keit	22	-je	177

Spanish	No.	French	No.	Italian	No.	Latin	No.
al	402	du	3976	i	5139	per	13616
una	396	est	1359	da	1123	ad	8890
es	162	avec	1114	della	477	seu	1490
el	148	une	314	degli	123	ejus	1484
los	86	il	278	che	107	quo	477
las	76	sont	126	nel	79	sine	92
para	52	chez	41	tra	77	-orum	13429
-ista	146	-oire	2641	-ino	1296	-ibus	7287
-encia	53	-elle	1286	-menti	595	-arum	6477
-emos	12	-eaux	315	-gio	581	-simus	36

Table 6.B: Sample Language Tables listing Function Words and Morphemes and their Corresponding Frequency Counts.

6.3 LANGUAGE RECOGNISER

Once the language tables were compiled, a recogniser was developed in Snobol4 using pattern matching. Pattern matching is the process of comparing two items of information to determine whether one is similar to the other [*Gimpel `73*]. It is an extremely powerful tool for string processing and was used here due to the fact that one can specify where in a word a pattern occurs. For example, if the recogniser is looking for an inflectional morpheme occurring at the end of a word, it can specify comparison at the end of the words in the entry.

The recognition process carried out by the recogniser followed the algorithm described below for each entry in a catalogue page [*Clarke `93*].

Algorithm - Version I

```
For each title Do
    For each of the seven language tables Do
        For each of the elements in the table Do
            If element contains a hyphen
                Then element is a morpheme.
                    Strip the hyphen.
                    Compare the element with morphemes occurring
                    at end of title words.
                    If there is a match
                        Then language of title = language of table.
                Else element is a word.
                    Compare the element with each of the title words.
                    If there is a match
                        Then language of title = language of table.
                End.
        End.
    End.
End.
```

This process is repeated until a match is found or all the language tables have been checked, in which case the language is labeled unrecognised.

When tested on a 100 page corpus of catalogue pages selected randomly from the database, this recognition method achieved a success rate of 70.38% which although encouraging left room for improvement. Its' deficiency lay in the elimination of duplicate function words and morphemes which prevented the recognition of a number of the entries.

e.g. *La femme est heureuse*
'la' and 'est' have been removed from the French language table
as they also occur in Spanish and Latin respectively.

6.4 LANGUAGE TABLE SIZE

If duplicated function words and morphemes are to be included in the language tables, the size of these tables will increase. How would this increased table size affect the language recognition?
To test the effect the size of the language tables have on language recognition, the algorithm was run repeatedly on a 50 page corpus of catalogue pages with varying numbers of function words and morphemes in the language tables. Initially the algorithm was tested with the ten most frequent function words and morphemes added to the table. The test was then repeated for every additional 15 most-frequently occurring elements added. The findings of this experiment are shown in Figure 6.1.

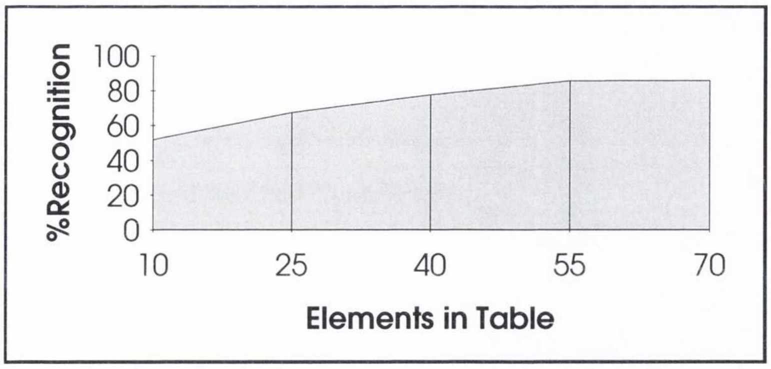


Figure 6.1: How the size of the Language Tables affect the Language Recognition Results.

As the number of elements in the language tables increase from 10 to 25 to 40 to 55, the success rate in language recognition increases simultaneously. However a saturation point is reached between 55 and 70 elements where the difference in the recognition success rate lies at only 1%. This indicates that once the number of elements in the language tables exceeds 55, additional elements will have little effect on the success rate of recognition. This suggests that a cutoff point for addition of elements to the language tables can be taken to be between 55 and 70 elements.

6.5 UPDATED LANGUAGE RECOGNISER

For the purpose of providing multilingual access, an attempt to improve the language recognition results was made. A new method of comparison was incorporated into the language recogniser to reduce the 30% loss in the recognition rate incurred with the previous algorithm. In addition Portuguese and Irish were added to the seven languages listed earlier - as they were found to occur occasionally in the catalogue. The new language recogniser searches for more evidence before selecting a correct language for a title. Because more than one piece of evidence (function word / morpheme) is used to determine the language of an entry, duplicate function words and morphemes, which were eliminated from the previous algorithm, are included in the language tables.

The titles are processed word by word and for each piece of evidence found in a language table, a score of 1 is assigned to that language. In addition, more experience with the languages concerned resulted in additional hand-picked elements to be added to the language tables. Yet even with these new elements and duplicated function words and morphemes, the number of elements contained in the tables remained small - approximately 70 elements on average.

Once enough evidence has been found to determine the correct language of an entry i.e. once the highest_language_score_so_far has reached some predefined threshold, the language is assigned to that entry and the next entry is processed. A threshold is employed so that processing of an entry can stop once enough evidence to determine a particular language has been found. Various threshold values were tested on the 100 page corpus and a threshold of 5 selected as safe in the determination of a language. The algorithm given below and the subsequent examples illustrate the working of this new language recogniser.

Algorithm - Version II

```
For each title Do
    While highest_score_so_far < threshold Do
        While there are still words in the title Do
            Get next word
            For each of the nine languages Do
                If word ∈ table for that language
                    Then language_score = language_score + 1 End.
                Else
                    Extract 2,3 and 4 letter suffixes from word.
                    If any of these suffixes ∈ table for that language
                      Then language_score = language_score + 1.
                    End.
                End.
                If language_score > highest_score_so_far
                    Then highest_score_so_far := language_score.
                        End.
            End.
        End.
    End.
End.
```

Examples:

The following examples explain how the algorithm works in more detail.

<u>TITLE 1</u>

"Leo Belgicus, hoc est, naratio exordii, progressus, &c., reipub. foederatarum Belgii regionum."

The word 'Leo' is first extracted. It exists in none of the language tables and its suffixes exist in none of the tables. So the next word in extracted. Again it does not appear in any of the language tables, however its 2 letter suffix '-us' appears in both English and Latin

and so both of these languages are awarded a score of 1. The highest language score so far is 1 and since this has not exceeded the threshold (which is 5), the next word is extracted. Neither the word ‘hoc’ nor its suffixes appear in any of the tables, so the next word ‘est’ is extracted. This word is found in French and Latin common word tables and so both languages are awarded a score of 1. Latin now has a score of 2 and so the highest score so far becomes 2. The scoring continues until the highest language score (Latin) exceeds the threshold 5.
The following list displays the evidence found by the algorithm.

Evidence List	Language
Suffix *-us*	English, Latin
Word *est*	French, Latin
Suffix *-ii*	Latin
Suffix *-us*	English, Latin
Suffix *-arum*	Latin
Suffix *-um*	Latin

Language Scores: ENGLISH:2; **LATIN:6**; GERMAN:0; DUTCH:0; FRENCH:1; SPANISH:0; ITALIAN:0; PORTUGUESE:0.

TITLE 2

“Iphigenia in Aulis: from the Greek of EURIPIDES. Adapted to the modern stage by J.W. Calcraft, with music by R. M. Levey.”

Evidence List	Language
Word *in*	Dutch, Italian, English, Latin German
Word *from*	English
Word *the*	English
Word *of*	Dutch, English
Word *to*	English
Word *the*	English

Language Scores: **ENGLISH:6**; LATIN:1;GERMAN:1, DUTCH:2; FRENCH:0; SPANISH:0; ITALIAN:1; PORTUGUESE:0.

Notes: With eleven of the words in the title processed, the correct language has been deduced and no further processing is required.

TITLE 3

"Romances N° 275, 275, 368 Romancero y cancionero sagrados, por J. de SANCHA]"

Evidence List	Language
Word *y*	Spanish, French
Suffix *-ados*	Spanish
Word *por*	Spanish, Portuguese
Word *de*	Spanish, Dutch, Italian, Latin, French, Portuguese
Language Scores	ENGLISH:0; LATIN:1; GERMAN:0; DUTCH:1; FRENCH:2; ITALIAN:1; **SPANISH:4**; PORTUGUESE:2.

Notes: All the words in the title have been processed and the highest_score_so_far (4) has not exceeded the threshold. If the highest_score_so_far applies to only one language - which it does here, then it may be taken to be the correct language for that entry.

The updated algorithm was tested on the same 100 pages corpus and a success rate of 88.21% achieved, meaning that the new algorithm improved the performance of the language recogniser by 17.83%.

6.6 RESULTS

This improvement in language recognition is significant, however there was still room for improvement. From the experience gained in using the recogniser, it was felt that the function words contributed more to the recognition of the catalogue entries. If this was the case, then a weighting scheme could be developed for the algorithm where function words would carry a heavier weighting then suffixes.

6.6.1 FUNCTION WORDS VERSUS SUFFIXES

This hypothesis was tested by applying the algorithm to the 100 page corpus with only function words used for recognition and again with only suffixes used for recognition. The results of this experiment may be seen in Figure 6.2.

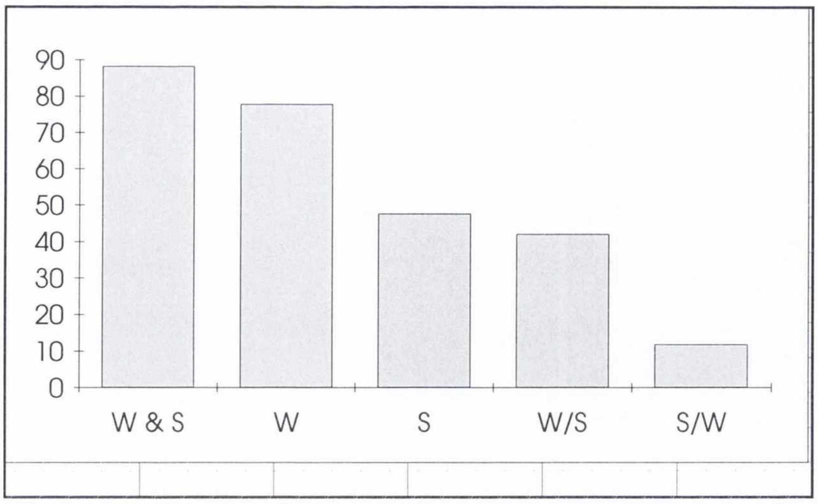


Figure 6.2: Function Words VS Suffixes

In Figure 6.2, *W&S* denotes function words and suffixes, *W* denotes function words only, *S* denotes suffixes only, *W/S* denotes the recognition that words alone got right and that suffixes got wrong and finally *S/W* denotes the recognition that suffixes alone got right that function words got wrong. There are two deductions that we can make from the graph.

1. Function words alone achieve a lower success rate than function words and suffixes together. Therefore both function words and suffixes are required for successful language recognition.
2. Suffixes alone achieve a success rate that is 30% lower than that of function words alone. Also the recognition that suffixes gets right which function words gets wrong (*S/*W) is 11.88% while the recognition function words gets right that suffixes gets wrong (*W/S*) is 30% higher (42.09%).

This verifies the hypothesis that suffixes are less powerful than function words in language recognition.

6.6.2 A NEW WEIGHTING SCHEME

A new weighting scheme could then developed with the function words carrying a heavier weighting than suffixes. But how much heavier should the weighting assigned to function words be?

Again an experiment was carried out on the 100 page corpus to choose which weighting scheme to assign to the algorithm. Figure 6.3 displays the differences obtained in language recognition rate when unique function words were given a heavier weighting than common words and suffixes.

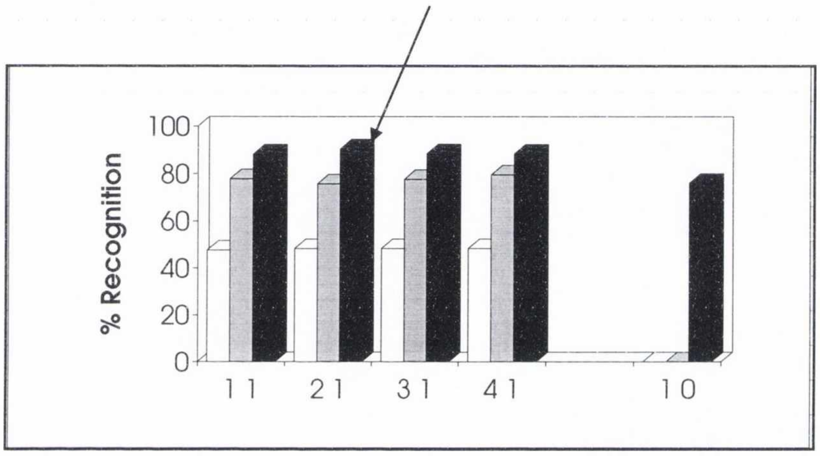


Figure 6.3: How Weighting of Function Words Affects Recognition

This graph displays the recognition rate obtained with suffixes only (white block), with function words only (light grey block) and with both suffixes and function words (black block) where suffixes were given a weight of 1 and unique function words were given a weight of 1,2,3 and 4 respectively. It may be observed that the recognition rate does not improve as the unique function words are given a higher weighting than suffixes to infinity (where the weighting for suffixes is 0 and the weighting for unique function words is 1). The graph demonstrates that the optimum weighting scheme to be used with the algorithm gives unique words a weight of 2, common words a weight of 1 and suffixes a weight of 1. The final version of the algorithm with the new weighting scheme is given below.

Algorithm - Version III

```
For each title Do
    While highest_score_so_far < threshold Do
        While there are still words in the title Do
            Get next word
            For each of the nine languages Do
                If word ∈ only one language table
                    Then language_score = language_score + 2.
                Elseif word ∈ more than one language table
                    Then language score = language_score + 1.
                    (for each of the tables containing the word)
                Else
                    Extract 2,3 and 4 letter suffixes from word.
                    If any of these suffixes ∈ table for that language
                      Then language_score = language_score + 1.
                    End.
                End.
                If language_score > highest_score_so_far
                    Then highest_score_so_far := language_score.
                        End.
            End.
        End.
    End.
End.
```

This version of the algorithm with the new weighting scheme achieved a success rate of 90.35%. The following entries, which the previous algorithms failed to recognise, have been recognised with this algorithm.

TITLE 4

"trad. en Franc. par Iaques de Miggrede"

Evidence List	Language	Weighting
Word *en*	Dutch, French	1
Word *par*	French	2
Word *de*	Spanish, Dutch, Italian, Latin, French, Portuguese	1
Suffux *-ede*	Dutch	1

Language Scores: ENGLISH:0; LATIN:1; GERMAN:0; DUTCH:3; **FRENCH:4**; ITALIAN:1; SPANISH:1; PORTUGUESE:1.

TITLE 4

"et cum notis mho. Crerii [tom. I, p.1 Musei Philo log. et histor.]"

Evidence List	Language	Weighting
Word *et*	French, Latin	1
Word *cum*	Latin	2
Word *et*	French, Latin	1

Language Scores ENGLISH:0; **LATIN:4**; GERMAN:0; DUTCH:0; FRENCH:2; ITALIAN:0; SPANISH:0; PORTUGUESE:0.

6.6.3 SYSTEMATIC EXTRACTION OF FUNCTION WORDS AND SUFFIXES

There is a final step to be taken in the improvement of the language recognition. The elements in the language tables, which were selected manually, were then compared to a frequency list of words in the catalogue. However with 90.35% of the entries in the catalogue correctly recognised - dictionaries for the various languages could be built up from the entries themselves. A program in Snobol4 was written to select the words in the titles for each of the recognised languages and place them in appropriate dictionaries. From these dictionaries, the 70 most frequently occurring function words and suffixes were selected and entered into the language tables[1] . The algorithm was run once more on the 100 page corpus with the new weighting scheme and the new language tables and a success rate of 91.32% achieved. One and two word titles were then removed from the corpus which raised the success rate to 92.56%. Table 6.C lists the search results obtained for the 100 page corpus.

[1] The language tables may be found in Appendix C.

Language	Actual	%Total	Recogniser	%Correct
ENGLISH	2617	59.86	2486	94.99
LATIN	1221	27.94	1118	91.56
FRENCH	251	5.74	219	87.25
DUTCH	102	2.33	97	95.09
GERMAN	73	1.67	68	93.15
ITALIAN	52	1.19	43	89.58
SPANISH	24	.56	12	50.00
PORTUGUESE	2	.05	2	100.00
LATIN&FRENCH	12	.27		
LATIN&ENGLISH	6	.14		
LATIN&GERMAN	2	.055		
ENGLISH&FRENCH	4	.095		
ENGLISH&GERMAN	1	.025		
ENGLISH&DUTCH	1	.025		
ENGLISH&ITALIAN	1	.025		
DUTCH&FRENCH	1	.025		
UNREC			265	
TOTAL	4370	100%	4370	92.56%

Table 6.C: Language Recognition Results obtained for 100 Page Corpus

The *Actual* column list the entries that occurred in the eight languages for the 100 page corpus. Some of these entries contained two languages e.g. ‘English & Latin’. The *%Total* column gives the percentage of entries occurring in the eight languages. The languages determined by the language recogniser are given in the *Recogniser* column and the *%Correct* columns displays the correct recognised entries. Those entries that could not be recognised by the language recogniser are listed in the *Unrec* (unrecognised) row. These unrecognised entries constitute 6.06% of the total number of recognised entries.

The confusion matrix given below shows exactly which entries the recogniser recognised both correctly and incorrectly {Table 6.D}.

		R	E	C	O	G		L	A	N	G
		E	**L**	**F**	**D**	**G**	**I**	**S**	**P**	**U**	**TOTAL**
A	**E**	2486	5	0	2	0	0	0	2	122	**2617**
C	**L**	6	1118	0	0	3	2	1	0	91	**1221**
T	**F**	2	3	219	0	0	0	0	0	27	**251**
U	**D**	0	1	0	97	1	0	0	0	3	**102**
A	**G**	0	0	0	1	68	0	0	0	4	**73**
L	**I**	0	2	0	0	0	43	0	2	5	**52**
	S	0	0	1	0	0	2	12	1	8	**24**
L	**P**	0	0	0	0	0	0	0	2	0	**2**
A	**L&F**		4	6						2	**12**
N	**L&E**	3	1							2	**6**
G	**L&G**					1				1	**2**
	E&F	3		1							**4**
	E&G	1									**1**
	E&D	1									**1**
	E&I						1				**1**
	D&F				1						**1**
	TOTAL	**2502**	**1134**	**227**	**101**	**73**	**48**	**13**	**7**	**265**	**4370**

Table 6.D: Language Recognition Confusion Matrix

Reading from left to right, we see that 2486 of the English entries were recognised as such, 5 of the English entries were recognised as Latin, 2 as Dutch, 2 as Portuguese and 122 were not recognised at all ('unrec'). Most of the languages presented in the confusion matrix follow suit in that few of the languages were recognised incorrectly. Most of the errors occurred as a result of the recogniser not being able to determine the language of an entry and hence marking the entries as unrecognised. This means that although the user will be unable to retrieve the documents that have not been recognised - at least they will retrieve few documents that have been recognised incorrectly.

The 92.56% success rate achieved by the language recogniser compares favourably with the other machine recognition methods describe in section 6.2.2. - especially since the 100 page corpus consists of noisy OCR output and the titles of the entries are short being 13 words (approximately 80 characters) in length on average.

6.7 SEARCHING THE DATABASE BY LANGUAGE OF TITLE

The 92.56% success rate of the language recognition algorithm enabled a major step to be taken towards multilingual access to the database. The languages of each of the entries were indexed in the same way the titles were indexed and a language hash table was compiled with each language pointing to the entries containing that language. Then a search was developed to allow users to search on entries in a particular language, thereby providing the user with access to entries in different languages.

However, the number of hits returned for each of the languages was far too large for users to browse through. For example, a search on 'language = English' returns 44,713 entries while a search on 'language = Latin' returns 121,357 entries.
How many users can afford the time to search that many entries?

It was for this reason that a language filter search was developed. The filter search ensures that the language search be used only in conjunction with another search. Examples are given below:

Author = Dura* and Language = French	**(21 hits)**
Author = Goethe and Language = German	**(9 hits)**

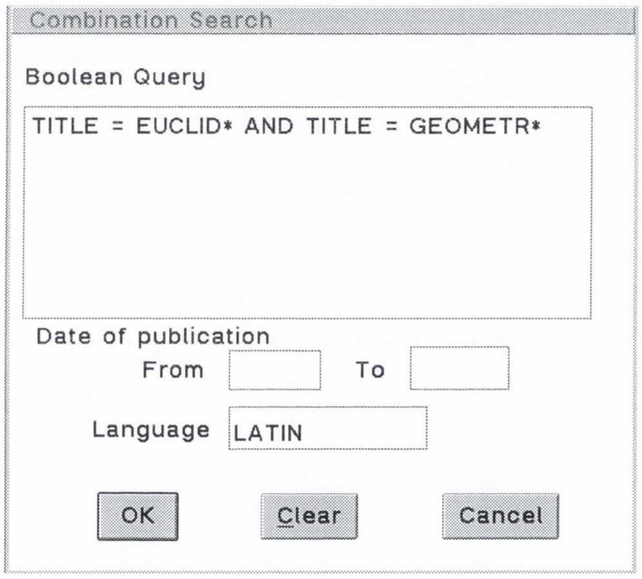
Title = euclid* and Title = geometr* and Language = Latin

(16 hits)

Location = Amsterdam and FromDate = 1600 and ToDate = 1750 and Language = Dutch **(50 hits)**

Location = Amsterdam and Language = English **(8 hits)**

In this way the number of documents retrieved is more manageable in size for the user to browse through. Figure 6.4 displays the combination search window which now enables the user to search by language of title.



TITLE = EUCLID* AND TITLE = GEOMETR*

Date of publication
From [ ] To [ ]
Language LATIN

Figure 6.4: Search by Language of Title

Language recognition has enabled an important step to be taken towards multilingual access to the 1872 Library Database. Users have the choice of refining their searches to their own native language or to the language of the book they are looking for. Users of any of the recognised languages (English, Latin, French, Italian, Portuguese, Dutch, Spanish, German) can now enjoy searching the catalogue for titles written in their own language.

CHAPTER 7 - THE IMPROVED USER INTERFACE

7.1 INTRODUCTION

Learning to use a computerised system can be frustrating with users often contracting such maladies as terminal terror, computer consternation and digital dismay! However, help for the user comes in the form of a user interface. The interface is the part of the system the user sees, hears and communicates with [*Sutcliff `88*]. Therefore it can make a substantial difference in learning time, performance speed and user satisfaction of any system. In information retrieval, the interface combines with a retrieval engine to form an IR system. The best retrieval engine is however useless without a good interface. What then can be defined as a 'good' interface.

According to Sutcliff a good interface must match what the user wants to do, must be easy to learn (and easy to remember after a period of disuse), must perform effectively and must satisfy the user. How then, can an interface that satisfies these criteria be designed?

7.2 INTERFACE DESIGN

Interface design is the process of designing interface software so that computer systems are efficient, easy to use and do what people want them to do. The reason interfaces are 'designed' at all is that they are the only part of the system the user sees. If an interface is poorly designed and difficult to use, it may result in user frustration, poor system performance and perhaps even user rejection.

Users have, in the past, accepted poor interfaces but this is unlikely to happen either at present or in the future as we become exposed to attractive and easy-to-use software. In addition, computer systems are becoming more and more interactive, meaning that more and more code is devoted to interface handling. For these reasons, good interface design is regarded as essential in the development of interactive systems, if they are to be accepted by today's users.

7.2.1 DESIGN ISSUES

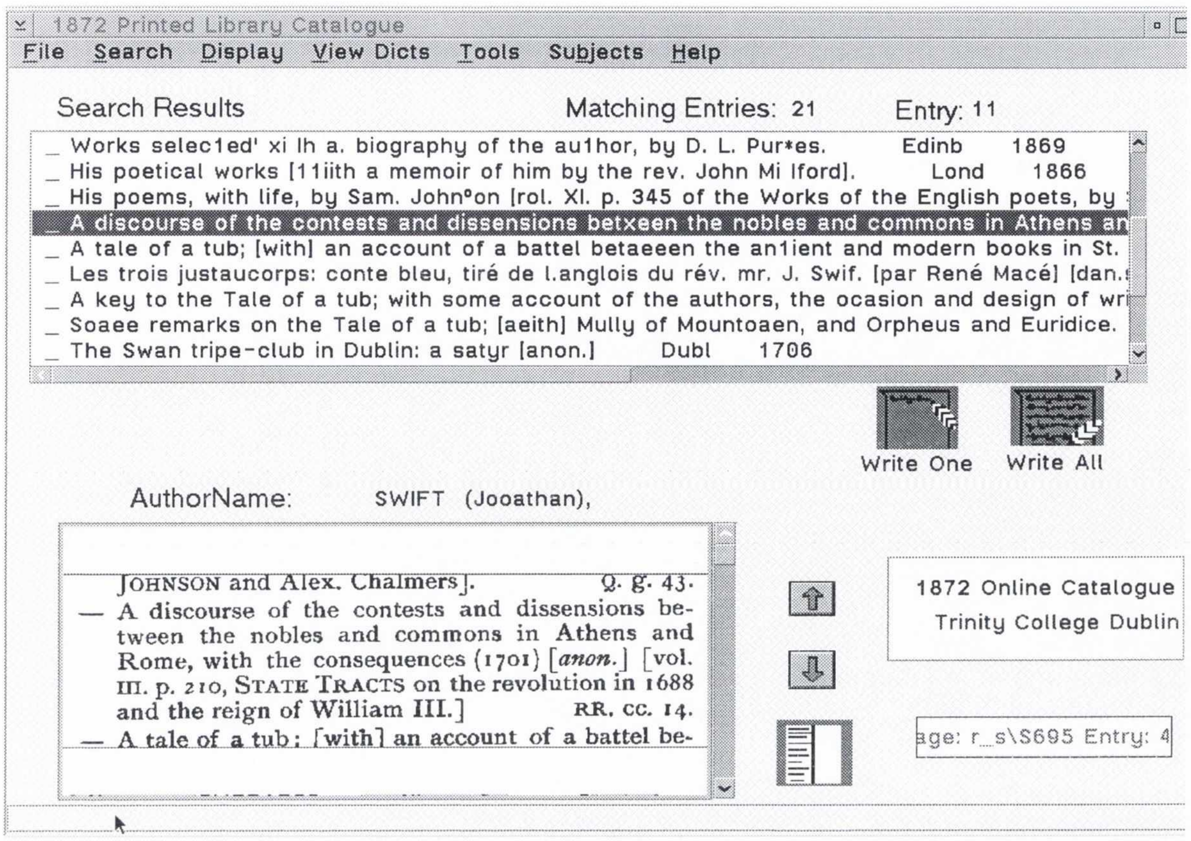
Every designer wants to build a high quality interactive system that is admired by colleagues, celebrated by users, circulated widely and frequently imitated [*Shneiderman `87*]. However, designing an efficient and user-friendly interface is a difficult task. After all, design is not an intuitive process [*Sutcliff `88*]. Sutcliff claims that design, for most people, has to be taught and the following are some of the important qualities for interface design that he outlines.

1. Consistency: is the similarity of patterns present in the interface. Consistency reduces the amount of learning required by the user and since humans recognise similar patterns easily, it improves usability of the system.
2. Compatibility: There should be compatibility between the interface and the user's expectation of it. This follows on from consistency. If the new interface design is compatible with user's previous interface experience then again learning is reduced and the interface will be easier to use.
3. Adaptability: The interface should adapt to its user in various ways. It should allow the user to be in control and hence to work at his own pace. It should also adapt to users skill levels by providing extra support for novice users and more challenging options for expert users.
4. Economy Interfaces should be designed so that they carry out an operation in the minimum number of steps necessary.
5. Guidance Not Control The interface should guide users through their tasks but not take control of these tasks.
6. Structure Structuring interfaces reduces complexity because humans process information by classifying and structuring it within a framework of understanding.

These principles have been drawn from human psychology and can be adapted in the design of any interface to improve effectiveness.

