LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Computerising a Library Catalogue using Optical Character Recognition

A thesis presented for the degree of
Master of Science
to the Faculty of Engineering,
University of Dublin

by

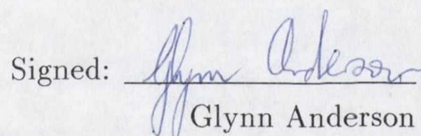
Glynn Anderson

Department of Computer Science,
Trinity College

TRINITY COLLEGE
10 MAR 1993
LIBRARY DUBLIN
THESIS
2842

Declaration

This thesis is entirely the work of the author, except where otherwise stated, and has not been submitted for a degree at any other University. This thesis may be copied and lent to others by the Library of the University of Dublin.

Signed: 

Glynn Anderson

October 26, 1992

Summary

Trinity College Library contains several million books. Catalogues for the more modern books have been computerised to allow readers a fast and efficient means of locating a book.

The 1872 Printed Catalogue which lists books owned by the library before 1872 has not yet been computerised. The catalogue lists 165,000 books, some of which are the most valuable in the library. The purpose of this project is to write a computer program that will automatically computerise the catalogue using optical character recognition (OCR). OCR is the process by which a digital picture of a portion of text is converted into computer readable text. Each character on the page is represented by a group or 'blob' of dots or pixels. The role of the computer is twofold; first to decide which pixels should be grouped together (ie which belong to the same character) and second to decide what character each of the blobs of pixels represents.

The output of the OCR program is sent to a database and will eventually be incorporated into the existing DYNIX© database, currently in use in the library.

The thesis contains a review of several different approaches to OCR, including feature vector analysis, discrimination trees, stroke analysis and neural networks. The implementation and results of a selection of these methods are described. The recognition or classification method used in this project, template matching, has not been implemented before as a primary classification method. The results of this thesis show that template matching compares very favourably with other classification methods. The thesis describes the considerable work undertaken in deriving a good matching algorithm which is the key to success of template matching. The segmentation of lines and characters is described in full including the development of a very efficient perimeter tracing algorithm.

Before the final chapters on results, conclusion and future work, there is a chapter

explaining how a state machine is used, while classifying, to delimit the fields within each entry on a catalogue page.

Acknowledgements

There are many people to thank for assistance with this thesis, but my supervisor Prof. J. G. Byrne must come first. Thanks to him for his guidance throughout the course of this thesis and earlier during my final-year project. Thanks also to Dr. David Abrahamson and James Mahon in the Computer Science department for their help and advice. Thanks to my office mates Mark Deegan and David Ngo. Special thanks also to Alan Judge, Martin O'Connor, Andrew Condon and David Kelly for invaluable advice, help and companionship. I'm grateful to Vincent Kinane of Trinity Library for help with the catalogue layout and history. Finally a very special thanks to my parents and Sally Ann McGrath for unwavering support and encouragement. Thanks to my parents also for considerable financial assistance in times of hardship.

Contents

1 Introduction **1**

1.1 Overview of this Chapter 1

1.2 Optical Character Recognition 2

1.3 The Printed Catalogue 3

1.3.1 Layout 3

1.3.2 History 7

1.4 Plan of the Thesis 8

2 Review of OCR methods **11**

2.1 Overview of this Chapter 11

2.2 Template Matching 11

2.2.1 Results and Performance for Template Matching 12

2.3 Feature Vectors 12

2.3.1 Line Adjacency Graphs 13

2.3.2 Convex and concave features 15

2.4 Discrimination Trees 16

2.4.1 Results and Performance for Discrimination Trees 17

2.5 Stroke Analysis 18

2.5.1 Complexity indices 19

2.6 Neural Networks 20

2.6.1 Results and performance for Neural Networks 21

2.7 Summary of this chapter 21

3 System Overview **23**

3.1 Overview of this Chapter 23

3.2 Development Steps and Modules 23

3.3 Classifying a full page 25
3.3.1 Classifying entries 27
3.4 Main data structures 28

4 Template Matching **30**
4.1 Overview of this Chapter 30
4.2 How template matching is used 30
4.3 Probability scoring 31
4.4 Filters 34
4.4.1 Analysis of success of filters 36
4.5 Multiple matching by shifting position 37
4.6 Problems with template matching 38
4.6.1 Erroneous match because of tilted text 38
4.6.2 Erroneous match because of bias 40
4.7 Training 41
4.7.1 Template Selection 42

5 Segmentation **43**
5.1 Overview of this Chapter 43
5.2 Segmenting entries 43
5.3 Segmenting lines and characters 44
5.3.1 Problems with multi-blob characters 45
5.3.2 The space character 46
5.4 Finding the bounding box 47
5.4.1 Connectivity 48
5.4.2 Perimeter tracing 48
5.5 Splits and merges 51
5.5.1 Splits 52
5.5.2 Merges 57
5.5.3 Thinning and expanding 59

6 Using a State Machine to Delimit Fields **61**
6.1 Overview of this Chapter 61
6.2 Delimiting Fields 61

6.3 State machine 63
6.3.1 Description of states 65
6.3.2 Description of trigger conditions 65
6.3.3 Reduced template sets 68
6.4 Errors introduced by the state machine 70

7 Postprocessing and Context **71**
7.1 Overview of this Chapter 71
7.2 Introduction to Postprocessing and Context 71
7.3 Context in this Project 72
7.4 Layout 73
7.5 Language 74
7.5.1 Spelling 74
7.5.2 Digrams & trigrams 75

8 Results and Performance **77**
8.1 Overview of this Chapter 77
8.2 Recognition time and speed 77
8.2.1 Time Stamping 78
8.3 Success rate 79
8.3.1 Errors 80
8.3.2 Counting Errors 82

9 Conclusion and Future Work **83**
9.1 Overview of this chapter 83
9.2 Conclusion 83
9.2.1 Making template matching a success 84
9.3 Future Work 85
9.3.1 Parallelism 85
9.3.2 Merged characters 86
9.3.3 Integration with the library database 87
9.3.4 Context and spelling 87
9.3.5 Averaging templates 87
9.3.6 Graphics user interface 88

A Rules for composing the printed catalogue **89**

B Equipment and software used **93**

B.1 Hardware 93

B.2 Software 93

C Run-length encoding **94**

D Photographs **96**

Bibliography **99**

List of Tables

3.1 Modules used in the project. 24

4.1 Score values for the character 'E' (highest values in bold). 32

4.2 Score values for the character 'e' (highest values in bold). 32

5.1 No. of ways n blobs can be grouped and no. of matches per grouping. 56

6.1 State names and descriptions. 63

6.2 Trigger Conditions. 64

6.3 Template sets. 69

6.4 Sets associated with states. 69

8.1 Compression Statistics. 78

8.2 Recognition time and speed. 78

8.3 Errors and success rate of the first test. 80

8.4 Errors and success rate of the second test. 80

C.1 127 run-length encoded in one byte. 94

C.2 128 run-length encoded in two bytes. 94

List of Figures

1.1 Page l60 4

3.1 Functional breakdown of the project. 26

4.1 The character 'H' can be mistaken for 'I-I' 33

4.2 Matching the character 'A' by Method 3. 35

4.3 Character 'o' has a much smaller perimeter than character 'c'. 36

4.4 Multiple matching in four extra positions. 38

4.5 A character is stored in the top left corner of the template array. . . . 40

5.1 3 Entries from page T12 of the catalogue. 44

5.2 Bounding boxes of two characters. 45

5.3 4 and 8-way connectivity. 48

5.4 Black pixels are 8-way, white pixels are 4-way connected. 49

5.5 Tracing the perimeter of the character 'C'. 49

5.6 Duck or chicken? 53

5.7 Finding splits with Method 1. 54

5.8 Finding splits with Method 2. 56

5.9 3 blobs can be grouped 4 ways, needing 8 matches. 57

5.10 Finding a merge by signature analysis. 59

6.1 Complete state machine. 66

9.1 Parallel matching 86

D.1 Screen photo. showing a hyphen split in two parts. 96

D.2 A 'P' template with its horizontal and vertical signatures. 97

D.3 A character enlarged within a window. 97

D.4 An entry composed almost entirely of Greek characters. 98

Chapter 1

Introduction

Trinity College Library currently contains almost 3 million books and is continually expanding. Catalogues for the more modern books have been converted to computer database form to allow readers a fast and efficient means of locating a book. The 1872 Printed Catalogue which lists books obtained by the library before 1872 has not yet been added to the database. The catalogue lists some 165,000 books, some of which are the most valuable in the library. The purpose of this project is to write a computer program that will automatically convert the catalogue into computer readable text, using optical character recognition. The output of the OCR program will be sent to a database and will eventually be incorporated into the existing DYNIX[1] database currently in use in the library. This thesis details the development of the project and describes the stages involved in converting a physical page of text to database readable form. Converting a physical page of text to database readable form involves several stages. The stages detailed in the thesis include: scanning, data compression and expansion, entry segmentation, line and character segmentation, field segmentation with a state machine, classification by template matching, split and merge detection and post processing.

1.1 Overview of this Chapter

This chapter describes the development of Optical Character Recognition over the last few decades and reviews the current status. There is also a description of

[1]DYNIX is a trademark of Sequent Computer Systems.

the layout of the catalogue pages and a factual history of the development of the catalogue itself from its conception in the mind of Bartholemew Lloyd in 1831 to its completion in 1887. The section on layout contains a picture of a typical catalogue page as well as a description of the format of an entry and a very detailed description of the format of shelf marks.

1.2 Optical Character Recognition

Optical character recognition (OCR) is the process by which a digital picture of a portion of text is converted into computer readable text. Each character on the page is represented by a group or 'blob' of dots or pixels. The role of the computer is twofold; first to decide which pixels should be grouped together (ie which belong to the same character) and second to decide what character each of the blobs of pixels represents.

Over the past four decades or more, much research has been undertaken into optical character readers or reading machines. The need for character readers sprang from the inability to store and process vast quantities of documents. There were other needs too, including the need to automatically find literature references, the need to translate human languages and the need to provide access to literature for the blind[2]. There was also the desire to replicate the functions of the human brain. Computers allowed huge amounts of data to be stored in very small physical areas but the problem of getting the data off the paper and into the computer still remained. In the late sixties and early seventies large, crude and extremely expensive character readers appeared on the market and were used in government departments and large institutions where large quantities of paper were produced. These machines cost one million dollars or more and were usually only good for one font type or page structure. A standard machine readable font was described to improve efficiency. The research of the seventies and eighties largely solved the problem of reading printed text and research switched to reading handwriting[14, 16, 20, 26, 27]. Recognition of handwriting *on the fly* is referred to as On-Line Character Recognition(OLCR) and is generally regarded as being easier than recognising pre-written text as the strokes can be followed as they are made. Much of this research was and is taking place in Japan where Chinese and Japanese ideographs provide particular problems [3, 9, 20, 23, 27].

[2] In conjunction with a speech synthesizer.

There are now several OCR packages available off the shelf for personal computers and these packages are, by in large, very successful. A variety of classification methods are used including feature extraction and stroke analysis. A few of the more popular and ingenious methods are described in Chapter 2. The primary classification method used in this project, template matching, is described in Chapter 4.

1.3 The Printed Catalogue

1.3.1 Layout

An example of a library catalogue page can be seen in Figure 1.1. The page has two columns of text separated by a vertical solid line. Above the two columns is a single line of text giving information about the page. On the left of this line are three letters showing the opening letters of the first entry on the page. On the right are three letters showing the opening letters of the last entry on the page. In the middle of the line, above the dividing center line is the page number enclosed in round brackets. Section 4.6.1 describes how this line can be used to find the slope or angle of tilt of the page. The two columns of text can logically be considered to be one continuous column. This logical column of text is broken up into entries. As this is a library catalogue, each entry describes either a book or an author. Where an author has several books in the library, each separate book entry is shown by an elongated hyphen '—'. Each entry usually contains several fields (described below) and takes up several lines of text. To segment the page, the entries are first segmented and then, for each entry, the individual lines and characters are segmented. With the *a priori* knowledge of the page layout, the top left corner of the first character in the first entry can be found (if the page is aligned correctly (See Section 4.6.1)). This gives a point of reference from where everything else on the page can be found.

The pages contain several different languages including English, French, German, Dutch and Latin. These languages use the Roman alphabet. In addition there is Hebrew, Syriac, Russian, Arabic and Greek which use their own alphabets. Figure D.4 in Appendix D is a screen photograph showing an entry consisting entirely of Greek characters. The typeface has been identified by two independent sources as *Old Style 2*. Most of the text is 11pt but the three-letter leaders at the top of the page are a larger size.

— Remarks upon certain objections published in the Dublin newspaper called The Warder, against the tithe composition bill now pending in parliament.
Lond. 1823. 8°. Gall. K. 20. 14. N°. 2.

— A sermon preached before the lord lieutenant, at the Castle Chapel, Dublin, upon Christmas-Day, 1825.
Dubl. 1825. 8°. D. gg. 18. N°. 10.

— A charge delivered at the triennial visitation of the province of Munster, in 1826.
Dubl. 1826. 8°. B. i. 17. N°. 10.

— A letter to him on the subject of a charge, purported to have been delivered at Killaloe & Limerick, June 20 & 22 [1826]. By N., *q. v.*
Gall. O. 8. 29. N°. 6.

— Remarks upon the laws affecting ecclesiastical pluralities, particularly in Ireland.
Dubl. 1829. 8°. Gall. K. 20. 14. N°. 3.

— Remarks upon the medical effects of the chlorides of lime and soda [*anon.*]
Dubl. 1832. 8°. Gall. K. 20. 14. N°. 4.

— On the existence of the soul after death: a dissertation.
Lond. 1834. 8°. C. i. 46. N°. 7.

— Extracts from a formulary for the visitation of the Saxon churches, A. D. 1528 [*anon.*]
Oxf. 1838. 8°. Gall. K. 20. 14. N°. 8.

— The visitation of the Saxon reformed church, in 1527, 8.
Dubl. 1839. 8°. G. r. 7. N°. 2.

— The Book of Job, arranged and pointed in conformity with the Masoretical text.—*Vid.* BIBLIA.
Gall. K. 20. 13. N°. 4.

— The Book of ENOCH the prophet, &c., translated.
O. rr. 63.

— Primi EZRÆ Libri versio Æthiopica; nunc primo in medium prolata, et Latine Angliceque reddita, a Ric. Laurence. I r. i. 33.

— The encyclical letter of pope GREGORY XVI., bearing date Aug. 16, 1832, translated, with notes.
Gall. K. 20. 14. N°. 5.

— Ascensio ISAIÆ vatis, opusculum pseudepigraphum, cum versione Latina Anglicanaque.
E. hh. 49.

LAURENCE (Robert French), M. A.—Remarks on the Hampden controversy.
Oxf. 1836. 8°. G. tt. 10. N°. 8.

— Examination of the theories of absolution and confession lately propounded in the university of Oxford.
Oxf. 1847. 8°. V. q. 20. N°. 1.

— Inquiry into the circumstances attendant upon the condemnation of dr. Hampden in 1836, in six letters.
Oxf. 1848. 8°. Gall. FF. 10. 3. N°. 15.

— Letter upon certain suggestions recently made by archdeacon Hare, as to the measures to be adopted for the removal of doubts on the doctrine of regeneration.
Oxf. 1850. 8°. Gall. FF. 12. 4. N°. 8.

— A letter to the rev. dr. M'Neile, on certain topics touched upon in his correspondence with archdeacon Wilberforce.
Oxf. 1850. 8°. Gall. HH. 10. 13. N°. 13.

— The order for the visitation of the sick, with supplemental services founded thereon.
Oxf. 1851. 12°. Gall. A. 9. 57.

— The churchman's assistant at holy communion.
Lond. 1860. 12°. Fag. E. 14. 20.

— Provision for the future: a sermon.
Lond. [*s. a.*] 8°. Gall. HH. 21. 51. N°. 15.

LAURENCE, or LAWRENCE (Roger).—Lay baptism invalid; sec. ed., enlarged, with an appendix, by a lay hand [and] a letter by the rev. Geo. Hickes [*anon.*]
Lond. 1709-13. 8°. 2 vols. LL. kk. 17. 18.
[the first part only]. GG. mm. 32.

— [with] Dissenter's baptism null and void. Reprinted from the 4th ed., 1723. With additions and illustrations, edited by Will. Scott.
Lond. 1841. 12°. B. pp. 17.

— Sacerdotal powers: or, the necessity of confession, penance, and absolution, 1710 [*anon.*]
Lond. 1711. 8°. RR. gg. 58. N°. 1

— [repr., under the title] An essay on confession, penance, and absolution. [With a preface by W. Gresley].
Lond. 1852. 8°. Gall. L. 12. 33. N°. 4.

LAURENCE (Z.)—Perspective simplified.
Lond. 1838. 8°. G. r. 17. N°. 8.

LAURENS (Bartholomæus).—Ioannis DAMASCENI opera, indice opera B. Laurentis adjecto. C. ee. 1.

— Ioh. NAUCLERI Chronica: cum auctario Nic. Baselii, ab A. D. 1501-14, et appendice nova memorante res interim gestas ab an. 1515 44 usque, rhapsodis partim Cunr. Tigemanno, partim B. Laurente. M. e. 1.

LAURENS (Honore de).—Panegyrique de l'edict de Henry III. roy de France, svr la reunion de ses subjets à l'eglise catholique, apostolique, & romaine, auec une exposition d'iceluy: & discours des moyens de purger les royaumes d'heresies, . . .
Bourdeaux, 1588. 8°. QQ. pp. 61. N°. 1.

LAURENT (——).—Histoire de l'empire Ottoman, traduite de l'Italien de SAGREDO. TT. 8. 39-15.

LAURENT (——).—Voyage autour du monde sur la corvette La Bonite, commandée par M. VAILLANT. Zoophytologie par M. Laurent. Fag. VV. 5. 20.

LAURENT (Achille). — Relation historique des affaires de Syrie, depuis 1840 jusqu'en 1842.
Paris, 1846. 8°. 2 tom. Gall. A. 7. 46. 47.

LAURENT (Émile).—The mercantile and bankrupt law of France: etc. By Henry DAVIES and E. Laurent. Gall. NN. 16. 44.

LAURENT (François).—Études sur l'histoire de l'humanité.
Paris, 1855— 8°. tom. 1— Gall. B. 15. 9—

— Le catholicisme et la religion de l'avenir.
Paris, 1870. 8°. 2me série. Gall. KK. 2. 20.
[This is a duplicate of a vol. of the preceding Études].

LAURENT (J. C. M.)—CLEMENTIS Romani ad Corinthios epistula: ed. J. C. M. Laurent.
Gall. 1. b. 10.

LAURENT (Peter Edmund).—Recollections of a classical tour through Greece, Turkey, and Italy, in 1818, 9.
Lond. 1821. 4°. P. bb. 13.

— Introduction to the study of ancient geography.
Oxf. 1830. 8°. NN. ii. 40.

— The nine books of the History of HERODOTUS, translated, . . . R. qq. 60, 61.

LAURENTIANUS (Laurentius).—GALENI de differentiis fœbrium libri duo, Lat., interprete Laurentio Laurentiano. K. l. 4. N°. 4.

LAURENTIE (P. S.)—La France au temps des croisades [10me série, p. 51, Bulletin du BIBLIOPHILE, par J. Techener]. Gall. 5. d. 18.

LAURENTIO (Nicolaus de), seu Cola di RIENZO, seu RIENZI, seu GABRINI.—Epistolæ duæ [tom. III. p. 136 Steph. BALUZII Miscell.] Q. c. 29.

— The life and times of Rienzi.
Lond. 1836. 12°. P. m. 71.

— Rienzi, the last of the tribunes. By the author of Eugene Aram, &c. [*i. e.* sir E. L. BULWER-LYTTON]. Gall. 21. 5. 12-14.

Figure 1.1: Page 160

Page statistics

There are, on average, 4800 characters per page, 48 entries per page, giving an average of 100 characters per entry. There are 152 lines per page, giving an average of 32 characters per line and 3 lines per entry.

Format of an entry

An entry can be several lines long and usually contains the name of the author, a description of his work, the place and date of publication and the shelf mark. Extra fields can also appear. When the catalogue was being devised, several rules were drawn up to aid the editors in the layout. These rules can be found in Appendix A. The first field in the entry is usually the author name. However, if an author has several books listed, an elongated hyphen will show the start of the entry. The author name field is always in capitals, with a few rare exceptions, for example *d'ARCY*. The author name or surname can be followed directly by the book description but is more usually followed by the author's first name. This usually appears in round brackets. The first name may be followed by the description or there may be another field that qualifies the author. The start of the book description is always signaled by an elongated hyphen. The description may contain all types of characters and can be several lines long. The location field does not always appear but when it does, it always begins a new line, after the description and is always in italics. The location is followed by the date field which is usually just four digits but can be surrounded by square brackets. The date field may be followed by a format field that can describe the size and/or the type of book. The format field is always followed by the shelf mark. The shelf mark occurs in the bottom right corner of the entry. There can be multiple shelf marks.

Shelf Marks

Most shelf marks have three separate sub-fields. These sub-fields show the *Bay*, *Shelf* and *Place* on the shelf where the book can be located. What can appear in each of the sub-fields depends on the location or what collection the book belongs to. The bay sub-field may be preceded by a prefix showing the building or collection. The following prefixes are possible:

- Gall. (for the gallery of the Long Room *or* for the east and west pavilions of the old library)
- Fag. (for the Fagel collection)
- Press. (for bookcases originally in the Librarian's office)
- Quin. (for the Quin collection)

Where there is no prefix, the book is shelved in the Long Room. The bay, shelf and place follow the following format, depending in the prefix:

- Gall. (Long Room Gallery)

  Bay: Capitals (A-Z, AA-ZZ)

  Shelf: number with at most 2 digits

  Place: number with at most 3 digits

  Example: Gall. DD. 18. 39.

- Gall. (east and west pavilion)

  Bay: number with at most 2 digits

  Shelf: lower case characters (a-z, aa-zz)

  Place: number with at most 3 digits

  Example: Gall. 5. k. 69.

- Fag.

  Bay: Capitals (A-Z, AA-ZZ)

  Shelf: number with at most 2 digits

  Place: number with at most 3 digits

  Example: Fag. C. 15. 110.

- Press.

  Bay: Capitals (A-Z)

  Shelf: number with at most 2 digits

  Place: number with at most 2 digits

  Example: Press. A. 1. 1.

- Quin.

  Quin is followed by a number in the range 1 to 127 and has the format: Quin, N° .127.

- <no prefix> (Long Room)

  Bay: Capitals (A-Z, AA-ZZ)

  Shelf: Lower case characters (a-z, aa-zz)

  Place: number

  Example: K. mm. 32.

Where a single work spans more than one volume, extended bay marks may appear. For example II. ee. 17, 18. shows that work occupies places 18 and 19 on the shelf and Y. e. 1-3. shows the work spans places 1 to 3 on the shelf.

1.3.2 History

In 1800[3], Trinity College Library contained about 50,000 volumes. In 1801, the Copyright Act was extended to Ireland which allowed the library to obtain a copy of every book published in Britain and Ireland. In the years following, the size of the library grew rapidly. In 1802, the collection of Hendrik Fagel, Chief Minister of Holland, was acquired. This added about 20,000 volumes.

In 1831, Bartholomew Lloyd was appointed Provost. Lloyd was a radical who immediately set about reorganising many areas of college. It was always evident to academics that the library was not being utilised to its fullest extent through the lack of a proper catalogue. It was under Lloyd's leadership that the college authorities began to consider undertaking the enormous task of compiling a catalogue. Up to that point and during the compilation of the catalogue, the library used the catalogue of the Bodleian library, with Trinity shelf marks added.

Work commenced on the catalogue unofficially in 1835. The initial cataloguing was done by preparing slips containing the basic information about the books. The catalogue was officially sanctioned on 11 Feb. 1837. Rev. James Henthorn Todd was appointed editor of what would be known officially as *Catalogus librorum impressorum qui in Bibliotheca Collegii ... Trinitatis ... adservantur*, known as the Printed

[3] Most of the information in this section has been gained from Kinane and O'Brien[19].

Catalogue. Dr. Thomas Fisher was appointed Todd's assistant in 1844. Todd and Fisher set about the task that would take fifty years to complete and who's completion neither would see. The design of the catalogue was based on the rules[4] used in completing the Bodleian catalogue and also that of the British Museum. It was to be a demy folio in small pica font (11pt.). Todd and Fisher made very slow progress. The first letter, 'A' was not completed until 1853 and it was not until 1864 that the volume containing 'A' and 'B' was printed. Todd died in 1867 and Fisher in 1869.

After the deaths of Todd and Fisher, the project was allowed to slide until 1872 when Henry Dix Hutton was appointed editor. Jan Hendrik Hessels, was appointed his assistant. Hessels was Dutch and was thought useful in cataloguing the largely Dutch Fagel collection. Slow progress continued and Hutton and Hessels did not work well together. Hessels favoured complete accuracy while Hutton was a pragmatist and was aware of time constraints. The letter 'C' was completed in 1874. Hessels eventually lost favour with college and his contract was not renewed in 1878. T.V. Keenan was appointed as Hessels replacement. Hutton and Keenan worked together to the completion of the project in 1887. At completion, it documented about 165,000 books.

The catalogue was printed by the Dublin University Press and was the biggest job they had ever undertaken. It has 5,606 pages and the total cost approached £20,000; a huge sum at that time.

Because of its age, the paper in the catalogue has become very delicate and several attempts have been made to preserve it. In the 1960s, a company in New York made an unauthorised microfilm version and in 1987, the centenary of the completion of the catalogue, college decided to authorise a microfiche version. The next stage was to combine the catalogue with the current library database of modern books and so this project was born.

1.4 Plan of the Thesis

Chapter 2 is a review of several different approaches to OCR. In most cases, there is a discussion of a documented implementation of the method and, where possible, a discussion of the merits and problems of the method and the results. There is also a brief description of template matching.

[4] See Appendix A.

Chapter 3 describes how the project was designed and discusses the modules, functions and data structures used.

Chapter 4 is a discussion of template matching in general as well as the particular implementation in this project. Several 'matching' algorithms are discussed as well as the reasons for choosing the particular one used in this project. Some methods for improving template matching such as filters and multiple matching are discussed as well as a description of the problems involved with template matching. System training is also discussed and advice is given on choosing the right instances of characters as templates.

Chapter 5 describes how individual characters are segmented prior to classification. The segmentation process is described in full, from segmenting entries to segmenting lines and characters by finding the bounding box through perimeter tracing. Segmentation problems are discussed as well as the considerable problem of splits and merges on which the author spent much time.

Segmenting fields within entries was done by means of a state machine which is the subject of Chapter 6. The structure and use of a state machine is explained and the states and trigger conditions of the state machine in this project are described. Reduced template sets are introduced and their merits and problems are discussed along with a discussion of the merits and problems of the state machine.

Chapter 7 is a description of how postprocessing and context can be used to improve the accuracy of an OCR system. Several documented methodologies are described as well as the methods used and investigated for this project.

The penultimate chapter is Chapter 8, discussing the results and performance of the system. Errors in the text output are described and discussed and the success rate for several pages are given in tabular form. The system speed is also discussed as well as the use of time stamps to discover bottlenecks.

Chapter 9 discusses the success of the whole system and of template matching in particular and includes a section on suggested future work the might enhance this system or make template matching more effective. There are three appendices: A, B and C. Appendix A is a transcription of the original rules used to construct the catalogue. These were useful to the author in writing the software and are included as being of interest to the reader. Appendix B is a list of the hardware and software used throughout the project and Appendix C is a description of run length encoding

which was used to in compressing the catalogue page. Finally there is a bibliography of the literature referred to throughout the project.

Chapter 2

Review of OCR methods

2.1 Overview of this Chapter

The primary OCR method used in this project is template matching. Template matching is described briefly in Section 2.2 and in detail in Chapter 4. It is usual in an OCR system to use several recognition or classification methods. In general, one method is the primary method and one or more secondary methods may be used to resolve ambiguities. For example, where a template matching system may have difficulty distinguishing between an 'e', 'c' and 'o', a feature vector or stroke analysis system may be used. Distinctive features such as the crossbar on the 'e' or the totally enclosed space of the 'o' can then be used to distinguish the character. The 'c' could be resolved by its area or perimeter length. Several other OCR methods are described in this chapter. Some of these have been used with some success on their own or may have been used with others. Where possible, results and performance have been included and the advantages and disadvantages of each method are evaluated.

2.2 Template Matching

Template matching is the primary OCR method used in this project and is described briefly in this section. The method is described in detail in Chapter 4.

Template matching requires at least one image or template, of every class of character that can be encountered, to be stored in memory. With each template

is stored the name (in ASCII) and any other useful information such as the width, height, perimeter length and area. When an input character is to be classified, it is compared with each of the templates in memory. The name of the template that best matches the input character is returned by the classifier. Matching is achieved by *superimposing* the input character on each of the templates and counting the pixels that match. Each template then has a score for the given input character. The template with the highest score is the *winner*! There are several scoring methodologies. The one used in this project was developed by the author and is described in Section 4.3. The advantages of template matching are its simplicity, ease of training and extendibility and the ability to readily identify the general structure of a character without having to view all the details. It also lends itself well to parallel processing. It's disadvantages are its sensitivity to noise and orientation and insensitivity to distinctive features.

2.2.1 Results and Performance for Template Matching

Results and performance for template matching in this project can be found in Chapter 8.

2.3 Feature Vectors

Feature vectors though not used in this project, are widely used as a classification method. The idea is to record several key independent features of each character to be classified. Feature vectors can be simple or complex. In the simpler systems, features can include width, height, perimeter and moments, to name but a few. More complex systems include arc lengths, enclosed areas and stroke sizes. Statistics are gathered for each character to be classified. For each feature, the mean and standard deviation are calculated. If n distinct features are used, each character can occupy a unique area in n dimensional space. The features of an input character are plotted in the n dimensional space and the character is classified as that which it is geometrically closest to. Ambiguities can arise when the standard deviations overlap. This can be overcome by choosing more features or by ensuring that those features chosen are indeed distinct. Choosing distinct features is non-trivial in itself as for example, there is always some relationship between the perimeter and area of

a character. A statistically based decision-theoretic feature system such as this was described as early as 1972 by Harmon[14, pages 1171—1172].

2.3.1 Line Adjacency Graphs

In a paper that is cited in many reports on OCR, Kahan *et.al.* [17] describe a classification system using Line Adjacency Graphs (LAGs). The aim was to recognise printed characters of any font and size. The line adjacency graph is really a method of representing the characters, the primary classification method being based on stroke-generated features. However according to the authors, the LAG allows for extremely fast thinning and feature extraction. LAG based thinning required 3 minutes per page as against 20 minutes for conventional pixel based thinning.

A LAG is constructed from a scanned and run-length encoded image file (see Appendix C). The LAG consists of nodes corresponding to black areas in the image and branches that join nodes whose corresponding black areas touch. Blobs are represented as connected components of the LAG. The LAG is used to thin (see Section 5.5.3) the input characters before classification, for contour or perimeter tracing and for feature extraction. When the LAG is traversed for thinning, the following features are extracted:

- character strokes
- number of holes
- hole location and size
- concavities in the skeleton
- stroke crossings
- vertical endpoints
- the minimum bounding box

A stroke is initially described by its two endpoint coordinates, $< x_1, y_1, x_2, y_2 >$. This rectangular description is transformed to a polar one thus: $< x, y, r, i >$ where $< x, y >$ is the center point of the stroke and $< r, i >$ represents the orientation and scaled length. When strokes are extracted and parameterised in this way, they

can be plotted in parameter space. Similar stroke lie close together in parameter space and dissimilar strokes lie far apart. When several examples of the same class of character are plotted together they tend to cluster in a small area of parameter space. Input characters whose strokes fall within some of these cluster regions can be recognised as the relevant class. The full classification algorithm is more complex and too lengthy to describe here. The output of the classifier is a list of possible matches in descending order of probability.

Where the match provided by the statistically based primary classifier belongs to a known *confusion group*, a secondary classifier based on contour analysis is invoked. In general, the confusion group contain characters that are generally 'compactly shaped and nearly convex.' Contour analysis using the LAG representation is faster than conventional contour analysis but the contour representation is an approximation. The contour is represented by eight line segments and classification is by means of a decision tree.

After classification, further improvements are made using layout and linguistic context. These methods are discussed in Chapter 7.

Results and Performance for Line Adjacency Graphs

Results and performance for the Line Adjacency Graph system of Kahan *et.al.*

- Scanner resolution: 200 dots per inch
- Page size: 1728 by 2048
- Processing speed: 5 characters per second on a DEC VAX 11/750
- Print quality: excellent

For testing, an alphabet of 70 characters were used, consisting of 26 upper case, 26 lower case (not including 'i' and 'j') and 20 other characters. Tests were done with both single and several fonts. Small punctuation characters, multi-blob characters[1], ligatures[2] and other special characters were omitted from the test and this should be remembered in considering the results. Training and test character sets were disjoint.

[1]See Section 5.3.1.

[2]See Section 5.5.2.

Results described here are only a summary of more complex results listed by the authors[17]. For single and multiple fonts of size 12 to 18 point, the recognition rate was better than 99.5%. At or below 10 point size characters, the recognition rate went below 98%. For six fonts, the rate was 97%.

The authors suggest that bad recognition rates with small size characters could be remedied by using a higher resolution scanner (e.g. 300 dpi as used in this project). For speed considerations, the authors suggest using purpose built hardware.

The authors claim speed of 5 characters per second for the whole process. It is useful to compare this to the results of this project (see Chapter 8). The library pages contain roughly 4800 characters per page. This would translate to a time of 16 minutes per page if the system of Kahan *et.al.* were used on the library pages, which is about twice the time taken by the software used in this project. It must also be remembered that only 70 characters were used in Kahan's test, whereas over 300 templates were used in this project. Remember also that Kahan's system ran on a VAX 11/750 while the software for this project runs on an InMos T800 transputer.

2.3.2 Convex and concave features

Yamamoto[27] describes a system for classifying characters by the number and shape of the convexities and concavities in the contour or perimeter of the character. After the perimeter of the character has been found, it is divided into convex and concave segments. The segments are found by tracing the perimeter in two directions, both clockwise and anti-clockwise. Segment start points are found on the clockwise trace and end points are found while traveling anti-clockwise. Equation (2.1) gives the criteria for a point i being the starting point of a convex segment,

$$\theta_i - \theta_e > S_1 \tag{2.1}$$

where θ_i is the angle perpendicular to the tracing direction, θ_e is the minimum angle between the point $i-1$ and the starting point and S_1 is a constant.

When each curve segment has been extracted, nine features are used to describe it:

1. C, the index of convexity or concavity
2. $G(G_x, G_y)$, the center of gravity of the segment

3. L1, the arc length of the segment

4. L2, the linear distance between the start point and end point

5. L_0 the degree of concavity

6. AG, the angle perpendicular to the line running through the start and end points

7. D12, the ratio of L1 to L2

8. L12, the difference between L1 and L2

9. H(r), the angular distribution of θ_i for the segment

A mask is constructed for each standard character that consists of the maximum and minimum value for each feature. During classification, the features of input characters are extracted and matched against each of the standard masks, the one with the biggest score being the winner.

Results and Performance for Convex and Concave Features

16,800 handwritten characters were used for testing. 9,600 of these were used for training data and the rest for test data. When the training data was tested against itself, a rate of 99.95% was obtained! When tested against the test data, a rate of 97.2% was obtained. Unfortunately no speed results were given.

2.4 Discrimination Trees

James Geller[11] describes a character recognition method using a quadtree. A quadtree is a tree structure representing a picture, where every node is a leaf or has exactly four sons.

Each character is divided exactly into four regions. These regions are subdivided into four further regions and so on recursively until a region is a single pixel. Each region is represented by a node on the tree. Each node of the tree is either a leaf or has four sons. Nodes contain a 'W' or a 'B' if the corresponding area of the character is all black or all white, or an 'M' if the region is grey. A pixel represents

one leaf on the tree. Every character could have its own tree but in Geller's system, all characters are combined into a single tree. At each node of the tree, a hint list is kept. The hint list lists all the characters with a region described by this node.

When a character is to be added to a tree (during the *learning* or *training* phase), it is analysed by dividing it up into relevant regions. The character name is added to the hint list at every node that represents a region of the character.

In the recognition or classification phase, an input letter is analysed in the same way as in the learning phase. While the character is being analysed, the corresponding nodes of the quadtree are traced through. If the input or unknown character contains a corresponding black or white area at the given level, then the hint list at that level is added to a result list. The letter returned by the classifier as the one being most similar to the input letter is the one that occurs most frequently in the result list. Geller uses an extra feature for recognition. At each node, a number between 0 and 1 is stored that represents the black pixel density, relative to the number of black pixels in the parent region. This density value is used during the recognition phase. Instead of keeping one hint list at every node, several may by kept, each corresponding to a different pixel density. For example, the following information may be stored at a node[11]: $((w\ 0.3\ (a\ e\ o)\ 0.4\ (g\ q))\ nil\ (m\ (...)\ 0.3\ (j\ k\ l)\ 0.41\ (m\ n)))$[3]. This shows, if the region is all white('w') and the black pixel density for the parent region is 0.3 then the suggested letters are 'a', 'e' and 'o'. If the region is all white and the density is 0.4 then the hint list contains 'g' and 'q'. The nil shows that there are no characters whose corresponding regions are all black. The 'm' shows the region is a grey area with different pixel densities for the hint lists (j k l) and (m n). During recognition, only those hint-lists whose black pixel densities match those of the corresponding regions of the input character are added to the result list.

2.4.1 Results and Performance for Discrimination Trees

Tests were carried out on a VAX-750. 4 classes of character were used for testing: 'a', 'b', 'g' and 'o'. The system was trained with one instance of each class and was then tested with ten distortions of each. Characters were handwritten. On the first run, 31 out of 40 or 77.5% were recognised. Better results could be obtained by raising the *effort level* of the program though this would inhibit performance. Learning times were less than 1 minute for a letter! Recognition times "varied

[3]LISP list.

between several seconds and almost a minute"! The average recognition time was fifteen seconds. These times make the system completely impractical.

In a second test, an entire alphabet was presented to the system. One alphabet set was used as the training data and one as the test data. 17 out of 26 or 65% of the letters were recognised. 8 letters required additional teaching.

The advantages of this system seem to be its tolerance of character size and stroke-width variation. Its disadvantages are its intolerance to rotation and its lack of use of the base-line position.

2.5 Stroke Analysis

Stroke analysis involves analysing and extracting the strokes or straight lines of a character and can be thought of as a subset of the feature extraction method. For each character to be classified, each stroke is extracted, analysed and stored. The strokes of the input characters are analysed and compared with the stored sets to find the best match. For example, a capital 'A' consists of three strokes. One left to right diagonal, one horizontal and one right to left diagonal. The length of each stroke might also be stored. This could be stored in pixels (ie 20 pixels long) or it could be quantised (eg one of small, medium or long). So, for the 'A', one stroke is short and the other two are long. Stroke analysis is particularly useful for hand printed characters.

Banno *et.al.*[3] describe a system using stroke analysis. In their system, the character is first skeletonised (a process similar to thinning (see Section 5.5.3)) which reduces it to pixel-wide strokes. Features such as junctions, end points, inflection points and line segments are extracted. Strokes are extracted by calculating the probability of line segments joining together. After the strokes have been extracted, features of each stroke, such as direction, length and mid point are extracted. Directions can be one of four:

- horizontal
- vertical
- left diagonal

- right diagonal

Length can be one of three:

- short
- medium
- long

Templates for each feature are created. There are four directions and three sizes so seven templates are used. For example, template one contains only the horizontal strokes in the character. Template five contains only the short strokes. Similarity is a function of the value of each template for each feature.

In addition to the stroke analysis, each 3×3 sub-area of the character is analysed for contour direction, junctions, end points and inflection points. Similarity of these features is a function of the difference between the input character and the standard character. Language processing is used to resolve errors.

2.5.1 Complexity indices

Saki and Mori[23] describe a preliminary classification system using stroke analysis. Their original paper is in Japanese and is described by Mori and Masuda[20] in English.

Complexity indices were developed for Chinese characters and give a value of how complex a character is in terms of the number of strokes and sub blobs it contains. Although developed for Chinese characters that have their own particular problems in relation to OCR, complexity indices may be useful as a quick pre-classification system for the Roman alphabet. Chinese Katakana characters are composed mainly of vertical and horizontal stroke components. The horizontal and vertical complexity indices for a character, given below, are defined as the ratio of its overall stroke length to the spread value of the pattern.

$$C_x = l_x / \sigma_y \tag{2.2}$$

$$C_y = l_y / \sigma_x \tag{2.3}$$

Where C_x is the vertical complexity index, C_y is the horizontal complexity index, l_x and l_y are the total stroke lengths in the horizontal and vertical directions respectively and σ_x and σ_y are spread values in each direction.

Diagonal strokes can be described in terms of horizontal and vertical components. The spread values are calculated from the second order moments around the central axes. Complexity indices could be used as a preliminary filter system, those standard characters whose complexity indices are not similar to the input character's indices, being filtered out.

Results and Performance for Complexity Indices

No results were given for this method.

2.6 Neural Networks

Rajavelu *et.al.*[22] describe a comprehensive system for character recognition, using a neural network. A computerised neural network mimics the neural interconnections of the human brain. A neural network has input nodes and output nodes and one or more layers of internal nodes with node interconnections having different weights. Rajavelu *et.al.* extract features from an input character before submitting it to the neural net. Twenty features are extracted using Walsh functions [4]. These twenty features provide twenty inputs to the neural net. As the output of the net is to be a single character, that being the one most similar to the input character, the number of outputs is relative to the number of characters that can be recognised. Each output can represent one bit of a character name. Therefore, if eight outputs are used, one character is represented. As Rajevelu *et.al.* don't need 256 characters, they use only seven outputs. For optimum convergence and recognition time, forty hidden nodes were used. Learning involves presenting each set of 20 features of each standard character, one at a time. Learning is implemented using back-propagation which involves changing the interconnection weights by feeding outputs back to inputs until convergence is attained.

2.6.1 Results and performance for Neural Networks

When tested under different conditions, recognition rates of between 97% and 99% were achieved. The system failed to distinguish between the characters 'l'(el) and '1'(one) but this is standard for OCR systems. Another problem was that, as characters are scaled to a standard size before recognition, similar upper and lower case characters can be confused. This could be remedied by adding a length feature to the input. In general, a problem of neural net systems is the slowness of the training phase.

The authors cite the advantages of this neural network system over existing techniques as being:

1. high speed of recognition
2. recognition for characters with improper alignment
3. font independence

The authors claim that the preprocessing techniques used are far less computationally expensive than those used in existing methodologies such as the *line adjacency system* of Kahan *et.al.* [17] while retaining efficiency and versatility.

2.7 Summary of this chapter

This chapter details several different character recognition methods. The methods cited vary from the well known (eg Stroke Analysis) to more unusual algorithms (eg using convex and concave features). Where possible, results obtained by the respective authors are shown. These results prove very difficult to compare and contrast because of the very different testing techniques, number and types of characters used and widely varying hardware. The recognition system used in this project, template matching, is described initially. Template matching was chosen as the primary classification method because of its simplicity, ease of training and extendibility and the ability to readily identify the general structure of a character without having to view all the details. It also lends itself well to parallel processing. In addition, global template matching has rarely been used as a primary classification method because

of its perception of being more simplistic than more sophisticated methods. As can been seen from the chapter on results, this project has shown template matching to perform very well.

Chapter 3

System Overview

3.1 Overview of this Chapter

This chapter details the design and development of the project. The project is described from a top-down orientation showing how a full page is classified first by segmenting entries and then individual characters. All the modules and main functions used are described. Finally, the main data structures are listed.

3.2 Development Steps and Modules

The aim of this project is to design and construct OCR software that will be used to computerise Trinity College Old Library Catalogue, that is, to scan each page of the catalogue, convert to ASCII code, segment the fields of each entry so that they may be added to a database.

The project has several steps:

1. Scan each page with a digital scanner, compress and save to disk.
2. Run the OCR software on each page.
3. Write all segmented entries and fields to a database.