7.3 THE NEW INTERFACE FOR THE 1872 LIBRARY CATALOGUE

The interface for the 1872 Search System was first designed in 1993 [*Culligan* `93]. In the design of the interface, Culligan has met some of the requirements outlined above in the design issues. The interface is consistent and structured - all the main options are presented to the user in a uniform fashion via menus; all data retrieved by the system is also presented consistently using windows (search results window, bitmap window, display entry window). The interface is also economical - it allows users to carry out a very fast search in just three steps - the selection of a search type (author, title, series or combination), the formulation of a search statement and the confirmation of the search request (by clicking on the OK button in the search windows to execute the search).
However this interface fell short on some of the important design issues - namely adaptability and guidance. The interface does not cater for users at different skill levels. Instead all users are assumed to have the same high level of experience with the system. No guidance or support is provided for the user either initially, in the formulation of a search statement, or on completion of an unsuccessful search. In addition, no knowledge about the catalogue characteristics - the fact that it is multilingual, uses a Latin alphabet and contains a variety of orthographies is presented to the user. This makes it difficult for users to carry out successful searches.
These limitations have led to the development of a user-oriented interface for the 1872 Search System. Although the basic layout of the interface has not been changed, many new features have been added to provide the adaptability and guidance required by today's users {Figure 7.1}.



```
Search Results                    Matching Entries: 21      Entry: 11
_ Works selec1ed' xi lh a. biography of the au1hor, by D. L. Pur*es.        Edinb    1869
_ His poetical works [11iith a memoir of him by the rev. John Mi lford].       Lond     1866
_ His poems, with life, by Sam. John°on [rol. XI. p. 345 of the Works of the English poets, by
_ A discourse of the contests and dissensions betxeen the nobles and commons in Athens an
_ A tale of a tub; [with] an account of a battel betaeeen the an1ient and modern books in St.
_ Les trois justaucorps: conte bleu, tiré de l.anglois du rév. mr. J. Swif. [par René Macé] [dan.
_ A key to the Tale of a tub; with some account of the authors, the ocasion and design of wr
_ Soaee remarks on the Tale of a tub; [aeith] Mully of Mountoaen, and Orpheus and Euridice.
_ The Swan tripe-club in Dublin: a satyr [anon.]        Dubl     1706

AuthorName:        SWIFT  (Jooathan),
```

JOHNSON and Alex. Chalmers]. Q. g. 43.
— A discourse of the contests and dissensions between the nobles and commons in Athens and Rome, with the consequences (1701) [*anon.*] [vol. III. p. 210, STATE TRACTS on the revolution in 1688 and the reign of William III.] RR. cc. 14.
— A tale of a tub: [with] an account of a battel be-

1872 Online Catalogue
Trinity College Dublin

age: r_s\S695 Entry: 4

Figure 7.1: The New 1872 Search System Interface

7.3.1 DESIGN STYLES

There have been several design styles created for today's interfaces - menus, icons, form-filling, command languages, windows etc. Each design style has different qualities and capabilities and so appropriate design styles must be chosen with care to match the user population.

Most interfaces utilise more than one design style to achieve the required level of sophistication for the user population. The design styles chosen for the 1872 Search System are described below.

7.3.1.1 Menus

A menu is a simple dialogue type suitable for inexperienced users. Menu selection systems are attractive because they can eliminate training and memorization of complex command sequences and are typically activated by a single keystroke [*Shneiderman '87*]. Yet using the menu selection style does not guarantee that the system will be easy to use. Consideration and testing will be required so that the menu items fit logically into categories and have readily understood meanings. The menu for the new 1872 interface is displayed below {Table 7.A}.[1] The first row is all that appears on the main window but by clicking on one of the elements in this row, the corresponding menu appears.

FILE	SEARCH	DISPLAY	VIEWDICTS	TOOLS	SUBJECTS	HELP
Exit	by Author	Entry Text	*English	*Notes	*View	*Index
*Create	by Title		*Latin	*TermFreq	*Build	General
*Open	by Series		*French	*PrevSearch	*Find	Using
						Help
*Save	Combination		*Dutch			Keys
*Print	GotoPage		*German			Product
			*Portuguese			Info.
			*Italian			
			*Spanish			

Table 7.A: Menu Options in the New User Interface

7.3.1.2 Icons

Pictures or icons can be used to represent functions of a system. To select a function the user points at an icon with the mouse and clicks the mouse button. Icons are very effective techniques if the icon pictures are realistic because the learning time is reduced and operation becomes very easy for inexperienced and experienced users alike [*Sutcliff*

[1] New features are indicated with an asterisk.

88]. Seeking to make the form of a symbol reflect its content or function is not a new concept. Leibniz wrote:

"In signs one sees an advantage for discovery that is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then, indeed, the labour of thought is wonderfully diminished" [*Kreiling `68*].

There are five icons present in the 1872 interface.

⇧ **Move up an entry on a page**

⇩ **Move down an entry on a page**

▥ **View entry surrounding current entries**

▨ **Write one entry to a file**

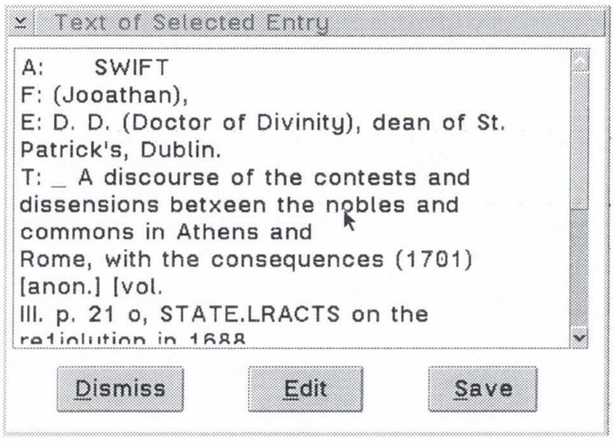
▦ **Write all entries to a file**

The 'write-one-entry' and 'write-all-entries' icons display a page and a quill (in an attempt to portray the era of the catalogue). To avoid user misinterpretation, text describing the functions of these icons is placed under the icons. In addition, information messages appear at the bottom of the main window for all five icons.

7.3.1.3 Windows

Windows subdivide the screen space so that different operations can be taking place on the screen at the same time. There is evidence that people can work concurrently on several tasks [*Sutcliff `88*] and windows enable this multitasking to take place. Windows have many uses. They can be used for messages, work areas and help. They can remain static on the screen or pop-up when initiated.

All the windows that have been added to the interface - term frequency window, dictionary windows, notes window etc. are pop-up windows and disappear once the user has finished working with them. The new editor window which displays the text version of the entry {Figure 7.2}, once opened by the user, can remain open should the user wish to edit text. With the bitmap window in view, the user can inspect the correct version of the entry while correcting the mistakes in the corresponding text version.



```
A:      SWIFT
F: (Jooathan),
E: D. D. (Doctor of Divinity), dean of St.
Patrick's, Dublin.
T: _ A discourse of the contests and
dissensions betxeen the nobles and
commons in Athens and
Rome, with the consequences (1701)
[anon.] [vol.
III. p. 21 o, STATE.LRACTS on the
re1iolution in 1688
```

Figure 7.2: Editor Window

7.4. DESIGN FEATURES

Once the design styles have been selected, decisions on the design features of an interface may be addressed. In their review of on-line search interface design Vickery & Vickery outline the design features found in today's on-line search interfaces [*Vickery & Vickery `93*]. Some of these features have been used in the design of the interface for the 1872 Search System and are described below.

7.4.1 INFORMATION ON USERS

Knowledge on user characteristics is essential if a system is to cater for the needs of its users. User characteristics include user's search experience, knowledge of the search area and user's preferences. These characteristics largely depend on the type of user, whether they be:

- naive - users who have not previously encountered computerised systems. These users may show fear of computers and are usually unfamiliar with the search operation.
- novice - users with some experience of computer systems, although they may be unfamiliar with the current system. These users will have little knowledge or experience with the new system and will be liable to make mistakes. Novice users need a high level of support.
- skilled - users who have gained considerable experience with the system and are proficient operators. Most skilled users become such over time and require rapid-to-use interfaces.

Since the 1872 Search System was developed for use within the library at Trinity College Dublin, most of its users would have had experience with other searching systems e.g. DYNIX. For this reason the type of user considered in the design of the interface excluded the naive user and included the novice and skilled user.
This meant that the support for searching provided by the interface had to be optional as novice users require a high level of support while skilled users require support to a lesser degree.

7.4.2 MODES OF ENTERING QUERIES

Vickery & Vickery outline two common modes of entering queries to on-line interfaces:
(a) The user may be asked to enter a query as a series of search terms. Each term is processed and AND'd or OR'd with the next term until the entire query has been

processed. This mode of entering queries is present in the INSERM interface [*Halpern & Sargeant `88*].

(b) A refinement to this mode is to present the user with the field names contained in the database. The user can then attach his query terms to the appropriate fields. The SHERLOCK interface displays this mode of query input [*Fiz Technik `91*].

Since the 1872 Search System interface was designed with both the novice and the skilled user in mind, ideas were taken from the two query input modes described above. Templates reflecting the fields present in the 1872 catalogue are provided as search windows to the database. For example, author, title and series windows are available which conduct searches on the corresponding fields of the catalogue when the user inputs query terms. This method of entering queries is extremely easy to use and is hence suited to the novice user as the search field is automatically specified.

A combination search window is also available in the 1872 Search System interface allowing the user to construct a Boolean query using a combination of fields. The user must specify the field to be searched with each query term he enters. Each query term and field is then AND'd or OR'd to the next query term and field until the query is processed in full. The combination search allows the skilled user to specify complex queries while the novice user may avail of it to construct simpler queries.

<u>Combination Queries -</u>

Simple: *title = 'Thames' and location = 'London'*

Complex: *(author = 'Smyth' or 'author = 'Smithe') and (title = 'engine' or title = 'engineer')* and *(location = 'Cambridge' or location = 'Oxford' or location = 'London')*

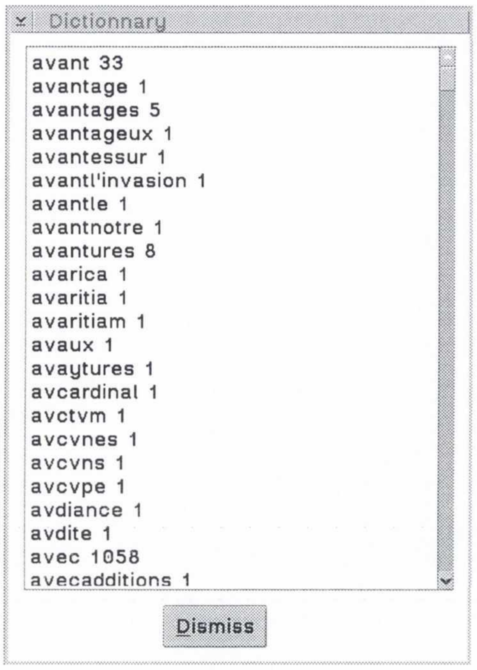
Now, the improved user interface provides an even simpler means of entering queries. With the expert subject trees option, automatic query formulation is available. All the user is required to do to input a query is to select a relevant term from a subject tree of interest and click on the query button. A Boolean query statement is then formulated containing not only the relevant term but all the terms lower than it in that subject hierarchy.

7.4.3 LEXICAL INFORMATION / DATABASE TERMINOLOGY

A lexicon is any repository of information about words and terms in the database. It may be presented to the user in alphabetical order, or in a hierarchical or even diagrammatic form and provides both linguistic and domain knowledge about the database concerned. Allowing users to browse through lexical information is important as it gives the user a feel for the database and its contents. Even though concepts of browsing are becoming more sophisticated, there is still a lingering tendency to see browsing in contrast to direct searching - to see it as a casual, don't-know-what-I-want behaviour that one engages in separate from regular searching [*Bates `89*].
However others - including Ellis - see browsing as being a semi-structured searching and recommend that it be available in automated systems [*Ellis `89*].

Browsing in the 1872 Search System is especially important for two reasons:

1. The vocabulary present in the database dates back to the fifteenth century. Not only have the spellings of words changed such as the English word 'catholic' being previously spelled as 'catholick' but also some of the words we use today did not exist at the time the catalogue was compiled and are therefore missing from the database. For example the word 'sun-dial' is not present in the catalogue, however many books on sun-dials are present in the database and may be found by looking up words such as 'gnomon' or 'horologium solarium'.
2. The 1872 database consists of a number of languages yet users tend to be familiar with just one or two of these languages. Making lexicons for the common languages in the database available could encourage the user to browse through these lexicons and conduct searches in languages he would otherwise have ignored.



```
avant 33
avantage 1
avantages 5
avantageux 1
avantessur 1
avantl'invasion 1
avantle 1
avantnotre 1
avantures 8
avarica 1
avaritia 1
avaritiam 1
avaux 1
avaytures 1
avcardinal 1
avctvm 1
avcvnes 1
avcvns 1
avcvpe 1
avdiance 1
avdite 1
avec 1058
avecadditions 1
```

Figure 7.3 Browsing through the French Lexicon

For these reasons in particular, lexicons were constructed for each of the languages recognised in the database i.e. English, Latin, French, German, Spanish, Italian, Dutch and Portuguese. These lexicons were made available to the searcher in the ViewDicts menu in the interface. From ViewDicts, users may select one of the languages and browse through the lexicon that appears in that language window {Figure 7.3}.

7.4.4 RELATIONS BETWEEN TERMS

A display of semantic links between terms can aid the searcher in the initial choice of search terms as well the modification of an unsuccessful or unsatisfactory search statement. The most common relations used are those found in a standard thesaurus - the synonym relation and the broader/narrower term relation.

Interfaces can use any set of these relations. The KIWI interface for example makes use of the 'part-of', 'component-of', 'cause-of' and 'application-of' relations [*Larsen `87*]. Other interfaces use the simpler 'related-term' relation which subsumes a number of relations.

Related terms in the 1872 Search System interface are presented to the user in tree form via semantic links as described in chapter 4. The general 'related-term' was chosen to link the terms because 1) the expert user is allowed to build a tree of knowledge himself and the aim of the tree construction option is to make it easy for him to do so. Reducing the type of links to just one eases the expert's task and increases the flexibility of the subject tree. 2) One of the thrusts behind the provision of subject trees was not only to relate terms within a language but also to relate terms across languages. The use of the 'related' term relation avoids incorporating additional relations for this purpose.

The provision of the subject trees option in the 1872 Search System interface helps the user get a feel for the variety of domains present in the catalogue and the scope of possible searches as well as aiding him in the formulation of a search statement.

7.4.5 MULTILINGUAL PROCESSING

There are many different levels of multilinguality present in today's interfaces. These include multilingual screens, acceptance of users' queries in more than one language and the translation of users' queries into the language of the database [*Vickery & Vickery `93*].

The 1872 Search System differs from others in the fact that the database itself is multilingual and since the retrieval system uses a free-text search, the user may enter a query term in any of the languages contained in the database (except for those in a non-Roman alphabet). In addition a multilingual tool to find translations of author names is available in the author search window (see chapter 5). There is however no provision for translating terms from one language to the other in this interface for a number of reasons.

(a) <u>Machine Translation Difficulties</u>

The linguistic key to machine translation is the resolution of ambiguity [*Bennet et al `86*]. Lexical ambiguity arises where multiple senses for a single word are common in all languages. For example the French word 'garçon' translates to either 'boy' or 'waiter' in English. Context would be required to select the appropriate translation. Such ambiguities contribute to the reasons why we are still far from the day when machines will be able to translate texts with the same fluency and accuracy as human translators.

(b) Lack of Appropriate Dictionaries

The age and orthography of the 1872 database means that modern electronic dictionaries are unsuitable for translation as many of the terms present in the database are absent from modern dictionaries and visa versa.

(c) Number of Languages Present

Due to the numerous languages present in the database, transfer-based translation across languages would be extremely time consuming. Even if appropriate electronic dictionaries were available, for just seven of the languages in the database, 7*7 dictionaries would be required in the translation of keywords.

The European Commission investigated the use of an artificial language - INTERLINGUA in an attempt to avoid incorporating so many dictionaries to translate their documents. Documents in the source language were translated into INTERLINGUA and then back into the target language. However INTERLINGUA requires the definition of language-free interlingual representation and therefore poses serious practical difficulties for use with the 1872 database [*Bennet et al `86*].

7.5 DESIGN FEATURES SPECIFIC TO THE 1872 CATALOGUE

In addition to the various design features suggested by Vickery & Vickery, specific design features were developed for the interface of the 1872 Search System. Some of these additional features were designed in the light of experience gained from using the system. Others were prompted by suggestions from users and librarians.

7.5.1 BIBLIOGRAPHY MANAGER

This option in the interface provides facilities for both the building of a bibliography and the printing and saving of bibliography files. These options may be found under the File menu in the main window.

Users can create their own personal file and add individual entries from the search results or the entire result of a search to this file by clicking on the ‘write one’ or ‘write all’ icons in the main window respectively {Figure 7.1}. When one of these icons is pressed, the full entry including author name, title, imprint and shelf mark is saved to the file.

Any number of personal files may be created by the user and existing files may also be opened should the user wish to add to them. On completion of his search session with the system, the user may either print his file directly {Figure 7.4} or save it to a floppy disk using the ‘print’ and ‘save’ options in the File menu.

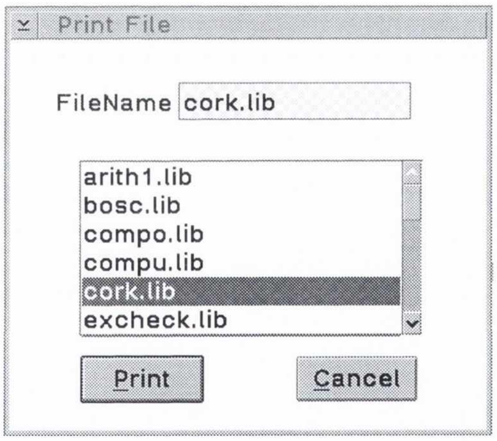


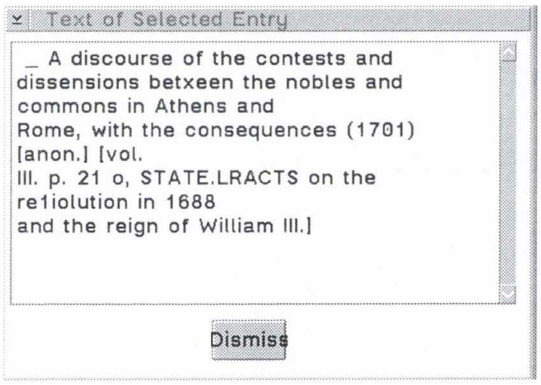
Figure 7.4 Printing a User File

This bibliography manager reduces the amount of ‘idle time’ the user spends with the system by eliminating the need to take down references to any relevant books found. Instead records of relevant books may be kept in the user’s own personal files at the touch of a button.

7.5.2 EDITOR

An editor was developed for the 1872 Search System interface with the goal of reducing the effects Optical Character Recognition errors have on searches. Because of the letters that have been misrecognised by OCR, some of the words and hence some of the books titles will never be retrieved by the user. By providing an editor, the misspellings caused by OCR errors may be corrected by the user, should he come across them while searching the database. These corrections can then be saved resulting in a continuous improvement in the database and hence in the retrieval of information.

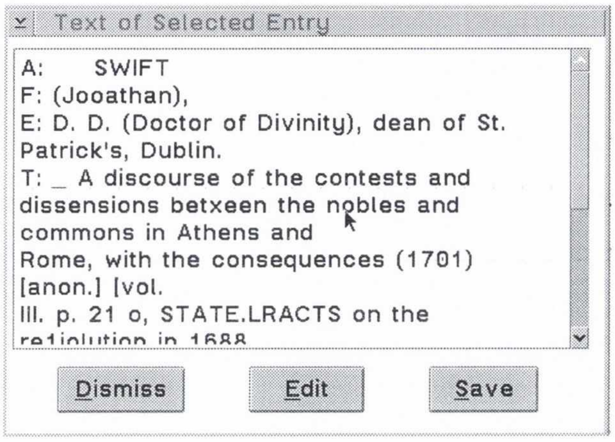
There was an option already available in the interface which allowed the user to view the entry selected. This option was originally provided so the user could see the entire entry as only the first line of each entry appears in the search results {Figure 7.5}.



```
_ A discourse of the contests and
dissensions betxeen the nobles and
commons in Athens and
Rome, with the consequences (1701)
[anon.] [vol.
III. p. 21 o, STATE.LRACTS on the
re1iolution in 1688
and the reign of William III.]
```

Figure 7.5 Display Entry Window

The editor was built on top of this option to allow users to correct any OCR errors present in the entry. Clicking on the 'edit' button in the new display entry window, causes the entry to be broken down into its constituent fields - 'A:' for author name, 'F:' for first name etc.{Figure 7.6}. By scrolling up and down the window, corrections to any one of these fields may be made by the user and then saved by clicking on the 'save' button. The changes are not saved to the actual file from which the entry originates, but to a log file containing updated files only. This file is used by a system administration file (written in Snobol4) to update the database files once a month, say. The administration program searches through the database files for the entries listed in the log file. Corrections are copied from the log file to the corresponding database files and the system is then reindexed. Reindexing the system causes the misspellings in the entries to be overwritten and their correct counterparts recognised for searching. Eventually the errors produced by OCR will be erased by this process.



```
A:      SWIFT
F: (Jooathan),
E: D. D. (Doctor of Divinity), dean of St.
Patrick's, Dublin.
T: _ A discourse of the contests and
dissensions betxeen the nobles and
commons in Athens and
Rome, with the consequences (1701)
[anon.] [vol.
III. p. 21 o, STATE.LRACTS on the
re1iolution in 1688
```

Figure 7.6 Editor Window

7.5.3 LIBRARIAN'S NOTES

Any comments about the system - how to use it effectively, what information is contained in it, what subjects and periods it spans etc. may be recorded in the librarian's notes window. These notes may be viewed by the user under the Tools menu in the interface and users are at liberty to add their own comments on the database. In this way users and librarians can share their experience and advice with fellow users.

7.5.4 HELP FACILITY

There is a powerful help facility provided by GPF - the interface development tool. The help pull-down option provided comprises a help index, general help, using help, keys help and product information. The help index allows the programmer to write his own help files for the various options he develops with the interface builder. The help index for the 1872 Search System interface is illustrated in Figure 7.7. General help, keys help and product information also provide space for the developer to add notes and help about the system.

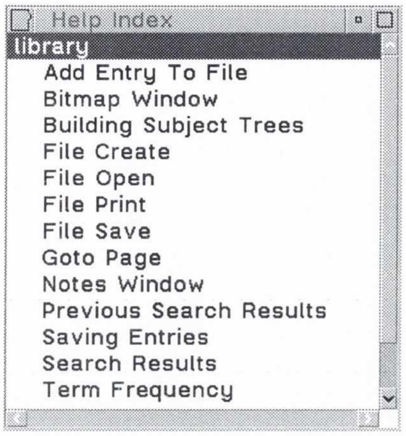

Help Index
library
Add Entry To File
Bitmap Window
Building Subject Trees
File Create
File Open
File Print
File Save
Goto Page
Notes Window
Previous Search Results
Saving Entries
Search Results
Term Frequency

Figure 7.7 Help Index for the 1872 Search System

But the most powerful help tool provided by GPF is the information message. The programmer can define an information message for controls, menu items, and windows within the application. When the user passes the mouse over any of these items, the information message appears in the bottom of the main widow. This means that the user is constantly informed of the options available in the interface. The following are two of the information messages that appear when using the Search System:

Display Entries Surrounding the Current Entry

Write all displayed entries to your personal file

7.6 INTERFACE EVALUATION

Evaluating an interface is considered an ongoing task. Search time, search accuracy, learning time and ease of use are just some of the variables that need to be examined over a period of time. Measures of evaluation may be either objective i.e. derived from controlled collections of data - or subjective, based on intuitive judgements and opinions gathered from users. Objective evaluation may be obtained by employing techniques such as video recordings, system logs and questionnaires.

Subjective evaluation has been carried out on the 1872 Search System interface and improvements to the interface have been made as a result of this evaluation. Such improvements include the incorporation of an editor, a bibliography manager and a more comprehensive help option. Although perfection is not attainable, percentage improvements are possible and worth pursuing [*Shneiderman `87*]. Once the 1872 Search System goes 'live' in the library at Trinity College Dublin, an objective evaluation of the interface will be carried out. For this purpose, a questionnaire adapted from Shneiderman will be used {Appendix D}. It is hoped that with user cooperation, further improvements may be added to the interface to improve user satisfaction and system performance.

CHAPTER 8 - CONCLUSIONS

8.1 INTRODUCTION

This thesis was aimed at providing user-oriented access to a multilingual database. In this conclusion, an initial step towards the evaluation of the improved search system will be taken to determine whether or not the goal of this thesis has been achieved. Complete evaluation will be postponed until the system has gone 'live' in the library at Trinity College Dublin. Two important questions will be posed in the evaluation of the system. Is the system user-oriented? Does the system improve access to the 1872 Catalogue?

8.2 A USER-ORIENTED SYSTEM

The first step in the evaluation of the system is to determine whether it has been designed with users in mind and if so, in what way. Rouse and Rouse conducted an extensive survey of the literature on information seeking behaviour of users and noted:

"Because information needs change in time and depend on the particular information seeker, systems should be sufficiently flexible to allow the user to adapt the information seeking process to his own current needs." [*Rouse & Rouse `84*].

If then, a system is to be user-oriented, it should provide flexible support for the users so they can adapt the system and search strategies to their needs. In the 1872 Search System, support for the user comes in the reduced effort required to use the system, the tools provided to simplify the searching process and the attention paid to the various user types utilising the system.

The naive user, whose knowledge of the system and search strategies is limited, may avail of the comprehensive help facility. Modes of inputting queries on author name, title and series are simplified while query formulation on subjects can be automatic with the aid of the subject trees. This means that successful searches can be carried out with little or no previous experience with the system.

The novice user can avail of some interactive tools to improve his retrieval of information. Term frequency information proves useful in determining the importance of query terms while previous search results highlight the various search strategies in use. In addition to the search by author, title and series, the novice user may conduct some combination searches which combine the above searches with searches by language of entry and date of publication. Browsing through the subject trees built by expert users can enhance the novice user's search results by encouraging him to search for titles in a variety of domains.

For the skilled user, the possibilities of creating complex searches are endless with the combination search option. The interactive tools help the skilled user, who has considerable experience with the system, predict the outcome of a search statement. Skilled users may also observe the structure of the subject trees and are encouraged, in doing so, to build up knowledge trees of their own - in their areas of specialisation.

The support provided for the user types has led to greater flexibility in their search statements and better search results. Automatic and interactive support is provided so that each user may choose the support required to adequately express his information need. The system can therefore be described as user-oriented.

8.3 IMPROVED ACCESS TO THE 1872 CATALOGUE

The next step in the evaluation process of the 1872 Search System is concerned with access to the database. Has this new improved system actually improved access to the entries of the 1872 Catalogue?

Unlike other information retrieval systems the 1872 Search System addresses some of the natural language processing problems in information retrieval and in this way, provides better access to the database. Montgomery declares that:

"In theory, the relationship between linguistics and Information Science is clear and indisputable.... In practice however, the relationship between the disciplines of linguistics and information science has not been exploited" [*Montgomery `72*].

In the development of a new user-oriented 1872 Search System, the relationship between linguistics and information science has been explored under the following headings:

8.3.1 GENERAL ACCESS

Automatic and interactive query expansion techniques have improved access to book titles listed in the 1872 Catalogue. The query expansion techniques have ensured that OCR errors, variant spellings and synonyms are catered for in the user's search statement before it is executed. The user's query is altered to better reflect the vocabulary of the database, which gives the retrieval engine a better chance of retrieving relevant documents.

8.3.2 SUBJECT ACCESS

Up until now, no subject index of the book titles contained in the catalogue has been made available in the search system. Users have formulated queries without knowing the scope of the titles in the catalogue. Now with the possibility of building subject trees of information, a comprehensive subject index is in the making. Dictionaries of some of the common languages in the catalogue have also been made available for browsing. With both the subject trees and the dictionaries, the scope and coverage of the catalogue can be presented to the user in a structured fashion. The user need no longer confine his searches to areas he knows to exist in the catalogue but may broaden his searches to include the variety of domains present.