Step 2 can be broken down into a number of sub steps:

1. Detect the beginning and end point of every entry on the page.
2. For every entry, segment the lines of text one by one.
3. For every line, classify every character in the line primarily by template matching.
4. Using a state table, run through every classified character and delimit individual fields.

The software was originally written in Turbo-C and ran on an IBM AT. Because of the memory limitations of MS-DOS and the lack of processing power, the project was ported to an InMOS T800 transputer running Parallel C. The main procedure of the program consists solely of a large case statement that traps key presses. The main OCR functions are called from this case statement as well as several test functions, some of which have been combined with the main OCR and some of which have been discarded. The program consists of several modules as shown in Table 3.1. There is a 'C' header file for each module as well as two extra header files, tocr_var.h and tocr_def.h which list variable definitions and define commands respectively.

Module name	Description
tocr.c	main case statement and declarations
tocr_an1.c	OCR high level functions
tocr_an2.c	OCR low level functions
tocr_io.c	input/output functions
tocr_util.c	utility functions
tocr_help.c	help text
tocr_scan.c	digital scanning functions
graphics.c	transputer to PC graphics

Table 3.1: Modules used in the project.

The 'tocr.c' module contains the mainline and consists of a large case statement that calls the other functions. The 'tocr_an1.c' module contains high level OCR functions that call the low level OCR functions of 'tocr_an2.c'. The 'tocr_io.c' module contains input/output functions concerned with screen, disk and printer. The 'tocr_util.c' module contains utility and user interface functions. The 'tocr_help.c' module contains a help function that provides help on all the operations available to the user. The 'tocr_scan.c' module contains functions for digitising a page in the

scanner. Finally, the 'graphics.c' function is concerned with drawing graphics in transputer memory and then transferring them to the PC host VDU via the Alien File Server.

Figure 3.1 is a representation of the structure of the project. Only the central functions are depicted. For clarity, all other functions are left out. The arrows from one function to another indicate that the target function is called by the other. The function may be called more than once. For clarity, no loops are depicted. The numbers in the bottom right corner of the function boxes indicate the level of the function. Thus, the level one function is broken down into level two functions, each of which are broken down into level three functions and so on. Only the important functions are broken down. No training or tracing functions are shown.

3.3 Classifying a full page

The program (see Figure 3.1) provides various functions to classify either a single character, a single line or a single entry. These were used as development tools to test the OCR software. When the software is invoked to fully classify an entire page, the *guesspage*[1] function is called. This is the upgraded version of the *old_guesspage* function that classified an entire page line by line. The guesspage function uses the full OCR software described in this project to classify a page entry by entry and using a state machine (see Chapter 6). Guesspage starts off by calling the *mark_entries* function. The mark_entries function finds the first character of each entry on the page and writes the coordinates of each of these characters into an entry array. In this way the location of each entry is known. Next, the *find_header* function is called. This function reads the two three-letter headers at the top of the page. These are used to gain alphabetical knowledge about the page (see Chapter 7). Next, the *first_and_last* function is called. This function reads the first and last surname on the page and finds which characters are common (see Chapter 7 again). Guesspage then calls the *guessentry* function, for every entry on the page found by the mark_entries function. The guessentry function does most of the heavy work and is described in detail in Section 3.3.1.

[1]In this section, functions are written initially in italics and thereafter in normal text.

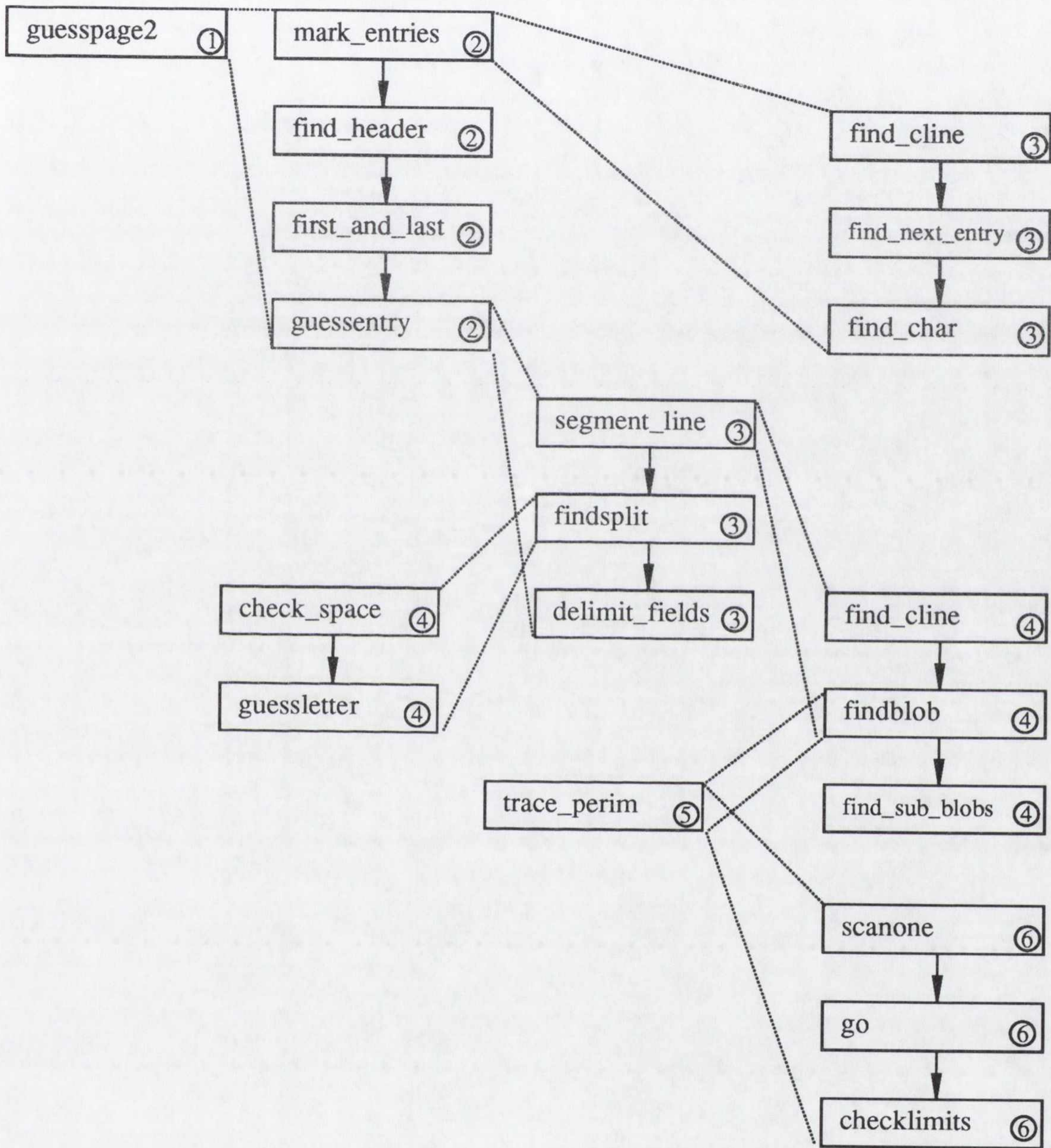


Figure 3.1: Functional breakdown of the project.

3.3.1 Classifying entries

The guessentry function takes the number of an entry as a parameter and then classifies that entry line by line, using a state machine. How an entry is segmented is described in Section 5.2. Guessentry first calls *segment_line* which searches for a line of blobs within the given coordinates and places each blob found in a linked list. The list is later used by the character recognition software. Next, guessentry calls the *findsplit* function. Findsplit looks at the blobs in the linked list created by segment_line and using character recognition, decides if any of these blobs are character fragments and should be joined together. When full characters are found and classified, they are passed on to the state machine in the function *delimit_fields.* Findsplit calls the function *check_space* to look for spaces between words and calls the function *guessletter* to classify characters. The main character recognition by template matching happens in the guessletter function and is described in Section 3.3.1. The delimit_fields function contains a large case (switch) statement where the switch condition is the current state. Thus when a character is sent into the delimit_fields function, the correct case arms test for a state trigger condition and may change the current state. Then delimit_fields returns and findsplit finds and classifies another character. The perimeter tracing functions are described in Chapter 5.

Classifying characters

The character recognition or classification happens inside the guessletter function. The coordinates of the bounding box of a blob are sent into the function. The function then compares this blob or input character with all the templates in each of the template sets associated with the current state. The name of the best match character is returned as the function. The function also returns the number of the state from which the template came, the number of the template within that set and the score for the match. Before matching, templates are filtered by width and height (see Section 4.4). The function contains a case (switch) statement to allow for different template scoring methods. The switch condition is the current *method.* Scoring methods are described in Section 4.3.

3.4 Main data structures

A digitised catalogue page was represented by a 2-D array, 3900 pixels long and 2544 pixels wide. An array of this size (over 1.2 Megabytes) would not fit in memory as one contiguous block, so memory for each line of the page was allocated dynamically. So the data structure used is:

```
#define WIDTH 2544
#define LENGTH 3900

unsigned char *page[LENGTH];

for(loop = 0; loop < LENGTH;loop++)
   if( (page[loop] = (char*)calloc(WIDTH/8,1)) == NULL){
      printf("Error [etc.]\n");
      exit(1);
   }
```

Memory for the template data structure is also allocated dynamically. As described in Section 6.3.3, templates are divided into sets. Each set is allowed a maximum of 100 templates. Templates consist of a bitmap and associated statistics including width and height as follows:

```
#define SETS 8
#define SIZE 48
#define ALPHABETSIZE 100

typedef struct{
   char name[5];
   char font;
   int width;
   int height;
   int perimeter;
   int width;
}st;

typedef struct{
```

```
    unsigned char bitmap[SIZE][SIZE/8];
    st stats;
}tmplt;

tmplt *alpha[SETS];

for(loop = 0;loop < SETS;loop++)
    if( (alpha[loop] = (tmplt *)calloc(ALPHABETSIZE,sizeof(tmplt))) == NULL){
        printf("Error [etc.]\n");
        exit(1);
    }
```

Chapter 4

Template Matching

4.1 Overview of this Chapter

In this project, a page is represented by a two dimensional array of pixels or picture elements. In the segmentation stage, this collection of dots is examined to decide which dots belong together and constitute a character. These collections of related pixels, which are called blobs are then presented to the classifier which decides which printed character they represent. The primary classification method used in this project is called template matching and is the subject of this chapter. Several matching algorithms are described and evaluated. Filters are introduced and their success in this project is analysed. The problems of template matching are described and ways to improve the process are discussed.

4.2 How template matching is used

Template matching can be used in two distinct ways:

1. To decide if a known pattern exists. eg. to decide if a picture contains a bird, a template of a bird would be tried in every possible position on the picture.
2. When presented with a pattern, to classify that pattern as one of a known type. eg. if we have a picture of a bird, the picture can be compared with a set of known bird templates to decide what type of bird it is.

For OCR, template matching is used in the latter fashion. Each segmented blob of pixels is presented to the classifier. During the classification stage, the blob is known as the input character. To decide what the input character is, ie A,B,C etc., the classifier compares it to an array of character templates it holds in memory. These characters are known as the golden templates and are explicitly identified to the computer during the training stage, before recognition. The classifier overlays the input character directly on each golden template and by matching pixel against pixel, a probability score is generated for each template. The template with the highest score is the one that most closely matches the input character.

4.3 Probability scoring

A probability score is generated for each template with respect to an input character by superimposing the template on top of the character and counting how many pixels match. While in essence, a simple operation, a number of difficulties arise. Should both black *and* white pixels be matched or just black pixels on their own? What should be done with pixels that don't match? What is the correct position to superimpose the template? The first two questions can be answered by an analysis of the pros and cons of different scoring methods. I have analysed five distinct methods of scoring as follows:

1. Give one point for each black pixel that matches.
2. Give one point for each black or white pixel that matches.
3. Give one point for each black pixel that matches and take away one point for every pixel that does not match.
4. Give one point for each black or white pixel that matches and take away one point for every pixel that does not match.
5. Take away one point for every pixel that does not match.

See Tables 4.1 and 4.2 for examples of each scoring method. Method 3 can be shown to be superior to method 1 because it will discriminate between characters that method 1 will render with the same score. Method 1 renders some characters

with the same score because some characters are complete subsets of other characters. For example the letter 'I' may be matched perfectly with the letter 'H' (using method 1) as the left leg of the 'H' forms a perfect 'I'. See Figure 4.1. While methods 2 and 4 give a higher score value, adding in the white pixels merely adds a fixed amount to the score of every template and is therefore not productive. For these reasons, I have chosen method 3. Note that through filtering based on width and height (see Section 4.4), not all templates have been matched. In Table 4.1, the top score for each method is shown in bold. The second highest score for each method is underlined. For both characters, 'E' and 'e', method 3 gives the greatest difference between the highest and second highest score. For character 'E', method 3, the highest score is 65, the second highest is 40 giving a difference of 25. The other differences are 18, 1, 1 and 1 showing clearly that method 3 discriminates best. For character 'e', method 3, the highest score is 92, the second highest is 46, giving a difference of 44. The other differences are 26, 1, 3 and 1 showing again that method 3 discriminates best.

Method	Max.	C	S	G	O	T	V	P	B	H	L	E	F
1	128	35	46	48	33	46	47	91	<u>105</u>	69	88	**123**	95
2	128	105	104	102	101	107	107	115	115	105	<u>119</u>	<u>119</u>	**120**
3	128	-121	-122	-124	-150	-98	-99	0	17	-88	29	**65**	<u>40</u>
4	128	83	80	77	74	87	87	102	102	83	<u>111</u>	<u>111</u>	**112**
5	128	106	105	103	102	108	108	116	116	106	<u>120</u>	<u>120</u>	**121**

Table 4.1: Score values for the character 'E' (highest values in bold).

Method	Max.	r	s	a	z	c	e	o	u	n
1	128	53	51	92	76	<u>92</u>	**118**	88	81	74
2	128	119	118	120	120	<u>124</u>	**125**	121	119	117
3	128	-53	-71	-2	-25	<u>46</u>	**92**	9	-27	-56
4	128	111	109	113	112	<u>120</u>	**123**	115	111	107
5	128	120	119	121	121	<u>125</u>	**126**	122	120	118

Table 4.2: Score values for the character 'e' (highest values in bold).

In practice, counting black pixels that match can be done with an AND function and counting pixels that are different can be done with an XOR function. For speed, operations can be done a byte at a time. One byte from the input character is ANDed with the corresponding byte from the template and then the bits in the result are counted. Again, for speed, the counting can be done with a look up table (LUT). The LUT is set up once initially by counting the bits in each number from

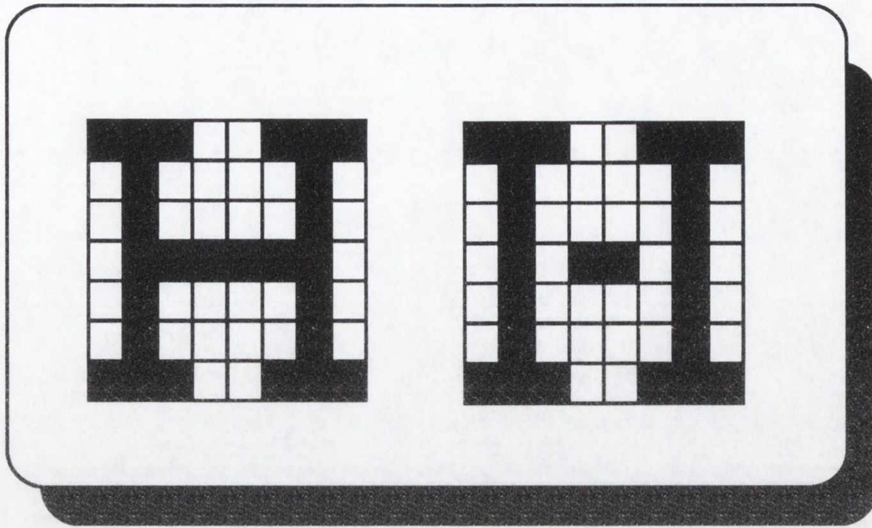


Figure 4.1: The character 'H' can be mistaken for 'I-I'

0 to 255. The score for a byte is then given by:

$$LUT[c\ AND\ t] - LUT[c\ XOR\ t] \tag{4.1}$$

where t is a byte from the template and c is the corresponding byte from the input character. Note that AND and XOR refer to numerical bitwise functions, not logical functions. The score for the whole character is given by:

$$score = \sum_{x=1}^{n} LUT[c_x\ AND\ t_x] - LUT[c_x\ XOR\ t_x] \tag{4.2}$$

where n is the number of bytes in the template.

In practice, there is a more efficient way to get the desired result. As mentioned in the above paragraph, the black pixel area of both the template and the input character are known. When the template and character are ANDed together, the value of the two blobs XORed together is given by the area minus the AND value as the only pixels left out of the area must be those that don't match. This must be done for both template and character. This means that, for a template containing n bytes, n XOR operations can be replaced by two subtractions and one addition operation as follows. From equation 4.2, the total number of black pixels (the AND value) that match is given by:

$$\mu = \sum_{x=1}^{n} LUT[c_x\ AND\ t_x] \tag{4.3}$$

The total number of pixels that do not match (the XOR value) is then given by:

$$\sum_{x=1}^{n} LUT[c_x\ XOR\ t_x] = (carea - \mu) + (tarea - \mu) \tag{4.4}$$

where *carea* is the number of black pixels in the input character and *tarea* is the number of black pixels in the template.

Substituting terms into equation 4.2, the total score is then given by (4.3) minus (4.4) ie:

$$score = \mu - (carea - \mu + tarea - \mu) \tag{4.5}$$

which gives:

$$score = \mu - carea + \mu - tarea + \mu \tag{4.6}$$

giving the total score as:

$$score = 3 * \mu - carea - tarea \tag{4.7}$$

or

$$score = 3 * \sum_{x=1}^{n} LUT[c_x \ AND \ t_x] - carea - tarea \tag{4.8}$$

thus eliminating the XOR term from equation (4.2).

If we wish to use a threshold with the character score, that is, we want a cut-off point for bad matches, then we need to normalise the score. The score is not naturally normalised because characters can vary in size and so the maximum score available for each template can be very different. To normalise the score, a percentage can be used. The normalised score then, is given by:

$$\frac{(real\ score) * 100}{(area\ of\ character)} \tag{4.9}$$

Note that the maximum score possible is given by the area of the character, i.e., the number of black pixels in the character. Note also, to optimise for speed, 128 may be used rather than 100 so that a shift operation is used rather than a multiply.

4.4 Filters

Filters can greatly reduce the number of template matches and can sometimes increase accuracy. Normally, to identify what template an input character is most like, it is compared to all the templates in a template set. Using a filter however negates the need to match against all templates. Some templates, which are obviously dissimilar to the input can be filtered out. Templates are 'obviously dissimilar' when

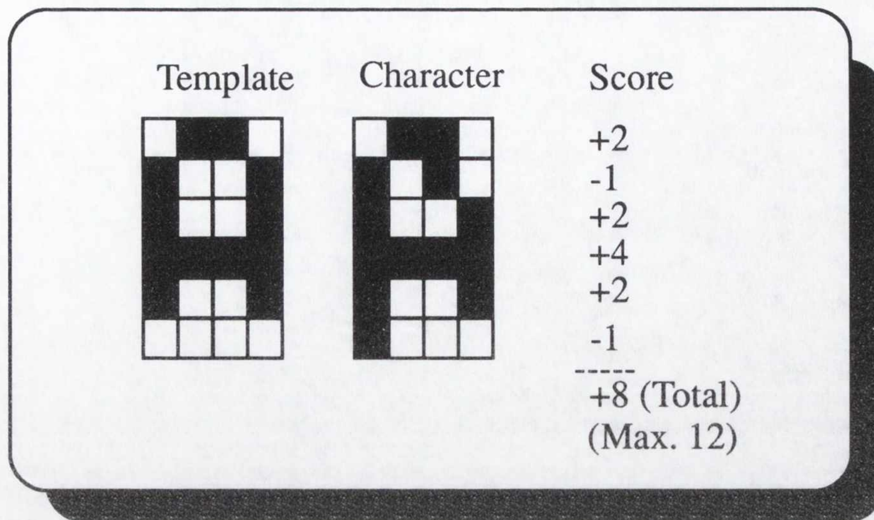


Figure 4.2: Matching the character 'A' by Method 3.

for example their width or height is completely different to that of the input. For example, comparing a 'W' template with a full stop character '.' wastes a lot of time when a simple check on the width difference can filter out that match. Refer back to Tables 4.1 and 4.2 again in Section 4.3. In Table 4.1, the character 'E' is matched only against 12 templates, not the full alphabet. In Table 4.2, the character 'e' is matched only against 9 templates. Close inspection will show that those templates matched against the characters are all similar in height and width.

Several filters can be used. Width and height filters were used in this project but perimeter and area filters could also have been used. It was found however that perimeters and areas deviated much more than widths and heights and were generally unreliable as filters. It must also be remembered that, for best results filter features must be independent of each other. Area and perimeter are not totally independent of width and height and so will generally filter out the same templates as the width and height filters, with no gain. The perimeter value was even more unreliable in cases where a character illegally touched itself thus enclosing an area and altering the perimeter length (see Figure 4.3). For example, if the end points of the 'c' character should join making an 'o', the perimeter length is thus made much smaller than it should be for a standard 'c'.

Filtering is achieved by subtracting the template value from the character value and comparing the absolute value of the result with an *a priori* threshold percentage and ignoring templates above the threshold percentage. If a high threshold value is used, eg. >50%, then the value of the filter is diminished as templates which are obviously dissimilar will be matched. On the other hand if a very low threshold

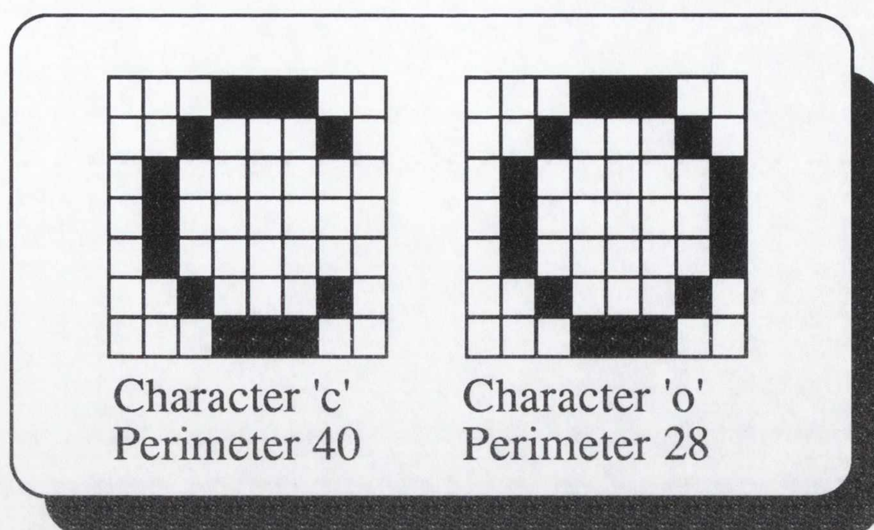


Figure 4.3: Character 'o' has a much smaller perimeter than character 'c'.

percentage is used, eg <5% then templates of the correct class may be erroneously removed. A value must be chosen that is high enough to ensure that 'good' templates are never removed. Experiment showed that character widths and heights could deviate from their means by as much as 10%. A threshold percentage of 15% was chosen to ensure that characters of the correct class were not ignored. Thus if a template's width and height were not within 15% of the input character's width and height respectively, the template was filtered out. For small characters (full stops, commas etc.), the 15% value was replaced by a constant value of 5 pixels because small characters tended to deviate by a lot more than the standard 10%. The width filter used in this project is given by:

$$filter = (abs(twidth - cwidth) < max(5, cwidth * 15/100)) \tag{4.10}$$

where twidth is the width of the template and cwidth is the width of the character.

4.4.1 Analysis of success of filters

At the time of testing the success of the filters there were 314 templates used by the system. Table 6.3 in Section 6.3.3 shows that these 314 templates were divided into 8 groups. When an input character was matched only against the one set to which it belonged, between 36% and 94% of templates were filtered out. On average 70% of templates were filtered out. When the same input character was matched against all 314 templates, between 78% and 99% of templates were filtered out. On average, 87% were filtered. The figure of 87% means that out of 314 matching operations, 273

could be avoided, leaving 41 templates to be matched. If a set were used, containing on average 40 templates, 70% percent or 28 of these would be filtered out leaving only 12 (or 4%) of the original 314 templates to be matched, making the matching process about 25 times faster.

The filtering percentage of course depends on the input character. For example, characters with distinctive heights or widths eg: '.', 'i', 't', 'W' are dissimilar to most other characters and hence most templates will be filtered. Other characters such as 'e', and 'L' have more average dimensions and so fewer templates will be filtered.

4.5 Multiple matching by shifting position

What is the correct position to superimpose the template on the input character?

In the segmentation stage, the bounding box of an input character is found by tracing the perimeter of the blob and recording the limits in four directions. The simplest way then to match the input character against the templates is to 'superimpose' the template on the input character so that the top left corners of the boxes match up. I will refer to this position as the central position. This may not be the optimum position. The input character is never perfect. If it was, the OCR process would be trivial. For various reasons, including dirt on the page, dust on the scanner window, excess ink when printing or shards of metal on the printing blocks, a character may contain 'spikes'. A spike will erroneously extend the bounding box of a character in a particular direction. Because of the spike, the central position is no longer the optimum position. To find the optimum position, a series of matches must be made, by shifting the template one or more pixels in every direction. Casey and Nagy[9] matched as much as seven pixels away from the center. 'Every' direction may mean the four points of the compass - top, bottom, left, right or may include the diagonals - top left, top right, bottom left, bottom right, making nine positions in all, including the center. The optimum position is then the one that gives the highest score. In the early seventies, when primitive OCR machines were large and expensive, shifting was done in hardware, using huge shift registers that allowed the characters to shift past the sensors at very high speed[7]. Hardware solutions are no longer absolutely necessary thanks to high speed processors.

If care is taken to keep the scanner and page as free of dust as possible and

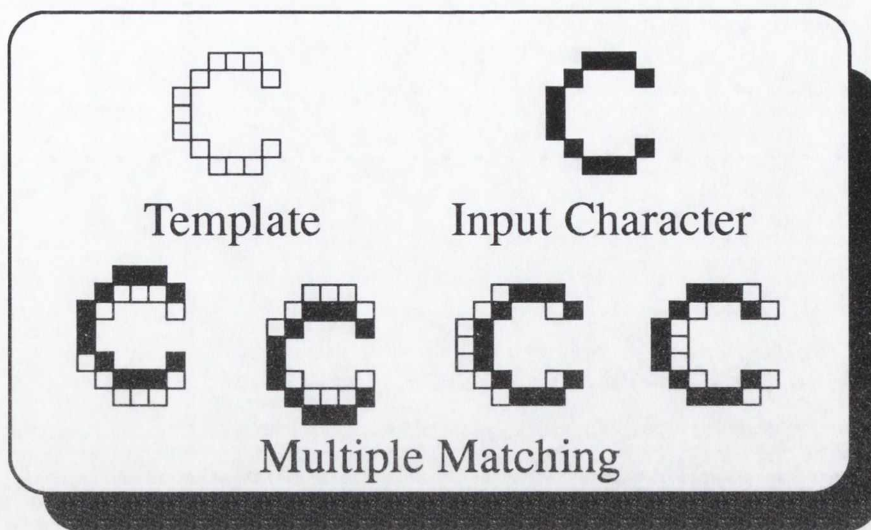


Figure 4.4: Multiple matching in four extra positions.

contrast and brightness settings are carefully chosen, then, in most cases, spikes are not significant and the central position will give a good match. This is important because matching a template six times over in shifted positions makes the program six times slower. Multiple matching is computationally expensive but can be useful in resolving doubtful matches. For example, it could be used only when a direct match at the central position has produced a very low match score. The author experimented with multiple matching and found it to be very useful. However, recognition speed is reduced dramatically if the method is used for every match. The usefulness of multiple matching seems to lie in resolving bad matches. Thus, when used only rarely, recognition speed will not drop dramatically.

4.6 Problems with template matching

As a pattern recognition system, template matching is very elementary and takes no notice of the shape or detailed physical characteristics of the character. As a result, some problems arise. These problems are described in the following sections.

4.6.1 Erroneous match because of tilted text

When an input character is to be matched against a set of templates, care must be taken to ensure that the character and templates are in the same plane. For example,

there is no guarantee that the page has been aligned properly in the scanner, or that the original text was not printed at an angle to the normal. It must also be made certain that all the templates are in the same plane and that that plane is known.

Let us examine what will happen if the page[1] is inserted in the scanner at an angle to the norm. If the tilt is very serious (ie greater than a few degrees), the first problems will be encountered in the segmentation stage as lines of text will not be straight. However, for the purpose of template matching, let us presume the characters have been segmented correctly. Let us presume also that the templates have been aligned correctly. Because template matching does not take account of the physical characteristics of the character, it cannot decide which pixels in the character and template correspond. Because it scores the pixels line by line in a horizontal grid, the wrong pixels will be matched and a bad score will result. The worse the angle of tilt, the worse the score.

To avoid this problem, one of several things can be done:

1. While placing the page in the scanner, make sure it is aligned properly by eye.
2. Calculate the angle of tilt of the page and adjust the character or templates accordingly.
3. Calculate the angle of tilt of the page and instruct the operator to readjust the page.
4. Calculate the angle of tilt of the page and rotate the page mathematically.

Initially, while developing the software, I relied on method 1 above. This proved to be unsatisfactory as the software was intolerant to a very small degree of tilt. This made it impossible to judge with the human eye. It was clear I needed to calculate the slope of the page.

Methods 2 and 4 both relied on a rotation algorithm. I disliked this for two reasons. First, rotation is time consuming and secondly I felt that the extra element of distortion brought in by the rotation would be unacceptable. The library pages have a title line containing two three character leaders (see Figure 1.1). I was able to read this line and then calculate the slope. In practice, this meant the scanner

[1]A picture of a library page can be seen in Figure 1.1, in Section 1.3.1 towards the front of the thesis.

reading a small portion of the top of the page, finding the title line, calculating the slope and then reporting to the operator if the slope was outside a certain tolerance and then rechecking when the operator had corrected the alignment. This is effectively method 3 above. This worked well in practice but it must be remembered, the method becomes more unstable as the angle of tilt increases. This was not a problem as the human eye can align the page sufficiently initially.

4.6.2 Erroneous match because of bias

Template matching as I have implemented it causes another problem which was not apparent to me initially.

When an input character is ready to be matched, it is copied from the page into a two dimensional array. The character is copied pixel by pixel into the top left of the array (see Figure 4.5). This causes problems as it presupposes that the top and left hand sides of the character are as they should be and do not contain spikes (see Section 4.5). If there are spikes on the top or left, the character will not be aligned properly with the template. The author has not addressed this problem but solutions include multiple matching as described in Section 4.5 or aligning the character with the template by finding the center point of each.

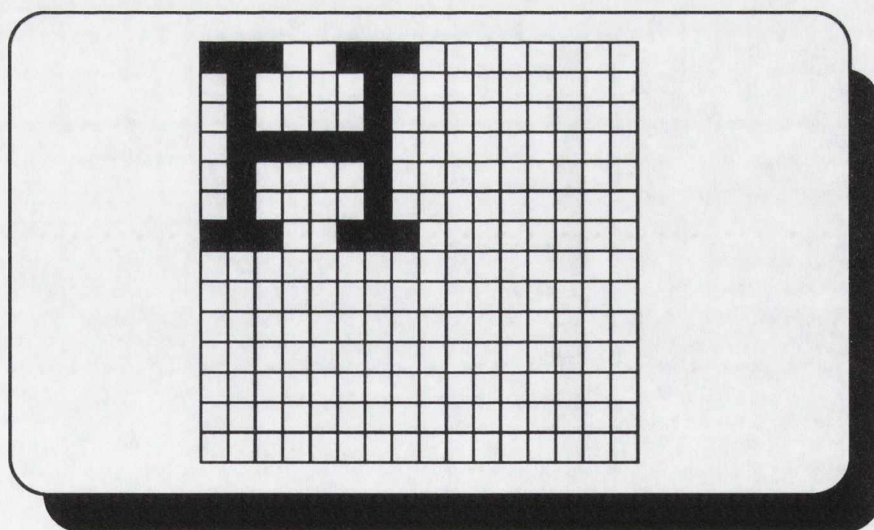


Figure 4.5: A character is stored in the top left corner of the template array.

4.7 Training

For any system to recognise characters, it must have some description of what each character looks like and its corresponding name. When a candidate character is ready for classification, it is compared with each character description or set of rules (depending on the classification system) and the name of the character it most closely matches is returned. The process by which a system is configured with the character names and descriptions is called training. Some OCR systems need minimal training and this can be *hard-wired* into the program so the user is oblivious to it. Such systems try to provide general purpose OCR and will attempt to recognise any font; best results being gained from standard printed text on A4 paper. For a system to give good results with unusual fonts and page layouts, it must be possible for the user to reconfigure it. Thus the package usually has a training option where the user can identify new fonts to the system. As this project is specifically concerned with recognising pages from the old library catalogue, a comprehensive run time training system was important. As the primary classification system used in this project is template matching, this section only concerns itself with template training.

In training mode, a blob or candidate character is found in the normal way. During normal recognition, this would be passed on to the classifier. In training however, the blob is highlighted on the screen. I chose to highlight the blob by drawing its bounding box rectangle. The blob could also be highlighted in reverse video. Blob statistics such as area, perimeter, width and height are also displayed. When the blob is highlighted on the screen, the user is invited to explicitly identify the character by name. After typing the name, the user is asked which character set the template should be put in. The use of reduced character sets is described in Section 6.3. The character bitmap or template is then copied from the page array to the end of the relevant template array if there is space for more templates. The new template set is not saved to disc until the user wishes it to be. Other information such as the character's perimeter, area width and height can also be saved. The following C structure is the template data structure used in the program:

```
#define TEMPLATESIZE 48

typedef struct {
    char name[5];
    char font;
```

```
    int width;
    int height;
    int perimeter;
    int area;
}st;

typedef struct {
    unsigned char bitmap[TEMPLATESIZE][TEMPLATESIZE/8];
    st stats;
}tmplt;
```

4.7.1 Template Selection

Selecting which instances of a character to store as a template is very important. Ideally, the template should be an *average* of all instances that can be encountered. This would ensure that the difference between any input character and its template would be kept to a minimum. In practice, the template is chosen by the operator. The template should not necessarily be a perfect representation of the character. If the system is biased in some way (for example, the scanner contrast settings are too high) then a template that reflects that bias should be chosen. Noise will not necessarily average out.

It is also important to realise that templates should not be taken from a page that is tilted too much. Tilted templates will obviously lead to bad results. This should not be a problem if some form of tilt correction system is being used (see Section 4.6.1). Where a page is tilted beyond a certain tolerance, I chose to have the page manually repositioned rather than doing a bitwise rotation to correct it. If a rotation algorithm were used, templates could be taken from any page at any angle and rotationally corrected.

Chapter 5

Segmentation

5.1 Overview of this Chapter

The process by which the computer decides which pixels should be grouped together to form a character, which characters form a word and which words form lines, is called segmentation and will be the subject of this chapter. The process of segmenting a character is described in depth from finding the bounding box by perimeter tracing, to resolving split and merged characters. Other problems such as multi-blob characters and what to do with inter-word spacing are also dealt with.

5.2 Segmenting entries

When the top left of the first character in the first entry has been found, that point is marked as the coordinate of the top left of the first entry. The perimeter of the blob can then be traced to find its bounding box (see Section 5.4.2 and Figure 5.2). As can be seen in Figure 5.1, the first word in each entry protrudes slightly to the left of the rest of the text in the entry. This means that the first character in the next entry can be found by scanning directly down from the first character until the first character in the next entry is located. From the top left of that character, we have both the top left coordinate of the second entry and the bottom left coordinate of the first entry. The distance from the left hand side of the text to the center line is known and hence the four coordinates of a box totally enclosing the entry can be calculated. By hopping from the first character in each entry, an array of bounding

TALLEYRAND (Henry de), compte de Chalais.—
Récit de son exécution [tom. V. p. 131, 2de série,
Archives curieuses de l'histoire de France par
F. DANJOU]. Gall. M. 14. 34.

TALLEYRAND - PERIGORD (Charles Maurice,
prince de).—Memoirs of him, containing the particulars of his private and public life, of his intrigues in boudoirs as well as in cabinets, by the author of the Revolutionary Plutarch.
Lond. 1805. 12°2 vols O. mm. 63, 64.

— Causeries sur les affaires du tems entre les deux premiers jongleurs de l'Europe [Louis Phillipe et Talleyrand].
[*car. tit.*] 8°. G. tt. 27. N°. 5.

Figure 5.1: 3 Entries from page T12 of the catalogue.

coordinates can be constructed for each entry. The next stage is to take one entry at a time and segment the lines and characters within that entry.

5.3 Segmenting lines and characters

To segment the lines in an entry, the top left coordinate of the entry is found, which is the top left of the first character in the entry and the bounding box of that character is found by tracing its perimeter. The next character is found simply by scanning to the right of the first character and tracing its perimeter in similar fashion. In this way each character in a line can be segmented. Segmenting characters must stop before hitting the center line which is a known distance from the beginning of text. When a character has been segmented, it can be immediately passed on to the recognition software to be classified or it can be retained in a buffer. In the early development stages of this project, the former method was used. The project was later changed so that all characters in a line are segmented, stored in a buffer and then passed one at a time to the classifier. By storing the characters in a buffer, they can be manipulated before classification. Why might it be desirable to manipulate

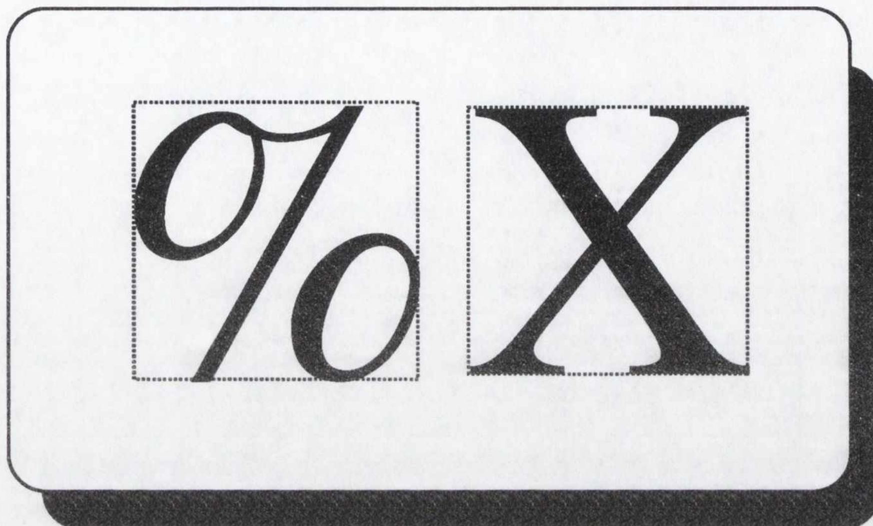


Figure 5.2: Bounding boxes of two characters.

the characters? Because characters are found by looking to the right of a previous character, two things can happen: some characters or parts of characters can be missed such as dots on 'i's (see Section 5.3.1) and punctuation marks and extra characters and noise can be interpreted as legitimate characters such as characters on the preceding or following line. By storing the characters in a buffer, the real high and low limits of the line can be decided and descenders can therefore be removed from the line above and ascenders from the line below and we can look for dots on 'i's. Rather than use an array, I used a doubly linked list to optimise for speed. Using a buffer can also help with splits and joins as described in Section 5.5

5.3.1 Problems with multi-blob characters

Some characters are composed of multiple blobs such as the characters 'i', 'j', ':',';','%','"','!'. Except for the quotation mark ("), all these characters are split in the vertical plane. The quotation mark (") is split in the horizontal plane and as such, it is usually resolved by the split detection system. For characters split in the vertical plane, the bounding boxes of the individual blobs must be merged together

to form the bounding box of the whole character. When a blob is found, the area directly above and below that blob must be scanned for sub-blobs. When a blob has been found, it is passed to the *find_sub_blob* function which scans above and below the blob within the confines of the current line. Sub-blobs of a multi-blob character are then *joined* by finding the combined bounding box of the constituent blobs. For example, the character 'i', is a multi-blob character containing two blobs: the main shaft of the 'i' and the dot. The main shaft is usually found first as it protrudes slightly more to the left. By scanning directly above the shaft, within its width, the dot can be found. There is no need to scan below the shaft as its base corresponds with the bottom of the line. If the dot is found first, scanning below will find the main shaft.

Scanning for sub-blobs presents problems when dealing with *italics* as italic characters tend to lean over each other. Because of this leaning, dots on 'i's etc. may be overlooked or the previous character may be re-traced with disastrous effects, so it is best to turn off sub-blob scanning while classifying italics and rely on split detection to resolve multi-blob characters. This means that the point where text changes to italics and the point where it changes back must be detected. Detecting these change points is relatively easy if template sets are used and if not, the change can still be detected by storing the font type and style with the template.

5.3.2 The space character

As well as the characters that compose words, the space characters (' ') which separate words must be detected. Once detected, a space can simply be converted to the ASCII space character and included in the output. There is no need for a 'space' template. Space characters are detected be measuring the gap between characters. This is done in conjunction with the split detection system. If the gap is below a certain threshold, a split character is suspected and investigated. Otherwise a space character (ASCII 32) is added to the output text. Sometimes, a large space is deliberately left between successive characters. This space can constitute several space characters and it is a matter of taste whether all these space characters should be processed. I chose to condense all space down to one character as the layout of the entries is not important to the output which is sent to a database.

```
struct space_structure {
        int character;
```

```
        int size;
  }
```

In this project, a simple structure was used for a space character containing two fields, a boolean, indicating if a space is present or not and a size field, showing the size of the space. When a space is detected, the space structure is set accordingly and this is passed to a state machine (see Section 6.3). The size field is necessary as it is used in some of the trigger conditions in the state machine.

5.4 Finding the bounding box

To find text on the page, the page is scanned backwards and forwards, line by line in an area of the page where something is expected to be found, until a black pixel is found. Having found the pixel, it must be ascertained if it is a piece of noise or part of a character. To do this, any other pixels that may be connected to it must be found. All neighbouring pixels could be examined recursively and a map of the blob built up or the perimeter of the blob associated with the pixel could be traced. As the first method is obviously slower, I chose the second. Using perimeter tracing, both the area and perimeter of the blob can be calculated as a byproduct of the perimeter trace algorithm. Perimeter tracing can also produce a chain code which is just a list of all the positions on the perimeter. Chain codes can give useful information about a blob, including the location and size of features such as strokes and corners. The author experimented with chain codes in an attempt to calculate the area of a character while tracing its perimeter. This was unsuccessful and was not pursued.

Banno *et.al.*[3] use vertical and horizontal signature analysis (see Section 5.5.2) to find all the bounding boxes of characters in a line. A horizontal signature analysis or projection is performed on the entire page, which delimits the lines of text. Vertical signature analysis is then done on each of these text lines to segment each blob in the line. As each blob is segmented, its bounding rectangle is revealed. This method was not used in this project. Banno *et.al.* do not say how how the signatures are analysed to find the individual blobs. The author of this project felt that this was non-trivial and that the straightforward perimeter tracing method described above, was at least as fast and rendered more useful information.

5.4.1 Connectivity

A brief discussion of connectivity will make the perimeter tracing process easier to understand. Connectivity is about which neighbouring pixels are connected to a central pixel. With 8-way connectivity, the central pixel is connected to all eight neighbouring pixels. With 4-way connectivity the central pixel is connected only to the four pixels at the poles of the compass. See Figure 5.3. In a binary image, background and foreground pixels must be of opposite connectivity (ie. 4-way and 8-way or vice versa). If they are not, impossible situations occur. In Figure 5.4, if both the white pixels and the black pixels are 8-way connected, then the two white pixels are connected to each other *and* the two black pixels are connected to each other which is impossible. If the white pixels are 4-way connected and the black pixels are 8-way connected then normality is restored. For perimeter tracing purposes therefore, background (white) pixels are considered 4-way connected and foreground (black) pixels are considered 8-way connected.