8.3.3 MULTILINGUAL ACCESS

Multilingual access to the 1872 Catalogue has been provided for the first time with the new 1872 Search System. The multilingual author translation tool provides translation of author names to and from Latin, and also provides access to variant spellings of author names. Being a nineteenth century catalogue with the author names listed predominantly in Latin and often in a variant spelling from the modern day author name, this tool is a valuable access aid to the 1872 Catalogue. Searching by language of entry is another multilingual tool developed for the system. The 92.56% success rate in language recognition enables users to search for book titles in English, Latin, German, Dutch, Irish, Italian, French, Spanish and Portuguese.
Users can now refine their searches to their native language if required. This reduces the amount of time users spend sifting through search results that are not in their own language (and perhaps cannot be understood).

8.3.4 INTERACTIVE ACCESS

The improved user interface developed for the 1872 Search System provides a more interactive environment for the user to search in. The help facility and information messages make the interface easier to use. The addition of a bibliography manager provides all users with the ability to create personal files to store search results of interest while the editor provides the librarians with an oportunity of adding notes and references and also a means of reducing the effects the OCR errors have on information retrieval.

This system has provided a more user-oriented and flexible approach to searching the 1872 Catalogue. It is a system that will improve over time as the subject trees grow and the OCR errors lessen in number. Since the improvement of the system will depend on the users, the intelligence of the 1872 Search System must be attributed to its' users.

8.4 FUTURE WORK

No system is without room for improvement and the 1872 Search System is no exception. The following ideas outline areas of future work which should be considered in the further development of this system.

8.4.1 SYSTEM ADMINISTRATION

By providing passwords for users and librarians, access to the various features of the system could be controlled. Each user could have a unique username and password and depending on his status (undergraduate, postgraduate, staff, librarian) would be allowed to perform certain tasks. For example, user files created by students and staff can be saved to floppy on completion of searching, and these files can be deleted by the system at the end of the day. Librarians' files however could be allowed to remain on the system. Editing of the catalogue entries and of the subject trees could be restricted to librarians only so that inconsistencies and corruption of catalogue entries may be avoided. In their review of text retrieval software, Edmunds and Monckton note that many of today's text retrieval packages separate the administrative tasks of looking after data from the reading and searching functions.
"This is a good idea, particularly in a networked environment when you don't want every user to have access to editing facilities." [*Edmunds & Monckton `95*].

8.4.2 REDUCING OCR-GENERATED MISSPELLINGS

Information retrieval in the system will improve as the number of OCR errors decline. Editing the catalogue entries to improve OCR is a lengthy process however and so to speed up the effects of reductions in OCR errors, other methods of correcting the spelling errors in the catalogue entries resulting from these errors could be explored. For example n-grams could be used to detect and correct misspellings. The Roman alphabet contains 28^2 (alphabet, apostrophe, blank) digrams and 28^3 trigrams, but many of these n-grams are either rare or nonexistent. Words containing rare occurrences of n-grams are considered to be misspellings. If words containing rare n-grams were matched to similar words containing frequently occurring n-grams, then perhaps corrections of the misspelled words could be deduced.

Heuristics could also be used in the detection and correction of spelling errors. Further analysis of the languages contained in the catalogue would be required to develop heuristic rules for spelling correction. Such rules could involve the sequence of letters in a word as in English where ‘q’ must be followed by ‘u’ and ‘i’ must precede ‘e’ except after ‘c’. A run of consonants or vowels could also form the basis for a heuristic rule. For example the run of greater than two consonants in English is considered a misspelling. These rules could be used in conjunction with n-grams to correct misspelled words in the catalogue entries.

8.4.3 IMPROVING MULTILINGUAL ACCESS

Multilingual access to the database could be improved by improving language recognition and thereby increasing the number of relevant documents returned from a language search. Providing some means of translating keywords would also improve user’s access to documents in the various of languages contained in the catalogue.

8.4.3.1 Improving Language Recognition

The 92.56% success rate obtained by the language recogniser could be improved by developing some rules to be used in the recognition process. The following rules for example, could improve results:

A) If the language of an entry cannot be determined and if the author of the entry is that of the preceding entry, then the language of the current entry could be assigned the language of the preceding entry;

B) If the language of an entry cannot be determined and if a place of publication exists for that entry, then the language of the entry could be assigned the language pertaining to the place of publication (i.e. if the place of publication is Paris, then the language assigned to the entry is French).

8.4.3.2 Providing Translation of Keywords

A crude short term solution to the problem of translating keywords would be to propose that queries and responses be translated from one language to another [*Montgomery* `72]. A long term solution would be to imply a high level of understanding - otherwise much information will be lost in the transition from one language to another. Such a solution would be difficult to achieve in the 1872 Search System due to the number of languages present and the lack of appropriate dictionaries. Poor translation results have an immediate negative psychological impact on the user. The reason for this is that information retrieval users are not used to seeing 'bad' output because if an IR system misses out on relevant documents, those documents simply do not appear in the output. Perhaps a medium term solution could involve the translation of keywords in the subject trees using not only modern dictionaries, but also dictionaries built up from the catalogue itself. Those translations that cannot be found in the modern dictionaries can perhaps be located in the catalogue's own dictionaries with the help of expert users. In this way - a subject index for a number of languages present in the catalogue would be built over time.

8.5 USER-ORIENTED EVALUATION

Evaluation is of fundamental importance to any information system. User-oriented evaluation further emphasizes this importance by examining the system within its operational context rather than as a isolated entity [*Bawden `90*]. The user plays an important role in the operational context of any system. Bawden suggests that if users are not to become passive consumers of systems, then their involvement in evaluating a system is essential.

Library catalogue evaluations have a very long history and have been generally carried out by a combination of user surveys, observations of users and failure analysis [*Hafter `79, Lancaster `77, Seymour & Schofield `73*]. It is intended to carry out user-oriented evaluation of the 1872 Search System as the final step in the evaluation process with the interface evaluation questionnaire in Appendix D.

8.6 CONCLUDING REMARKS

This thesis has improved acess to the 1872 Catalogue by expanding users' queries to better reflect their information need and the orthography of the databse; by providing subject access via 'knowledge trees' of information built up by expert users on their areas of sepcialisation; by providing multilingual access with the author name translation tool and the search by language of title; and by supporting users in their formulation of queries with the use of interactive tools such as the term frequency count and previous search results.
It it hoped that users' retrieval of information will continue to improve with this system as the number of subject trees increase, the number of OCR errors decrease and the user gains more experience with the system. In the meanwhile, the users of this system may now enjoy enhanced access to some of Europe's rarest and most valuable books.

BIBLIOGRAPHY

Aitchison, J. (1991) *Language Change: Progress or Decay,*
Cambridge University Press, Cambridge.

Alberico R., Micco M. (1990) *Expert Systems for Reference and Information Retrieval,*
Meckler Corporation, London.

Ammeraal, L. (1991) *C++ for Programmers,*
John Wiley & Sons, Chichester.

Anderson, G. (1992) *Computerising a Library Catalogue using Optical Character Recognition,*
M.Sc. Thesis.
Trinity College, Dublin.

Bates, M. (1986) *Subject Access in On-line Catalogs: A Design Model,*
Journal of the American Society for Information Science, Vol. 37, No. 6, pp. 357-376.

Bates, M. (1987) Book Review of *Interaction in Information Systems: A Review of Research from Document Retrieval to Knowledge Based Systems,* Belkin N., Vickery A..
Information Processing & Management, Vol. 25, No. 2, pp. 215-216.

Bawden, D (1990) *User-Oriented Evaluation of Information Systems and Services,*
Gower Publishing Company, Vermont.

Beerel, A. (1987) *Expert Systems - Strategic Implications and Applications,*
Ellis Horwood Ltd., Chichester.

Belkin, N. (1983) *Recent Trends in Information Science Research in the UK, USA, and Canada,*
Overview of Research Development, Internal Report, 12p, Berlin.

Belkin N., Vickery A. (1985) *Interaction in Information Systems,*
British Library, London.

Bench-Capon, T. (1990) *Knowledge Representation - an Approach to Artificial Intelligence,*
Academic Press Ltd., London.

Bennet P., Johnson R., McNaught J., Pugh J., Sager J., Somers H. (1986) *Multilingual Aspects of Information Technology,*
Gower Publishing Company, Vermont.

Borgman, C. (1986) *Why are On-line Catalogs Hard to Use? Lessons Learned from Information Retrieval Studies,*
Journal of the American Society for Information Science, Vol. 37, No. 6, pp. 387-400.

Buckley C., Salton G., Allan J. (1994) *The Effect of Adding Information in a Relevance Feedback Environment,* In "Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval" Dublin, July 3-6, pp. 292-302.
Springer Verlag, London.

Cavner W., Trenkle J. (1994) *N-gram Based Text Categorization,* In "Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval", 11-13 April, pp. 161-169.

Chomsky, N. (1990) *Three Models for the Description of Language,*
PGIT, pp. 113-124.

Clarke, R. (1993) *Optical Character Recognition Output Corrector,*
B.A. Final Year Project.
Trinity College, Dublin.

Clerin, B. (1992) *GPF Manual,*
GPF Systems, Inc..

Culligan, B. (1993) *Design of an on-line Database Query System for the 1872 Printed Library Catalogue,*
B.A.I. Final Year Project.
Trinity College, Dublin.

Cunningham P., Veale T., Conway A. (1992) *Knowledge Acquisition for Concept Indexing in Document Retrieval,*
Expert Systems for Information Management, Vol. 5, No. 1, pp. 25-41.

Cushman W., Pernundu O., Daniels D. (1991) *Usable OCR - What are the Minimum Requirements,* In "CHI-90 Conference Proceedings, Special Issue of ACM SIGCHI Bulletin", Seattle, April 1-5, pp. 145-151.

Devitt M., Sterenly K. (1987) *Language and Reality: an Introduction to the Philosophy of Language,*
Basil Blackwell, Oxford.

Edmunds N., Monckton P. (1995) *Getting Your Own Back,*
PC Magazine, Vol. 4, No. 4 , pp. 225-254.

Ellis, D.(1989) *A Behavioural Approach to Information Retrieval System Design,*
Journal of Documentation, Vol. 45, No. 3, pp. 171-212.

Fiz Technik (1991) *SHERLOCK - The New On-line Information System for Business and Industry,*
Fachinformationszentrum Technik, Frankfurt.

Fox, P. (1986) *Treasures of the Library,*
Trinity College, Dublin.

Fujisawa, H. (1994) *Character Recognition Technologies and Applications - the Current Status and the Future,*
Central Research Laboratory, Hitachi Ltd., Tokyo.

Fujisawa H., Kato K., Kojima K., Tomohiro S., Wakayama S. (1994) *Concept Centered Document Management and Full-text Search for Information Sharing,* In "International Federation for Information and Documentation, 47th FID Conference and Congress" Sonic City Omiya, Saitama, Japan, Oct 5-8, pp. 1-8.

Fujisawa H., Shima Y., Masashi K. (1992) *Automatically Organizing Document Bases Using Document Understanding Techniques,* In "Proceedings of the Second Far-East Workshop on the Future Database Systems", (Ed.) Chen,Q. Kyoto, Japan, April 26-28 pp. 244-253.
World Scientific, Singapore.

Gimpel, R. (1973) *A Theory of Discrete Patterns and their Implementation on Snobol*
CACM, Vol. 16, No. 2, pp. 91-100.

Glantz, H. (1957) *On the Recognition of Information with a Digital Computer,*
JACM, Vol. 57, No. 4, pp. 178-188.

Gotlieb C., Kuman S. (1968) *Semantic Clustering of Index Terms,*
JACM, Vol. 15, No. 4, pp. 493-513.

Griffith, R. (1982) *Three Principles of Representation for Semantic Networks,*
ACM Transactions on Database Systems, Vol. 7, No. 3, pp. 417-442.

Griswold R., Poage J., Polonsky I. (1971) *The Snobol4 Programming Language,*
Prentice-Hall, London.

Hafter, R. (1979) *The Performance of Card catalogue: A Review of Research,*
Library Research, Vol. 1, pp. 199-222.

Halpern J., Sargeant H. (1988) *A New End-User Interface for Bilingual Searching of Medline,*
Online Information, Vol. 2, pp. 427-444.

Hildreth C. (1983) *To Boolean or Not to Boolean,*
Information Techniques & Libraries, Vol. 2, No. 3, pp. 235-237.

Ingle, N. (1976) *A Language Identification Table,*
The Incorporated Linguist, Vol. 15, No. 4, pp. 98-101.

Ingwersen, P. (1992) *Information Retrieval Interaction,*
Taylor Graham, London.

Jackson, P. (1990) *Introduction to Expert Systems,*
Addison Wesley, Wokingham.

Katzner, K. (1986) *The Languages of the World,*
Routledge, London.

Khoo C., Poo D. (1994) *An Expert System Approach to On-line Catalog Subject Searching,*
Information Processing & Management, Vol. 30, No. 2, pp. 223-238.

Kinane V., O'Brien A. (1988) *The Vast Difficulty of Cataloguing : The Printed Catalogue of Trinity College, Dublin (1864-1887),*
Libraries & Journals: A Journal of Library History, Vol. 23, No. 4, pp. 427-453.

Knuth D. (1981) *The Art of Computer Programming,* 2nd ed.
Addison Wesley, London.

Kreiling, F. (1968) *Leibniz,*
Scientific American, Vol. 218, No. 5, pp. 90-100.

Kulikowski, S. (1991) *Using Short Words: A Language Identification Algorithm,*
Unpublished Technical Report,
Provided by Fuji Xerox Palo Alto Laboratory.

Lancaster, F. (1977) *The Measurement and Evaluation of Library Services,*
Information Resources Press, Washington D.C..

Lancaster F., Fayer E. (1973) *Information Retrieval On-Line,*
Melville, California.

Larsen, H. (1987) *KIWI - Knowledge Based User-Friendly System for the Utilisation of Information Bases,* In "Knowledge Engineering", (Ed.) Wormell, I..
Taylor Graham, London.

Lee, J. (1994) *Properties of Extended Boolean Models in Information Retrieval,* In "Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval"
(Dublin City University), Dublin, July 3-6, pp. 182-192.
Springer Verlag, London.

Markey K., Visine-Goetz D. (1988) *Increasing the Accessibility of Library of Congress Subject Headings in On-line Bibliographic Systems,*
Annual Review of OCLC Research, pp. 32-34.

Martin J., Oxman S. (1988) *Building Expert Systems - a Tutorial,*
Prentice-Hall, London.

Matthews J., Lawrence G., Ferguson D. (1988) *Using On-line Catalogs: A Nationwide Survey,*
Neal-Schuman, New York.

Matthews, P. (1991) *Morphology,*
Cambridge University Press, Cambridge.

Montgomery, C. (1972) *Linguistics and Information Science,*
Journal of the American Society for Information Science, Vol. 2, No. 3, pp. 195-219.

Morris, A. (1992) *Overview of Expert Systems,* In "The Application of Expert Systems in Libraries and Information Centres", (Ed.) Morris, A., pp. 1-34.
Bowker Sauer Ltd., London.

Nebendahl, D. (1980) *Expert Systems,*
John Wiley & Sons Ltd., Chichester.

Noreault T., Koll M., McGill M. (1977) *Automatic Ranked Output from Boolean Searches in SIRE,*
Journal of the American Society for Information Science, Vol. 28, No. 6, pp. 333-339.

O'Grady W, Dobrovolsky M., Aronoff M. (1989) *Contemporary Linguistics,*
St. Martin's Press, New York.

Patterson D. (1990) *Introduction to Artificial Intelligence and Expert Systems,*
Prentice-Hall, New Jersey.

Peat H., Willett P. (1991) *The Limitations of Term Co-occurrence Data for Query Expansion in Document Retrieval Systems,*
Journal of the American Society for Information Science, Vol. 42, No. 5, pp. 378-383.

Price, C. (1990) *Knowledge Engineering Toolkits,*
Ellis Horwood Ltd., Chichester.

Renfrew, C. (1994) *World Linguistic Diversity,*
Scientific American, Vol. 270 , No. 1 (January), pp. 104-110.

Rouse W., Rouse S. (1984) *Human Information Seeking and Design of Information Systems,*
Information Processing and Management, Vol. 20, No. 1-2, pp. 129-135.

Salton, G. (1971) *The SMART Retrieval System,*
Prentice-Hall, New Jersey.

Salton, G. (1975*) Dynamic Information and Library Processing,*
Prentice-Hall, Inc., London.

Salton, G. (1983) *Introduction to Modern Information Retrieval,*
McGraw-Hill, Inc., London.

Salton G., Fox E., Wu H. (1983) *Extended Boolean Information Retrieval,*
CACM ,Vol. 26, No. 12, pp. 1022-1036.

Saracevic T. (1987) *Changing Agenda for Information Retrieval,* (Editorial)
Information Processing & Management, Vol. 23, No. 5, pp. i-ii.

Saussure, F. de (1915/1959) *Cours de Linguistique Generale, Paris.*
English Translation by Baskin, W., "Course in General Linguistics",
Fontana, London.

Sell, P. (1985) *Expert Systems - a Practical Introduction,*
Macmillan, Basingstoke.

Seymour C, Schofield J. (1973) *Measuring Reader Failure at the Catalogue,*
Library Resources and Technical Services, Vol. 17., pp. 6-24.

Shastri, L. (1988) *Semantic Networks - an Evidential Formalization and its Connectionist Realization,*
Pitman, London.

Shneiderman, B. (1987) *Designing the User Interface,*
Addison-Wesley, Massachusetts.

Sibun P., Spitz A. (1994) *Language Determination: Natural Language Processing from Scanned Document Images,*
Submitted to ANLP-94. Provided by the author.

Smeaton A., van Rijsbergen C. (1983) *The Retrieval Effects of Query Expansion on a Feedback Document Retrieval System,*
Computer Journal, Vol. 26, No. 3, pp. 239-246.

Smith F., Devine K. (1983) *QUILL - an On-Line Retrieval System,*
Queen's University, Belfast.

Sowizral, M. (1985) *Expert Systems,*
Annual Review of Information Sciences and Technology (ARIST), Vol. 20, pp. 179-199.

Spink, A. (1994) *Term Relevance Feedback and Query Expansion: Related to Design,* In "Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval"
(Dublin City University), Dublin, July 3-6, pp. 81-91.
Springer Verlag, London.

Stroustrup, B. (1991) *The C++ Programming Language,*
Addison-Wesley, Wokingham.

Sutcliff, A. (1988) *Human-Computer Interface Design,*
Macmillan, London.

Taghva K., Borsack J., Condit A. (1994) *Results of Applying Probabilistic IR to OCR Text,* In "Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval"
(Dublin City University), Dublin, July 3-6, pp. 202-211.
Springer Verlag, London.

Taghva K., Borsack J., Condit A., Erva S. (1994) *The Effects of Noisy Data on Text Retrieval,*
Journal of the American Society for Information Science, Vol. 45, No. 1, pp. 50-58

Taylor, R. (1968) *Question Negotiation and Information Seeking in Libraries,*
College and Research Libraries, Vol. 29, pp. 178-194.

Tseng, G. (1992) *Expert Systems in On-line Information Retrieval,* In "The Application of Expert Systems in Libraries and Information Centres", (Ed.) Morris, A., pp. 167-194.
Bowker Sauer Ltd., London.

Tuhrim S., Reggia J., Floor M. (1988) *Expert System Development : Letting the Domain Specialist Directly Author Knowledge Bases,* In "Expert Systems - the User Interface" (Ed.) Hendler, J., pp. 37-57.
Ablex Publishing Corporation, New Jersey.

Turk C., Powers D. (1989) *Machine Learning of Natural Language,*
Springer Verlag, London.

Vadera, S. (1989) *Expert System Applications,*
Sigma Press, Wilmslow.

van Rijsbergen C. (1975) *Information Retrieval,*
Butterworth & Co. Ltd., London.

Vickery B., Vickery A. (1993) *On-line Search Interface Design,*
Journal of Documentation, Vol. 49, No. 2, pp. 103-187.

Voorhees, E. (1994) *Query Expansion using Lexical Semantic Relations,* In "Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval" (Dublin City University), Dublin, July 3-6, pp. 61-71.
Springer Verlag, London.

Weiss S., Kulikowski C. (1984) *A Practical Guide to Designing Expert Systems,*
Rowman & Allenheld, New Jersey.

Willet P., Ingwersen P. (1994) *An Introduction to Information Retrieval,*
Tutorial for the 17th International Conference on Research and Development in Information Retrieval, (Dublin City University), Dublin, July 4th.

Yazdani, M. (1993) *Multilingual Multimedia - Bridging the Language Barrier with Intelligent Systems,*Intellect, Oxford.

APPENDIX A:

List of Author Title Abbreviations used In Automatic Query Expansion. (Taken from Chambers Twentieth Century Dictionary)

A. B.	(Bachelor of Arts)
A. C.	(Before Christ)
A. D.	(anno domini)
A. M.	(Master of Arts)
B. A.	(Bachelor of Arts)
B. C. L.	(Bachelor of Civil Law)
B. D.	(Bachelor of Divinity)
B. L.	(Bachelor of Law)
B. M.	(Bachelor of Medecine)
C. C. C.	(Corpus Christi College)
C. E.	(Civil Engineer)
C. J.	(Chief Justice)
D. C. L.	(Doctor of Civil Law)
D. D.	(Doctor of Divinity)
F. A. S.	(Fellow of the Society of Arts)
F. C. S.	(Fellow of the Chemical Society)
F. G. S.	(Fellow of the Geological Society)
F. M.	(Field Marshal)
F. R. A. S.	(Fellow of the Royal Astronomical Society)
F. R. C. S.	(Fellow of the Royal College of Surgeons)
F. R. G. S.	(Fellow of the Royal Geographical Society)
F. R. S.	(Fellow of the Royal Society)
F. S. A.	(Fellow of the Society of Arts)
F. S. E.	(Fellow of the Society of Engineers)
F. T. C. D.	(Fellow of Trinity College Dublin)
G. C. B.	(Grand Cross of the Bath; Knight)
G. C. H.	(Grand Cross of Hanover; Knight)
J. C.	(Jurisconsult)
J. P.	(Justice of the Peace)
J. U. D.	(Doctor of Canon and Civil Law)
K. C. B.	(Knight Commander of Bath)
K. C. H.	(Knight Commander of Hanover)
K. C.	(King's Council)
L. P.	(Lord Provost)
LL. B.	(Bachelor of Laws)
LL. D.	(Doctor of Laws)
LL. M.	(Master of Laws)
M. A.	(Master of Arts)
M. B.	(Bachelor of Medicine)
M. D.	(Doctor of Medicine)
M. P.	(Member of Parliament)
M. R. C. C.	(Member of the Royal College of Chemistry)
M. R. C. P.	(Member of the Royal College of Physicians)
M. R. C. S.	(Member of the Royal College of Surgeons)
M. R. G. S.	(Member of the Royal Geographical Society)
O. S. A.	(Order of Saint Agustine)
O. S. B.	(Order of Saint Benedict)
O. S. F.	(Order of Saint Francis)
P. P.	(Parish Priest)

Ph. D.	(Doctor of Philosophy)
ph. D.	(Doctor of Philosophy)
ph. d.	(Doctor of Philosophy)
Ph. d.	(Doctor of Philosophy)
Q. C.	(Queen's Council)
K. C.	(King's Council)
R.	(King \ Queen; Rex \ Regina)
R. A.	(Royal Academy)
R. C.	(Roman Catholic)
R. E.	(Royal Engineers)
R. N.	(Royal Navy)
SC. B.	(Bachelor of Science)
SC. D.	(Doctor of Science)
S. J.	(Society of Jesus (Jesuit))
S. L.	(Solicitor at Law)
S. M.	(Sergeant Major)
S. S.	(Saints)
S. T. B.	(Bachelor of Theology)
S. T. D.	(Doctor of Theology)
S. T. M.	(Master of Theology)
S. T. P.	(Professor of Theology)
T. C. D.	(Trinity College Dublin)
T. D.	(Territorial Decoration)
Th. D.	(Doctor of Theology)
U. J. D.	(Doctor of Canon and Civil Law)
abp.	(Archbishop)
adm.	(Admiral)
archb.	(Archbishop)
archbp.	(Archbishop)
archd.	(Archdeacon)
archeveque	(Archbishop)
archiep.	(Archbishop)
archiepisc.	(Archbishop)
archiepiso	(Archbishop)
bp.	(Bishop)
capt.	(Captain)
card.	(Cardinal)
coll.	(College)
com.	(Commander)
commr.	(Commander)
dr.	(Doctor)
doct.	(Doctor)
doc.	(Doctor)
duc.	(Duke)
dux.	(Duke)
episc.	(Bishop)
episcopus	(Bishop)
eveque	(Bishop)
esq.	(Esquire)
facult.	(faculty)
knt.	(Knight)
kt.-bav.	(Knight)
kt.	(Knight)
lieut.-col.	(Lieutenant Colonel)
lieut.-general	(Lieutenant General)
lieut.	(Lieutenant)
maj.-gen.	(Major General)
maj.	(Major)
marq.	(Marquis)

mart.	(Martyr)
med. pr.	(Medical Practitioner)
med.	(Medical)
miss.	(Missionary)
ord. frat. min.	(Order of Friars Minor; Franciscans)
ord. praed.	(Order of Preachers; Dominicans)
ord. S. August.	(Order of Saint Augustine)
ord. S. Bened.	(Order of Saint Benedict)
ord. S. Franc.	(Order of Saint Francis)
parl.	(Parliament)
phil.	(Philosophy)
phys.	(Physiology)
presb.	(Presbyterian)
prof.	(Professor)
rex.	(King)
regina.	(Queen)
rom. cath.	(Roman Catholic)
rt. hon.	(Right Honourable)
sen.	(Senior)
sr.	(Sister)
vic.	(Vicar)
visct.	(Viscount)
col.	(Colonel)
A.	(Academician)
D.	(Doctor)
J.	(Judge)
S.	(Saint)

APPENDIX B:

Extracting Translation Rules from Author Names (Snobol4 Program)

```
*-INCLUDE "D:\SPT\host.inc"
      &TRIM = 1
*This program searches the author field of each entry searching
*for any author fields containing alternate author names
*e.g.  "WADDING seu WADDINGUS "
*The program extracts rules from the alternate author names so
*that one name can be mapped to the other
*In the above example NULL -> "US"
*A list of these rules is then compiled and stored in the file "srules.txt"

        DEFINE('FINDRL(X,Y)')
         DEFINE('CLERA(AUTHNME)')
        TERMINAL = 'enter name of file containing file names'
ASK     FILENAME = TERMINAL
        INPUT('FLE', 1, FILENAME)
        INPUT('SEU', 2, "C:\SPT\SRULES.TXT")
        OUTPUT('OUT', 4, "C:\SPT\SRULES.DAT")              :F(NOOUT)
        DIGIT = ('0123456789')
        PUNCT = ('.,\:- ')
        CAPS = &UCASE
        CHARS = &LCASE &UCASE DIGIT PUNCT
        LISTSEU = TABLE()
        seu1 = (" sive" | " or" | " ou" | " seu" | " [hod.")
        MATCH = BREAK(CAPS) SPAN(CAPS) ("," | "") SEU1
        MATCHLST = BREAK(CAPS) SPAN(CAPS)
        SRNMS = MATCH . AUTHOR1 (MATCH . AUTHOR2  | "")
+       (MATCHLST . AUTHOR3 | "")
************************
* Start of processing *
************************
*Read in seu list so far
LIST1   ONE = SEU                                          :F(READ)
        LISTSEU<ONE> = 1                                   :(LIST1)
*Read in file names to be processes
READ    FILENAME1 = FLE                                    :F(OUT)
        EXTEN = LEN(3) . EXT RPOS(0)
        EXT = EXT
        NME = RPOS(8) LEN(4) . FN
        FILENAME1 EXTEN :S(RR1)
RR1     FILENAME1 NME   :S(RR2)
RR2     FNEXT = FN '.' EXT
        TERMINAL = FNEXT
*Input each file
        INPUT('INQ',3, FNEXT)                              :F(NOFILE)


*Produce an output file for each input file
RDLN    ENTRY = INQ                                        :F(NXTF)
        ENTRY "A: " REM . AUTH                             :F(RDLN)
```

```
*Check for seu in author surname field
          AUTH SEU1                                       :F(RDLN)
          AUTH SRNMS                                      :F(RDLN)
          IDENT(AUTHOR2,NULL)                             :F(CHK12)
          FINDRL(AUTHOR1,AUTHOR3)                         :(CLR)
CHK12 FINDRL(AUTHOR1,AUTHOR2)
          IDENT(AUTHOR3,NULL)                             :S(CLR)
          FINDRL(AUTHOR1,AUTHOR3)
CLR      AUTHOR1 = AUTHOR2 = AUTHOR3 =                    :(RDLN)
NXTF    ENDFILE(3)                                        :(READ)
OUT      LISTSEU = CONVERT(LISTSEU, 'ARRAY')
          LISTSEU = SORT(LISTSEU)
          I = 1
AGAIN  OUT = LISTSEU<I,1> " : " LISTSEU<I,2>  :F(CLSF)
          I = I + 1                                       :(AGAIN)
CLSF    ENDFILE(4)                                        :(END)
NOOUT TERMINAL = "Cannot output to file"                  :(END)
NOFILE TERMINAL = 'File does not exist'                   :(END)
*---------------------------------------------------
*This function finds rules for mapping one author name to the other
FINDRL AUTH1 = X
          AUTH2 = Y
          COMMON = ""
          BEGIN = 'FALSE'
          MIDDLE = 'FALSE'
          ENDA = 'FALSE'
          BEG1 = BEG2 = ""
          END1 = END2 = ""
          MID1 = MID2 = ""
*Clear author names of spaces
          AUTH1 = CLERA(AUTH1)
          AUTH2 = CLERA(AUTH2)
          TERMINAL = "1 " AUTH1 " ; 2 " AUTH2
REST    ONE = TWO =
          AUTH1 TAB(1) . ONE =                            :F(DIFSTR)
          AUTH2 TAB(1) . TWO =                            :F(DIFSTR)
*Compare letter
          IDENT(ONE,TWO)                                  :F(DIFSTR)
          COMMON = COMMON ONE
          MIDDLE = 'TRUE'                                 :(REST)
*If there are no more letters in 1st string then map null to remaining
*characters in 2nd string to make a rule
DIFSTR  IDENT(AUTH1,NULL)                                 :F(TEST2)
          IDENT(AUTH1,AUTH2)                              :F(EMPT1)
          ENDING1 = ONE
          ENDING2 = TWO
          ENDA = 'TRUE'                                   :(MKRL)
EMPT1  ENDING1 = ONE "#"
          ENDA = 'TRUE'
          ENDING2 = TWO AUTH2                             :(RULEE)
TEST2   IDENT(AUTH2,NULL)                                 :F(DIFF2)

*If there are no more letters in 2nd string then map null to remaining
*characters in 1st string to make a rule
EMPT2  ENDING2 = TWO "#"
          ENDA = 'TRUE'
          ENDING1 = ONE AUTH1                             :(MKRL)
DIFF2   IDENT(MIDDLE,'TRUE')                              :S(MIDS)
*If there have been no letters in common then the first letters of
```

```
*the author names are different and need to be recorded
        BEG1 = BEG1 ONE
        BEG2 = BEG2 TWO
        BEGIN = 'TRUE'                                  :(REST)
MIDS    MID1 = MID1 ONE
        MID2 = MID2 TWO
        AUTH1 AUTH2                                     :F(OTHER)
        AUTH2 TAB(1) . TWO =                            :F(MKRL)
REP1    AUTH1 TAB(1) . ONE =                            :F(MKRL)
        IDENT(ONE,TWO)                                  :S(MKMD)
        MID1 = MID1 ONE                                 :(REP1)
OTHER   AUTH2 AUTH1                                     :F(ENDS)
        AUTH1 TAB(1) . ONE =                            :F(MKRL)
REP2    AUTH2 TAB(1) . TWO =                            :F(MKRL)
        IDENT(ONE,TWO)                                  :S(MKMD)
        MID2 = MID2 TWO                                 :(REP2)
MKMD    RULE2 = MID1 " \ " MID2
        RULE2B = MID2 " \ " MID1
        IDENT(LISTSEU<RULE2>,NULL)                      :S(SECB)
        LISTSEU<RULE2> = LISTSEU<RULE2> + 1             :(ENDFR)
SECB    IDENT(LISTSEU<RULE2B>,NULL)                     :S(ENTER)
        LISTSEU<RULE2B> = LISTSEU<RULE2B> + 1           :(ENDFR)
ENTER   LISTSEU<RULE2> = LISTSEU<RULE2> + 1             :(ENDFR)

ENDS    ENDING1 = ONE AUTH1
        ENDING2 = TWO AUTH2
        ENDA = 'TRUE'
MKRL    LEQ(BEGIN,'TRUE')                               :F(RLEE)
        RULE1 = BEG1 " \ " BEG2
        RULE1B = BEG2 " \ " BEG1
        IDENT(LISTSEU<RULE1>,NULL)                      :S(SECB1)
        LISTSEU<RULE1> = LISTSEU<RULE1> + 1             :(RLEE)
SECB1   IDENT(LISTSEU<RULE1B>,NULL)                     :S(ENTR1)
        LISTSEU<RULE1B> = LISTSEU<RULE1B> + 1           :(RLEE)
ENTR1   LISTSEU<RULE1> = LISTSEU<RULE1> + 1             :(RLEE)
RLEE    LEQ(ENDA,'TRUE')                                :F(ENDFR)
        RULE3 = ENDING1 " \ " ENDING2
        RULE3B = ENDING2 " \ " ENDING1
        TERMINAL = RULE3
        TERMINAL = RULE3B
        IDENT(LISTSEU<RULE3>,NULL)                      :S(SECB3)
        LISTSEU<RULE3> = LISTSEU<RULE3> + 1             :(ENDFR)
SECB3   IDENT(LISTSEU<RULE3B>,NULL)                     :S(ENTR2)
        LISTSEU<RULE3B> = LISTSEU<RULE3B> + 1           :(ENDFR)
ENTR2   LISTSEU<RULE3> = LISTSEU<RULE3> + 1             :(ENDFR)
ENDFR                                                   :(RETURN)
*----------------------------------------------------

*This function clears a string of spaces, commas and 'seu/sive' etc
CLERA   WORD = AUTHNME
        AUTHNME SEU1 =
SPSP    AUTHNME " " =                                   :S(SPSP)
        AUTHNME "," =
BGSPS   AUTHNME " " POS(0) =                            :S(BGSPS)
        CLERA = AUTHNME                                 :(RETURN)
END
```

List of Rules Produced in Descending Order of Frequency

IUS# : US# - 201
US# : - 139
IUS# : - 101
US# : E# - 90
US# : I# - 77
E : I - 72
Y : I - 71
I : E - 66
A : E - 58
E : A - 56
#C : #CH - 55
INUS# : US# - 53
CK : K - 49
AE : E - 48
O : U - 42
SIUS# : S# - 42
C : K - 41
IUS# : E# - 41
ST : T - 41
US# : O# - 41
L : LL - 37
IO : O - 35
C : CH - 34
IUS# : I# - 32
T : TT - 30
ERUS# : ER# - 27
E : IE - 26
S# : Z# - 26
AEUS# : US# - 25
I : EI - 24
LE : L - 24
A : AI - 23
T : R - 23
AEUS# : EUS# - 22
D : T - 22
E : AE - 22
S# : ES# - 22
TZ : Z - 22
#C : #K - 21
IA : A - 21
NS : S - 21
S : Z - 21
E : EI - 20
O : E - 20
S : T - 20
T : TH - 20
S : SS - 18
K# : KE# - 16
E : Y - 15
EUS# : E# - 14
IUS# : ER# - 14
U : OU - 14
V : W - 14
A : AU - 13
IUS# : ES# - 13
C : CK - 12
C : G - 12

D : L - 12
LO : L - 12
N : NN - 12
F : P - 11
IUS# : Y# - 11
NUS# : NI# - 11
Y# : IE# - 11
C : CC - 10
CIUS# : TIUS# - 10
E : EA - 10
AEUS# : - 9
E : EE - 9
F : FF - 9
IUS# : EN# - 9
S : CH - 9
TA : TE - 9
#C : #G - 8
E# : I# - 8
F : PH - 8
IER# : ER# - 8
R : RR - 8
#E : #HE - 7
AL : OL - 7
C : QU - 7
CUS# : CK# - 7
ERIUS# : IER# - 7
IUS# : IO# - 7
M : MM - 7
P : PP - 7
S : X - 7
Z : ZZ - 7
#C : #Z - 6
ART# : ARD# - 6
CUS# : KE# - 6
DE : T - 6
G : GG - 6
IS# : ES# - 6
IS# : ISIUS# - 6

APPENDIX C:

Language Tables used in the Language Recognition Program

ENGLISH	LATIN	FRENCH	DUTCH	GERMAN	ITALIAN	SPANISH	PORTUGUESE
the	de	de	van	von	di	de	pele
of	et	et	de	der	de	por	pelo
by	in	des	en	und	per	el	sua
and	per	par	in	die	da	la	em
in	ad	la	den	des	della	del	ao
to	cum	du	ende	in	et	espana	hum
on	ii	les	het	vnd	delle	en	çam
with	lat	le	een	zu	con	los	çao
from	ex	en	tot	uber	del	autores	dos
an	ibi	sur	der	mit	in	su	da
for	libri	roy	door	das	la	las	ar
or	vita	dv	op	den	gio	nueva	de
his	acc	avec	vande	ur	le	orden	do
church	ejus	france	met	dem	ramusio	ribera	delle
england	jo	es	zee	herrn	il	sancha	por
ireland	notis	sa	die	dr	roma	primera	nos
transl	ab	pour	staten	ein	sopra	tratado	os
sermon	liber	paris	vanden	vol	iu	-ana	ou
rev	scriptt	lettre	voor	zur	si	-res	si
history	sive	au	ter	an	dei	-ero	na
letter	jac	ses	uyt	so	gli	-era	no
at	opera	fran	is	oder	alla	-ria	sobre
life	actt	ou	dat	einer	citt	-les	nossa
notes	quae	un	uit	hagen	dell	-tio	que
ofthe	vett	dans	daer	jahrg	ital	-ada	para
new	auctore	trad	haer	ueber	nella	-tes	e
john	ac	pr	ofte	leiden	spagna	-eca	des
its	pro	ois	verhael	raumer	viaggi	-ica	-ados
english	hist	paix	aen	herausg	scriptt	-ado	
being	aliorum	voyage	des	im	io	-ico	
into	inter	qui	hem	aus	re	-ano	
lord	seu	lui	zeelant	fur	col	-tas	
sir	eo	louis	te	man	don	-ini	
as	part	ville	alle	san	note	-gio	
him	est	moires	heer	abth	isaac	-rca	
some	steph	lettres	west	luth	nuoua	-con	
dr	andr	av	heeft	mart	uiaggi	-ura	
th	rer	ce	musch	nach	volume	-las	
account	pars	me	stadt	denen	francia	-ade	
st	rerum	del	graven	marte	lettere	-ron	
upon	miscell	une	ghedaen	stadt	miscell	-sas	
letters	duo	vn	utrecht	unter	pittura	-nto	
book	germ	aux	na	bucher	ac	-ial	
that	poett	vie	nu	embden	al	-esi	
irish	qui	hist	of	ersten	ru	-eva	

ENGLSIH	LATIN	FRENCH	DUTCH	GERMAN	ITALIAN	SPANISH	PORTUGUESE
state	pauli	mars	als	goethe	che		
law	patrum	povr	toe	jungen	des		
sermons	bibl	trait	eene	naturw	nel		
is	thes	flandre	gene	schulz	piu		
present	etiam	recueil	jaar	seiner	sua		
works	notae	se	oude	worden	vera		
their	eiusdem	cour	copye	beitrag	degli		
people	an	sacr	haere	carmina	libro		
great	graec	entre	johan	freuden	nuova		
french	regum	publi	vader	sprache	-lla		
prayer	-rum	saint	vrede	-hen	-ella		
two	-orum	autres	deelen	-ung	-one		
king	-ium	egypte	eerste	-ben	-lle		
james	-iae	espagne	-nde	-chen	-elle		
italy	-tis	ont	-den	-gen	-ione		
other	-bus	que	-ten	-che	-oni		
answer	-ibus	roi	-oor	-den	-ioni		
-ing	-tio	-res	-ende	-eben	-ica		
-and	-bid	-ire	-hen	-ber	-ato		
-ons	-arum	-ent	-ren	-ten	-ere		
-ith	-nis	-que	-che	-her	-ell		
-land	-ita	-ires	-tie	-sche	-ggi		
-ent	-atio	-oire	-sche	-der	-aggi		
-ions	-tum	-tes	-ert	-nde	-nti		
-ers	-nes	-nce	-nge	-uch	-tio		
-rom	-num	-ues	-inge	-ner	-ani		
-ter	-one	-tre	-eren	-hte	-bid		
-ted	-ica	-les	-len	-chte	-sio		
-ment	-ione	-ant	-ande	-ers	-pra		
-ish	-ones	-ment	-nden	-ift	-usio		
-nce	-ris	-ance	-gen	-cher	-ali		
-the	-tus	-our	-chen	-ngen	-oli		
-ory	-onis	-ique	-ers	-rift	-ita		
-rch	-lis	-ques	-ien	-cht	-ore		
-mon	-ius	-ons	-ndt	-ler	-tte		
-tory	-erum	-urs	-andt	-sen	-ata		
-urch	-sis	-vec	-ven	-rrn	-tto		
-ies	-ini	-lle	-aer	-tung	-ria		
-cal	-que	-ions	-aten	-errn	-imo		
-tes	-iis	-ens	-ste	-sch			
-ical		-ers	-ene				
-ity		-eurs	-ken				
-son		-aire	-ant				
-ure		-eur					
-rmon		-des					

APPENDIX D:

Interface Evaluation Questionnaire - adapted from Shneiderman `87.

EVALUATION OF 1872 SEARCH SYSTEM INTERFACE FORM

Please circle the numbers that most appropriately reflect your impressions about using the 1872 search system.

You may add written comments on the extra sheet provided.

Your cooperation is appreciated.

On-line Help	confusing clear
	0 1 2 3 4 5 6 7 8 9 10
Learning to Search	difficult easy
	0 1 2 3 4 5 6 7 8 9 10
Use by Different Levels of Experience	not accommodated / accommodated
	0 1 2 3 4 5 6 7 8 9 10
Facilities Provided	not useful useful
Bibliography Manager	0 1 2 3 4 5 6 7 8 9 10
Language Dictionaries	0 1 2 3 4 5 6 7 8 9 10
Term Frequency	0 1 2 3 4 5 6 7 8 9 10
Librarian's Notes	0 1 2 3 4 5 6 7 8 9 10
Previous Searches	0 1 2 3 4 5 6 7 8 9 10
Subject Trees	0 1 2 3 4 5 6 7 8 9 10
Exploration of Facilities	discouraged encouraged
	0 1 2 3 4 5 6 7 8 9 10
Screen Display	confusing clear
	0 1 2 3 4 5 6 7 8 9 10
Menu Display	confusing clear
	0 1 2 3 4 5 6 7 8 9 10
Error Messages Helpful	never always
	0 1 2 3 4 5 6 7 8 9 10

Overall Reactions	terrible	wonderful
	0 1 2 3 4 5 6 7 8 9 10	
	frustrating	satisfying
	0 1 2 3 4 5 6 7 8 9 10	
	uninteresting	interesting
	0 1 2 3 4 5 6 7 8 9 10	
	difficult	easy
	0 1 2 3 4 5 6 7 8 9 10	

Thank you for your cooperation.

APPENDIX E:

SOURCE CODE LISTING - C++

(All include Files are Available on Request)

Searching the Database - Search for Author, Title and Series Keywords
Author has been broken up into Surname, FirstName and Title (e.g.S.J.)

Srchwin.Hpp

```
#ifndef _SEARCHWIN_HPP_
#define _SEARCHWIN_HPP_

#include "stdtyp.hpp"
#include <gpfparms.h>
#include "entryfld.hpp"
#include "refer.hpp"
#include "stdwin.hpp"
#include "hash.hpp"

class SearchWindow
{
        friend class LatWin;
        friend class TreeWin;
        friend class TermWin;
        private:
                EntryField efield, efield2, efield3;
                StdWin win;
                int entryFieldID, entryFieldID2, entryFieldID3;
                HashTable *pHashTable, *pHashTable2, *pHashTable3;
                Boolean authwin;
                char *pText, *pText2, *pText3;


        public:
                Boolean adjacent;
                int count;
                char *remadj[5];
                SearchWindow(int ID, HashTable &hashtable);
                SearchWindow(int ID, HashTable &hashtable, int ID2,
HashTable &hashtable2, int ID3, HashTable &hashTable3);
                void Init(HWND hwnd);
                void InitAthWin(HWND hwnd);
                        // Called when Clear button is clicked
                void Clear();
                        // Called when the OK button for the dialog
                        // is clicked - commences search
                void OK();
                void Completed();
};

extern SearchWindow authorWindow;
extern SearchWindow titleWindow;
extern SearchWindow seriesWindow;

#endif
```

SrchWin.Cpp

```
#include <iostream.h>
#include <math.h>
#include <string.h>
#include "stdtyp.hpp"
#include "library.ext"
#include "hash.hpp"
#include "refer.hpp"
#include "document.hpp"
#include "aux.hpp"
#include "library.ids"
#include "elist.hpp"
#include "error.hpp"
#include "map.hpp"
#include "snode.hpp"
#include "parse.hpp"
#include "srchwin.hpp"
#include "glob.hpp"
#include "line.hpp"
#include "expand.hpp"

//declare instances of search window classes
SearchWindow authorWindow(ID_EFSEARCHSURNME, authorHash,
ID_EFSEARCHFIRNME, firnmeHash, ID_EFSEARCHATHWIN, autitleHash);
SearchWindow titleWindow(ID_EFSEARCHTITLE, titleHash);
SearchWindow seriesWindow(ID_EFSEARCHSERIES, seriesHash);

static char fileName[MAXPATHLEN];

//Constructor to set up variables
SearchWindow::SearchWindow(int ID, HashTable &hashTable)
{
        pHashTable = &hashTable;
        entryFieldID = ID;
        pText = NULL;
        authwin = false;
}
//Overloaded function to set up variables for author search window.
//Author search window has three entry fields unlike those of title
//search window and series search window
//The three fields are for surname, firstname and author title (D.D.)
SearchWindow::SearchWindow(int ID, HashTable &hashTable, int ID2,
HashTable &hashTable2, int ID3, HashTable &hashTable3)
{
        pHashTable = &hashTable;
        pHashTable2 = &hashTable2;
        pHashTable3 = &hashTable3;
        entryFieldID = ID;
        entryFieldID2 = ID2;
        entryFieldID3 = ID3;
        pText = NULL;
        pText2 = NULL;
        pText3 = NULL;
        authwin = true;
}

//Initialize titel and series windows
void SearchWindow::Init(HWND hwnd)
{
        win.AttachHandle(hwnd);
        efield.AttachID(hwnd, entryFieldID);
        if (pText != NULL)
             efield = pText;
}
```

```
//Initialize authro window with three entry fields
void SearchWindow::InitAthWin(HWND hwnd)
{
        win.AttachHandle(hwnd);
        efield.AttachID(hwnd, entryFieldID);
        efield2.AttachID(hwnd, entryFieldID2);
        efield3.AttachID(hwnd, entryFieldID3);
        if (pText != NULL)
             efield = pText;
        if (pText2 != NULL)
             efield2 = pText2;
        if (pText3 != NULL)
             efield3 = pText3;
        authwin = true;
}



//Clear entry fields
void SearchWindow::Clear()
{
        efield = "";
        efield2 = "";
        efield3 = "";
}


//Execute Search
void SearchWindow::OK()
{
        Boolean found,loop1;
        Boolean one, two, three;
        EntryLoc entryLoc;
        Token token;
        Token tokexp;
        SearchNode *pRoot, *pRoot2, *pRoot3;
        SearchNode *pNode, *pPlus;
        char logs[150];
        char *word;
        char *sav1, *sav2, *sav3;
        char *stre;
        pText = efield;
        if (authwin == true)
           {
           pText2 = efield2;
           pText3 = efield3;
           }
        int r = 0;
        count = 0;
        adjacent = false;
        loop1 = true;
        one = two = false;

        sav1 = (char *)new char[20];
                  //stores adjacent word variables
        sav2 = (char *)new char[20];
        sav3 = (char *)new char[20];
        stre = (char *)new char[100];

//Log users search for Previous Search Results
//Attach labels to searches before they are logged
        if (entryFieldID == ID_EFSEARCHSURNME)
            {
            strcpy(logs,"author = ");
            strcat(logs, pText);       //surname
            strcat(logs, ", ");
            strcat(logs, pText2);      //first name
            strcat(logs, "; ");
            strcat(logs, pText3);       //author title
            }
        else
            {
            if (entryFieldID == ID_EFSEARCHTITLE)
                strcpy(logs,"title = ");
              else
                {
                if (entryFieldID == ID_EFSEARCHSERIES)
                    strcpy(logs,"series = ");
                }
             strcat(logs, pText);
             }
        strupr(logs);
        expand.LogSearch(logs);        //log users search

      if (pText != NULL)
       {
//Process each word of user's search.
        Segment segment(pText);
// get first token in search string
        word = segment.GetToken(token);
        if (word != NULL)
        {
              one = true;
              int i = strcmp(word, "+");
                  //adjacent keyword sign to avoid false drops
              if (i == 0)
                 {
                 errorMsg.Warn("+ - cannot occur in itital
position");
                 return;
                 }
              strcpy(sav1,word);
              remadj[r] = sav1;                //store 1st word
              count++;
              r++;
              pRoot = new WordNode(word, pHashTable);
              assert(pRoot != NULL);
              char *expword = expand.Ocrerrs(word);
              Segment seg(expword);
// if token expands than 'or' the results
              while ((expword = seg.GetToken(tokexp)) != NULL)
                     {
                     pNode = new WordNode(expword, pHashTable);
                     assert (pNode != NULL);
                     pRoot = new UnionNode(pRoot, pNode);
                     assert (pRoot != NULL);
                     }
                        // else get next token
              while ((word = segment.GetToken(token)) != NULL)
              {
                   i = strcmp(word, "+");
                        //check for adjacent sign
                   if (i == 0)
                     {
                      word = segment.GetToken(token);
                        //skip '+' sign
                      adjacent = true;
                      if (loop1 == false)
                         {
                         strcpy(sav3,word);
                         remadj[r] = sav3;
                        //store next word
```

```
                                }
                             else
                                {
                                 strcpy(sav2,word);
                                 remadj[r] = sav2;
                               //store next word
                                }
                             count++;
                             r++;
                            }
                         else
                            {
                              if (adjacent == false)
                                 {
                                 if (loop1 == false)
                                    {
                                    strcpy(sav3,word);        //2nd loop
                                    remadj[r - 1] = sav3;
                                    }
                                 else
                                    {
                                    strcpy(sav2,word);        //1st loop
                                    remadj[r - 1] = sav2;
                                    }
                                 }
                            }
                         pPlus = new WordNode(word, pHashTable);
                         assert (pPlus != NULL);
                         expword = expand.Ocrerrs(word);
                         Segment seg(expword);
                         while ((expword = seg.GetToken(tokexp)) != NULL)
                         {
                                   pNode = new WordNode(expword, pHashTable);
                                   assert (pNode != NULL);
                                   pPlus = new UnionNode(pPlus, pNode);
                                   assert (pPlus != NULL);
                          }
                         pRoot = new IntersectionNode(pRoot, pPlus);
                         assert (pRoot != NULL);
                         pPlus = 0;
                         loop1 = false;
                              //end of first time through loop
                    }
               }
           }

                if (pText2 != NULL && authwin == true)
                   {
                      Segment segment(pText2);
                      word = segment.GetToken(token);
                      if (word != NULL)
                         {
                          two = true;
                          pRoot2 = new WordNode(word, pHashTable2);
                          assert (pRoot2 != NULL);
                          char *expword2 = expand.Ocrerrs(word);
                          Segment seg(expword2);
                            // if token expands than 'or' the results
                          while ((expword2 = seg.GetToken(tokexp)) !=
NULL)
                           {
                             cout << "\n2 " << expword2;
                             pNode = new WordNode(expword2, pHashTable2);
                             assert (pNode != NULL);
                             pRoot2 = new UnionNode(pRoot2, pNode);
                             assert (pRoot2 != NULL);
```

```
            }
           while ((word = segment.GetToken(token)) != NULL)
           {
                 pPlus = new WordNode(word, pHashTable2);
                 assert (pPlus != NULL);
       }
          pRoot2 = new IntersectionNode(pRoot2, pPlus);
          assert (pRoot2 != NULL);
      } //while
         if (one == true)
             {
             pRoot = new IntersectionNode(pRoot, pRoot2);
             assert (pRoot != NULL);
             }
          } //word
   } //pText2

   if (pText3 != NULL && authwin == true)
    {
       Segment seg(pText3);
       char* expn = expand.ProcTitle(pText3);
       cout << "end1 " << expn << "#";
       strcpy(stre,expn);
       int i = strcmp(expn,"");
       cout << "i " << i;
       if (i > 0)
          {
          strcpy(pText3,stre);
          }
       Segment segment(pText3);
       word = segment.GetToken(token);
       cout << "word " << word;
       if (word != NULL)
          {
           three = true;
           pRoot3 = new WordNode(word, pHashTable3);
           assert (pRoot3 != NULL);
           while ((word = segment.GetToken(token)) != NULL)
           {
             cout << "word2 " << word;
             pPlus = new WordNode(word, pHashTable3);
             assert (pPlus != NULL);
             pRoot3 = new IntersectionNode(pRoot3, pPlus);
             assert (pRoot3 != NULL);
           } //while
          cout << "one " << one;
          cout << "two " << two;
          if (one == true)
             {
             pRoot = new IntersectionNode(pRoot, pRoot3);
             assert (pRoot != NULL);
             }
            else
                {
                if (one == false && two == true)
                 {
                  pRoot2 = new
                  IntersectionNode(pRoot2,pRoot3);
                  assert (pRoot != NULL);
                 }
                }
          } //word
   } //pText3

      cout << "\n one " << one;
      cout << "\n two " << two;
```

```
                    if (token == TOK_ERROR)
                       cout << "Invalid character detected when reading
token" << endl;
                    else
                       {
                       if (one == true)
                          elist.AsyncDisplayList(pRoot, &win);
                       else
                          {
                          if (two == true)
                             elist.AsyncDisplayList(pRoot2, &win);
                          else
                             {
                             if (three == true)
                                elist.AsyncDisplayList(pRoot3, &win);
                             }
                          }
                       }
```

Expanding Queries - to include OCR errors in users' queries, to include synonymns and mappings of place names in users' queries, to expand author titles.