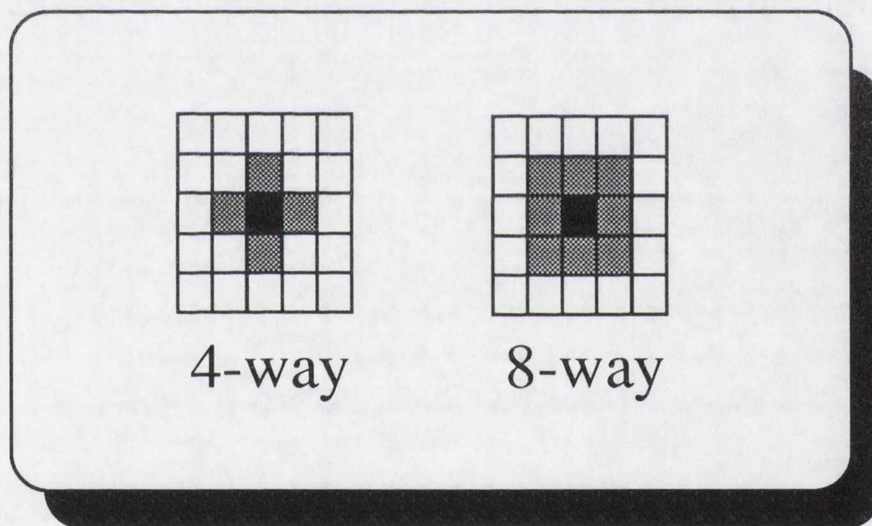


Figure 5.3: 4 and 8-way connectivity.

5.4.2 Perimeter tracing

In the wider area of computer vision and pattern matching, perimeter tracing is often referred to as contour analysis. In a binary image as used here, the term 'perimeter tracing' is adequate.

To trace the perimeter of a blob, a start point is needed. This start point is provided by an initial search procedure as described above. The function of the

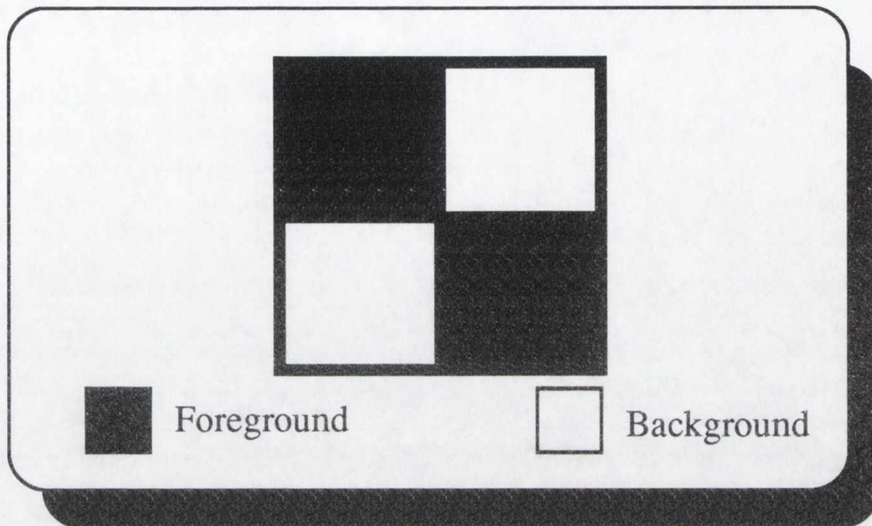


Figure 5.4: Black pixels are 8-way, white pixels are 4-way connected.

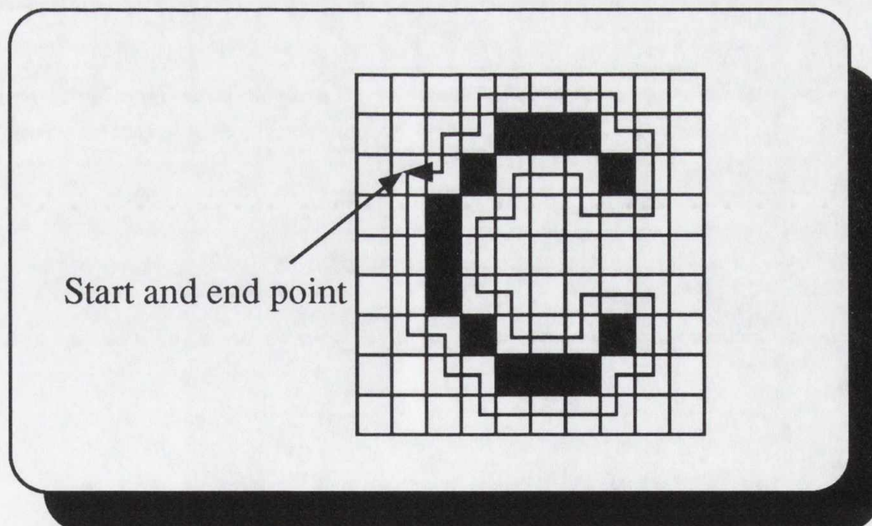


Figure 5.5: Tracing the perimeter of the character 'C'.

perimeter tracer is to move around the perimeter of the blob until it returns to the start point, while recording the bounding box. The perimeter length and area can also be calculated 'on the fly' as described below. My trace algorithm follows below. It is a refinement of a more primitive algorithm described and constructed previously by myself [1]. It traces the blob in one direction (eg down) while trying to change direction (eg right). A blob may be traced clockwise or anti-clockwise. I deemed that the direction would be anti-clockwise so the sequence of directions is: DOWN → RIGHT → UP → LEFT. My perimeter trace algorithm follows below. The key variable is the direction variable 'dir'. This contains the direction we are currently traveling around the blob. It may have values in the range [0..3] representing the four directions. The perimeter variable is used to calculate the perimeter by incrementing after every step. The perimeter value returned is four more than it should be and needs to be corrected. This is because the algorithm does not actually count the perimeter pixels but counts all the pixels surrounding the blob. For example, consider a blob containing just one pixel. It's perimeter length is four but the number of pixels surrounding it is eight. So, for this algorithm, four must be subtracted from the perimeter length before it is returned.

```
/* initially go in the DOWN direction but try to change to */
/* the RIGHT direction */
/* down:0, right: 1, up:2, left:3 */

dir = 0; /* ie DOWN */
perimeter = 0; /* set perimeter counter to zero */

do{
   check bounding box
   if the pixel in the next (dir+1 mod 4) direction is on{
      if the pixel in the current (dir) direction is on
         if the pixel in the previous (dir+3 mod 4) direction is on
            /* we're in a cul-de-sac so reverse direction */
            dir = (dir+2) mod 4
         else /* travel out around the obstruction */
            /* go back to the previous direction */
            dir = (dir+3) mod 4
   }
   else /* increment the direction */
```

```
        dir = (dir+1) mod 4
    go one step forward in the dir direction
    perimeter++
}while not back at the starting point
```

Calculating the area while perimeter tracing

The author investigated the possibility of calculating the area of the blob, that is the number of black pixels, while tracing the perimeter. A simple way of calculating this area was developed, for all characters that do not contain enclosed space as found in an 'A' or 'O' for example. Because of the existence of characters containing enclosed space, it proved easier to simply count the pixels row by row rather than calculate the enclosed space.

5.5 Splits and merges

There are two related problems that pose a challenge to character recognition. The first is where a character may be split up into two or more pieces and the second is where a character may become merged with a neighbouring character. Because of these problems, it is not merely sufficient to segment single blobs and pass them on to the classifier, one after the other. To have any chance of approaching a 100% recognition rate some attempt has to be made to decide if a blob is really two or more blobs stuck together or just a small fragment of a larger blob. These problems are non-trivial. I had much more trouble with splits than merges.

Splits and merges in characters can occur for several reasons. These include: bad scanning parameters, insufficient or too much ink on the printing blocks, badly finished or worn printing blocks, dirt on the scanner screen or page, bad thresholding in the scanner, or even ink fading. Some of these problems can be addressed such as choosing the best scanner contrast and brightness settings, cleaning the scanner screen and even choosing a good quality scanner. The other problems remain and must be resolved with software. Figure D.1 in Appendix D is a screen photograph showing three entries each indicated by a leading hyphen. The first hyphen is split in two parts.

5.5.1 Splits

My initial approach to resolving splits was to take a simple ad hoc approach. This solution proved unworkable but is worth commenting on as it demonstrated the complexity of the problem. Because the library pages are of standard format (see Section 1.3.1), the maximum character width and the average inter character gap are known. With this information, a primitive split detection system can be constructed. If it is known that the average gap between characters is, x pixels wide, it can be supposed that gaps smaller than x constitute a split in a character. This information alone is not enough. As the maximum character width is also known, it can be said that, any two blobs whose combined width is less than or equal to the maximum width *and* whose inter character gap is less than the average gap, constitute a character with a split. This was the basic split detection system used in the early stages of the project. Initially, the project was only concerned with segmenting and classifying the author names. Author names are all in capitals and this meant that the combined width of almost any two blobs would not exceed the maximum character width. This meant that with two combined blobs whose widths were less than the maximum width, it could be said that they constituted two fragments of a split character. The system was quite successful under these conditions.

As the system was developed, it became clear (as I suspected) that the ad hoc solution was not sufficient. With a mixture of fonts, the standard deviation of character widths was very high so that two lower case characters could be joined together, without exceeding the maximum character width, in the mistaken belief that they formed two fragments of an upper case character. The inter character gap also varied enough to make it unreliable. This method raised the question of whether it would ever be possible to find splits during the segmentation stage without resorting to character recognition. This was a fundamental question for me that I couldn't answer initially. If it could be done without character recognition, the system would probably be faster as recognition is very CPU intensive. Clearly however, using character recognition can give a good idea whether blobs are fragments or whole characters. Character recognition is used to see if blobs constitute fragments or whole characters by looking at the probability scores for a match. If the score is low, the blob is probably a fragment. If the score is high, the blob is likely to be a whole character though more testing is then needed.

Is character recognition necessary to resolve splits properly? To answer the question, it is useful to consider how the human vision system solves the problem.

The human brain processes massively in parallel and can afford to use as much high level information as necessary. Consider the characters 'c' 'l' and 'd'. Where a 'c' is closely followed by an 'l', how does the system decide that the 'c' and 'l' are not two fragments of a 'd'? Take a look at Figure 5.6. Is the bird in the picture a duck or a chicken? Where it can, the human brain uses context, spelling and even higher level information to resolve ambiguities. It is interesting to note that humans have a 4% error rate when reading without the aid of context [16]. Clearly the ad hoc method could never match the recognition method in terms of accuracy. Accuracy was all important. The speed problem could always be overcome with more powerful machines.

Using character recognition during the segmentation phase meant that segmentation and classification were no longer two distinct phases. It was important, for performance considerations, to avoid classifying the same blob twice. My first approach to resolving splits using recognition used an array as a circular buffer and is described below.

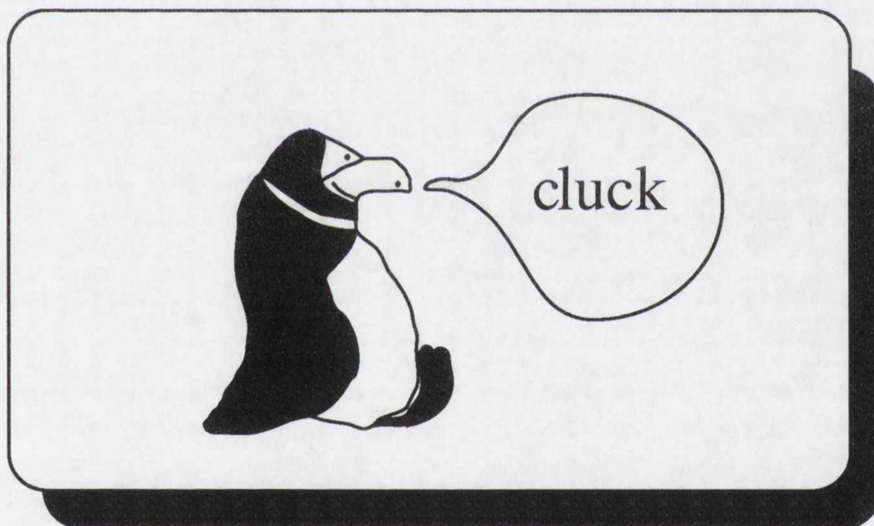


Figure 5.6: Duck or chicken?

Method 1

This method was the first implemented, after the ad hoc method, using character recognition. It was superseeded by Method 2, described in the next section.

When a blob is discovered, the four coordinates delimiting its bounding box, are placed in a circular buffer. The blob to the right of the current blob is also placed in the buffer if the following conditions are met:

1. The inter-blob gap is sufficiently small.
2. The overall width of all the blobs in the buffer does not exceed the maximum character size.

This process is continued until adding another blob would break one of the preceding conditions. Let us suppose that there are n blobs in the buffer. The classifier works backwards through the buffer, first matching n blobs, then $n-1$, $n-2$ etc. until one of the combined set of blobs returns a match score that is higher than an *a priori* threshold. See Figure 5.7. If no combination exceeds the threshold, then the very last blob is taken by default. When a good match is found, the x blobs that constituted the match are removed from the buffer and the process is started again with new blobs being added to the $n-x$ blobs remaining in the buffer if conditions 1 and 2 above are met.

There are several problems with this method. First, in taking the first match that reaches the threshold, a better match can be missed. Second, when no match reaches the threshold, the best match should be taken, not the last one. In trying to overcome these drawbacks, a new method was developed which is described below.

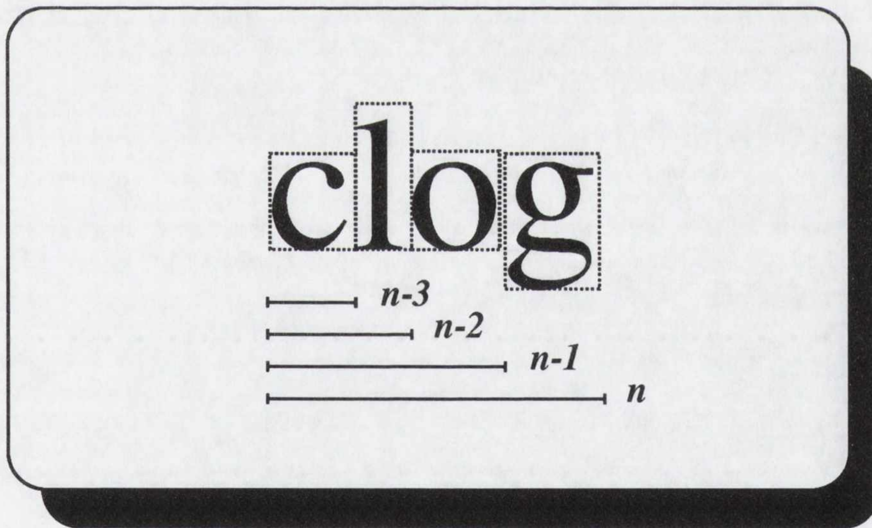


Figure 5.7: Finding splits with Method 1.

Method 2

This is the second method attempted for resolving split characters using character recognition.

Two side by side blobs are traced and pointers to them are placed in a buffer. Each blob is matched separately and their scores are recorded. The combined blob is then matched and its score is recorded. The scores for each individual blob are averaged. If the averaged score is greater than that of the combined blob, then it is presumed that there are two distinct characters present. If the averaged score is less than that of the combined blob, then it is presumed that the two blobs are one character split in two. The conditions regarding the inter character gap and maximum width described for Method 1 still hold. Consider two examples. In Figure 5.8, two blobs representing the characters 't' and 'o' are side by side. When the two blobs are matched individually, let us assume that the classifier matches the first blob as a 't' with 70% probability and the second blob as an 'o' with 80% probability. The average match then for the two blobs is 75%. Let us then assume that the combined 'to' blob is then matched and the classifier matches it as a 'W' with probability 5%. As the average probability (75%) is larger than the combined probability (5%) it is assumed that there is no split and hence two separate characters. Also in Figure 5.8, two blobs representing two fragments of the character 'b' are side by side. Again, when the two blobs are matched individually, let's assume that the first fragment is matched as an 'l' with probability 50% and the second fragment is matched as an 'o' with 30% probability. The average match for the two blobs is 40%. The combined blob is then matched as a 'b' with probability 95%. As the average probability (40%) is smaller than the combined probability (95%) it is assumed that there is only one character present, split into two fragments. The advantage of this method is that no threshold value is needed and, as all three possible combinations have been tried, we can be reasonably sure that the best match has been found. One disadvantage is that it is presumed there is no more than one split per character. I settled on this second method as a compromise.

The definitive solution to split detection

The ideal solution would be to put as many blobs as possible in a buffer and match every legal combination of blobs and take the highest score or combination of scores. Obviously, this would reduce performance drastically as the number of matches for each character would increase greatly. The author did some investigation to work out the cost. For n blobs in the buffer, there would be 2^{n-1} legal ways in which the blobs could be grouped (see Table 5.1). Thus for example, 3 blobs could be grouped 4 different ways. See Figure 5.9. Working out the number of matches per grouping is a little more difficult. When the groups are drawn out and counted, binomial

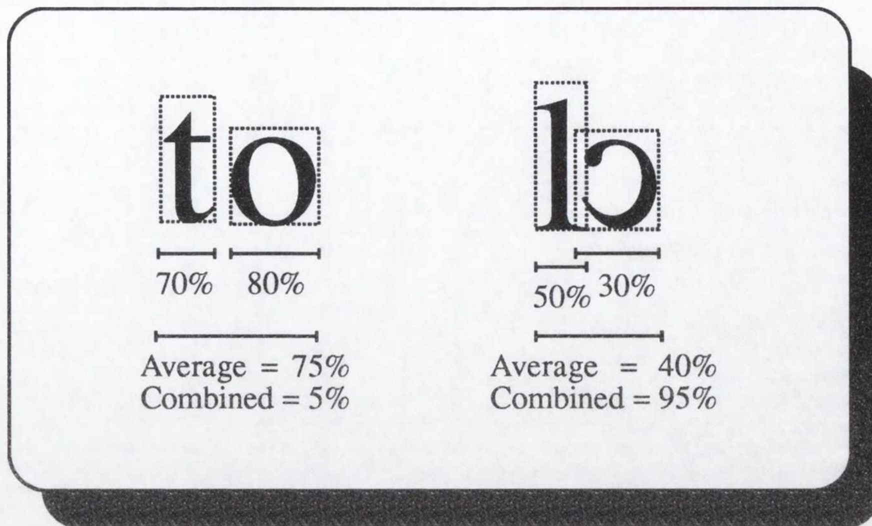


Figure 5.8: Finding splits with Method 2.

coefficients appear as in Pascal's triangle. Investigation shows the formula for the number of matches (μ) for n blobs is given by Equation 5.1:

$$\mu = \sum_{r=0}^{n-1} \binom{n-1}{r} .(r+1), \qquad n > 0 \tag{5.1}$$

Thus, 3 blobs, grouped 4 ways would require 8 matches or, 6 blobs, grouped 32 different ways would require 112 matches.. This would be very expensive but the number of blobs grouped together can always be minimised by the maximum width of a character (see condition 2 for Method 1). Thus, if characters were very badly fragmented, the system would be far too slow to be useful but if characters were only split into one or two blobs, the number of matches needed to resolve the splits could be kept at a reasonable level. In any case, if the text was that badly fragmented it is likely that recognition would be extremely poor anyway.

This method is mentioned by Banno *et.al.*[3, page 177]. Unfortunately the authors provide no figures for the success or cost of their system.

n blobs	1	2	3	4	5	6
no. of groupings(2^{n-1})	1	2	4	8	16	32
no. of matches(μ)	1	3	8	20	48	112

Table 5.1: No. of ways n blobs can be grouped and no. of matches per grouping.

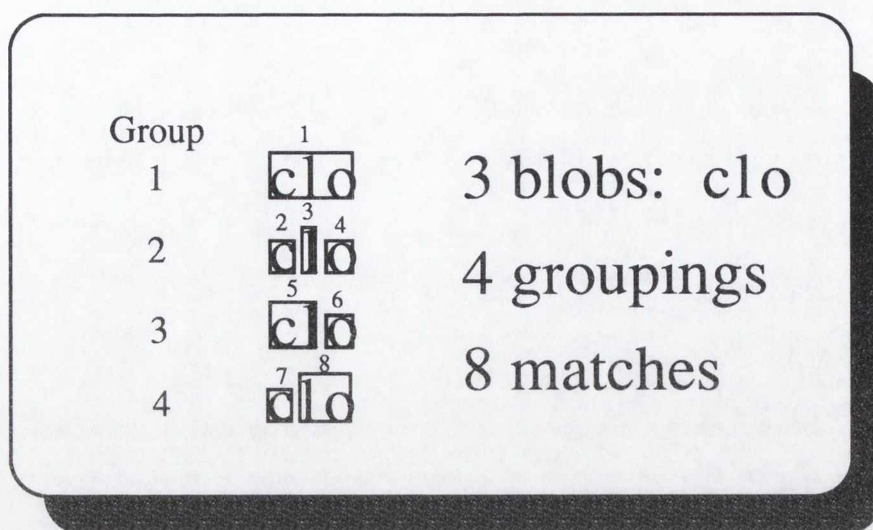


Figure 5.9: 3 blobs can be grouped 4 ways, needing 8 matches.

5.5.2 Merges

Merges occur when two separate characters become joined at a point to form a single blob. Reasons why merges can occur are described in Section 5.5. In this project, merges occurred very rarely and so no action was taken to resolve them. However, in the interests of striving to attain 100% recognition, I will describe a few merge resolving methodologies. As with splits, it must be realised that resolving some merges can only be done with a considerable amount of contextual information. Consider Figure 5.6 once again. It must also be remembered that some characters are naturally merged. For example, the ligatures 'Æ' and 'œ'. The best solution to natural ligatures, and the one implemented in this project, is to treat them as a single character and allow template names to be more than one character long. Thus, the name of the æ template would be 'ae'. This project originally ran on an IBM PC and so some well known ligatures were included in the character set. This meant that single character names could be used. When the project was ported to a transputer that did not have an extended character set, it became clear that template names of more than one character were the answer. I allowed templates have names up to five characters long. Another advantage of this approach was that merges that occurred more frequently than others (like 'ry' in this project) could be handled by treating them as a single character with a two-character name thus avoiding merge resolving problems for that particular character pair.

Signature Analysis for Merge Detection

Kahan *et.al.*[17] describe three stages in resolving merged characters. They identify what they call the *Recognition* stage where a blob is flagged as being merged, the *Segmentation* stage where the position of the merge is located and lastly, the *Reclassification* stage where the components of the blob are classified both individually and as a merged character. They use a very simplistic system for flagging a character as merged. If the probability score for the character, as returned by the classifier, is less than the *a priori* probability for that character, then the character is flagged as merged.

In the *Segmentation* stage, two different types of merges are identified: serif joins and double-o joins. Serif joins are those caused by the very top or very bottom of a character merging at extreme points. Double-o joins are caused by round bulbous characters touching at the extreme points of their arcs. The authors use the vertical signature or, vertical projection to find the position of the merge. A vertical signature is simply a function mapping a horizontal position to the number of pixels in the vertical column at that position. See Figure 5.10. (Figure D.2 in Appendix D is a screen photograph showing a window with a 'P' template. The template's horizontal signature is to its right while its vertical signature is below.) The merge point can be identified as a low point on the signature. All low points however are not necessarily merges. To verify if a low point on the signature corresponds to a merge, it is necessary to look for a sharp dip and a sharp rise in the signature, before and after the low point. The authors recommend looking one place before and one place after the low point. A breakpoint criterion function can be derived from the signature function V. The function recommended is

$$V(x-1) - 2 * V(x) + V(x+1) \tag{5.2}$$

which is quite straightforward. The author of this project did some experimentation with this function and found it to be very successful but it was not used in the project because merges were not a serious problem and the cost on recognition speed could not be justified.

Finally, in the *Reclassification* stage, the left and right fragments of the merged blob are classified individually. If the probability scores for the individual blobs are worse than for the merged blob then the merge is ignored. Note that this system could use recursion to find multiple merges.

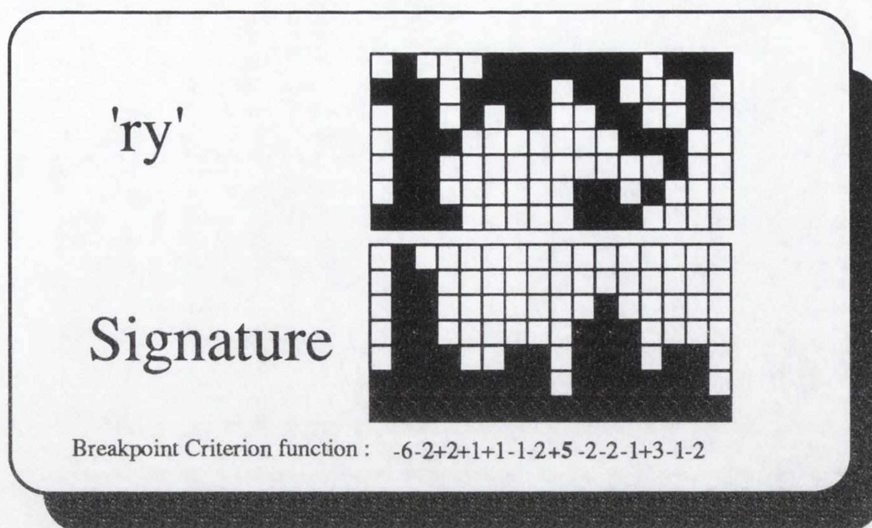


Figure 5.10: Finding a merge by signature analysis.

5.5.3 Thinning and expanding

Thinning[1] and expanding[2] are two operations used widely in the area of computer vision that may be used to resolve merges and splits respectively. Thinning is the process by which one or more pixels are removed from the boundary of a blob, at every point on the boundary. Expanding is similar except that pixels are added rather than removed. Pixels can be added or removed either by tracing the perimeter and removing or adding pixels during the trace or by placing a small filter window (eg 3×3) at every position on the character and changing the contents of the window depending on the operation required. There has been much research in this area and several different thinning filters have been devised.

Using thinning to resolve merges

Thinning can be used to resolve merges as follows. After a blob has been found, one pixel is removed from the perimeter of the blob, all the way around. The perimeter of the blob is then retraced from the original point. If the perimeter is different, we can presume that we have split the blob into sub blobs. If the individual blobs are then classified, we can get a good idea from their probability scores, whether their is a legitimate merge present.

[1] Also referred to as *skeletonising*.

[2] Also referred to as *region growing*.

Using expanding to resolve splits

Using expanding to resolve splits is a similar process to that of using thinning to resolve merges. When a blob has been found, one pixel is added to the perimeter of the blob, all the way around. The perimeter of the blob is then retraced starting from the same point as before. If the perimeter is different, it can be assumed that the blob has merged with a neighbouring blob. This combined blob can then be classified and its probability score examined to decide if a legitimate split is present.

In this project, splits were detected by the method described in Section 5.5.1 and no merge detection was necessary so thinning and expanding were not used.

Chapter 6

Using a State Machine to Delimit Fields

6.1 Overview of this Chapter

The aim of this chapter is to describe how fields were delimited within entries using a state machine. The structure of a field is described in Section 1.3.1. State machines in general are discussed and the state machine used in this project is described in detail. The states used are described in Section 6.3.1 and Figure 6.1 shows a comprehensive picture of the state machine. Table 6.2 shows all state transitions and associated trigger conditions. Finally, reduced template sets are introduced.

6.2 Delimiting Fields

As the ASCII output from this system was to be fed into a database and that database was in turn to be combined with the current database in use in the library, it was useful to be able to delimit fields within entrys. This meant that text passed on to the database would contain control characters showing the starting positions of the different fields present. Thus the following entry:

PALIN (William).—Cheshire farming: a report on
the agriculture of Cheshire.
Lond. 1845. 8°. A. r. 46. N°. 14.

would become:

≡APALIN
≡F(William).
≡T—Cheshire farming: a report on the agriculture of Cheshire.
≡L*Lond.*
≡D1845.
≡f8°.
≡SA. r. 46. N°. 14.

where the beginning of each field is indicated by the following tokens:

- '≡A' indicates the author surname
- '≡F' indicates the first name
- '≡T' indicates the text
- '≡L' indicates the publishing location
- '≡D' indicates the date of publishing
- '≡f' indicates the format of the volume
- '≡S' indicates the shelf number

To decide where one field ends and another field begins, the conditions that end and/or begin a field must be known. For example, the *text* field starts with an elongated hyphen '—'. The start/end conditions are not always so simple however and often depend on the preceding field. For example, the *text* field can be followed by the *location* field or the *shelf* field, or by no field. So the *text* field can end under different conditions. Thus, to delimit the fields, a list of trigger conditions for each different field transition must be known. One solution to these problems is to use a state machine. Delimiting the fields can be done either in a post-processing stage or at the recognition stage. However, if done during the recognition stage, considerable benefits in time and accuracy can be gained by using reduced template sets (see Section 6.3.3).

6.3 State machine

A state machine consists of a number of different states (including a start and end state) and a set of trigger conditions for each state transition. Thus each state may only be entered under certain conditions and may only be left under certain other conditions. This lends itself well to the problem of delimiting fields. Fields can be made correspond to states. Thus, for example, the *author* field corresponds to the *author* state. A transition can be made from the *start* state to the *author* state if the first letter encountered in an entry is a capital. If, on the other hand, the first letter encountered is an elongated hyphen '—', the transition is from the *start* state to the *text* state.

As each character in an entry is classified, it is passed to the state machine. The machine resets to state 0 (the start state) at the beginning of each entry. A transition from one state to another is triggered by special trigger characters associated with each state. Each state has a number of trigger conditions associated with it and a number of reduced template sets (see Section 6.3.3). Trigger conditions are listed in Table 6.2. Table 6.1 lists each of the possible states.

Num.	State	Description
0	START	Start state
1	SUR	Surname
2	FIRST	First name
3	ALT	Alternative
4	TEXT	Descriptive Text
5	QUAL	Qualification
6	LOC	Publication location
7	DATE	Date of publication
8	FORMAT	Format of book
9	SHELF	Shelf number
10	HEADER	Three letter headers
11	TRAINING	Template training
12	FIR_QUAL	Dummy state

Table 6.1: State names and descriptions.

State Transition	Trigger condition
START→TEXT	'–'
START→SUR	not '–'
SUR→TEXT	'–'
SUR→ALT	',' or '['
SUR→FIRST	'('
ALT→TEXT	'–'
ALT→FIRST	'('
FIRST→TEXT	'–'
FIRST→FIR_QUAL	')'
FIR_QUAL→TEXT	'–'
FIR_QUAL→QUAL	not '–'
QUAL→TEXT	'–'
TEXT→END	no more characters
TEXT→LOC	character is italic and starts a new line
TEXT→SHELF	new line and large space and not italics
TEXT→SHELF	no new line but large space
LOC→DATE	digit or '['
DATE→FORMAT	'.' or '['
FORMAT→SHELF	'.'
SHELF→LOC	character is italic and starts a new line
SHELF→SHELF	new line and large space before character
SHELF→FORMAT	new line and not italic and no space
SHELF→END	no more characters

Table 6.2: Trigger Conditions.

6.3.1 Description of states

State 0 is a dummy state, the START state. State 1 (SUR) is the author surname state. State 2 (FIRST) is the author first name state. State 3 (ALT) is a state for the alternative name for the author. State 4 (TEXT) is the state containing book title text.State 5 (QUAL) is the state for the author's qualifications or other descriptive text. State 6 (LOC) is the state in which the location of publication appears. State 7 (DATE) is the state in which the date of publication appears. State 8 (FORMAT) is the state in which the format of a book appears. This can be the size of the book and/or information on the number of volumes, edition or other information (see Section 6.3.2). State 9 (SHELF) is the state for the shelf marks. A detailed description of the sub-fields that can occur in the shelf field appears in Section 1.3.1. State 10 (HEADER) is a special state which is used purely for the two 3-letter headers that appear at the top of each catalogue page. State 11 (TRAINING) is a special state which is used exclusively in training mode (see Section 4.7). State 12 (FIR_QUAL) is a dummy state which is used between the FIRST and QUAL states. A dummy state was necessary here because the FIRST state is terminated with a closing bracket (')'). All other states are terminated by the character that starts the next state. When the FIRST state terminates with a closing bracket, the next state entered is the FIR_QUAL state. This is a transition state and another state may be immediately entered, depending on the first character after the FIRST state.

A more detailed description of the fields associated with each state appears in Section 1.3.1.

6.3.2 Description of trigger conditions

The TEXT state always begins with an elongated hyphen ('—')[1] so transitions from the START, SUR, ALT, FIRST and QUAL states to the TEXT state occur on an elongated hyphen. The START to SUR transition and the FIR_QUAL to QUAL transition occur when the character is *not* an elongated hyphen. The transition from SUR to ALT occurs when the character is either a comma (',') or an opening square bracket ('[') as these are the characters that can start the ALT field. The FIRST field always starts with an opening round bracket ('(') so this is the transition character from both the SUR state and the ALT state to the FIRST state. The FIRST state

[1]The character '—' is represented by an underscore character ('_') to distinguish it from a normal hyphen ('-').

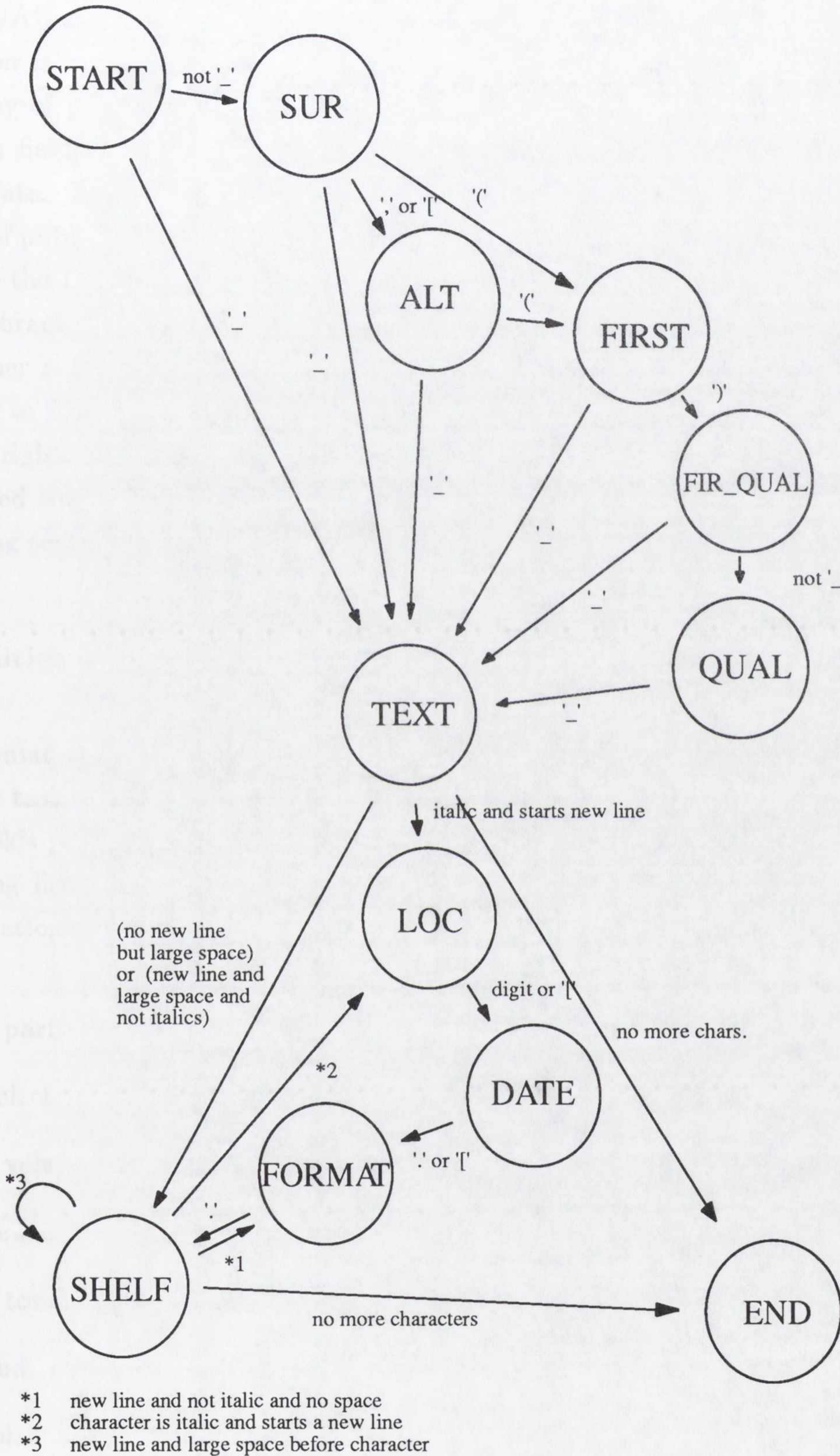


Figure 6.1: Complete state machine.

ends with a closing round bracket (')') so this is the transition character to the FIR_QUAL state. The condition for the transition from TEXT to LOC is that a character is in italics (set 4) and starts on a new line. The new line proviso is the only way of preventing italics within the text field being taken as the start of the location field. This is also the transition condition from the SHELF state to the LOC state. The SHELF to LOC transition is needed as some books have several places of publishing as well as different shelf marks. For the transition from the LOC state to the DATE state, the trigger character is either a digit (set 5) or an opening square bracket ('['). The trigger characters for the DATE to FORMAT transition are either a full stop ('.') or an opening left bracket ('['). The condition for the SHELF to SHELF transition is that the character starts a new line and is well over on the right. Detecting the boundaries between the SHELF and FORMAT fields presented some difficulties and the transition conditions used are described in the following section.

Difficulties with the format and shelf mark fields

The format and shelf fields vary enormously and detecting their boundaries was a difficult task that was not completed successfully. The format field usually contains the book's page size, of the form (8°). but other information can appear here. The following list shows the sort of thing that can occur in the format field. Several combinations of these types of sub-field can occur.

- 2 parties en 1 tom.
- vol. 1 [Family library]
- 2 vols.
- new ed.
- 2 tom.
- 2nd. ed.
- fol.

The variations made it impossible to settle on a trigger condition. The one settled on for the FORMAT to SHELF transition was a full stop character ('.') but this

is clearly insufficient. Detecting the transition from TEXT to SHELF, or DATE to FORMAT was also very difficult. The TEXT to SHELF transition settled on was the following condition: the character must (be an italic and start a new line and be well over on the right) *or* must (not start a new line and start well over on the right).

The successful determination of the fields may be possible using postprocessing as described in Chapter 7.

6.3.3 Reduced template sets

One of the advantages of using a state machine is that reduced template sets may be used. A reduced template set is simply a subset of all the templates stored. In certain states, certain characters or groups of characters can never occur (for example, only digits and certain punctuation marks can occur in the date field). In these states, it is not necessary to match candidate characters against all templates. Thus, the complete template set can be split up into subsets or reduced template sets and all or some of these sets can be combined together depending on the current state. This means that both performance and accuracy improve, sometimes dramatically. Performance improves because the number of template matches is greatly reduced. Accuracy improves because there are fewer templates and thus less chance of error. So, by using a state machine, performance and accuracy improve and individual fields have been delimited. Table 6.3 shows the template sets used in this project. Table 6.4 shows the template sets associated with each state.

Description of sets

Set 0 includes all punctuation marks and is used in every state except the SHELF and HEADER states[2]. Set 1 contains all large capital letters. Set 2 contains all lower case letters. Set 3 contains small capital letters. This font is used mainly in the shelf mark but also appears in the 'alt' and text fields. Set 4 contains italics. Italics are used mainly in the location field but can also appear in the text field. Set 5 contains the digits 0 to 9. Set 6 contains the header templates. These templates are used exclusively for the 3-letter headers that appear over both columns of the catalogue page and which are slightly larger than the other fonts on the page. Set 7

[2]The shelf set has its own punctuation templates.

is a set used exclusively in the SHELF state. The characters that appear in the shelf field are of the same fonts that appear elsewhere in the entry. However, because the shelf field gave particular problems (as described in Section 6.3.2) and because of its importance, it was felt that templates used in the shelf field should only be taken from shelf marks in the belief that perhaps the same printing blocks were used for all shelf marks. Thus the SHELF state has its own template set, the shelf set (set 7).

Set	Name
0	Punctuation
1	Capitals
2	Lower case
3	Small capitals
4	Italics
5	Digits
6	Header
7	Shelf

Table 6.3: Template sets.

State	Associated template sets							
	0	1	2	3	4	5	6	7
START	√	√						
SUR	√	√						
FIRST	√	√	√					
ALT	√	√	√	√				
TEXT	√	√	√	√	√	√		
QUAL	√	√	√					
LOC	√				√	√		
DATE	√					√		
FORMAT	√		√			√		
SHELF					√			√
HEADER							√	
TRAINING	√	√	√	√	√	√	√	√
FIR_QUAL	√	√	√					

Table 6.4: Sets associated with states.

6.4 Errors introduced by the state machine

Overall, the state machine improves accuracy and recognition speed. However, it does introduce some errors of its own. Problems can arise when a trigger character is misclassified. Because reduced template sets are associated with each state, missing a trigger character can mean that the following field may be classified with the wrong template sets with disastrous consequences.

This is a very difficult error to deal with as any character can be misclassified. One way of reducing the chances of the error would be to weight the template set with several instances of the character. However, there are other non character based trigger conditions than can be misidentified.

Chapter 7

Postprocessing and Context

7.1 Overview of this Chapter

This chapter describes how text output from the classification system may be corrected and improved both after classification in a postprocessing phase and during classification with a feedback mechanism. Several types of context are described including layout, spelling and grammer with reference to several documented systems and the context used in this project is discused.

7.2 Introduction to Postprocessing and Context

Postprocessing is useful in any OCR system, to correct characters that have been misclassified by the recognition system. In all but the most rigidly structured and scientific documents, the text will contain some context. Context is useful to any OCR system including humans and it has been said many times in OCR literature that human readers often misidentify single letters taken out of context[14, 25]. It is apparently possible to rewrite English text using only a quarter of the characters required[6, 24] though according to Chapanis[10] if more than 25% of characters are randomly deleted, humans cannot restore the text by context.

There are several levels of context including, layout, spelling, grammar and the actual idea conveyed by the text. Spelling, grammar and sometimes alphabetical order may be used to correct errors and with so much research being done in the area

of artificial intelligence it may not be long before the idea inherent in the sentences may also be available. Other simple rules can be utilised such as detecting a capital letter between two lower case ones.

Errors can be corrected in two different ways. First, contextual information may be used during classification to further filter out the possible candidate characters. Second, the errors can be corrected when all classification has ceased, in a postprocessing stage. Here, characters that the classifier knows are ambiguous are flagged for the postprocessor but the postprocessor can also look for characters that have simply been misclassified, unbeknownst to the classifier.

7.3 Context in this Project

In this project, authors' names and works are listed alphabetically in the catalogue. This alphabetical information allows some assumptions to be made during or after classification. For example, if the previous author's name was 'NOLAN' and the classifier returns the current author name as 'NGLAN', it can be reasonably assumed, because of alphabetical order, that the 'O' was misclassified as a 'G'. This information could also have been given by a dictionary of proper names. The alphabetical information can be applied at the postprocessing stage or during classification. If used during classification, character templates which cannot occur for alphabetical reasons may be filtered out of the matching process.