<u>Expand.Hpp - Header File</u>

```
#ifndef _EXPAND_HPP_
#define _EXPAND_HPP_

#include "stdtyp.hpp"
#include "hash.hpp"
#include "stdwin.hpp"
#include "refer.hpp"
#include <gpfparms.h>


class Expand
{
        private:
                FILE *fp;
                HashTable *pHashTable;
                StdWin win;
                char *replce;
                char *ocrerr;
                char *ocrrpl;
                int t;
        public:
                int sofar;
                char *refarr1[10];
                char *refarr2[10];
                char *Ocrerrs(char *query);
                void LatAth();
                char *Locating(char *srchstn);
                char *ProcTitle(char *inits);
                char *AuTitleCmb(char *cmbinits);
                void LogSearch(char *searchl);

extern Expand expand;

#endif
```

Expand.Cpp

```
#include <iostream.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "elist.hpp"
#include "error.hpp"
#include "stdtyp.hpp"
#include "library.ext"
#include "library.ids"
#include "glob.hpp"
#include "hash.hpp"
#include "line.hpp"
#include "parse.hpp"
#include "expand.hpp"
#include "snode.hpp"

Expand expand;

static char fileName[MAXPATHLEN];
static char fileName2[MAXPATHLEN];
static char fileName3[MAXPATHLEN];
static char fileName4[MAXPATHLEN];
```

//OCR Errors

```
//This function expadns the query to include possible OCR errors
//It returns the query
char *Expand::Ocrerrs(char *query)
{
        char *lne;
        char *newlne;
        char *newstr;
        char *test;
        char *p;
        Boolean second;

        strcpy(fileName, glob.IndexDir());
        strcat(fileName, "ocrerrs");
        if ((fp = fopen(fileName, "r")) == NULL)
                {
                perror(fileName);
                return NULL;
                }
        newstr = (char *)new char[100];
        test = (char *)new char[MAXQUERLN];
        replce = (char *)new char[MAXQUERLN];
        lne = (char *)new char[MAXQUERLN];
        newlne = (char *)new char[MAXQUERLN];
        ocrerr = (char *)new char[5];
        ocrrpl = (char *)new char[5];


        strnset(newstr, 0, MAXQUERLN);
        while ((lne = line.GetLine(fp)) != NULL)
        {
                strcpy(test,query);
                second = false;
                if ((ocrerr = line.GetW(lne)) != NULL)
                   {
                     p = strstr(test,ocrerr);
                     int len1 = strlen(ocrerr);
```

```
               while (p != NULL)
                 {
                  t = 0;
                  while (test[t] != ocrerr[0])
                    {
                    replce[t] = test[t];
                    t++;
                    }
                  if (second == true)
                    {
                     replce[t] = test[t];
                     t++;
                     while (test[t] != ocrerr[0])
                       {
                        replce[t] = test[t];
                        t++;
                        }
                    }
                  int stp = t;
                  stp++;
                  newlne = line.RemWord(lne,ocrerr);
                  if ((ocrrpl = line.GetW(newlne)) != NULL)
                     {
                      int len2 = strlen(ocrrpl);
                        for (int i = 0; i < len2; i++)
                          {
                          replce[t] = ocrrpl[i];
                          t++;
                          }
                      if (len1 == len2)
                       {     //replacing same amount of letters
                       while (test[t] != 0)
                           {
                            replce[t] = test[t];
                            t++;
                            }
                         }
                      else if (len1 < len2)
                       {   //replacing one letter
                           //with more than one
                           while (test[stp] != 0)
                           {
                             replce[t] = test[stp];
                             t++;
                             stp++;
                           }
                         }  //replacing more than one
                            //letter with one letter
                        else
                         {
                             stp++;
                             while (test[stp] != 0)
                             {
                               replce[t] = test[stp];
                               t++;
                               stp++;
                              }
                          }
                       replce[t] = 0;
                       strcat(replce, " ");
                       strcat(newstr,replce);
          }
                  if (second == false)
                     {
                     p = strstr(replce, ocrerr);
                     if (p != NULL)
```

```
                                        {
                                         second = true;
                                         strcpy(test,query);
                                         strnset(replce,0,MAXQUERLN);
                                        }
                                     }
                                 else p = 0;
                                }
                            }
          }
fclose(fp);
return newstr;
}
```

//Translation Rules for Latinizing Author Names

```
//This function reads in the translation rules to Latinize an Author
Name
void Expand::LatAth()
{
        int i = 0;
        char *lin;
        lin = (char *)new char[20];
        char *rules[10];

        strcpy(fileName4, glob.IndexDir());
        strcat(fileName4, "seurules");
        if ((fp = fopen(fileName4, "r")) == NULL)
           {
           perror(fileName4);
           return;
           }
        while ((lin = line.GetLine(fp)) != NULL)
              {
              rules[i] = lin;
              i++;
              }
       return;
}
```

//Country of Publication Expansion

```
//This function tests to see if a place of publication is a country.
//If it is - the country is replaced with appropriate cities.
char* Expand::Locating(char *srchstn)
{
        char *loc, *newloc;
        char *expstr, *expqer;
        char *city, *country, *lines;
        char *found, *newfd;
        char *remn;
        char *locstr;
        char finale[250];
        char reff[10];
        char reff1[10];
        char spcr[50] = " ";
        char *locat;
        Boolean locbrak;
        Boolean plcbrak;
        Boolean set;
        Boolean endr;
        Boolean first;
        FILE *file;

//allocate space for new variables
        loc = (char *)new char[MAXSRCHLN];
        locstr = (char *)new char[MAXSRCHLN];
        newloc = (char *)new char[MAXSRCHLN];
        expqer = (char *)new char[MAXSRCHLN];
        city = (char *)new char[MAXQUERLN];
        country = (char *)new char[MAXQUERLN];
        lines = (char *)new char[MAXSRCHLN];
        expstr = (char *)new char[150];
        newfd = (char *)new char[150];
        locat = (char *)new char[150];
        remn = (char *)new char[MAXSRCHLN];

        locbrak = plcbrak = set = false;
        first = true;
        strcpy(expstr,srchstn);
        strcpy(locat, " OR LOCATION = ");
        strcpy(expqer,expstr);
        strcat(expstr, " @");        //mark end of string
//Clear variables

        strnset(loc, 0, MAXSRCHLN);
        strnset(locstr, 0, MAXSRCHLN);
        strnset(newloc, 0, MAXSRCHLN);
        strnset(newfd, 0, 150);
        strnset(finale, 0, 250);

//Look for a location entry in the search string
        if ((found = strstr(expstr, "(LOCATION = ")) == NULL)
           found = strstr(expstr, "LOCATION = ");
        while (found != NULL)
          {
//Find place of publication - ie. word that follows 'location ='
             strnset(newfd,0,150);      //clear var
             strnset(remn,0,MAXSRCHLN); //clear var
             strcpy(remn,found);
             endr = false;
             char *p = strtok(found, " ");  // skip 'location'
             strcat(newfd, p);
             strcat(newfd, " ");
             //check for bracket around location
             char *brk = strstr(p,"(");
             if (brk != NULL)
                 locbrak = true;
             p = strtok(NULL, " "); // skip = sign
             strcat(newfd, p);
             strcat(newfd, " ");
             p = strtok(NULL, " ");       //skip place name
             strcat(newfd, p);
             strcat(newfd, " ");
             strcpy(loc,p);               //save place name
             //check for bracket around place name
             brk = strstr(loc,")");
             if (brk != NULL)
                {                               //if there remove it
                   int len = strlen(loc);
                   loc[len - 1] = 0;
                   plcbrak = true;
                }
             p = strtok(NULL, " ");
             int ii = strcmp(p, "@");     //check for end of string
             if (ii == 0)
                {
                endr = true;
                }
              else
                {
                  strcpy(reff, p);
                char *rtt = strstr(newfd,reff);
                if (rtt == NULL)
                    p = strstr(remn,reff);
                else
                  {
                   strcat(spcr, reff);
                   p = strstr(remn,spcr);
                   p = strstr(p,reff);
                  }
                cout << "\nremstr" << p;
               }
//Open file where countries are mapped to cities
            strcpy(fileName2, glob.IndexDir());
            strcat(fileName2, "country");
            if ((file = fopen(fileName2, "r")) == NULL)
               {
               perror(fileName2);
               cout << "can't open file";
               }
//Get lines from 'country' file

            while ((lines = line.GetLine(file)) != NULL)
                 {
                  if (country = strtok(lines, " "))
                     {
                     int i = strcmp(country,loc);
                     cout << "country" << country << "@" << endl;
                     cout << "location" << loc << "@" << endl;
//If user's location and country are the same then expand the country
                     if (i == 0)
                        {
                         set = true;
                         cout << "matches " << endl;
                         if ((city = strtok(NULL, " ")) != NULL)
                            {
                             if (locbrak == true)
                                  strcpy(newloc, "((");
                             else strcpy(newloc, "(");
                             strcat(newloc, "LOCATION = ");
                             strcat(newloc,city);
                             while ((city = strtok(NULL, " "))!= NULL)
                                    {
                                    strcat(newloc, locat);
                                    strcat(newloc, city);
                                     }
                             if (city = strtok(NULL, "\n"))
                                    {
                                     strcat(newloc, locat);
                                     strcat(newloc, city);
                                     }
                             if (plcbrak == true)
                                  strcat(newloc, "))");
                             else strcat(newloc, ")");
                             cout << "newloc" << newloc << endl;
                             expqer=
                               line.ReplcWds(expstr,newfd,newloc);
                             strcat(finale,expqer);
                               } //if
                          } //if
                     } //if
                 } //while
          if (endr == true)
             {
             if (set == false)
                {
                strcat(finale, " ");
```

```
                    strcat(finale, expqer);
                    }

                int lf = strcspn(finale, "@");
                cout << "lf " << lf << endl;
                if (lf != NULL)
                    finale[lf] = 0;
                cout << "final1 " << finale << endl;
                strcpy(locstr,finale);
                return locstr;
                }
            if (set == false)
              {
                    if (first == true)
                       {
                       first = false;
                       strcat(finale,expqer);
                       strcpy(reff1,reff);
                       }
                    else
                       {
                       strcat(finale, reff1);
                       strcat(finale, " ");
                       strcat(finale,newfd);
                       }
                }
            else
                set = false;
            strcpy(expqer,p);           //process remainder of string
            strcpy(expstr,p);
            plcbrak = locbrak = false;
            if ((found = strstr(expstr, "(LOCATION = ")) == NULL)
               found = strstr(expstr, "LOCATION = ");
           } //while
           strcat(finale, expstr);
           int lf2 = strcspn(finale, "@");
           if (lf2 != NULL)
               finale[lf2] = 0;
           strcpy(locstr,finale);
           return locstr;
}


//Author Description Expansions
//This function expand the author title field initials
// The author field contains the author description e.g. doctor
//However these are often in initials e.g. S.J. = jesuit
//These initials are expanded in this function so that searching may
continue
//Called from author window search
char *Expand::ProcTitle(char *inits)
{
        char store[20];
        char *lnes;
        char result[200];
        char *found;
        char *abbrev;
        FILE *file;

        lnes = (char *) new char[200];
        strcpy(store, inits);
        strnset(result,0,200);
        strcpy(fileName3, glob.IndexDir());
        strcat(fileName3, "autitle");
                                    //open autitle file for reading
        if ((file = fopen(fileName3, "r")) == NULL)
```

```
            {                   //file contains expansions of initials
            perror(fileName3);
            errorMsg.Warn("Cannot open autitle file in d:\indexdir");
            return NULL;
            }

        while ((lnes = line.GetLine(file)) != NULL)
             {
             if (abbrev = strtok(lnes, " "))
                {
                int i = strcmp(abbrev,store);
                if (i == 0)
                   {
                   while (abbrev = strtok(NULL, " "))
                         {
                         strupr(abbrev);
                         strcat(result, abbrev);
                         strcat(result, " ");
                         }
                   cout << "\n res " << result;
                   return result;
                   }
                }
             }
return "";
}

//This function expand the author title field initials
// The author field contains the author description e.g. doctor
//However these are often in initials e.g. S.J. = jesuit
//These initials are expanded in this function so that searching may
continue
//Called from combination window search
char *Expand::AuTitleCmb(char *cmbinits)
{
        char *store, *store2;
        char *lnes;
        char *result, *retstr;
        char *found;
        char *abbrev, *change;
        char *atitle;
        char autle[300] = "(AUTITLE = ";
        FILE *file;

        lnes = (char *) new char[200];
        atitle = (char *) new char[30];
        result = (char *) new char[250];
        retstr = (char *) new char[250];
        change = (char *) new char[50];
        store = (char *) new char[250];
        store2 = (char *) new char[250];
        strnset(result,0,250);
        strnset(retstr,0,300);
        strnset(store,0,250);
        strnset(store2,0,250);
        strnset(atitle,0,30);
        strnset(change,0,50);

        strcpy(store, cmbinits);
        strcpy(store2, cmbinits);

        if ((found = strstr(store, "AUTITLE = ")) != NULL)
            {
            char *p = strtok(found, " ");    //skip 'autitle'
            strcat(change,p);
            strcat(change," ");
            p = strtok(NULL, " ");     //skip '='
            strcat(change,p);
            strcat(change," ");
            p = strtok(NULL, " ");     //get author title
            strcat(change,p);
            strcpy(atitle,p);

            strcpy(fileName3, glob.IndexDir());
            strcat(fileName3, "autitle");
                                      //open autitle file for reading
            if ((file = fopen(fileName3, "r")) == NULL)
               {          //file contains expansions of initials
               perror(fileName3);
               errorMsg.Warn("Cannot open autitle file");
               return store2;
               }
            while ((lnes = line.GetLine(file)) != NULL)
                {
                if (abbrev = strtok(lnes, " "))
                   {
                    int i = strcmp(abbrev,atitle);
                    if (i == 0)
                      {
                       if (abbrev = strtok(NULL, " "))
                                      //if one word append to result
                          {
                           strupr(abbrev);
                           strcat(result, abbrev);
                          }
                       if (abbrev = strtok(NULL, " "))
                                      //if second word, add bracket
                          {
                           strupr(abbrev);
                           strcat(autle,result);
                           strcat(autle, " AND AUTITLE = ");
                           strcat(autle, abbrev);

                           while (abbrev = strtok(NULL, " "))
                              {
                              strupr(abbrev);
                              strcat(autle, " AND AUTITLE = ");
                              strcat(autle, abbrev);
                              }
                           strcat(autle, ")");
                           retstr =
                              line.ReplcInits(store2,change,autle);
                           return retstr;
                          }
                       retstr = line.ReplcInits(store2,atitle,result);
                       return retstr;
                    } //if  (i)
                  } //if  (abbrev)
              } //while (lnes)
         } //if (found)
return store2;
}
```

//Log Users Search for Use in Previous Search Results

```
void Expand::LogSearch(char *searchl)
{
        FILE *fpl;
        char *srchlne;

        srchlne = (char *)new char[150];
        strcpy(srchlne, "\n");
        strcat(srchlne,searchl);
        if ((fpl = fopen("logsearch.txt", "a")) == NULL)
           {
           errorMsg.Warn("Cannot open file LOGSEARCH.TXT");
           return;
           }

        fputs(srchlne, fpl);
        fclose(fpl);
        delete srchlne;
}
```

User Tools Code - Term Frequency / Previous Search Results / Librarians Notes

Tools.Hpp

```
#ifndef _TOOLS_HPP_
#define _TOOLS_HPP_

#include "stdwin.hpp"
#include "stdtyp.hpp"
#include "entryfld.hpp"
#include "line.hpp"
#include "listbox.hpp"
#include "mle.hpp"

class TermWin
{
        friend class TreeWin;
        friend class LatWin;
        private:
                EntryField tfefield, htefield;
                StdWin win;
                Boolean inittf;
                char *tfText;
                char *htText;
                int nmhits;

        public:
                TermWin() { inittf = false; }
                void Init(HWND hwnd);
                int TermFrq(char *term);
                int AthTermFrq(char *term);
};

extern TermWin termWin;

class NotesWin
{
        private:
                MLE mle;
                ListBox pslbox;
                StdWin win;
                Boolean ntsinits;
        public:
                NotesWin() {ntsinits = false;}
                void Init(HWND hwnd);
                void InitPS(HWND hwnd);
                void Show();
                void ShowPS();
                void Save();
};
```

```
extern NotesWin notesWin;
extern NotesWin prvsrchWin;

#endif
```

Tools.Cpp

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "library.ids"
#include "elist.hpp"
#include "error.hpp"
#include "glob.hpp"
#include "tools.hpp"
#include "parse.hpp"
#include "snode.hpp"
#include "srchwin.hpp"

TermWin termWin;
NotesWin notesWin;
NotesWin prvsrchWin;

//Initialize Term Frequency Window
void TermWin::Init(HWND hwnd)
{
        win.AttachHandle(hwnd);
        tfefield.AttachID(hwnd, ID_EFTERMFREQ);
        htefield.AttachID(hwnd, ID_EFNUMHITS);
        cout << "in init termf" << endl;
        nmhits = 0;
        tfText = NULL;
        htText = NULL;
        inittf = true;
}


int TermWin::TermFrq(char *term)
{
        SearchNodePtr pNode, pRoot;
        Token token;
        char buffer[10];
        char *word;
        char *tText;

        tText = (char *)new char[30];
        strcpy(tText,term);

        Segment segment(tText);
        word = segment.GetToken(token);
        if (word != NULL)
           {
           elist.nodisplay = true;
           pRoot = new WordNode(word,authorWindow.pHashTable);
           assert(pRoot != NULL); //in author field
           pNode = new WordNode(word,authorWindow.pHashTable2);
           assert(pNode != NULL);
           pRoot = new UnionNode(pRoot,pNode);
           assert(pRoot != NULL);
           pNode = new WordNode(word,titleWindow.pHashTable);
           assert(pNode != NULL); //in title field
           pRoot = new UnionNode(pRoot,pNode);
           assert(pRoot != NULL);
           pNode = new WordNode(word,seriesWindow.pHashTable);
```

```
            assert(pNode != NULL); //in series field
            pRoot = new UnionNode(pRoot,pNode);
            assert(pRoot != NULL);
            elist.AsyncDisplayList(pRoot,&win);
            if (elist.addentry == true)
               {   //return number of hits found if any
               nmhits = elist.freqcnt;
               return nmhits;
               }
            }
return 0;
}

//This function looks for a term in author field only.
//It is used in latinizing names to see if translated names
// are legal - by checking to see if they are in the catalogue.
int TermWin::AthTermFrq(char *term)
{
        SearchNodePtr pNode, pRoot;
        Token token;
        char buffer[10];
        char *word;
        char *tText;

        tText = (char *)new char[30];
        strcpy(tText,term);

        Segment segment(tText);
        word = segment.GetToken(token); //get word
        if (word != NULL)
           {
           elist.nodisplay = true;
           pRoot = new WordNode(word,authorWindow.pHashTable);
           assert(pRoot != NULL); //lokk in author surname field
           elist.AsyncDisplayList(pRoot,&win);
           if (elist.addentry == true)
              {
              nmhits = elist.freqcnt;
              return nmhits;
              }
           }
return 0;
}

//Initialize Libraian's Notes Window
void NotesWin::Init(HWND hwnd)
{
        ntsinits = true;
        cout << "in init noteswin " << endl;
        win.AttachHandle(hwnd);
        mle.AttachID(hwnd, ID_MLENOTES);
}

//Show Notes on Catalogue
void NotesWin::Show()
{
        FILE *nf;
        char *lnes;

        lnes = (char *) new char[500];
        notesWin.mle.Clear();
        if (ntsinits == false)
           return;

        //open notes file
        if ((nf = fopen("notes.txt", "r")) == NULL)
```

```
                {
                errorMsg.Warn("Cannot open notes file");
                return;
                }

            while ((lnes = line.GetLine(nf)) != NULL)
              {
              notesWin.mle+=lnes;
              cout << lnes << endl;
              }
          fclose(nf);
}

//Save any additional notes made
void NotesWin::Save()
{
        FILE *nf;
        char *nText;
//mle buffer size = 500
        nText = (char *) new char [500];
        if (ntsinits == false)
            return;

        if ((nf = fopen("notes.txt", "w")) == NULL)
           {
           errorMsg.Warn("Cannot open notes file");
           return;
           }

        nText = notesWin.mle.GetText();
        fputs(nText,nf);
        fputs("\n", nf);
        fclose(nf);
}

//Initialize Previous Search window
void NotesWin::InitPS(HWND hwnd)
{
        ntsinits = true;
        cout << "in init previous srch " << endl;
        win.AttachHandle(hwnd);
        pslbox.AttachID(hwnd, ID_LBPRVSRCH);
}

void NotesWin::ShowPS()
{
        FILE *nf;
        char *lnes;

        lnes = (char *) new char[500];
        prvsrchWin.pslbox.Clear();

        if (ntsinits == false)
           return;




      // open file that logs search results
        if ((nf = fopen("logsearch.txt", "r")) == NULL)
           {
           errorMsg.Warn("Cannot open previous search file");
           return;
           }
        //display contents of file in previous search window
          while ((lnes = line.GetLine(nf)) != NULL)
            {
            prvsrchWin.pslbox+=lnes;
            cout << lnes << endl;
            }
         fclose(nf);
}
```

Knowledge Tree Construction Tool code

<u>TreeWin.Hpp</u>

```
#ifndef _TREEWIN_HPP_
#define _TREEWIN_HPP_

#include "stdwin.hpp"
#include "stdtyp.hpp"
#include "entryfld.hpp"
#include "listbox.hpp"

struct tNode
{
        char *label;
        tNode *next_sib;
        tNode *first_child;
};


class TreeWin
{
        private:
                ListBox treelistbox, viewlistbox;
                EntryField refield, pefield, cefield;
                EntryField frefield, fefield, bqefield;
                StdWin win;
                char *rText, *bqtext;
                char *cText, *pText, *all;
                tNode *arrles[20];
                char *treecd[20];
                tNode *mainroot, *at, *store;
                int subj, t;
                Boolean level2,found,parent,exist;
                Boolean start;

        public:
                TreeWin() {rText = NULL; found = false; parent =
false;}
                Boolean initroot;
                void Root();
                void AddChild();
                void InitR(HWND hwnd);
                void InitAddC(HWND hwnd);
                void AddNode(char *chdtext, tNode *parent);
                void AddFirstNode(char *chdtext, tNode *parent);
                void ChngParnt();
                void Back();
                void InitView(HWND hwnd);
                void FindSubj();
                void ListSubjs();
                void ViewTree(tNode *hdnode);
                void FindCInit(HWND hwnd);
                void FindConcept();
                void FindNode(tNode *fNode, char *ftext);
                tNode *FindPrev(tNode *fpNode);
                tNode *FindPrnt(tNode *cdNode);
                void DelNode();
                void InitSearchWin(HWND hwnd);
                void Query();
                void GetSubj(tNode *sNode, int s);
                void Search();
                char *Remhits(char *ntext);
                void Recursrch(tNode *rNode);
                void GoSearch(char *text);
};

extern TreeWin treeWin;

#endif
```

•

TreeWin.Cpp

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "library.ids"
#include "elist.hpp"
#include "error.hpp"
#include "parse.hpp"
#include "snode.hpp"
#include "srchwin.hpp"
#include "tools.hpp"
#include "treewin.hpp"

TreeWin treeWin;

//This function initializes the root window
void TreeWin::InitR(HWND hwnd)
{
        win.AttachHandle(hwnd);
        refield.AttachID(hwnd,ID_EFSUBJHEAD);
        initroot = true;
        exist = false;
}

//This function takes the text in the efield and attaches it
//to a new root node
void TreeWin::Root()
{
        char rhits[5];
        char *reText;
        rText = (char *) new char[30];
        reText = (char *) new char[30];
        tNode *froot;

        reText = refield;
        int len = strlen(reText);
        if (len > 3)
           {
            int num = termWin.TermFrq(reText);
            strcat(reText, " (");
            itoa(num,rhits,10);
            strcat(reText, rhits);
            strcat(reText, ")");
           }

        strcpy(rText,reText);

        if (mainroot == NULL)
            mainroot = new tNode;
        else
          {
          froot = mainroot->first_child;
          cout << "tst " << froot->label << endl;
          while (froot != NULL)
            {
            int i = strcmp(froot->label,reText);
            if (i == 0)
               {
               errorMsg.Warn("This subject already exists");
               exist = true;
               return;
               }
            else froot = froot->next_sib;
            }
          }
        char *mText = refield;
        AddFirstNode(mText,mainroot);
        froot = mainroot->first_child;
        cout << froot->label << endl;
        cout << "trxt " << rText << endl;
}

//This function initializes the addchild window
void TreeWin::InitAddC(HWND hwnd)
{
        win.AttachHandle(hwnd);
        pefield.AttachID(hwnd,ID_EFSUBJPRNT);
        cefield.AttachID(hwnd, ID_EFSUBJCHLD);
        treelistbox.AttachID(hwnd, ID_LBCHLDRN);
        tNode *cdNode;

        pefield = rText;
        treelistbox.Clear();
        t = 1;
        level2 = false;
        if (exist == true)
          {
          exist = false;
          cdNode = new tNode;
          cdNode = mainroot->first_child;
          int i = strcmp(cdNode->label,rText);
          while (i != 0)
              {
              cdNode = cdNode->next_sib;
              i = strcmp(cdNode->label,rText);
              }
          cdNode = cdNode->first_child;
          while (cdNode != NULL)
              {
              treelistbox += cdNode->label;
              cdNode = cdNode->next_sib;
              }
          }
}

//this function takes the childs name in the efield
//and calls a newprocedure to link that child to
//its parent.
void TreeWin::AddChild()
{
       tNode *parent;
       tNode *rtNode;
       int cmp;
       parent = new tNode;
       rtNode = new tNode;
```

```
        pText = pefield;
        cText = cefield;
        rtNode = mainroot->first_child;
        cout << "rtnode " << rtNode->label << endl;
        cout << "pText " << pText << endl;
        int i = strcmp(rtNode->label,pText);
        if (i == 0)
           AddNode(cText,rtNode);
        else
           {
           level2 = true;
           parent = rtNode->first_child;
           cmp = strcmp(parent->label,pText);
           while (cmp != 0)
                {
                parent = parent->next_sib;
                cout << parent->label << endl;
                cmp = strcmp(parent->label,pText);
                }
           AddNode(cText,parent);
           }
        cefield = "";
}

//This function adds the label given by chdtext to the node parent
void TreeWin::AddNode(char *chdtext, tNode *parent)
{
         tNode *child;
         char *cldtxt;
         char *chtext;
         char hits[5];
         char *sps = "   ";

         child = new tNode;
         cldtxt = (char *) new char[30];
         strcpy(cldtxt,sps);
         if (level2 == true)
            strcat(cldtxt,sps);
         strcat(cldtxt,chdtext);
         int len = strlen(chdtext);
         if (len > 3)
            {
             int num = termWin.TermFrq(cldtxt);
             strcat(cldtxt, " (");
             itoa(num,hits,10);
             strcat(cldtxt, hits);
             strcat(cldtxt, ")");
            }
         child->label = cldtxt;
         child->next_sib = parent->first_child;
         parent->first_child = child;
         child->first_child = NULL;
         treelistbox += child->label;
         treecd[t] = new char[strlen(cldtxt) + 1];
         strcpy(treecd[t],cldtxt);
         t++;
}

//This function adds the label given by chdtext to the node parent
void TreeWin::AddFirstNode(char *chdtext, tNode *parent)
{
         tNode *child;
         char *cldtxt;
         char *chtext;
         char hits[5];
        child = new tNode;
        cldtxt = (char *) new char[30];
        strcpy(cldtxt,chdtext);
        int len = strlen(chdtext);
        if (len > 3)
           {
            int num = termWin.TermFrq(cldtxt);
            strcat(cldtxt, " (");
            itoa(num,hits,10);
            strcat(cldtxt, hits);
            strcat(cldtxt, ")");
           }
        child->label = cldtxt;
        child->next_sib = parent->first_child;
        parent->first_child = child;
        parent->next_sib = NULL;
        child->first_child = NULL;
}

//This function changes the parent node to one of the children
//to allow new children to be added to that child.
void TreeWin::ChngParnt()
{
        int i = 0;
        char *ndtext;
        tNode *trnode;
        tNode *rtNode;

        ndtext = (char *) new char[30];
        strnset(ndtext,0,30);
        int rown = treelistbox.GetSelection();
        if (rown == -1)
            {
            errorMsg.Warn("No entry selected-select to continue");
            return;
            }
        rown++;
        cout << "row " << rown;
        rtNode = mainroot->first_child;
        if (rown != LIT_NONE)
           {
           trnode = rtNode->first_child;
           strcpy(ndtext,trnode->label);
           i = strcmp(ndtext, treecd[rown]);
           while (i != 0)
                {
                trnode = trnode->next_sib;
                strcpy(ndtext,trnode->label);
                i = strcmp(ndtext,treecd[rown]);
                }
           if (trnode != NULL)
               {
               pefield = trnode->label;
               treelistbox.Clear();
               }
           }
}
```

```
//This function moves back up the tree to display a higher level
void TreeWin::Back()
{
        tNode *tnode;
        tNode *rtNode;

        treelistbox.Clear();
        rtNode = mainroot->first_child;
        pefield = rtNode->label;
        tnode = rtNode->first_child;
        int tr = 1;
        while (tnode != NULL)
              {
              treelistbox += tnode->label;
              treecd[tr] = new char[strlen(tnode->label) + 1];
              strcpy(treecd[tr],tnode->label);
              cout << "tr" << tr << treecd[tr] << endl;
              tr++;
              tnode = tnode->next_sib;
              }
}


//This procedure initialises the view tree window
void TreeWin::InitView(HWND hwnd)
{
        tNode *rtNode;

        win.AttachHandle(hwnd);
        viewlistbox.AttachID(hwnd, ID_LBVIEWLIST);
        viewlistbox.Clear();
        cout << "init view " << endl;
        if (mainroot == NULL)
           errorMsg.Warn("No subject tree has been created");
        else
           {
           rtNode = mainroot->first_child;
           while (rtNode != NULL)
                 {
                 viewlistbox += rtNode->label;
                 rtNode = rtNode->next_sib;
                 }
           start = true;
           }
}


//This function finds the subject heading for a given subject
void TreeWin::FindSubj()
{
        tNode *rtNode;

        rtNode = mainroot->first_child;
        int where = viewlistbox.GetSelection();
        if (where != LIT_NONE)
           {
           while (where != 0)
                 {
                 rtNode = rtNode->next_sib;
                 where--;
                 }

          viewlistbox.Clear();
          viewlistbox += rtNode->label;
          at = rtNode;
          rtNode = rtNode->first_child;
```

```
            start = false;
            ViewTree(rtNode);
            }
}

//This procedure list all the subjec tress that have been built
void TreeWin::ListSubjs()
{
        tNode *rtNode;
        rtNode = mainroot->first_child;
        viewlistbox.Clear();
        while (rtNode != NULL)
              {
               viewlistbox += rtNode->label;
               rtNode = rtNode->next_sib;
               }
}

//This function uses a depth first search to traverse the tree
//displaying the labels of each of the nodes in the listbox.
void TreeWin::ViewTree(tNode *hdnode)
{
        if (hdnode != NULL)
           {
           viewlistbox += hdnode->label;
           ViewTree(hdnode->first_child);
           ViewTree(hdnode->next_sib);
           }
}

void TreeWin::FindCInit(HWND hwnd)
{
        win.AttachHandle(hwnd);
        fefield.AttachID(hwnd,ID_EFFINDCONC);
        frefield.AttachID(hwnd,ID_EFFINDRSLT);
}


void TreeWin::FindConcept()
{
        char *fText;
        char *srchText;
        char rhits[5];
        tNode *fdNode;
        char *pos = "Found!";
        char *neg = "Not found";

        if (mainroot == NULL)
           {
           errorMsg.Warn("The subject tree is empty");
           return;
           }
        srchText = (char *) new char[30];
        fdNode = new tNode;
        found = false;
        fText = fefield;
        strcpy(srchText,fText);
        cout << "ftext " << fText << endl;
        int len = strlen(fText);
        if (len > 3)
           {
           int num = termWin.TermFrq(fText);
           strcat(srchText, " (");
           itoa(num,rhits,10);
           strcat(srchText, rhits);
           strcat(srchText, ")");
```

```
            cout << "stext " << srchText << endl;
            }
         fdNode = mainroot->first_child;
         FindNode(fdNode,srchText);
         if (found == false)
            frefield = neg;
}


//This function checks to see if a subject already exists in the tree.
//It takes in a subject name and searches the tree breadth-first
//for a node with that label. If found - a boolean variable 'found' is
set
//to true and false otherwise.
void TreeWin::FindNode(tNode *fNode, char *ftext)
{
         char *fdText;
         char *spText;
         char *sps = "   ";
         fdText = (char *)new char[30];
         spText = (char *)new char[30];

         strcpy(spText,sps);
         strcat(spText,ftext);
         strcpy(fdText,ftext);
         cout << "fn fd" << fdText << endl;
         cout << "fn fd" << spText << endl;
         if (fNode != NULL)
            {
            int i = strcmp(fNode->label,fdText);
            if (i != 0 && fNode != NULL)
               {
               FindNode(fNode->next_sib,fdText);
               FindNode(fNode->first_child,spText);
               }
            if (i == 0)
               {
               found = true;
               cout << "before call" << endl;
               tNode *nde = FindPrnt(fNode);
               cout << "nde " << nde->label << endl;
               frefield = nde->label;
               }
            }
}


//This function finds previous node in the tree
tNode *TreeWin::FindPrev(tNode *fpNode)
{
         tNode *pNode;
         tNode *dad;

         pNode = new tNode;
         dad = new tNode;
         pNode = mainroot->first_child;
         dad = pNode;
         while (pNode != NULL)
         {
               if (pNode->first_child == NULL)
                  {
                  if (pNode->next_sib == fpNode)
                      {
                      cout << "ret" << pNode->label << endl;
                      return pNode;
                      }
```

```
                    else
                        {
                        if (pNode->next_sib != NULL)
                            pNode = pNode->next_sib;
                        else
                            pNode = dad->next_sib;
                        }

                    }
                else
                  {
                    dad = pNode;
                    pNode = pNode->first_child;
                    if (pNode == fpNode)
                        {
                        parent = true;
                        cout << "ret" << dad->label << endl;
                        return dad;
                        }
                  }
            }
return NULL;
}


//This function finds the parent of a node
tNode *TreeWin::FindPrnt(tNode *cdNode)
{
        tNode *pNode;
        tNode *dad;

        pNode = new tNode;
        dad = new tNode;
        pNode = mainroot->first_child;
        dad = pNode;
        if (dad == cdNode)
           return dad;
        while (pNode != NULL)
        {
              if (pNode->first_child == NULL)
                 {
                 if (pNode == cdNode)
                     return dad;
                 else pNode = pNode->next_sib;
                 }
              else
                {
                 dad = pNode;
                 pNode = pNode->first_child;
                 if (pNode == cdNode)
                     return dad;
                }
            }
return NULL;
}


//Deletes a node
void TreeWin::DelNode()
{

        int i = 0;
        char *ndtext;
        tNode *rtNode;
        tNode *curNode;
        tNode *pNode;
        pNode = new tNode;
        curNode = new tNode;
        ndtext = (char *) new char[30];
        strnset(ndtext,0,30);
        int rown = treelistbox.GetSelection();
        if (rown == -1)
            {
            errorMsg.Warn("No entry selected - select to continue");
            return;
            }
        rown++;
        rtNode = mainroot->first_child;
        if (rown != LIT_NONE)
           {
           curNode = rtNode->first_child;
           strcpy(ndtext,curNode->label);
           i = strcmp(ndtext, treecd[rown]);

            while (i != 0)
                 {
                 curNode = curNode->next_sib;
                 strcpy(ndtext,curNode->label);
                 i = strcmp(ndtext,treecd[rown]);
                 }
           if (curNode != NULL)
                {
                 pNode = FindPrev(curNode);
                 if (parent)
                    pNode->first_child = curNode->next_sib;
                 else
                    pNode->next_sib = curNode->next_sib;
                }
           Back();
         }
}

//Initialise the authomatic query formulation window
void TreeWin::InitSearchWin(HWND hwnd)
{
        win.AttachHandle(hwnd);
        bqefield.AttachID(hwnd,ID_EFBUILDQUERY);
        bqefield = "";
        cout << "init srch" << endl;
}

//Build a Query from the subject heading chosen by the user
void TreeWin::Query()
{
        tNode *bqNode;
        bqNode = new tNode;
        bqtext = (char *) new char[30];

        int whr = viewlistbox.GetSelection();
        cout << "start " << start << endl;
        cout << "where " << whr << endl;
        if (whr != LIT_NONE)
           {
           if (start == true)
              {
              bqNode = mainroot->first_child;
              while (whr != 0)
                {
                bqNode = bqNode->next_sib;
                whr--;
                }
              strcpy(bqtext,bqNode->label);
              store = bqNode;
              cout << "txt" << bqtext << endl;
              bqefield = bqtext;
              }
           else GetSubj(at,whr);
           }
delete at;
}


//Recursive function which finds all children subjects
void TreeWin::GetSubj(tNode *sNode, int s)
{
        if (sNode != NULL)
          {
          if (s == 0)
              {
              strcpy(bqtext,sNode->label);
              store = sNode;
              bqefield = bqtext;
              }
          else
              {
              s--;
              GetSubj(sNode->first_child,s);
              GetSubj(sNode->next_sib,s);
              }
         }
}


void TreeWin::Search()
{
        char *text;
        text = (char *)new char[30];
        all = (char *)new char[30];

        strcpy(all, " ");
        DismissCreatSearchWindow(false);
        DismissViewTreeWindow(false);

        if (store->first_child != NULL)
           {
           Recursrch(store);
           GoSearch(all);
           }
        else
           {
           text = Remhits(store->label);
           cout << "text" << text << "@" << endl;
           GoSearch(text);
           }
       cout << "all " << all << endl;
}

//Remove hits from subject string
char *TreeWin::Remhits(char *ntext)
{
       char *total;
       total = (char *) new char[30];
       strcpy(total,ntext);
       cout << "total1 " << total << endl;
       int i = strcspn(total,"(");
       cout << "i" << i << endl;
       if (i != 0)
          total[i - 1] = 0;
        cout << "total1 " << total << endl;
        return total;
}



//Recursive function to build query from tree
void TreeWin::Recursrch(tNode *rNode)
{
         char *text;
         text = (char *)new char[30];
         strnset(text,0,30);

         if (rNode != NULL)
            {
            text = Remhits(rNode->label);
            cout << "recur " << text << endl;
            strcat(all, text);
            Recursrch(rNode->next_sib);
            Recursrch(rNode->first_child);
            }
}

//Call search calss to execute the search
void TreeWin::GoSearch(char *text)
{
         SearchNodePtr pRoot, pPlus, pNode;
         Token token;
         char *word;
         char *gText;

         gText = (char *)new char[30];
         strcpy(gText, text);
         Segment segment(gText);
         if ((word = segment.GetToken(token)) != NULL)
              {
              cout << "word " << word << endl;
              pRoot = new WordNode(word, authorWindow.pHashTable);
              assert(pRoot != NULL);
              pNode = new WordNode(word, titleWindow.pHashTable);
              assert(pNode != NULL);
              pRoot = new UnionNode(pRoot, pNode);
              assert(pRoot != NULL);
              }
         while ((word = segment.GetToken(token)) != NULL)
              {
              cout << "word " << word << endl;
              pNode = new WordNode(word, authorWindow.pHashTable);
              assert(pRoot != NULL);
              pRoot = new UnionNode(pRoot, pNode);
              assert(pRoot != NULL);
              pNode = new WordNode(word, titleWindow.pHashTable);
              assert(pNode != NULL);
              pRoot = new UnionNode(pRoot, pNode);
              assert(pRoot != NULL);
              }
         elist.AsyncDisplayList(pRoot, &win);
}
```

Latinize Author Name Code

Latwin.hpp

```
#ifndef _LATWIN_HPP_
#define _LATWIN_HPP_
```

```
#include "stdwin.hpp"
#include "stdtyp.hpp"
#include "entryfld.hpp"
#include "line.hpp"
#include "listbox.hpp"

class LatWin
{
        private:
                ListBox latlistbox;
                EntryField efield;
                StdWin win;
                Boolean initlat;
                char *lText;
                char *rText;
                int sv,vr;

        public:
                char *latrules1[80];
                char *latrules2[80];
                char *saverls[100];
                char *validrls[100];
                LatWin() { initlat = false; }
                void Init(HWND hwnd);
                void FindCog();
                void ChgCog();
                void SaveChg(char *svword);
                char *RemSgn(char *word);
                void ReplcBeg(char *name, char *one, char *two);
                void ReplcEnd(char *name, char *one, char *two);
                void ReplcMid(char *latm, char *onem, char *twom);
                void Check(char *nwname);
};

extern LatWin latWin;

#endif
```

<u>Latwin.cpp</u>

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "library.ids"
#include "elist.hpp"
#include "error.hpp"
#include "glob.hpp"
#include "latwin.hpp"
#include "parse.hpp"
#include "snode.hpp"
#include "srchwin.hpp"
#include "tools.hpp"

LatWin latWin;

//Initialize Latinize Window
void LatWin::Init(HWND hwnd)
{
        win.AttachHandle(hwnd);
        latlistbox.AttachID(hwnd, ID_LBLATNAME);
        efield.AttachID(hwnd, ID_EFLATNAME);
        rText = NULL;

        if (authorWindow.pText == NULL)
```

```
                rText = authorWindow.efield;


         if ((rText == NULL) && (authorWindow.pText == NULL))
            {
              errorMsg.Warn("No author surname selected");
              DismissLatAthWindow(false);
              return;
            }
         else
            {
            if (rText != NULL)
                efield = rText;
            else efield = authorWindow.pText;

            lText = efield;
            initlat = true;
            sv = 0;
            vr = 0;
            cout << "init lat " << initlat << endl;
            for (int k = 0; k < 100; k++)
                        //allocate space for cognates array
                 saverls[k] = new char[30];
            for (int u = 0; u < 20; u++)
                        //allocate space for valid cognates array
                 validrls[u] = new char[30];
            latlistbox.Clear();
            FindCog();
            }
}

//Check for Translation Rules inthe Author Name
void LatWin::FindCog()
{
        char *rule1;
        char *rule2;
        char *found;
        char *rl1;
        char *rl2;
        char *gen;
        char *lattext;
        char *again;
        int posr1;
        int posr2;
        Boolean beg,end,emp,mid;
        Boolean first,secd,fndrl;

        lattext = (char *) new char[30];
        rule1 = (char *) new char[10];
        rule2 = (char *) new char[10];
        again = (char *) new char[30];

        strnset(lattext,0,30);

        secd = false;
        int i = 0;
        while (latrules1[i] != NULL)
           {
           beg = end = emp = mid = false;
                           //set all boolean vars to false
           first = fndrl = false;
            strcpy(lattext,lText);
           rule1 = latrules1[i];
                           //get rule from array
           rule2 = latrules2[i];
```

```
char *r = strstr(rule1,"#");
if (r == NULL)      //if '#' not found
   {                // rule occurs in  middle of name
    mid = true;
    if ((found = strstr(lattext,rule1)) != NULL)
         {
         ReplcMid(lattext,rule1,rule2);
         fndrl = true;
         first = true;
         }
    else
         {          //invert rule and try again
         if ((found = strstr(lattext,rule2)) != NULL)
            {
            ReplcMid(lattext,rule2,rule1);
            fndrl = true;
            }
         }
   }

if (r != NULL)  //if rule contains #
   {          // then remember where it occurs and strip it
   posrl = strcspn(rule1,"#");
   if (posrl == 0)
       {
       gen = RemSgn(rule1);        //remove # from rule
       if (gen == NULL)
           emp = true;
       else beg = true;
       rl1 = gen;                  //newrule = rl1
       }
   else
       {
       end = true;
       gen = RemSgn(rule1);        //remove # from rule
       rl1 = gen;                  //newrule = rl1
       }

   if (beg == true || end == true)  // if rule not empty
     {
         gen = RemSgn(rule2);
         rl2 = gen;
         found = strstr(lattext,rl1);
         if (found != NULL)
            {
            first = true;
            fndrl = true;
            if (beg == true)
               ReplcBeg(lattext,rl1,rl2);
            else
               ReplcEnd(lattext,rl1,rl2);
            }
         else
           {
           found = strstr(lattext,rl2);
           if (found != NULL)
              {
              fndrl = true;
              if (beg == true)
                  ReplcBeg(lattext,rl2,rl1);
              else
                  ReplcEnd(lattext,rl2,rl1);
              }
           }
     }
```

```
                  if (emp == true)
                    {
                    posr2 = strcspn(rule2,"#");
                    if (posr2 != NULL)
                       {
                       fndrl = true;
                       if (posr2 == 0)
                          beg = true;
                       else end = true;
                       gen = RemSgn(rule2);
                       rl2 = gen;
                       if (beg == true)
                          ReplcBeg(lattext,rl1,rl2);
                       else ReplcEnd(lattext,rl1,rl2);
                       }
                    }
              }
          //if rule has been found then
          //apply rule found to names already latinized
          if (fndrl == true)
            {
             int whr = 0;
             if (sv > 1)          //if names already found
                {
                 int mx = sv - 1;
                 while (whr != mx && sv <= 99)
                   {
                    strcpy(again,saverls[whr]);
                    if (mid == true)
                             //if rule occurs in middle of name
                       {
                       if (first == true)
                             //replc first with second
                            ReplcMid(again,rule1,rule2);
                       else ReplcMid(again,rule2,rule1);
                       }
                    if (emp == true)     //if mapping in rule is null
                       {
                        if (beg == true)
                                   //replc at beginning of name
                            ReplcBeg(again,rl1,rl2);
                        else ReplcEnd(again,rl2,rl1);
                        }
                    if (beg == true && emp != true)
                             //if rule occurs at beginning of name
                        {
                        if (first == true)
                             //replc first with second
                           ReplcBeg(again,rl1,rl2);
                        else ReplcBeg(again,rl2,rl1);
                        }
                    if (end == true && emp != true)
                             //if rule occurs at end of name
                       {
                        if (first == true)
                             //replc first with second
                           ReplcEnd(again,rl1,rl2);
                        else ReplcEnd(again,rl2,rl1);
                       }
                    whr++;
                   }
                }
            }
           i++;
          }
}
```

```
//This fuction removes the # sign from the rule
char *LatWin::RemSgn(char *word)
{
         char *newword;
         char *sendstr;

         newword = (char *) new[30];
         sendstr = (char *) new[30];

         strcpy(newword,word);
         strnset(sendstr,0,30);

         int len = strlen(newword);
         int p = strcspn(newword,"#");
         if (p == 0)
            {
            while (newword[p + 1] != NULL)
               {
               newword[p] = newword[p + 1];
               p++;
               }
            newword[p] = 0;
            strcpy(sendstr,newword);
            }
        else
             {
             newword[len - 1] = 0;
             strcpy(sendstr,newword);
             }
        return sendstr;
}

//This function substitutes one (part of rule) with two (translation)
//at beginning of name
void LatWin::ReplcBeg(char *name,char *one, char *two)
{
        char *bnewauth;
        char *authorn;
        char *rone;
        char *rtwo;
        int num = 0;


        bnewauth = (char *)new char[30];
        rone = (char *)new char[10];
        rtwo = (char *)new char[10];
        strnset(bnewauth,0,30);

        strcpy(authorn,name);
        strcpy(rone,one);
        strcpy(rtwo,two);

         if (rone == NULL)
           {
           strcpy(bnewauth,rtwo);
           strcat(bnewauth,authorn);
           }
        else
          {
          strcpy(bnewauth,rtwo);
          int len = strlen(rone);
          int ln = len;
          while (authorn[len] != NULL)
             {
             authorn[num] = authorn[len];
             len++;
```

```
              num++;
              }
          authorn[len - ln] = 0;
          strcat(bnewauth,authorn);
          }
          cout << "replc beg" << bnewauth << endl;
          SaveChg(bnewauth);
}

//Replace one with two at end of name
void LatWin::ReplcEnd(char *name,char *one, char *two)
{
        char *enewauth;
        char *authorn;
        char *rone;
        char *rtwo;
        char *hello2;

        enewauth = (char *)new char[30];
        rone = (char *)new char[10];
        rtwo = (char *)new char[10];
        strnset(enewauth,0,30);

        strcpy(authorn,name);
        strcpy(rone,one);
        strcpy(rtwo,two);

         if (rone == NULL)
           {
           strcpy(enewauth,authorn);
           strcat(enewauth,rtwo);
           }
        else
          {
//Needed for applying rule to names already found
//If name has been changes by a previous rule it might not
//contain the current rule

          int alen = strlen(authorn);
          int roneln = strlen(rone);
          char *tst = strstr(authorn,rone);
          cout << "tst " << tst << endl;
          cout << "alen " << alen << endl;
          if (tst == NULL)
              return;
          int pp = strcspn(authorn,rone);
          int newlen = alen - roneln;
          if (newlen != pp)
             return;
          authorn[pp] = 0;
          strcpy(enewauth,authorn);
          strcat(enewauth,rtwo);
          }
         cout << "replc end" << enewauth << endl;
//         latlistbox += enewauth;
         SaveChg(enewauth);
}


//This function substitutes one with two in middle of name
void LatWin::ReplcMid(char *latm, char *onem, char *twom)
{
         char *authorn;
         char *mnwauth;
         int na = 0;
         char rune[10];
         char rdeux[10];
         authorn = (char *)new char[30];
         mnwauth = (char *)new char[30];

         strnset(mnwauth,0,30);          //claer variables
         strnset(authorn,0,30);

         strcpy(authorn,latm);
         strcpy(rune,onem);
         strcpy(rdeux,twom);

        int len1 = strlen(rune);
        int len2 = strlen(rdeux);

//Needed for applying rule to names already found
//If name has been changes by a previous rule it might not
//contain the current rule
       char *tst = strstr(authorn,rune);
        cout << "tst " << tst << endl;
        if (tst == NULL)
            return;
        while (authorn[na] != rune[0])
             {                           //copy initial segment
             mnwauth[na] = authorn[na];
             na++;
             }

        int remt = na;
        for (int j = 0; j < len2; j++)  //copy rule
            {
            mnwauth[na] = rdeux[j];
            na++;
            }
        if (len1 == len2)
           {
           while (authorn[na] != NULL)
              {
              mnwauth[na] = authorn[na];
              na++;
              }
           }

        if (len1 < len2 || len2 < len1)
           {
           remt = remt + len1;
           while (authorn[remt] != NULL)
              {
              mnwauth[na] = authorn[remt];
              na++;
              remt++;
              }
           }

         mnwauth[na] = 0;
        cout << "Rplc mid" << mnwauth << endl;
        SaveChg(mnwauth);
}

//This finction changes the entry field in the author window to
//the cognate found in FindCog
void LatWin::ChgCog()
{
         int row = latlistbox.GetSelection();
         if (row != LIT_NONE)
            authorWindow.efield = validrls[row];
}
```

```
//This function saves the cognate found in an array of cognates
//to be accessed when the user clicks on a row in the listbox
void LatWin::SaveChg(char *svword)
{
        char *newsave;

        newsave = (char *)new char[30];

        strcpy(newsave,svword);
        int slne = strlen(newsave);
        if (slne < 4)   //if word length less than 4 then ignore it
           return;
        if (sv < 100)
           {
           strcpy(saverls[sv],newsave);
           Check(newsave);
           sv++;
           cout << "sv at " << sv  << endl;
           }
        return;
}


//This function checks to see if new name exists before
//displaying it in the listbox
void LatWin::Check(char *nwname)
{
        SearchNodePtr pNode;
        Token token;
        char *word;
        char *chktxt;
        int chk = 0;
        chktxt = (char *)new char[30];

        strnset(chktxt,0,30);
        strcpy(chktxt,nwname);
        int i = termWin.AthTermFrq(chktxt);
             if (i > 0)
              {
              latlistbox += chktxt;
              cout << "valid name" << chktxt << endl;
              strcpy(validrls[vr],chktxt);
              vr++;
              }
}
```

Bibliography Manager Code -Create User File, Open Existing User File, Save User File and Print User File

NewFile.Hpp

```
#ifndef _NEWFILE_HPP_
#define _NEWFILE_HPP_

#include "stdtyp.hpp"
#include "stdwin.hpp"
#include "entryfld.hpp"
#include "line.hpp"
#include "listbox.hpp"
//#include<gpfparms.h>
```

```
class NewFile
{
        private:
                StdWin win;
                ListBox liblistbox;
                int entryFieldID;
                int listboxID;
                int l;
                char *nText;
                CHAR buf[20];
                char *libs[50];

        public:
                int create,open;
                EntryField efield;
                NewFile(int ID);
                NewFile(int ID1, int ID2);
                ~NewFile();
                void Init(HWND hwnd);
                void Clear();
                void Create();
                void Open();
                void Print();
                void SaveFloppy();
                void ListFls();
                Boolean exist, listb;
                FILE *fileo;
};

extern NewFile newFile;
extern NewFile openFile;
extern NewFile printFile;
extern NewFile saveFile;

#endif
```

NewFile.Cpp

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "error.hpp"
#include "library.ext"
#include "library.ids"
#include "newfile.hpp"
#include "stdtyp.hpp"

NewFile newFile(ID_EFCREATEFILE);
NewFile openFile(ID_EFOPENFILE, ID_LBOPENFLE);
NewFile printFile(ID_EFPRINTFILE, ID_LBPRINFLE);
NewFile saveFile(ID_EFSAVEFILE, ID_LBSAVEFLE);

NewFile::NewFile(int ID)
{
        entryFieldID = ID;
        exist = false;
        listb = false;
        create = open = 0;
        nText = NULL;
}


//Set Up New File Variables
NewFile::NewFile(int ID1, int ID2)
{
        entryFieldID = ID1;
        listboxID = ID2;
        exist = false;
        listb = true;
        create = open = 0;
        l = 0;
        nText = NULL;
}

//Close File
NewFile::~NewFile()
{
        fclose(fileo);
        delete [] libs;
}

//Initialize New File
void NewFile::Init(HWND hwnd)
{
        win.AttachHandle(hwnd);
        efield.AttachID(hwnd, entryFieldID);
        if (listb == true)
           {
           liblistbox.AttachID(hwnd, listboxID);
           liblistbox.Clear();
           for (int k = 0; k < 50; k++)
              libs[k] = new char[15];
           }
        if (nText != NULL)
           efield = nText;
}

void NewFile::Clear()
{
        efield = "";
}

//Create New User File
void NewFile::Create()
{
        char *p;
        char *r;
        nText = efield;
        p = strstr(nText, ".lib");
        r = strstr(nText, ".LIB");
        if (p == NULL && r == NULL)
           {
           errorMsg.Warn("The file must have a '.lib' extension");
           return;
           }
        DismissNewFileWindow(false);
        if ((fileo = fopen(efield, "w")) == NULL)
           errorMsg.Warn("Cannot open file");
        create = 1;
        exist = true;
}

//Open Existing User File
void NewFile::Open()
{
        char *opfle;
        opfle = (char *)new char[20];
        int row = liblistbox.GetSelection();
        if (row != LIT_NONE)
           {
           cout << "row " << row << endl;
```

```
              opfle = libs[row];
              cout << "opfle " << opfle << endl;
              efield = opfle;
              }
          nText = efield;
          strcpy(opfle,nText);
          cout << "nText " << opfle << endl;
          DismissSaveFileWindow(false);
          if ((fileo = fopen(opfle, "r")) == NULL)
              {
              errorMsg.Warn("No such file exists");
              return;
              }
          fclose(fileo);
          if ((fileo = fopen(opfle, "a")) == NULL)
              {
              errorMsg.Warn("Cannot open file");
              return;
              }
          open = 1;
          cout << "finish open " << endl;
          exist = true;
          delete opfle;
}

//Save User File to Floppy
void NewFile::SaveFloppy()
{
          char svcomm[60];
          char *svfle;
          svfle = (char *)new char[15];

          fclose(fileo);
          int row = liblistbox.GetSelection();
          if (row != LIT_NONE)
              {
              cout << "row " << row << endl;
              svfle = libs[row];
              cout << "opfle " << svfle << endl;
              efield = svfle;
              }
          nText = efield;
          DismissOpenFileWindow(false);
          if ((fileo = fopen(efield, "r")) == NULL)
              {
              errorMsg.Warn("No such file exists");
              return;
              }
          strcpy(svcomm, "copy ");
          strcat(svcomm,nText);
          strcat(svcomm," a:");
          cout << "command " << svcomm << endl;

          system(svcomm);
          delete svfle;
}

//Print User File
void NewFile::Print()
{
          static char command[20];
          char *ppfle;
          ppfle = (char *)new char[15];


          int row = liblistbox.GetSelection();
```

```
        if (row != LIT_NONE)
           {
           cout << "row " << row << endl;
           ppfle = libs[row];
           cout << "ppfle " << ppfle << endl;
           efield = ppfle;
           }
        nText = efield;
        DismissPrintFileWindow(false);
        if ((fileo = fopen(efield, "r")) == NULL)
           {
           errorMsg.Warn("No such file exists");
           return;
           }
        strcpy(command, "print ");
        strcat(command,nText);
        system(command);
        delete ppfle;
}

//List all files with .lib extension
void NewFile::ListFls()
{
        FILE *lis;
        char *liblns;
        static char command[20];
        liblns = (char *) new char[20];

        strcpy(command, "dir *.lib > lfiles ");
        system(command);
        strcpy(command, "spitbol remext");
        cout << "com" << command << endl;
        system(command);
        if ((lis = fopen("libfiles", "r")) == NULL)
           {
           errorMsg.Warn("Cannot find list of lib files");
           return;
           }

        while ((liblns = line.GetLine(lis)) != NULL)
              {
              liblistbox += liblns;
              strcpy(libs[l],liblns);
              l++;
              }
        return;
}
```