As described in Section 1.3.1, a three-letter leader appears at the top of each of the two columns on every catalogue page. These two sets of three letters are the first three letters of the first and last surnames on the page respectively. If these two headers are compared to see which initial letters are common to each, then it can be assumed that every surname on the page starts which those common letters. For example, if the two headers are 'NIL' and 'NIM' respectively, then the initial letters common to both are 'NI' and therefore every surname on the page begins with 'NI'. Once again, this information may be used during the classification stage or after, as part of a larger postprocessing stage. This method can be taken a stage further. If the very first and very last surnames on the page are examined, and the common initial characters are extracted in the same way, then, again, it can be presumed that each entry begins with those common characters. For example, if the first name is 'PALLADIUS' and the last name is 'PALLIOT' then the four characters 'PALL' are common to all entries on that page. If the first surname is the same as the last

surname, then, that name can be filled in for all authors and no processing need be done on any surname. This method was used in this project.

When a surname has been found, characters that are 'common' to the start of every surname on the page are written over it. This 'common' string of characters is deduced from the first and last entry as explained previously. The surname is then compared with the previously classified surname. If a character of the current surname is alphabetically less than a character of the previous surname then it is assumed that this character has been misclassified and it is replaced with the corresponding character of the previous surname. The comparison is done from the end of the 'common' characters to the end of the surname or until a character of the current surname is alphabetically greater than the corresponding character of the previous surname. The algorithm used (in C) follows:

```
/* first write in common characters */
for(count = 0; count < strlen(common); count++)
   current[count] = common[count];
/* now compare current surname with previous surname */
minimum = min(strlen(prev_surname),strlen(current));
for(count2 = count; count2 < minimum; count2++){
   if(current[count2] > prev_surname[count2])
      break;
   if(current[count2] < prev_surname[count2])
      current[count2] = prev_surname[count2];
}
```

7.4 Layout

Kahan *et.al.*[17] use layout context in their now well known paper. The authors argue that layout context is usually language independent and hence use it as a first stage of postprocessing, before spelling correction. For a small segment of text, four guiding lines are determined. Two of these guide lines delimit the top and bottom of a 'normal' character, the other two delimit the tops and bottoms of characters with ascenders or descenders or capital letters. Each character has an expected position with respect to these lines and the actual position calculated from the bounding box. If these two positions are different then the character has probably been misclassified.

7.5 Language

7.5.1 Spelling

Some character recognition errors can be found by looking for misspelled words. This involves the use of a dictionary. Spelling correction is most effective when the document is written in one language and scientific and proper words are seldom used. Alas, the opposite is the case with the library catalogue. Several languages are used and are mixed liberally and the pages are littered with foreign language names so a spelling corrector was not used although it would have been of some use. It might have been possible to write a *language recogniser* which would either look up words in several dictionaries or would detect the language being used by perhaps letter frequency and then apply the correct dictionary to correct the spelling. Because of time considerations, this was not explored.

Kahan *et.al.*[17] describe a system for detecting and correcting spelling errors. The UNIX[1] 'spell' program is used. When a word is rejected, a list of variant spellings is generated. Each variant has an associated cost and variants are tested in increasing order of cost. Checking stops with a correctly spelled variant or when the cost is too high. In the training section, for each pair of characters, a probability of misclassification is generated. Thus for characters of classes X and Y, the probability that a character of class X was misclassified as a character of class Y is P_{XY}. The cost of a spelling variant C is given by:

$$C = -\sum_{x=1}^{i=n} P_{c_0^i c^i} \tag{7.1}$$

where the word emitted by the classifier is given by:

$$W_0 = c_0^1 c_0^2 \cdots c_0^n \tag{7.2}$$

and the spelling variant is given by:

$$W = c^1 c^2 \cdots c^n \tag{7.3}$$

Variants are generated in order of increasing cost. When a word has been identified as misspelled, n variants are generated by replacing a character of the word by the character with which it is most often confused. Variants are examined by

[1]UNIX is a trademark of AT&T.

order of least cost. When a variant is rejected, it is replaced by variants obtained by altering other characters in the rejected variant. Attempts are made to avoid generating duplicate variants. The authors point out that this correction scheme is asymptotically exponential to both the total number of characters and the number of misclassified characters in the word. Because few character sequences are rejected and most rejected sequences have only one error, the authors reckon the spelling correction time can be kept to a small percentage of the total recognition time.

7.5.2 Digrams & trigrams

A digram is a sequence of two characters. A trigram is a sequence of three characters. Digrams and trigrams were used as early as the nineteen fifties to improve recognition of text[13, 14, 25]. There are 26 characters in the English language therefore there are 26^3 or 17576 possible trigrams. The frequency of occurrence of trigrams in the any language varies greatly. For example, in the English language, the characters 'zzz' never occur (except perhaps in proper names) whereas the characters 'the' are very common. This information can be used in the postprocessing phase to correct errors. According to Harmon[14, page 1172] the procedure is based on two facts:

1. "in large samples of common English publication text, 42 percent of all possible digram combinations never occur"
2. "random substitution of one letter in a word by another letter which is selected with equal likelihood from the other 25 of the alphabet will produce at least one new digram which has zero probability 70 percent of the time in the sense of 1"

In Harmon's system errors are corrected by substituting a letter, which will construct a legal n-gram, for the errant one. If possible, the letter selected will be one that is often confused with the errant one by the system. Harmon claimed that 40% of all single-letter errors in ordinary English text are corrected in this way. Mori and Masuda[20] cite Hanson *et.al.*[12] as reducing substantial errors from 45% to 1-2% using trigrams.

The author experimented with a trigram program[2] but it was not used in the project due to the numerous languages used. Trigram frequencies are generated by

[2]Thanks to James Mahon for the program.

constructing a $26 \times 26 \times 26$ matrix where each element of the matrix represents one trigram. The matrix is then filled by feeding as much text as possible (similar to that which will undergo OCR) into a program that extracts the trigrams and fills the matrix. Alternatively, the trigrams can be generated as they are used. The trigrams can then be used to correct errors.

Chapter 8

Results and Performance

8.1 Overview of this Chapter

This chapter details the success rate, recognition rate and recognition speed of the software. The program was run on several pages of the catalogue and the errors are counted, classified and discussed. Two tests were carried out on each page, the second after some bugs had been ironed out (see Section 8.3). This led to a higher success rate and slower recognition time for the second test. There is a brief discussion of time stamping which is used to discover where most time is being spent in the code.

8.2 Recognition time and speed

Table 8.1 shows the compression time statistics for several catalogue pages. The machine used was a single InMos transputer. The page numbers have two parts, a single character followed by a 2 or 3 digit number. For example, 'n80' is page 80 from the 'n' section. Decompression time was about 30 seconds. This contributed to between 4 and 5% of the overall recognition time. The scanning time was of the same order so that scanning and compression/expansion contributed to about 10% of the overall recognition time. Delays due to page misalignment and page realignment were not taken into account (see Section 4.6.1).

Table 8.2 shows the recognition times and speed for the same set of pages. Recognition time was in the order of 13 minutes and with about 4800 characters per page,

this gave a recognition speed of about 6 characters per second. This compares favourably with those systems discussed in Chapter 2.

Page	Compressed size (in bytes)	Decompression time (in seconds)
l165	348839	32.8
l167	328315	32.2
n45	370272	37.1
n77	371283	35.0
n78	368146	31.5
n79	368601	34.0
n80	379218	34.5
p207	355387	35.7
p208	348752	34.9

Table 8.1: Compression Statistics.

Page	Recognition time (in minutes)		Chars./sec.	
	test 1	test 2	test 1	test 2
l165	12.98	13.33	6.16	5.99
l167	12.02	12.42	6.30	6.09
n45	12.98	13.38	6.24	6.05
n77	13.98	14.23	5.88	5.76
n78	14.39	14.70	5.36	5.49
n79	13.37	13.70	6.24	6.08
n80	14.02	14.28	6.08	5.97
p207	12.32	12.64	6.34	6.17
p208	12.67	13.08	6.24	6.04

Table 8.2: Recognition time and speed.

8.2.1 Time Stamping

Time stamping is a very valuable way of determining where bottlenecks may occur in software. It's a simple procedure that involves marking or *stamping* the current time at important points in the program and writing these times to a time stamp array. When the program terminates, the array can be analysed and intermediate times calculated thereby showing the percentage time spent in each function.

The following C code is the timestamp structure used in this project:

```
#define MAX_STAMPS 100

typedef struct{
   char comment[NAMESIZE];
   int time;
   int diff;
}t_entry;

t_entry tstamps[MAX_STAMPS];
```

This allows for comments to be stored with the times to mark where the time stamp occurred. The tstamps array is written to by a *timestamp* function. A typical call to timestamp would be:

```
timestamp("At start of guessletter",timer_now());
```

The tstamps array can be displayed with a *show_times* function that shows the time of each stamping and the time difference between successive stamps. The call to *timer_now* is a transputer dependent call that returns the elapsed time from the start of the program in transputer ticks. This is easily converted to seconds in the show_times function.

8.3 Success rate

Tests were carried out on full catalogue pages. These pages contain about 4800 characters on average, in a mixture of several fonts including Hebrew, Arabic and Greek (see the screen photograph in Figure D.4 in Appendix D). Characters not from the Roman alphabet were ignored in success rate statistics. In the first test, 314 templates were used, split up into the templates sets in Table 6.3. The test consisted of full classification of several catalogue pages. The results of the first test are displayed in Table 8.3. The errors shown in Table 8.3 are described in Section 8.3.1. Some of these errors were caused by bugs that were corrected and a second test was carried out. Note that no changes were made to the template set that would aid the results of the second test, excepting that two character classes that were not represented in the template set were added to it so that 316 templates were used in

the second test. The results of the second test are shown in Table 8.4. The success rate is calculated by counting the number of errors on the page and working out the percentage. Errors must be counted manually as some are detectable and some are not (see below). An attempt was made to write code that would automatically count all errors and this is described in Section 8.3.2.

Page	No. of chars.	Total errors	Merge errors	Score errors	Other errors	Success rate
l165	4794	211	14	11	186	95.59%
l167	4546	187	14	1	172	95.89%
n45	4860	251	21	11	219	94.84%
n77	4931	103	15	5	83	97.91%
n78	4626	188	8	12	168	95.94%
n79	5007	191	11	6	174	96.19%
n80	5115	142	10	3	129	97.22%
p207	4688	221	28	5	188	95.29%
p208	4745	250	14	8	228	94.73%

Table 8.3: Errors and success rate of the first test.

Page	No. of chars.	Total errors	Merge errors	Score errors	Other errors	Success rate
l165	4792	152	11	12	129	96.83%
l167	4541	142	8	2	132	96.88%
n45	4853	193	18	11	164	96.02%
n77	4920	91	13	8	70	98.15%
n78	4842	176	15	26	135	96.37%
n79	4994	159	11	16	132	96.82%
n80	5111	121	11	3	107	97.63%
p207	4680	190	18	9	163	95.94%
p208	4738	193	7	9	177	95.93%

Table 8.4: Errors and success rate of the second test.

8.3.1 Errors

Characters can be misclassified for several reasons. Sometimes the classifier realises that it cannot classify a character and an error character is written to the output file. These *detectable* errors fall into two different categories:

Detectable errors

- filter errors
- score errors

Filter[1] errors occur where **all** templates are filtered out. This can happen when a blob or groups of blobs are badly merged or split[2], the resulting blob bearing no resemblance to any template. In this project, filter errors were marked by writing a solid block character in place of the correct character. Score errors occur when the best score for an input character matched against a set of templates falls below a threshold score. In this project, score errors were marked by writing an upside-down question mark '¿' in place of the correct character.

Non-detectable errors

Most of the errors made by the classifier are unknown to it. These errors come under the 'Other errors' column in Tables 8.3 and 8.4. On average, only about 45% of these errors are mismatches. That is, only 45% of errors occur because the template matcher simply made a bad match. The rest of the errors are made up mainly of segmentation and state machine errors. The first (Table 8.3) test showed that 30% of the other errors were caused by a bug in the code which searches for multi-blob characters (see Section 5.3.1), thus causing problems with characters such as 'i', ':' and ';'. This bug was fixed before a second test was undertaken. 'Other errors' were split up as follows:

- mismatches (45%)
- multi-blob characters (30%)
- merged characters (10%)
- line segmentation (5%)
- no template (4%)
- incorrect position (3%)

[1]For a description of filtering see Section 4.4.

[2]For a description of splits and merges see Section 5.5.

- missed trigger characters (2%)
- split characters (1%)

10% of errors were caused by merged characters. Line segmentation errors occur when characters on the line above or line below are taken to be on the current line. 4% of errors occurred because no template was stored for the class of character encountered. This can happen because characters such as 'ü' and 'ñ' may not have been previously encountered. 3% of errors occurred because characters have different names depending on their position on a line. For example, the comma (,) and the apostrophe (') are the same character but in a different position. Missing a trigger condition in the state machine can mean that the wrong templates sets can be used for the following fields. Sometimes this can mean that large sections of text are misclassified. Finally, 1% of characters were misclassified because they were badly split. Note that this number is low, owing to the success of the split correction system (see Section 5.5.1).

8.3.2 Counting Errors

Working out the success rate means counting the errors. As there are almost 5000 characters per page and a 5% error rate, this means searching for about 250 errors amongst the 5000 characters. This is time consuming and tedious. The author experimented with automated character counting[3]. This process involves first constructing a file for each page that is a perfect ASCII representation of the page with no errors. This *golden* page is then compared with the file containing the errors. Success depends on finding matching points in the two files. If there are too many errors or there are several errors grouped together, this can be a problem. Because of time constraints, the author did not get this to work properly but it can give a quick representation of how fast the success rate is improving.

[3]Thanks to James Mahon for help with this.

Chapter 9

Conclusion and Future Work

9.1 Overview of this chapter

Results are drawn from several chapters and interpreted. The factors affecting the speed and success rate are discussed as well the merits of template matching. In the section on future work, improvements for both hardware and software are discussed.

9.2 Conclusion

Template matching has proved to be a very successful means of character classification. From Chapter 8 it can be seen that success rates of about 96% were achieved for this project. This was under operating conditions using almost 45000 input characters and over 300 templates for several fonts. This compares extremely well with more complex classification systems described earlier in this work and with off-the-shelf software whose claims of 99+% success rates are only seen in practice under perfect conditions. A recognition speed of about 6 characters per second was achieved with a single transputer. This rate compares well with the systems described in Chapter 2 but is much slower than some of the high cost industrial text reading machines. The speed must be seen in the context of the hardware used. On a state-of-the-art work station, the speed could be perhaps doubled or tripled. As stated earlier, template matching lends itself well to parallelisation. Using several transputers to take advantage of this is discussed in the section on Future Work.

9.2.1 Making template matching a success

In this project, the template matching process was enhanced using filters, reduced template sets and a good scoring algorithm. Other enhancements such as multiple matching were also discussed. It is this enhancement of template matching which makes it a success.

Filters

Section 4.4.1 analyses the success of filters. Tests showed that for a full set of templates, an average of 87% could be filtered out. For a reduced set, that is a set containing templates of the same font or group, on average 70% could be filtered out. The figure for reduced templates is smaller as templates within reduced sets tend to resemble each other more than they resemble templates from other sets. Filtering templates has a direct beneficial effect on the processing speed and an indirect beneficial effect on the success rate. The effect on the success rate is indirect as it could be argued that none of the filtered-out templates would have been chosen as the match in any case.

Reduced template sets

Reduced template sets were introduced in Section 6.3.3. A reduced template set is merely a subset of all the templates used. Sets are used to gather like templates together. In general, these sets can then be used when a section of text can be predicted to contains only those characters which appear in the set or sets used. Specifically, in this project, a state machine was used to delimit fields within entries and reduced templates sets were then used in classifying the fields as it was known *a priori* what characters could legally appear in each field. As with using filters, reduced template sets reduce the number of candidate templates before classifying an input character. If for example, three template sets are used with 40 templates each out of a total of 300 templates, then 60% of the templates have been rejected. If, as in this project, reduced template sets are combined with filtering, there can be further reduction. If we take the figure of 70% filtering from the previous section, it can be seen that up to 70% of the remaining 120 templates can be filtered, leaving only 36 or just over 10% of the original 300 templates.

Splits

Splits and split resolving are discussed in detail in Section 5.5. Resolving split characters reduces the recognition speed of the system dramatically but to ignore them would be to admit that a 100% success rate is not attainable. Split and merge resolution has been the subject of much research and the author of this project spent much time on the problem. There is a definitive solution and this is described in Section 5.5.1 but this is an exhaustive solution and can take an enormous amount of time. The ideal is to find a practical solution that can be executed in a reasonable time. If the text is too badly fragmented, the classification system will most likely not work anyway. In Section 5.5.1 the author explains a few of the theories and methods used in this project. The method settled on effectively increases the processing time by about 50% as all side-by-side lower case characters are classified to check for splits. This must be taken into consideration when considering the recognition speed.

9.3 Future Work

9.3.1 Parallelism

As pointed out earlier, template matching lends itself well to parallelisation. As the software was developed on a single transputer, parallelisation is most definitely attainable. It could be approached in two ways:

1. split each page into sections
2. split each character into sections for template classification (see Figure 9.1)

Splitting each page into sections is probably the most straightforward means to parallelisation. The page could be split manually, before the software is run or it could be split by the software. Problems arise however if a line of text of broken in the horizontal plane across a split line. Some form of preprocessing would be needed to ensure this didn't happen. In any case, this method of parallelisation is not very pretty and is really a very crude way of using the transputers. The second method, where characters are split prior to classification, is much more attractive.

A root transputer would be in charge of the splitting operation. All segmentation and management operations would be handled by the root transputer. The input character could be easily split on byte boundaries. The x transputers being used would then process the same probability scoring algorithm concurrently, on their own piece of the character, with the sub-scores for each piece being returned to the root transputer. If the pieces of the character are not too small, it can be presumed that the recognition speed will increase in proportion to the number of transputers used. At some high number of transputers, X, it can be presumed that the pieces would be so small that the overhead in transputer communication would inhibit the recognition speed from increasing. This is a very practical example of future work that could be undertaken. It's not very labour intensive and would make better use of transputer technology.

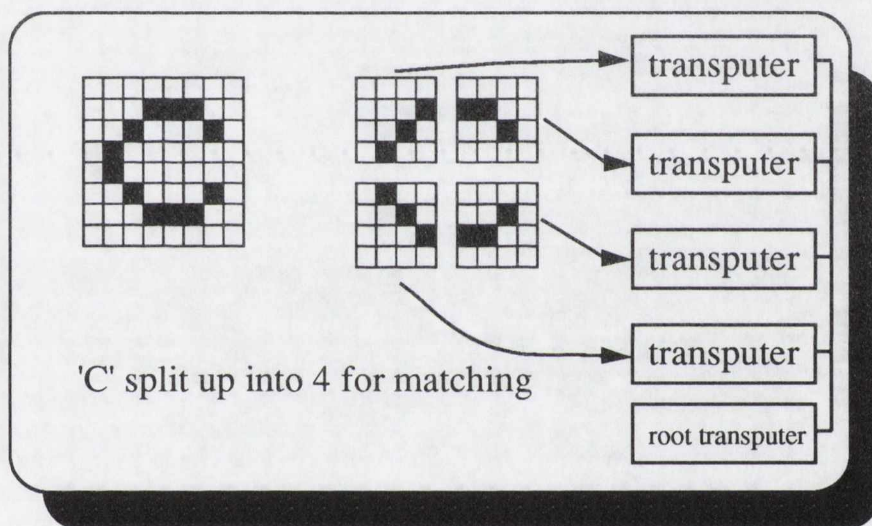


Figure 9.1: Parallel matching

9.3.2 Merged characters

Merged characters are discussed in Section 5.5.2. Merged characters are two adjacent characters that merge into one at one or more points. The merge can be caused at print time or scan time, or by dirt. In this project, no merge detection code was included although techniques are well discussed. The author would recommend that, for future work, the signature analysis method be implemented, using a breakpoint criterion function. This will of course lead to a drop in recognition speed.

9.3.3 Integration with the library database

It is planned that the output of this project will eventually be combined with the current library Dynix© database. At present, the output is embedded with symbols which delimit fields within entries (see Section 6.2). Work is presently being undertaken to analyse this embedded output with a free-text analyser. Pages will need to be manually corrected first.

9.3.4 Context and spelling

Context and spelling can be used to great effect to improve the output of an OCR system. These topics are discussed fully in Chapter 7. Within this project, advantage was taken of the inherent alphabetic order of the entries. This can be used to reduce the number of candidate templates prior to classification or can be used in a post-classification stage to correct errors. Through the use of a dictionary, spelling errors can be detected. This was not implemented in the project but is recommended for future work. It should be noted however that the use of several languages and other alphabets in the catalogue means that spell checking is a non-trivial task. It would be an interesting project to attempt to detect the language being used from the local text and to then invoke the correct dictionary for spell checking. The use of digrams and trigrams was also discussed. These could be very useful for error checking, but again, the use of different languages must be taken into consideration.

9.3.5 Averaging templates

There are two distinct methods of averaging templates that may improve the success rate. In the first method, templates for each character class are generated from several instantiations of the class. In this way, the template stored is more generally representative of the class of character it is representing. In the second method, again, several instantiations of a given class of character are used to represent that class. In this case however, the instantiations are not mathematically averaged but are stored together. Each template is then made out of several layers. During classification, an input character is scored against each layer in the template and the scores are added. This effectively weights some pixels above others. A more *average* template is stored and the success rate is bound to improve. It is clear that one of

the drawbacks of the second method is the huge increase in storage space needed. Both methods have the drawback of a much more complicated training phase.

9.3.6 Graphics user interface

No user friendly interface was developed for the project. It was felt unimportant for development and was left as future work. If the software is to be used by non-technical people, a simple user-friendly interface should be developed.

Appendix A

Rules for composing the printed catalogue

The following rules were those devised to guide the editors of the printed catalogue. They were taken from Dr. Thomas Fisher's notebook[1], c. 1860. The rules in Fisher's original notebook were heavily amended in pencil. The rules are listed verbatim, including the original errors. The rules were described as *Rules to Be Observed in Composing the Catalogue of Printed Books in the Library of Trinity College Dublin.* Notes have been added in square brackets.

1. Titles to be arranged alphabetically under the surname of the author when ascertainable. In the alphabetical arrangement initial prepositions, letters or articles to be regarded or not as part of the name as ordinary usage may direct. When this is doubtful, the name is to be used primarily *exclusive* of the prefixes and a cross-reference to be made from the same name with the prefixes added.