APPENDIX F

SOURCE LISTING CODE - SNOBOL4

FindLang.Spt - Language Recognition Program - Final Version

```
* This is the latest language recognition program written by R. Clarke
07.02.'95
* This program runs on ocr files of the 1872 catalogue pages.
* Spitbol version ; runs only on SPITBOL PHARLAP
        &TRIM = 1
*Function to recognize language of each record
        DEFINE('RECOGL(LREC)')
*Patterns to be used
PATS    DIGIT = '0123456789'
        UPPERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        LOWERS = 'abcdefghijklmnopqrstuvwxyz'
        CHARS = (UPPERS LOWERS '"')
        LETTERS = UPPERS LOWERS
        OR = ("sive" | "seu" | "or" | "ou")
        AUTSEP = ('("\?",')
        AUTSPR = ('),.')
*Direct Translation Patterns
        DIRTRANS = ARRAY('10,2')
*Language Indicators for each language
        SPANISHIND = TABLE()
        DUTCHIND = TABLE()
        ITALIANIND = TABLE()
        FRENCHIND = TABLE()
        ENGLISHIND = TABLE()
        LATININD = TABLE()
        GERMANIND = TABLE()
        PORTUGIND = TABLE()
*Language characteristics for recognition
        SPANISHREC = TABLE()
        DUTCHREC = TABLE()
        ITALIANREC = TABLE()
        FRENCHREC = TABLE()
        ENGLISHREC = TABLE()
        LATINREC = TABLE()
        GERMANREC = TABLE()
        PORTUGREC = TABLE()
*Suffix characteristics for recognition
        SPANISHSUF = TABLE()
        DUTCHSUF = TABLE()
        ITALIANSUF = TABLE()
        FRENCHSUF = TABLE()
        ENGLISHSUF = TABLE()
        LATINSUF = TABLE()
        GERMANSUF = TABLE()
        PORTUGSUF = TABLE()
*Table to keep scores
*         SCORES = TABLE()
*         SCORES<'SPANISH'> = 0
*         SCORES<'DUTCH'> = 0
*         SCORES<'ITALIAN'> = 0
*         SCORES<'FRENCH'> = 0
*         SCORES<'ENGLISH'> = 0
*         SCORES<'GERMAN'> = 0
*         SCORES<'PORTUG'> = 0
*Record each natural language name
        LANGNME = ARRAY('10')
*Constants - eight languages
        NUMLGS = 8
*-----------------------------------------------------------------
*Read in direct translations
        dt = 1
        INPUT('trans',3, 'C:\spt1\trans.lst')                :F(NOTRNS)
TRANS   inp = trans                                          :F(ENDTRS)
        inp break('=') . pat len(1) rem . lng
RSPS    pat ' ' rpos(0) =                                    :S(RSPS)
LSPS    lng ' ' =                                            :S(LSPS)
        dirtrans<dt,1> = pat ; dirtrans<dt,2> = lng
        dt = dt + 1                                          :(TRANS)
NOTRNS  terminal = "Trans.lst does not exist"                :(END)
ENDTRS  endfile(3)                                           :(begin)

*Read indicators lists from a file
*First, read in possible languages from a file
IFILE   INPUT('indlang',4, 'C:\spt1\indics.lst')             :F(NOISFL)
GETILNG ilangs = indlang                                     :F(LICLSE)
        ilangs break('.') . ifn len(1) len(3) . iext
        nextilng = ifn '.' iext
        newtable = ifn 'ind'
*Create table for each  indicator file
        newtab = $newtable
*Then, read in characteristics from each file
        INPUT('indic',5, 'C:\spt1\' nextilng)                :F(LGIERR)
LIREAD  recind = indic                                       :F(NOILMR)
LICHAR  newtab<recind> = 1                                   :(LIREAD)
NOILMR  endfile(5)                                           :(GETILNG)
LGIERR  TERMINAL = 'Unable to access indicator file ' NEXTLANG
:(END)
NOISFL  TERMINAL = 'File indics.lst does not exist'          :(END)
LICLSE  endfile(4)

*First read in list of suffix files from a file
SFILE   INPUT('suflang',6, 'C:\spt1\suffix.lst')             :F(NOSFFL)
GTSFLNG slangs = suflang                                     :F(SFCLSE)
        slangs break('.') . lsfx len(1) len(3) . sext
        nxtsflng = lsfx '.' sext
        sftable = lsfx 'suf'
*Create table for each suffix file
        suftable = $sftable
*Then, read in characteristics from each file
        INPUT('suffix',7, 'C:\spt1\' nxtsflng)               :F(LGSFER)
LSFRED  recsuf = suffix                                      :F(NOMRSF)
SFCHAR  suftable<recsuf> = 1                                 :(LSFRED)
NOMRSF  endfile(7)                                           :(GTSFLNG)
LGSFER  TERMINAL = 'Unable to access suffix file ' nxtsflng    :(END)
NOSFFL  TERMINAL = 'File suffix.lst does not exist'          :(END)
SFCLSE  endfile(6)

BEGIN
*Read in charcteristics to recognize languages
        L = 0
*First, read in possible languages from a file
*Ask user for name of file containing list of languages
        TERMINAL = "Please enter name of file containing languages "
        TERMINAL = "Use full file name e.g. C:\SPT1\LANGS.LST
(default)"
        filelist = "C:\SPT1\LANGS.LST"
        ans = terminal
        leq(ans, '')                                         :S(OFILE)
        filelist = ans ; ans =
OFILE   INPUT('difflang',8,filelist)                          :F(NOLSFL)
GETLANG LANG = difflang                                       :F(LCLOSE)
        L = L + 1
        LANG BREAK('.') . LFN LEN(1) LEN(3) . LEXT            :F(LGERR)
LREC    NEXTLANG = LFN '.' LEXT
*Get appropriate table
        LTABLE = LFN 'REC'
*        terminal = ltable
        LANGTABLE = $LTABLE
*Store language names
        LANGNME<L> = LFN
*Then, read in characteristics from each file
        INPUT('nlang',9, 'c:\SPT1\' NEXTLANG)                 :F(LGERR)
LREAD   RECWRD = nlang                                        :F(NOMORE)
LCHAR   LANGTABLE<recwrd> = 1                                 :(LREAD)
NOMORE  ENDFILE(9)                                            :(GETLANG)
LGERR   TERMINAL = 'Unable to access language file ' NEXTLANG
:(END)
NOLSFL  TERMINAL = 'File '  filelist ' does not exist'  :(END)
LCLOSE  ENDFILE(8)
*------------------------------------------------------------------
*Start processing
        terminal = "              -*-   LANGUAGE RECOGNIZER   -*- "
        terminal =
        terminal = "This program processes .ocr files to produce .rec
files"
        terminal = "Please enter extension - (default .rec)"
        output(.outr,13, "c:\spt1\lngres\fmglst.txt")
        ext = "rec"
        ans = terminal
        leq(ans,'')                                           :s(page_nm)
        ext = ans ; ans =
* Read page file name and constructing directory
page_nm terminal = "Enter page file name with full path (default
c:\spt1\pages.cat)"
        pgfile = "c:\spt1\pages.cat"
        ans = terminal
        leq(ans,'')
:s(in_file)
        pgfile = ans ; ans =
in_file input(.direc,1, pgfile)
        terminal = "Enter records directory (default d:\records)"
        rec_dir = "d:\records"
        ans = terminal
        leq(ans,'')
:s(ndir)
        rec_dir = ans ; ans =
ndir    fline = direc
:f(out)
        fline break(':') . dir break(letters) rem . fline
        dir ' ' rpos(0) =
        dir = rec_dir '\' dir
        terminal = fline
        terminal = dir

*********************************************************
*Get numbers of pages to be processed
nseq    fline  break(letters) len(1) . flet break(digit)
+       span(digit) . num1 break(digit) span(digit) . num2 =
:f(ndir)
        pnum = num1
*Open file containing pagenames
in_page fill = "00" lt(pnum,10)                               :s(npg)
        fill = "0"  lt(pnum,100)                              :s(npg)
        fill = ""
```

```
npg     page_name = dir '\' flet fill pnum "." ext        :f(no_page)
        out_page = flet fill pnum ".rec"
        out_lang = "c:\spt1\lngres\" flet fill pnum ".NLG"
        terminal = "Out   " out_page
        terminal = "Page " page_name
* Open file for specified page
        input(.page,11, page_name)                        :f(no_page)
*Produce an output file for each input file

        OUTR = FLET FILL PNUM
        output(.outm,12, out_page)                        :(read1)
next    endfile(11)
        endfile(12)
npg1    pnum = pnum + 1
        le(pnum,num2)
:s(in_page)f(nseq)
no_page terminal = page_name " does not exist"            :(npg1)
*---------------------------------------------------------------
*Start processing each record
READ1   ENTRY = PAGE                                      :F(NEXT)
        TITLE =
        ENTRY "T:" REM . TLE                              :S(MORE)
        OUTM = ENTRY                                      :(READ1)
MORE    TITLE = TITLE ' ' TLE
        OUTM = ENTRY
        ENTRY = PAGE                                      :F(NEXT)
        TLE = ENTRY
*Imprint field marks end of title field
        ENTRY "I:"                                        :S(CNTO)
*For records that have not been split properly only title and shelf
field
        ENTRY "S:"                                        :F(MORE)
CNTO    OUTM = ENTRY
REPEAT  ENTRY = PAGE                                      :F(NEXT)
        ENTRY "L:" REM . OLDLNG                           :S(RELN)
        OUTM = ENTRY                                      :(REPEAT)
RELN    RECOGL(TITLE)
        LNME LEN(1) . LGRES
        OUTR = LGRES
        OUTM = "L:   " OLDLNG ' ' LNME                    :(READ1)
OUT     TERMINAL = "COMPLETED"                            :(END)
*--------------------------------------------------------------
*Function to recognize languages
RECOGL  BEGWORD = ' ' SPAN(CHARS)
        ENDWORD = BREAK(CHARS) SPAN(CHARS) (' ' | '?' | '!' | ';' |
':')
        SEPR = ANY('_.,')
        NSEP = ('_., :;[]()')
        WORDPAT = BREAK(LETTERS) (SPAN(LETTERS)) . WORD BREAK(NSEP)
        SUF2 = LEN(2) RPOS(0)
        SUF3 = LEN(3) RPOS(0)
        SUF4 = LEN(4) RPOS(0)
        KEEPT = LREC
        DR = 1
        LNME = 'UNREC'
*Table to keep scores
        SCORES = TABLE()
*Max score so far
        MAXSOFAR = 0
*Begin Processing
        WORDMATCH = 'TRUE'
*Look for direct translations - if any found, then return
DTST    LT(DR,DT)                                         :F(HERE)
        LREC DIRTRANS<DR,1>                               :S(FNDDR)
        DR = DR + 1                                       :(DTST)
FNDDR   TERMINAL = "Direct Translation "  DIRTRANS<DR,1>:(RETURN)
```

```
*        TERMINAL = "**Language Chosen " DIRTRANS<DR,2>
*        WAIT = TERMINAL                                    :(RETURN)
TSTFIN
HERE
        LREC = REPLACE(LREC,&UCASE,&LCASE)
nxtwrd  LREC WORDPAT REM . LREC                            :F(VWSCR)
        preslg = 0
*        TERMINAL = "----NEXT WORD " WORD
        WORDMATCH = 'FALSE'
SRCIND  LGIND = 1
*Look for language indicators
*Get language indicator table from array
NXTLID  NAME = LANGNME<LGIND>
*Evaluate string to get language table
        NAMER = NAME 'REC'
        LNAME = $NAMER
*Check how many tables word is in
        IDENT(LNAME<WORD>,NULL)                            :S(NXTPRS)
*        TERMINAL = "Word " WORD " found in language " LANGNME<LGIND>
        WORDMATCH = 'TRUE'
        preslg = preslg + 1
nxtprs  eq(lgind,numlgs)                                   :S(CKSUF)
        lgind = lgind + 1                                  :(NXTLID)

cksuf   eq(preslg,0)                                       :F(SETWGT)
*        "Word " word "not found"
cntsuf  LGIND = 1
nxtsf   SF4 = SF3 = SF2 =
*Get length of word to see which suffixes to strip
        WRD_LEN = SIZE(WORD)
*Word length less than 2, get next word
        LE(WRD_LEN,2)                                      :S(NXTWRD)
*Word length = 3, strip 2 letter suffix
        LE(WRD_LEN,3)                                      :S(LET2a)
*Word length = 4, strip 2 and 3 letter suffix
        EQ(WRD_LEN,4)                                      :S(LET3a)
*Word length = 5 or more, strip 2, 3 and 4 letter suffix
*Extract 4 letter suffix from word
LET4a   WORD SUF4 . SF4 ; SF4 = '-' SF4
*Extract 3 letter suffix from word
LET3a   WORD SUF3 . SF3 ; SF3 = '-' SF3
*Extract 2 letter suffix from word
LET2a   WORD SUF2 . SF2 ; SF2 = '-' SF2
*        TERMINAL = WORD
*        TERMINAL = "SF4 " SF4 " SF3 " SF3 " SF2 " SF2
*Evaluate string to get suffix table
suffs   NAME = LANGNME<LGIND>
        NAMER = NAME 'REC'
        LNAME = $NAMER
*Check if suffixes are in suffix tables
        IDENT(LNAME<SF4>,NULL)                            :S(COUNT3a)
*        TERMINAL = "Suffix " SF2 " found in language " LANGNME<LGIND>
        preslg = preslg + 1                               :(nxtsfl)
COUNT3a IDENT(LNAME<SF3>,NULL)                            :S(COUNT2a)
*        TERMINAL = "Suffix " SF2 " found in language " LANGNME<LGIND>
        preslg = preslg + 1                               :(nxtsfl)
COUNT2a IDENT(LNAME<SF2>, NULL)                           :S(NXTsfl)
*        TERMINAL = "Suffix " SF2 " found in language " LANGNME<LGIND>
        preslg = preslg + 1                                :(nxtsfl)
nxtsfl  eq(lgind,numlgs)                                   :s(ckfor)
        lgind = lgind + 1                                  :(nxtsf)
ckfor   eq(preslg,0)
:s(nxtwrd)f(setwgt2)
```

```
setwgt
*       wgt = 1.0 / preslg
        wgt = 1
*If word contained in only one language than weight = 2 else weight=1
        gt(preslg,1)                                    :S(TRYWRD)
        wgt = 2
*        terminal = 'prels ' preslg ' wgt ' wgt
*Check if word is in the language table
TRYWRD
        LGIND = 1
NXTLID2 NAME = LANGNME<LGIND>
        NAMER = NAME 'REC'
        LNAME = $NAMER
        IDENT(LNAME<WORD>,NULL)                         :S(CONT)
        WORDMATCH = 'TRUE'
*        TERMINAL = "Word " WORD " found in language " LANGNME<LGIND>
*If it is, add 1 point to that language score
        SCORES<LANGNME<LGIND>> = SCORES<LANGNME<LGIND>> + wgt
*Check for max score so far
        GT(SCORES<LANGNME<LGIND>>,MAXSOFAR)             :F(CONT)
        MAXSOFAR = SCORES<LANGNME<LGIND>>
        LNME = LANGNME<LGIND>
CONT    EQ(LGIND,NUMLGS)                                :F(NEXTL)
        leq(wordmatch,'FALSE')                          :(NXTWRD)
nextl   LGIND = LGIND + 1                               :(NXTLID2)
setwgt2
*         wgt = 1.0 / preslg
*        terminal = 'preslg ' preslg ' wgt' wgt
        wgt = 1
trysfx  LGIND = 1
        SF4 = SF3 = SF2 = ""
*Get length of word to see which suffiexes to strip
        WRD_LEN = SIZE(WORD)
*Word length less than 2, get next word
        LE(WRD_LEN,2)                                   :S(RESET)
*Word length = 3, strip 2 letter suffix
        LE(WRD_LEN,3)                                   :S(LET2)
*Word length = 4, strip 2 and 3 letter suffix
        EQ(WRD_LEN,4)                                   :S(LET3)
*Word length = 5 or more, strip 2, 3 and 4 letter suffix
*Extract 4 letter suffix from word
LET4    WORD SUF4 . SF4 ; SF4 = '-' SF4
*Extract 3 letter suffix from word
LET3   WORD SUF3 . SF3 ; SF3 = '-' SF3
*Extract 2 letter suffix from word
LET2    WORD SUF2 . SF2 ; SF2 = '-' SF2
*        TERMINAL = WORD
*        TERMINAL = "SF4 " SF4 " SF3 " SF3 " SF2 " SF2
*Evaluate string to get suffix table
NXTSFX  NAME = LANGNME<LGIND>
        NAMES = NAME 'REC'
        LNAME = $NAMES
*Check if suffixes are in suffix tables
COUNT4  IDENT(LNAME<SF4>,NULL)                         :S(COUNT3)
*        TERMINAL = "Suffix " SF4 " found in language " LANGNME<LGIND>
*If it is, add 1 point to that language score
        SCORES<LANGNME<LGIND>> = SCORES<LANGNME<LGIND>> + wgt
*Check for max score so far
        GT(SCORES<LANGNME<LGIND>>,MAXSOFAR)             :F(COUNT3)
        MAXSOFAR = SCORES<LANGNME<LGIND>>
        LNME = LANGNME<LGIND>
COUNT3  IDENT(LNAME<SF3>,NULL)                          :S(COUNT2)
*        TERMINAL = "Suffix " SF3 " found in language " LANGNME<LGIND>
*If it is, add 1 point to that language score
        SCORES<LANGNME<LGIND>> = SCORES<LANGNME<LGIND>> + wgt
*Check for max score so far
```

```
        GT(SCORES<LANGNME<LGIND>>,MAXSOFAR)              :F(COUNT2)
        MAXSOFAR = SCORES<LANGNME<LGIND>>
        LNME = LANGNME<LGIND>
COUNT2  IDENT(LNAME<SF2>,NULL)                           :S(INCSFX)
*        TERMINAL = "Suffix " SF2 " found in language " LANGNME<LGIND>
*If it is, add 1 point to that language score
        SCORES<LANGNME<LGIND>> = SCORES<LANGNME<LGIND>> + wgt
*Check for max score so far
        GT(SCORES<LANGNME<LGIND>>,MAXSOFAR)              :F(INCSFX)
        MAXSOFAR = SCORES<LANGNME<LGIND>>
        LNME = LANGNME<LGIND>
*Check next language for suffixes
INCSFX  EQ(LGIND,NUMLGS)                                 :S(RESET)
        LGIND = LGIND + 1                                :(NXTSFX)
RESET   WORDMATCH = 'TRUE'                               :(nxtwrd)
*View scores for all languages
VWSCR   s = 1
*       TERMINAL = "SCORES-"
*Test to see if maxsofar is present in more than one language
*If this is the case - language cannot be recognized
        COUNT_EQ = 0 ; CT = 1
TSTCNT  EQ(SCORES<LANGNME<CT>>, MAXSOFAR)                :F(INCTST)
        COUNT_EQ = COUNT_EQ + 1
INCTST  EQ(CT,NUMLGS)                                    :S(SETLNG)
        CT = CT + 1                                      :(TSTCNT)
SETLNG  GT(COUNT_EQ,1)                                   :F(NXTSCR)
        LNME = 'UNREC'
NXTSCR
*       TERMINAL = LANGNME<S> ' : ' SCORES<LANGNME<S>> '   ' LANGNME<S
+ 1>
*+       ' : ' SCORES<LANGNME<S + 1>>
*        EQ(S,5)                                          :F(PLUSS)
*        TERMINAL = LANGNME<S + 2> ' : ' SCORES<LANGNME<S + 2>>
*       TERMINAL = KEEPT
*       TERMINAL = "** Language Chosen " LNME
*       WAIT = TERMINAL
*                                                         :(RETURN)
*PLUSS   S = S + 2                                        :(NXTSCR)
LFIN    RECOGL = LNME                                     :(RETURN)
RECOGL.END
*----------------------------------------------------------------------
END
```

NewDict.Spt - Program that builds dictionaries for each of the languages

```
*This program creates dictionaries for each language and stores
*all the words in titles into the relevant dictionaries.
* Spitbol version ; runs only on SPITBOL PHARLAP
        &TRIM = 1
-INCLUDE "D:\spt\host.inc"
        DEFINE('CONVUL(WD)')
        terminal = 'PLease enter file name'
        name = terminal
        INPUT('INP', 3, name)                  :(OUTVAR)
*Function to convert a lower case letter to an upper case letter
        DEFINE('CPWORD(LWRD)')
*Shelf No. patterns
OUTVAR  OUTPUT('OUTENG',15,"DICT.ENG")    :F(WARN)
        OUTPUT('OUTFRN',16,"DICT.FRN")    :F(WARN)
        OUTPUT('OUTLAT',17,"DICT.LAT")   :F(WARN)
        OUTPUT('OUTITL',18,"DICT.ITL")  :F(WARN)
        OUTPUT('OUTDUT',19,"DICT.DUT")  :F(WARN)
        OUTPUT('OUTGER',20,"DICT.GER")  :F(WARN)
        OUTPUT('OUTSPN',21,"DICT.SPN")  :F(WARN)
PATS    DIGIT = '0123456789'
        UPPERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        LOWERS = 'abcdefghijklmnopqrstuvwxyz'
        CHARS = (UPPERS LOWERS '"')
        NUM1OR2 = (ANY(DIGIT) $ DIG *DIG) | ANY(DIGIT)
        NUMBER = BREAK(DIGIT) SPAN(DIGIT)
        CAP = BREAK(UPPERS) SPAN(UPPERS)
        LC  = BREAK(LOWERS) SPAN(LOWERS)
        CAP1OR2 = (ANY(UPPERS) $ CH *CH) | ANY(UPPERS)
        LC1OR2 = (ANY(LOWERS) $ CH *CH) | ANY(LOWERS)
        DIG2 = ANY(DIGIT) ANY(DIGIT)
        DIG3 = DIG2 ANY(DIGIT)
        NUMB1 = ANY(DIGIT)
        NUMB2 = DIGIT LEN(2)
        NUMB3 = DIGIT LEN(3)
        NUMB4 = DIGIT LEN(4)
        SDSP = SPAN ('. ')
        SKSP = BREAK(LOWERS)
        EXDIGIT = DIGIT ',-  '
        NMAT = ' 1 o'
        EXTRA = ( (',' | '-') . X SPAN(EXDIGIT) . SREST BREAK('.')  .
PREST
+       '.') | '.'
        NOS = (' NØ. ' | ' NØs. '  | (' N' NOTANY('N') '.')) . YYX
+       SPAN(EXDIGIT) . TREST '.'
*       NOS = (' N' LEN(1) '.' ) . YYX BREAKX(DIGIT) SPAN(DIGIT) . +
        TREST
        PAGEAUTHORS = ARRAY('100,2')
*       SHELFMARKS = ARRAY('100,2')
* Word patterns etc
        TOTAL = 0
        T = TABLE()
        PLACE = TABLE()
        CHARS = UPPERS LOWERS DIGIT '"&'
        CHAR = UPPERS LOWERS "'&\" DIGIT
        SEP = ' ,.;:[]()!-"'
        WORDPAT = LEN(1) BREAK(CHARS) SPAN(CHARS) . WORD BREAK(SEP)
        WDPAT = LEN(1) BREAK(CHAR) SPAN(CHAR) . WORD BREAK(SEP)
        ITEMS = DIGITS CHAR SEP
        ITEM = ANY(ITEMS)
        ERCNT = TABLE()
        CORRECT = TABLE()
        POSSIB = ARRAY('10')
        CW = 1
        GLBCT = 0
        CHARCT = 0
        AC = 0
        MOR = 0
        LANGCT = 0
        TIMES = 0
        ND = 0
        AUTPR = 0
        OCRERR = 'TT'
*Count number of records recognised by language recognition function
        LCNT = 0
        SUM = 0
        COR = 0
        NFIX = TABLE()
*Creat new dictionnary of words
        NEWDICT = TABLE()
```

```
***********************
* Start of processing *
***********************
*Set up tables for trigram counting
CRTAB   ENGLDICT = TABLE()
        FRENDICT = TABLE()
        ITALDICT = TABLE()
        GERMDICT = TABLE()
        LATIDICT = TABLE()
        SPANDICT = TABLE()
        DUTCHDICT = TABLE()
        OUTPUT('OPAT',4, "OCRERR.TXT")                      :F(NOFIL)
*Read in charcteristics to recognize languages
        L = 0
*Start processing
*Read in first file from list of file names
READ    FILENAME1 = INP             :F(T2)
*       FILENAME1 BREAK(' ') . FN SPAN(' ') SPAN(UPPERS LOWERS) . EXT
        EXTEN = LEN(3) . EXT RPOS(0)
        NME = RPOS(8) LEN(4) . FN
        FILENAME1 EXTEN :S(RR1)
RR1     FILENAME1 NME   :S(RR2)
RR2     FNEXT = FN '.' EXT
*       FN LEN(1) REM . FN
*Produce an output file for each input file
        TERMINAL = FNEXT
        INPUT('INQ',7,FNEXT)                                :F(NOFILE)
*Read in lines of file
READ1   ENTRY = INQ                                         :F(NEXTF)
        TITLE =
*Find title field in file
        ENTRY "T:" REM . TLE                                :F(READ1)
MORE    TITLE = TITLE ' ' TLE
        ENTRY = INQ                                         :F(NEXTF)
        TLE = ENTRY
        ENTRY "C:"                                          :F(MORE)
REPEAT  ENTRY = INQ                                         :F(NEXTF)
*Fine language of title
        ENTRY "L:" rem . lnme                               :F(REPEAT)
cntrmv  lnme ' ' =                                          :S(CNTRMV)
reln    LEQ(LNME, 'UNREC')                                  :S(READ1)
        LNME '2' =
        WDICT = LNME 'DICT'
GOGO    WHDICT = $WDICT
*Add each word in the title to the appropriate dictionary
T1      TITLE WORDPAT =                                     :F(READ1)
        WORD OCRERR                                         :F(PLACE)
*Output specified ocr errors to file "ocrerr.txt"
        OPAT = WORD
PLACE   WHDICT<WORD> = WHDICT<WORD> + 1       :(T1)
*Output dictionnary for each language to a file
T2      ENDFILE(4)
        ENGLDICT = CONVERT(ENGLDICT, 'ARRAY')               :F(CREFRE)
        ENGLDICT = SORT(ENGLDICT)
        Q = 1
*Count number of elements in each array
FILE1   DIFFER(ENGLDICT<Q,1>,"")                            :F(ODIC2)
        Q = Q + 1                                           :(FILE1)
ODIC2   NUMB = Q / 4
*Set variables for padding words
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING ENGLISH DICTIONARY - PLEASE WAIT"
CONTE   X1 = SIZE(ENGLDICT<X,1>) ; Y2 = SIZE(ENGLDICT<Y,2>)
        Z1 = SIZE(ENGLDICT<Z,1>)
*Output words in four columns to file
```

```
        OUTENG = ENGLDICT<X,1> LPAD(ENGLDICT<Y,1>,(25 - X1))
+       LPAD(ENGLDICT<Z,1>,(25 - Y2)) RPAD("",15) ENGLDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                      :F(CONTE)
        ENDFILE(15)
CREFRE  FRENDICT = CONVERT(FRENDICT, 'ARRAY')        :F(CRELAT)
        FRENDICT = SORT(FRENDICT)
        Q = 1
FILE2   DIFFER(FRENDICT<Q,1>,"")                     :F(ODIC3)
        Q = Q + 1                                    :(FILE2)
ODIC3   NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING FRENCH DICTIONARY - PLEASE WAIT"
CONTF   X1 = SIZE(FRENDICT<X,1>) ; Y2 = SIZE(FRENDICT<Y,2>)
        Z1 = SIZE(FRENDICT<Z,1>)
        OUTFRN = FRENDICT<X,1> LPAD(FRENDICT<Y,1>,(25 - X1))
+       LPAD(FRENDICT<Z,1>,(25 - Y2)) RPAD("",15) FRENDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                      :F(CONTF)
        ENDFILE(16)
CRELAT  LATIDICT = CONVERT(LATIDICT, 'ARRAY')        :F(CREITL)
        LATIDICT = SORT(LATIDICT)
        Q = 1
FILE3   DIFFER(LATIDICT<Q,1>,"")                     :F(ODIC4)
        Q = Q + 1                                    :(FILE3)
ODIC4   NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING LATIN DICTIONARY - PLEASE WAIT"
CONTL   X1 = SIZE(LATIDICT<X,1>) ; Y2 = SIZE(LATIDICT<Y,2>)
        Z1 = SIZE(LATIDICT<Z,1>)
        OUTLAT = LATIDICT<X,1> LPAD(LATIDICT<Y,1>,(25 - X1))
+       LPAD(LATIDICT<Z,1>,(25 - Y2)) RPAD("",15) LATIDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                      :F(CONTL)
        ENDFILE(17)
CREITL  ITALDICT = CONVERT(ITALDICT, 'ARRAY')        :F(CREGER)
        ITALDICT = SORT(ITALDICT)
        Q = 1
FILE4   DIFFER(ITALDICT<Q,1>,"")                     :F(ODIC5)
        Q = Q + 1                                    :(FILE4)
ODIC5   NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING ITALIAN DICTIONARY - PLEASE WAIT"
CONTI   X1 = SIZE(ITALDICT<X,1>) ; Y2 = SIZE(ITALDICT<Y,2>)
        Z1 = SIZE(ITALDICT<Z,1>)
        OUTITL = ITALDICT<X,1> LPAD(ITALDICT<Y,1>,(25 - X1))
+       LPAD(ITALDICT<Z,1>,(25 - Y2)) RPAD("",15) ITALDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                      :F(CONTI)
        ENDFILE(18)
CREGER  GERMDICT = CONVERT(GERMDICT, 'ARRAY')        :F(CRESPA)
        GERMDICT = SORT(GERMDICT)
        Q = 1
FILE5   DIFFER(GERMDICT<Q,1>,"")                     :F(ODIC6)
        Q = Q + 1                                    :(FILE5)
ODIC6   NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING GERMAN DICTIONARY - PLEASE WAIT"
```

```
CONTG   X1 = SIZE(GERMDICT<X,1>) ; Y2 = SIZE(GERMDICT<Y,2>)
        Z1 = SIZE(GERMDICT<Z,1>)
        OUTGER = GERMDICT<X,1> LPAD(GERMDICT<Y,1>,(25 - X1))
+       LPAD(GERMDICT<Z,1>,(25 - Y2)) RPAD("",15) GERMDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                         :F(CONTG)
        ENDFILE(20)
CRESPA  SPANDICT = CONVERT(SPANDICT, 'ARRAY')           :F(CREDUT)
        SPANDICT = SORT(SPANDICT)
        Q = 1
FILE6   DIFFER(SPANDICT<Q,1>,"")                        :F(ODIC7)
        Q = Q + 1                                       :(FILE6)
ODIC7   NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING SPANISH DICTIONARY - PLEASE WAIT"
CONTS   X1 = SIZE(SPANDICT<X,1>) ; Y2 = SIZE(SPANDICT<Y,2>)
        Z1 = SIZE(SPANDICT<Z,1>)
        OUTSPN = SPANDICT<X,1> LPAD(SPANDICT<Y,1>,(25 - X1))
+       LPAD(SPANDICT<Z,1>,(25 - Y2)) RPAD("",15) SPANDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                         :F(CONTS)
        ENDFILE(21)
CREDUT  DUTCHDICT = CONVERT(DUTCHDICT, 'ARRAY')         :F(END)
        DUTCHDICT = SORT(DUTCHDICT)
        Q = 1
FILE7   DIFFER(DUTCHDICT<Q,1>,"")                       :F(END1)
        Q = Q + 1                                       :(FILE7)
END1    NUMB = Q / 4
        X = 1 ; Y = (NUMB + 1)
        Z = (NUMB * 2) + 1 ; W = (NUMB * 3) + 1
        TERMINAL = "COMPILING DUTCH DICTIONARY - PLEASE WAIT"
CONTD   X1 = SIZE(DUTCHDICT<X,1>) ; Y2 = SIZE(DUTCHDICT<Y,2>)
        Z1 = SIZE(DUTCHDICT<Z,1>)
        OUTDUT = DUTCHDICT<X,1> LPAD(DUTCHDICT<Y,1>,(25 - X1))
+       LPAD(DUTCHDICT<Z,1>,(25 - Y2)) RPAD("",15) DUTCHDICT<W,1>
        X = X + 1 ; Z = Z + 1
        Y = Y + 1 ; W = W + 1
        EQ(W,Q)                                         :F(CONTD)
        ENDFILE(19)                                     :(END)
*Error messages
NEXTF   ENDFILE(7)                                      :(READ)
NOFILE                                                  :(READ)
NOFIL   TERMINAL "CANNOT OPEN OUTPUT FILE"              :(END)
WARN    TERMINAL = 'CANNOT OUTPUT TO FILE'              :(END)
ER1     OUTPUT = "Error in \page :" ENTRY               :(END)
ER2     OUTPUT = "Error in \entries :" ENTRY            :(END)

END
```

Rectit.Spt - Expands author title abbreviations in author fields of each entry

```
*-INCLUDE "D:\SPT\host.inc"
        &TRIM = 1
*This program expands the initials and abbreviations in the author
title field
*in order that this information can be used in the search system of
the library
*catalogue.
*e.g. S.J. - Society of Jesus (jesuit)
        TERMINAL = 'enter name of file containing file names'
ASK     FILENAME = TERMINAL
        INPUT('SOFAR', 7, FILENAME)
        INPUT('TITL', 1, "C:\SPT\AUTD.DAT")
        INPUT('OCRR', 2, "C:\SPT\AUTOCR.DAT")
        OCRTAB = ARRAY('150,2')
        AUTHRT = ARRAY('200,2')
        DIGIT = ('0123456789')
        PUNCT = ('.,\:- ')
        CHARS = &LCASE &UCASE DIGIT PUNCT
***********************
* Start of processing *
***********************
        I = 1 ; R = 1
*Read in substitutions for initials
SUBSTI  INN = TITL                                  :F(ROCR)
        INN SPAN(CHARS) . ONE BREAK(';') LEN(1) REM . TWO
RMSPS   ONE " " RPOS(0) =                           :S(RMSPS)
        AUTHRT<I,1> = ONE
RMSPS2  TWO POS(0) " " =                            :S(RMSPS2)
        AUTHRT<I,2> = TWO
        I = I + 1                                   :(SUBSTI)
*Read in ocr errors for author title field
ROCR    OCR = OCRR                                  :F(SET)
        OCR SPAN(CHARS) . ONE BREAK(';') LEN(1) REM . TWO
RMSPS3   ONE " " RPOS(0) =                          :S(RMSPS3)
        OCRTAB<R,1> = ONE
RMSPS4  TWO POS(0) " " =                            :S(RMSPS4)
        OCRTAB<R,2> = TWO
        R = R + 1                                   :(ROCR)
SET     LASTI = I - 1
        LASTR = R - 1
READ    FILENAME1 = SOFAR                           :F(OUT)
        EXTEN = LEN(3) . EXT RPOS(0)
        EXT = EXT
        NME = RPOS(8) LEN(4) . FN
        FILENAME1 EXTEN :S(RR1)
RR1     FILENAME1 NME   :S(RR2)
RR2     FNEXT = FN '.' EXT
        NEWF = FN '.OCR'
        TERMINAL = FNEXT
*Input each file
        INPUT('INQ',3, FNEXT)                      :F(NOFILE)
        OUTPUT('OUT',4, NEWF)                      :F(NOOUT)
*Produce an output file for each input file
READ1   ENTRY = INQ                                :F(NXTF)
        I = 1 ; R = 1
NEXTE   ENTRY "E:" REM . EDU                       :S(TEST1)
        OUT = ENTRY                                :(READ1)
TEST1
TEST    EQ(R,LASTR)                                :S(SUBST)
        EDU OCRTAB<R,1>                            :S(REPLC)
        R = R + 1                                  :(TEST)
REPLC   EDU OCRTAB<R,1> = OCRTAB<R,2>
```

```
        ENTRY OCRTAB<R,1> = OCRTAB<R,2>
        R = R + 1                          :(TEST)
SUBST   EQ(I,LASTI)                        :S(NEENY)
        EDU AUTHRT<I,1>                    :S(REPL2)
        I = I + 1                          :(SUBST)
REPL2   ENTRY AUTHRT<I,1> = AUTHRT<I,1> " " AUTHRT<I,2>
        ENTRY AUTHRT<I,2> REM . EDU
        IDENT(EDU, "")                     :S(NEENY)
        I = I + 1                          :(SUBST)
NEENY   OUT = entry                        :(READ1)
NXTF    ENDFILE(3)
        ENDFILE(4)                         :(READ)
NOOUT   TERMINAL = "Cannot output to file"    :(END)
NOSOF   TERMINAL = "Cannot convert array to table" :(END)
OUT     TERMINAL = "Processing Completed"     :(END)
NOFILE  TERMINAL = 'File does not exist'      :(END)
END
```

Rules.Spt - Program that attempts to find cognate rules for pairs of words in different languages

```
*This program compares words in two different languages and checks if
*they are related.
*The sounds in the words are used to determine how related the words
are.
*If the word pair is deemed to be over 65% related - then translation
rules
*are developed from this word pair.
*The rules developped are context sensitive.
        DEFINE('READIN(X),STORE')
        DEFINE('CMETHOD(CONSON1,CONSON2)')
        DEFINE('VMETHOD(VOWEL1,VOWEL2)')
        DEFINE('VOICING(CN1,CN2)')
        DEFINE('ISCON(CHARC)')
        DEFINE('ISVWL(CHARV)')
        DEFINE('SETSOUND(ARRAY1)')
        DEFINE('SETRULE(RUL1,RUL2)')
        VOWELS = 'AEIOUY'
        CONSONANTS = 'BCDFGHJKLMNPQRSTVWXZ'
        DOUBLECONS = (ANY(CONSONANTS) $ CAP *CAP)
        COMPOUND = ('TH' | 'GH' | 'CK' | 'CH' | 'PF' | 'NG')
        CEITHER = (DOUBLECONS | COMPOUND | ANY(CONSONANTS))
        COMPVWLS = ('EE' | 'EA' | 'IE' | 'EI' | 'AI' | 'EA' | 'AY' |
+       'OI' | 'OH' | 'UH' | 'AU')
        DOUBLEVLS = (ANY(VOWELS) $ VOL *VOL)
        VEITHER = (COMPVWLS | DOUBLEVLS | ANY(VOWELS))
        GLBCNT = 0
        ARRAY1 = ARRAY('10,2')
        FIRST = ARRAY('10,2')
        SECOND = ARRAY('10,2')
        RES1 = ARRAY('10,2')
        RES2 = ARRAY('10,2')
        FRONT = ('EE' | 'EA' | 'IE' | 'EI' | 'AI' | 'EA' | 'I' | 'E' |
'AY')
        CENTRAL = ('E' | 'AU' | 'Y' | 'U' | 'A')
        BACK = ('OI' | 'OO' | 'OH' | 'UH' | 'OU' | 'O' | 'U')
        BILABIAL = ('P' | 'B' | 'M')
        LABIODENTAL = ('PF' | 'F' | 'V')
        DENTAL = 'TH'
        SINGLE = ('BDGVZJMNRWL')
        VOICED = ('TH' | 'DG' | ANY(SINGLE))
        VSINGLE = ('PTKFS')
        VOICELESS = ('PF' | 'TH' | 'CH' | ANY(VSINGLE))
        ALVEOLAR = ('T' | 'D' | 'SH' | 'S' | 'Z' | 'N' | 'R' | 'L' |
'X')
        PAL_ALVEOLAR = ( 'CH' | 'J' | 'DG')
        PALATAL = 'J'
        VELAR = ('K' | 'CH' | 'C' | 'G' | 'NG' | 'GN' | 'W' | 'Q')
        GLOTTAL = 'H'
* Possible phonetic groups
        CONSOUNDS = ARRAY(8);           GRPVAL = TABLE()
        CONSOUNDS<1> = BILABIAL;        GRPVAL<1> = 'BILABIAL'
        CONSOUNDS<2> = LABIODENTAL;     GRPVAL<2> = 'LABIODENTAL'
        CONSOUNDS<3> = DENTAL;          GRPVAL<3> = 'DENTAL'
        CONSOUNDS<4> = ALVEOLAR;        GRPVAL<4> = 'ALVEOLAR'
        CONSOUNDS<5> = PAL_ALVEOLAR;    GRPVAL<5> = 'PAL_ALVEOLAR'
        CONSOUNDS<6> = PALATAL;         GRPVAL<6> = 'PALATAL'
        CONSOUNDS<7> = VELAR ;          GRPVAL<7> = 'VELAR'
        CONSOUNDS<8> = GLOTTAL;         GRPVAL<8> = 'GLOTTAL'
MAIN    TERMINAL = 'PLEASE ENTER FIRST LANGUAGE'
        LANG1 = TERMINAL
        LANG1 LEN(2) . L1
        TERMINAL = 'PLEASE ENTER SECOND LANGUAGE'
        LANG2 = TERMINAL
        LANG2 LEN(2) . L2
        DEFRULES = L1 L2 'RULE'
        STORERLS = L1 'TO' L2 '.RLS'
        DEFRULES = TABLE()
        POSSRULES = ARRAY('20')
        OUTPUT('OUTR',4,STORERLS)
        INPUT('INP1',2,LANG1)                           :F(NOSUCH1)
        INPUT('INP2',3,LANG2)                           :F(NOSUCH2)
REPEAT  ONE = INP1                                      :F(END1)
        ONE = REPLACE(ONE,&LCASE,&UCASE)
        TERMINAL = ONE
        TWO = INP2                                      :F(END2)
        TWO = REPLACE(TWO,&LCASE,&UCASE)
        TERMINAL = TWO
*The 'Readin' procedure reads the sounds of a word into an array
        FIRST = READIN(ONE)
*The 'Setsound' procedure determines each of the sounds in the array -
i.e.
* whether the sounds are inetrvovalic, initial, final etc
        FIRST = SETSOUND(FIRST)
*Same as above  with second word
        SECOND = READIN(TWO)
        SECOND = SETSOUND(SECOND)
*Scan through consonants in each array and compare them
        FINFIR = 'FALSE'
        FINSEC = 'FALSE'
        CNT1 = CNT2 = 1;
        CONS = 'FALSE'
        EXTRA = 0
        PR = 0
*Read in the first sound of both words
CMPTST  LEQ(FIRST<CNT1,1>,NULL)                 :F(SECFIN)
        FINFIR = 'TRUE'                         :(RELATE)
SECFIN  LEQ(SECOND<CNT2,1>,NULL)                :F(CMPFS)
        FINSEC = 'TRUE'                         :(RELATE)
*        TERMINAL = FIRST<CNT1,1> CNT1 ' SEC ' SECOND<CNT2,1> CNT2
CMPFS   FIRST<CNT1,1> CEITHER                   :S(CHKSEC)
        SECOND<CNT2,1> VEITHER                  :S(TRYVL)
        FIRST<CNT1,1> (DOUBLEVLS | COMPVWLS)    :S(INCSEC)F(INCFIR)
CHKSEC  SECOND<CNT2,1> CEITHER                  :S(CMET)
        SECOND<CNT2,1> (DOUBLEVLS | COMPVWLS)   :S(INCFIR)F(INCSEC)
*If sounds are both consonants - then compare these consonants
```

```
CMET     CMETHOD(FIRST<CNT1,1>,SECOND<CNT2,1>)     :(INCBTH)
INCFIR   RES1<CNT1,1> = FIRST<CNT1,1>
         EXTRA = EXTRA + 1
*Next sound in first word
         CNT1 = CNT1 + 1                            :(CMPTST)
INCSEC   RES2<CNT2,1> = SECOND<CNT2,1>
         EXTRA = EXTRA + 1
*Next sound in second word
         CNT2 = CNT2 + 1                            :(CMPTST)
INCBTH   RES1<CNT1,1> = FIRST<CNT1,1>
         RES2<CNT2,1> = SECOND<CNT2,1>
         CNT1 = CNT1 + 1
         CNT2 = CNT2 + 1                            :(CMPTST)
*Else if sounds are both vowels - then compare these vowels
TRYVL    VMETHOD(FIRST<CNT1,1>, SECOND<CNT2,1>)    :(INCBTH)

RELATE   C = D = 1;
         TOTAL = NO = 0;
         EXCHARS = 0
*If one word is shorter than the other
*then find out by how many chars/sounds they differ and
*add this to the total number of sounds.
LENGTH   LEQ(FINFIR,'TRUE')                         :F(FIRLNG)
         LEQ(SECOND<CNT2,1>,NULL)                   :S(EXTRCH)
SECLNG   LEQ(SECOND<CNT2,1>,NULL)                   :S(EXTRCH)
         EXTRA = EXTRA + 1
         CNT2 = CNT2 + 1                            :(SECLNG)
FIRLNG   LEQ(FIRST<CNT1,1>,NULL)                    :S(EXTRCH)
         EXTRA = EXTRA + 1
         CNT1 = CNT1 + 1                            :(FIRLNG)
EXTRCH   EXCHARS = EXTRA / 1.5
TSTFIN   GT(C,CNT1)                                 :S(CKRES2)
*For each sound in the first word
*If current sound has a score then add this score to the total
CHKVAL   LEQ(RES1<C,2>,'')                          :S(INCC)
         TOTAL = TOTAL + RES1<C,2>
*         TERMINAL = RES1<C,2>
         NO = NO + 1
INCC     C = C + 1                                  :(TSTFIN)
*For each sound in the second word
*If current sound has a score then add this score to the total
CKRES2   GT(D,CNT2)                                 :S(FINISH)
         LEQ(RES2<D,2>,'')                          :S(INCRD)
         TOTAL = TOTAL + RES2<D,2>
*         TERMINAL = RES2<D,2>
         NO = NO + 1
INCRD    D = D + 1                                  :(CKRES2)
FINISH   TOTAL = TOTAL * 2
         NO = NO + EXCHARS
*Calculate percentage of related words
         PRECEN = TOTAL / NO
         TERMINAL = 'TOT ' TOTAL ' NO ' NO
         PRECEN = PRECEN * 10
         GT(PRECEN,65)                              :F(STATE)
         NUMB = 1
TSTEND   GT(NUMB,PR)                                :S(STATE)
         TEMP = POSSRULES<NUMB>
         DEFRULES<TEMP> = DEFRULES<TEMP> + 1
         TEMP =
         NUMB = NUMB + 1
         GLBCNT = GLBCNT + 1                        :(TSTEND)
STATE    TERMINAL = 'THERE IS A ' PRECEN '% CHANCE OF THESE WORDS BEING
RELATED'
         TERMINAL =
         TERMINAL = 'DO YOU WISH TO CONTINUE - YES / NO '
         ANS = TERMINAL
         LEQ(ANS,'YES')                           :S(CNTIN)
         LEQ(ANS, 'yes')                          :S(CNTIN)
         LEQ(ANS,'NO')                            :S(ENDT)
         LEQ(ANS,'no')                            :S(ENDT)
CNTIN    TERMINAL = 'NEXT PAIR'                   :(REPEAT)
*---------------------------------------------------
CMETHOD
*This procedure compares two consonant sounds
         CONS1 = CONSON1
         CONS2 = CONSON2
         SCR = 0
*If two sounds are identical - then set score to 5
         LEQ(CONS1,CONS2)                         :F(CHKGRP)
         SCORE = 5                                :(SETSCR)
*Otherwise find out to which group each of the sounds belong
CHKGRP   CI = 1
         DI = 1
PHON1    CONS1 CONSOUNDS<CI> REM . REST           :S(CON1SET)
INCCI    GT(CI,8)                                 :S(PROB)
         CI = CI + 1                              :(PHON1)
CON1SET  CONS1 DOUBLECONS                         :S(STVLS)
         LEQ(REST,NULL)                           :F(INCCI)
STVLS    SNDONE = GRPVAL<CI>
PHON2    CONS2 CONSOUNDS<DI> REM . RESTD          :S(CON2SET)
INCDI    GT(DI,8)                                 :S(PROB)
         DI = DI + 1                              :(PHON2)
CON2SET  CONS2 DOUBLECONS                         :S(STVLS2)
         LEQ(RESTD,NULL)                          :F(INCDI)
STVLS2   SNDTWO = GRPVAL<DI>
*Are both consonant sounds of the same phonetic group - i.e. bilabial,
dental
*If voicing of both consonants is the same then increase the score
         SCR = VOICING(CONS1,CONS2)
*If both sounds belong to the same phonetic group - then set score = 4
         IDENT(SNDONE,SNDTWO)                     :F(DIFGRP)
         SCORE = 4                                :(SETSCR)
*If both sounds belong to different phonetic groups - then
*check how close these phonetic groups are and set score accordingly
DIFGRP   GT(CI,DI)                                :S(BIGR)
         DIFF = DI - CI                           :(SETVAL)
BIGR     DIFF = CI - DI
SETVAL   SCORE = 4
CONV     EQ(DIFF,0)                               :S(SETSCR)
         EQ(SCORE,0)                              :S(SETSCR)
         SCORE = SCORE - 1
         DIFF = DIFF - 1                          :(CONV)
PROB     SCORE = 0
SETSCR   SCORE = SCORE + SCR
         SETRULE(CONS1,CONS2)
         RES1<CNT1,2> = SCORE
*         TERMINAL = CONSON1 'SC' SCORE
*         TERMINAL = CONSON2 'SC' SCORE
         RES2<CNT2,2> = SCORE                     :(RETURN)
*---------------------------------------------------
VMETHOD
*This procedure compares two vowel sounds
         VWL1 = VOWEL1
         VWL2 = VOWEL2
         SCORE = 0
         SECTIME = 'FALSE'
*If both vowel sounds are identical - then set the score to 5
         LEQ(VWL1,VWL2)                           :F(SETONE)
         SCORE = 5                                :(SETVLS)
SETONE   CMVWL = VWL1
*Otherwise find out to whcih group the vowle sounds belong
CKVL     CMVWL DOUBLEVLS . DB                     :F(CKVLGP)
```

```
        DB LEN(1) . CMVWL
CKVLGP  CMVWL POS(0) FRONT REM . REST            :F(CENTRL)
        LEQ(FREST,NULL)                          :F(CENTRL)
        FL = 1
        FLAG = 'FRONT'                           :(SECN)
CENTRL  CMVWL CENTRAL REM . CREST                :F(BACK1)
        LEQ(CREST,NULL)                          :F(BACK1)
        FL = 2
        FLAG = 'CENTRAL'                         :(SECN)
BACK1   CMVWL BACK REM . BREST                   :F(MINSC)
        LEQ(BREST,NULL)                          :F(MINSC)
        FL = 3
        FLAG = 'BACK'
SECN    LEQ(SECTIME,'TRUE')                      :S(FINVL)
        CMVWL = VWL2
        SECTIME = 'TRUE'
        FL1 = FL
        FLAG1 = FLAG                             :(CKVL)
*If vowel sounds belong to same group - then score = 4
FINVL   LEQ(FLAG1,FLAG)                          :F(DIFFRV)
        SCORE = 4                                :(SETVLS)
*Else check how close the vowel sounds are and set screo accordingly
DIFFRV  GT(FL1,FL)                               :F(LWRR)
        DIFVL = FL1 - FL                         :(WHICH)
LWRR    DIFVL = FL = FL1
WHICH   EQ(DIFVL,1)                              :F(DIFTWO)
        SCORE = 3                                :(SETVLS)
DIFTWO  EQ(DIFVL,2)                              :F(MINSC)
        SCORE = 2                                :(SETVLS)
MINSC   TERMINAL = 'MINSC ' CMVWL
        SCORE = 0
SETVLS  SETRULE(VWL1,VWL2)
        RES1<CNT1,2> = SCORE
        RES2<CNT2,2> = SCORE                     :(RETURN)
*-----------------------------------------------------------
VOICING
*This procedure checks if both consonant sounds are voiced
*or both consonant sounds are voiceless
        EL1 = CN1
        EL2 = CN2
        EL1 VOICED                               :F(NOVCE)
        EL2 VOICED                               :F(NOTSIM)
* If both sounds have the same voicing quality -
*then award an extra 0.5 to the score
VOICE   SCR = 0.5                                :(COMPLT)
NOVCE   EL2 VOICELESS                            :S(VOICE)
NOTSIM  SCR = 0
COMPLT  VOICING = SCR                            :(RETURN)
*-----------------------------------------------------------
ISCON
*This function checks whether the current char is a consonant
        CON = CHARC
        CON CEITHER                              :F(NOT)
        TEMP = 'TRUE'                            :(RETVAL)
NOT     TEMP = 'FALSE'
RETVAL  ISCON = TEMP                             :(RETURN)

*-----------------------------------------------------------
ISVWL
*This function checks whether the current char is a vowel
        VWL = CHARV
        VWL VEITHER                              :F(VNOT)
        CURRENT = 'TRUE'                         :(RETVWL)
VNOT    CURRENT = 'FALSE'
RETVWL  ISVWL = CURRENT                          :(RETURN)
```

```
*-----------------------------------------------------------
SETSOUND
*This procedure determines the context of each sound
* i.e. intervocalic, initial, final etc
        WORDAR = ARRAY1
        INITIAL = 'TRUE'
        P = 0
        PREV = ''
NEXTCH  R = P + 2
        LEQ(WORDAR<R,1>,NULL)                   :S(FINAL)
        VAL = ISCON(WORDAR<R,1>)
        LEQ(VAL,'TRUE')                         :F(ISVL)
        NEXT = '_#C'                            :(INCR)
ISVL    VAL = ISVWL(WORDAR<R,1>)
        LEQ(VAL,'TRUE')                         :F(NEITH)
        NEXT = '_#V'                            :(INCR)
NEITH   NEXT = '#'
INCR    P = P + 1
        WORDAR<P,2> = PREV WORDAR<P,1> NEXT
        VAL = ISCON(WORDAR<P,1>)
        LEQ(VAL,'TRUE')                         :F(ISVL2)
        PREV = '#C_'                             :(NEXTCH)
ISVL2   VAL = ISVWL(WORDAR<P,1>)
        LEQ(VAL,'TRUE')                         :F(NEITH2)
        PREV = '#V_'                             :(NEXTCH)
NEITH2  PREV = '#'                              :(NEXTCH)
FINAL   P = P + 1
        WORDAR<P,2> = PREV WORDAR<P,1>
        SETSOUND = WORDAR                       :(RETURN)

*-----------------------------------------------------------
SETRULE
*This procedure determines translation rules
        ELEM1 = RUL1
        ELEM2 = RUL2
        F1 = 1; S1 = 1
FIND1   LEQ(FIRST<F1,1>,ELEM1)                  :S(SELCT)
        F1 = F1 + 1                             :(FIND1)
SELCT   A = FIRST<F1,2>
FIND2   LEQ(SECOND<S1,1>,ELEM2)                 :S(SELCT2)
        S1 = S1 + 1                             :(FIND2)
SELCT2  B = SECOND<S1,2>
        RULE = A ' / ' B
        PR = PR + 1
        POSSRULES<PR> = RULE
*        IDENT(DEFRULES<RULE>, '')              :F(NOSET)
*If a rule has occurred once already - then it becomes a 'definite
rule'
*and is placed in the defrules table
*        IDENT(POSSRULES<RULE>,1)               :S(SETDEF)
*Otherwise it is a 'possible rule' and is placed in the possrue table
*        POSSRULES<RULE> = 1                    :(NOSET)
*SETDEF  POSSRULES<RULE> = ''
*        DEFRULES<RULE> = 1
*        GLBCNT = CLBCNT + 1
NOSET                                           :(RETURN)
*-----------------------------------------------------------

READIN
*This procedure reads the sounds of each word into an array
        WORD = X
        I = 1
        STORE = ARRAY('15,2')
LOOP    WORD POS(0) (DOUBLECONS | COMPOUND | ANY(CONSONANTS)) . CNS
:S(STCNS)
VWLS    WORD POS(0) (SPAN(VOWELS) (BREAK(CONSONANTS) | '')) . VLS =
        STORE<I,1> = VLS
        I = I + 1
        WORD POS(0) VLS =
        EQ(SIZE(WORD),0)                            :S(FINEND)F(LOOP)
STCNS   STORE<I,1> = CNS
        I = I + 1
        WORD POS(0) CNS =
FIN     EQ(SIZE(WORD),0)                            :F(LOOP)
FINEND  READIN = STORE                              :(RETURN)
*-----------------------------------------------------
*Write out the definite rules to a file
END1    ENDFILE(2)                                  :(ENDT)
END2    ENDFILE(3)
ENDT    DEFRULES = CONVERT(DEFRULES, 'ARRAY')       :F(END)
        AR = 1
        TERMINAL = 'STORING RULES TO TRANSLATE ' LANG1 ' TO ' LANG2
        OUTR = 'RULES FOR CONVERTING FROM ' LANG1 ' TO ' LANG2 ':'
CUNN    GT(AR,GLBCNT)                      :S(END)
        TERMINAL = DEFRULES<AR,1>
        OUTR = DEFRULES<AR,1>
        AR = AR + 1                        :(CUNN)
NOSUCH1 TERMINAL = 'THERE IS NO ' LANG1 ' FILE '      :(END)
NOSUCH2 TERMINAL = 'THERE IS NO ' LANG2 ' FILE '      :(END)
END
```

•