2. If more than one name occurs in the title as the author, the first mentioned to be taken as the heading of the primary entry.

3. Sovereigns or Princes of sovereign houses to be entered under their Christian or first name.

4. Works of Jewish Rabbis and of oriental writers to be entered under their first name.

[1]See Section 1.3.2.

5. Works of friars to be entered under the Christian name also persons canonized. Patronymics to be used as surnames. [This rule crossed out in the manuscript and left out in later copies.]

6. The Respondent in a Thesis to be considered its author except when it unequivocally appears to be the work of the Præses.

7. Any document purporting to be issued by a corporate body (except in the case of Learned Societies or scientific bodies, for which a specific rule is made afterwards) to be entered in distinct alphabetical series under the name of the place from which they derive their denomination, under the name of the place whence their acts are issued.

8. When the name of an author appears in different languages, the entry to be made in that in which appears in his several works. If they vary, Latin to have precedence, then English, French and German. [Crossed out in pencil and left out of later copies.]

9. Surnames of noblemen to be used in all cases as the heading with a cross-reference from the Title to the name. The same rule to be followed with respect to bishops if the surnames are ascertainable; if not then the name of the see to be taken. No cross-reference necessary in the case of bishops.

10. Initials and pseudonyms (such as "Philoveritas" "Irenæus," etc.) to be disregarded except in those cases where they have acquired such a degree of notoriety or importance as to entitle them to be taken as substitutes for the real names of anonymous authors.

11. When the author's name does not appear in the work, it is to be treated as anonymous, and placed under some specific heading according to the rules hereafter mentioned. If the name of the author can be ascertained then the title in an abridged form is to be placed under the anonymous heading and a reference to be made to this author's name under which the full title is to appear, with the term (Anon.) appended.
(If the author gives his initials only, the book to be entered under the initial of his last name—as a book by J. F. is to be entered under F(J) [later addition].)

12. If an anonymous publication should be a reply to or a defense of or consist of remarks upon the work of a known writer; the name of that writer may be taken as the heading of the anonymous work. (But all such anonymous works are to be enumerated after the list of the author's own works and separated

by a line, [or placed in the lists distinguished by Italics][crossed out in pencil] the work to which they refer but enclosed in brackets [ ]. [All of this left out in later copies.])

13. Next to the names of authors, the names of other persons forming the subject of anonymous publications (when the authors [*sic*] names cannot be ascertained) to be selected as the subject of the heading; next to them to rank the names of places.

14. In most other cases (of anonymous books) that noun substantive in the title which will but express the subject of the work will form the most suitable heading. If no one of these is sufficiently characteristic, a compound heading (ex. gr. "Future State," "Poor Laws," "Roman Catholic Emancipation Bill") may be formed and in some few cases a title may be selected altogether different from any words to be found in the title page (ex. gr. "Bank of England" for the "History of the Old Lady of Threadneedle St")—but such exceptions to the general rule to be avoided as much as possible.

15. In any series of printed works, which embrace the collected productions of various writers upon particular subjects the work is to be entered under the name of the editor. (But the particular authors are also to be entered, with reference to the collection [inserted].)

16. Works of several authors published together but under a collective title, to be catalogued under the name of the first author, although an editor's name may appear in the work.

17. If the editor's name do [*sic*] not appear, the whole collection is to be entered under the collective title as is the case of anonymous works.

18. Translations to be entered primarily under the name of the original author, secondarily under the name of the translator. The same rule to be observed with respect to the work of commentators, if accompanied with the complete text.

19. A cross-reference to be made from the name of any person who is the subject of a biography or narrative to its authors or in general, from any name which may be reasonably conceived to have equal claim to that selected for the principal entry.

20. The list of works under each heading to be arranged (first in order of the languages Oriental, Greek, Latin, English, French, German next [crossed out

in the manuscript and not used in later copies]) in chronological order only that subsequent editions of any work are to be placed immediately after the original edition. Volumes without date or the date of which cannot be supplied, to be entered last.

21. All Acts, Annuals, Memoirs, Transactions, Journals, &c of Academies, Universities or other learned Societies to be catalogued under 3 separate heads. 1. Memoirs, Acts or Transactions &c. as the case may be. 2. Academy, Society or Institute &c. & 3. The place from which such Transactions, &c. are issued. *Academy, Society* &c. to be made primary [order changed in pencil: 3 becomes 1, 1 becomes 2, 2 becomes 3].

    Where pseudonymous or fictitious names are used, the book to be entered under the real name & a reference to it given under the Pseudonym, except when this letter [*sic*] is better known than the real name—as Melanchton, whose real name Schwartzerde is known to few, Mathias Flaccius Illyricus &c.—but in these cases there may be a reference from the Pseudonym to the real name. In cases where the real name cannot be ascertained the book to be entered under the Pseudonym.

    Books published under such names as A patriot, a clergyman, a member of Parliament to be treated as purely anonymous.

22. Anonymous works relating to the Church Catholic or to the doctrines, principles or government of *churches in general* to be entered under the head *Ecclesia* if in Latin, or any foreign language, or Church if in English. Those which relate to separate national churches—as of Rome, England &c are to be entered under the name of the respective nation in which the Church is situated.

23. Concordances, Cyclopædias, Dictionaries and Lexicons to be entered primarily under the head of the author and secondarily under the respective words, Concordance &c. Dictionaries or Cyclopædias published under the sanction of an Academy or Society to be entered secondarily under the head *Academy, Society,* &c.

Appendix B

Equipment and software used

B.1 Hardware

- IBM PC AT
- Inmos T800 Transputer
- Microtek MS-300C flatbed scanner

B.2 Software

- Borland Turbo C
- Transputer Parallel C
- Microtek Image Scanner function-call package

Appendix C

Run-length encoding

Files used in this project were compressed by run-length encoding. This is a very simple means of compression, and expansion is relatively fast. Because of its simplicity, compression rates are not very high. Catalogue pages (A3 size) were scanned at 300 pixels per inch. After some of the surrounding white space had been removed, the image dimensions were 2544×3900 pixels so each page had 1,240,200 bytes or just under 1.2 megabytes. Run-length encoded files were between about 300 and 400 kilobytes, that is, between about 25% and 30% of their original size.

0	1	1	1	1	1	1	1

Table C.1: 127 run-length encoded in one byte.

1	0	0	0	0	0	0	0		1	0	0	0	0	0	0	0

Table C.2: 128 run-length encoded in two bytes.

Images are compressed by counting the alternate runs of black and white pixels. For example, if the first line of an image contains a run of 1000 white pixels followed by 2 black pixels followed by 1542 white pixels, then the first three entries in the run-length encoded file would be the numbers 1000, 2 and 1542. Compression is highest when the image contains large sections of all black or all white pixels and is lowest when there are frequent changes from black to white and back again. The files start with a 15 character text leader, followed by a three byte end of text file marker (DOS file) making a leader of 18 characters in all before the run data begins. Each line of the encoded file starts with a white run. If the actual image line begins with a black pixel, then a 0, indicating a white run, 0 pixels long, is entered at the

start of the corresponding line in the compressed file. Runs are stored in one or two bytes. If a run is 127 bits long or less, one byte is used with its most significant bit (msb) set to 0 (see Table C.1). If a run is greater than 127, a second byte is used. This is signaled by setting the msb of the first byte to 1 (see Table C.2). This allows for runs between 0 and 32767 pixels long. If images wider than 32767 bits are used, then an extra byte will be needed.

Appendix D

Photographs

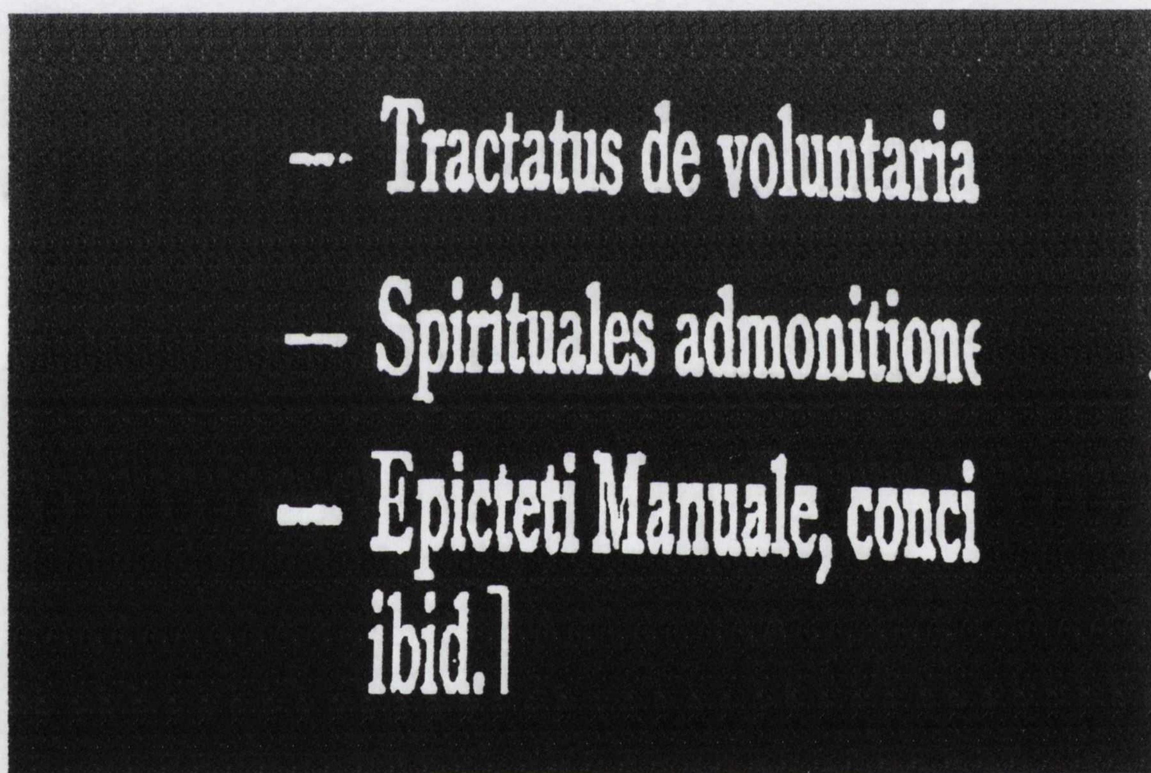


Figure D.1: Screen photo. showing a hyphen split in two parts.

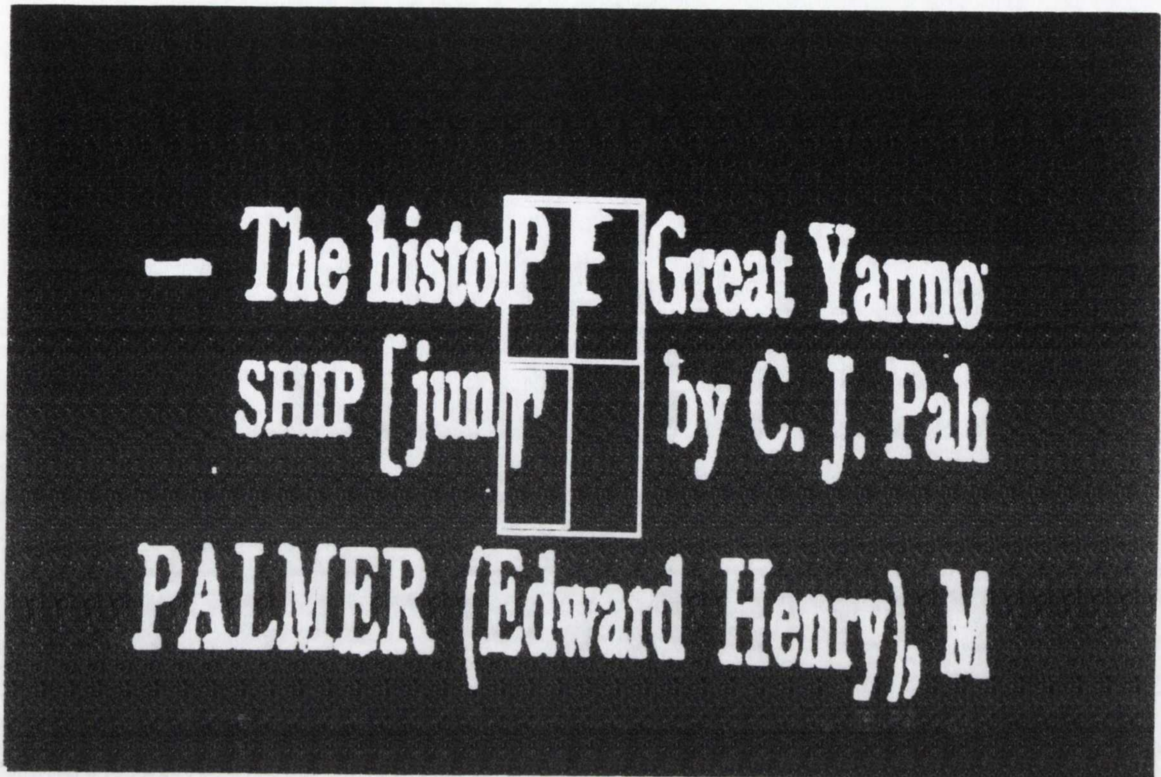


Figure D.2: A 'P' template with its horizontal and vertical signatures.

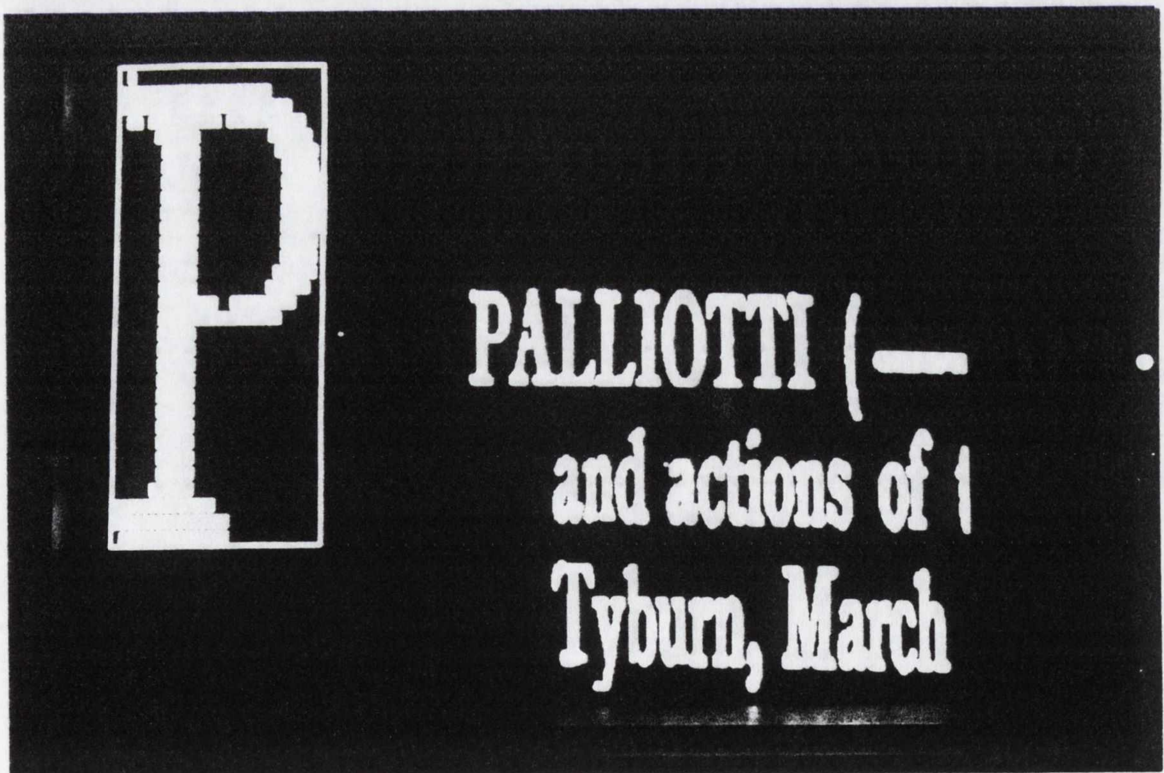


Figure D.3: A character enlarged within a window.

NILUS, [illegible]rchiepiscopus Thessa
αἰτίων τῆς ἐκκλησιαστικῆς δυστ
τοῦ πάπα. Τόποι ὀλίγοι τῷ αἰ
κοντες. Ἀρχὴ διαλέξεως τινος

Figure D.4: An entry composed almost entirely of Greek characters.

Bibliography

[1] Glynn Anderson. The Application of Optical Character Recognition to the Computerisation of Old-Library Catalogues. Technical report, University of Dublin, Trinity College, 1989.

[2] Glynn Anderson and Mark Deegan. Computerising the printed catalogue using optical character recognition. *Alumnus. Journal of Graduate Students Union, Trinity College*, pages 65 - 71, 1991.

[3] Kozo Banno, Takenori Kawamata, Keiji Kobayashi, and Hajime Nambu. Text Recognition System For Japanese Documents. *IEEE*, pages 176 - 180, 1988.

[4] K. G. Beauchamp. *Walsh Functions and their applications.* Academic Press, London, 1975.

[5] F. Bergadano and L. Saitta. A General Framework For Knowledge-based Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 1(2):13 - 29, February 1987.

[6] N. G. Burton and J. C. R. Licklider. Long-range constraints in the statistical structure of printed English. *American Journal of Psychology*, 68:650–653, 1955.

[7] D. A. Bush and J. A. Weaver. OCR and its Application to Documentation A state of the art review. Technical Report agard-ag-216, North Atlantic Treaty Organisation, March 1976.

[8] Carlos A. Cabrelli and Ursula M. Molter. Automatic Representation of Binary Images. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, PAMI-12(12):1190 - 1196, December 1990.

[9] R. Casey and G. Nagy. Recognition of printed Chinese Characters. *IEEE Trans. Electronic Computing*, EC(15):91 - 101, February 1966.

[10] A. Chapanis. The reconstruction of abbreviated printed messages. *Journal Experimental Psychology*, 48:496–510, 1954.

[11] James Geller. The Teachable Letter Recogniser. Technical report, Dep. of Computer Science University of New York at Buffalo.

[12] A. Hanson, E. Riseman, and E. Fisher. Context in word recognition. *Pattern Recognition*, 8:35 – 45, 1976.

[13] L. D. Harmon and E. J. Sitar. Method and apparatus for correcting errors and in mutilated text. U.S. Patent 3188609, issued June 8, 1965.

[14] Leon D. Harmon. Automatic Recognition of Print and Script. In *Proceedings Of The IEEE*, volume 60, pages 1165 – 1176, October 1972.

[15] Murray J. J. Holt. A Fast Binary Template Matching Algorithm For Document Image Data Compression. Technical report, Dep. of Electronic and Electrical Engineering Loughborough University of Technology.

[16] P. J. Hutton. Handwriting recognition: a gentle introduction. *The Computer Bulletin*, pages 18 – 21, December 1989.

[17] Simon Kahan, Theo Pavlidis, and Henry S. Baird. On the Recognition of Printed Characters of Any Font and Size. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, PAMI-9(2):274 – 288, March 1987.

[18] David P. M. Kelly. Automatic Document Structure Recognition. Master's thesis, University of Dublin, Trinity College, August 1991.

[19] Vincent Kinane and Ann O'Brien. "The Vast Difficulty of Cataloguing":The Printed Catalogue of Trinity College Dublin (1867-1887).

[20] Kenichi Mori and Isao Masuda. Advances in Recognition of Chinese Characters. *IEEE*, pages 692 – 702, 1980.

[21] Joseph O'Rourke and Richard Washington. Curve Similarity via Signatures. *Computational Geometry*, pages 295 – 317, 1985.

[22] A. Rajavelu, M. T. Musavi, and M. V. Shirvaikar. A Neural Network Approach to Character Recognition. *Neural Networks*, 2(2):387 – 393, April 1989.

[23] K. Sakai and K. Mori. Clustering of Chinese character patterns. *Transactions of IECE of Japan*, PRL73(14), 1973. in Japanese.

[24] C. E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.

[25] E. J. Sitar. Machine recognition of cursive script: The use of context for error detection and correction. Bell Labs., Murray Hill N.J.(unpublished), 1961.

[26] Jean Renard Ward and Barry Blesser. Interactive Recognition of Handprinted Characters for Computer Input. *IEEE CG&A*, pages 24 - 34, September 1985.

[27] Kazuhiko Yamamoto. Recognition Of Handprinted Characters By Convex And Concave Features. *IEEE*, pages 708 - 711, 1980.

[28] Hiroshi Yashiro, Tatsuya Murakami, Yoshihiro Shima, Yasuaki Nakano, and Hiromichi Fujisawa. A New Method of Document Structure Extraction using General Layout Knowledge. *International Workshop On Industrial Applications Of Machine Intelligence And Vision*, pages 282 - 287, April 1989.

```
#define SPOTWIDTH 2544
#define SPOTLENGTH 3900
#define SPOTHEADER 18
#define ALPHABETSIZE 100
#define AUTHORSIZE 70
#define PINDEXSIZE 50
#define TEMPLATESIZE 48
#define BLOBNUM 100
#define NAMESIZE 60
#define LINE_HEIGHT 35
/* LINE_HEIGHT changed from 35 to 30 to 40 to 50 to 35, 7,10/6/91 GPA */
#define OPEN 0
#define CLOSE 1
#define ESC 0x1b
#define RET 0x0d
#define FALSE 0
#define TRUE 1
#define TRANS_TICKS 15019.0
#define LFT 75
#define RGT 77
#define UP  72
#define DWN 80
#define NOISE 20        /* noise size changed from 10 to 20 14/8/90 Glynn */
#define SPACE 7         /* this is the space between two adjacent words   */
/* space changed from 7 to 10, 13/8/90 Glynn */
/* change back to 7, 15/8/90, back to 10, 16/11/90, back to 7, 4/4/91 */

#define max(a,b)        (((a) > (b)) ? (a) : (b))
#define min(a,b)        (((a) < (b)) ? (a) : (b))
#define getbit(x,y)     (page[y][x/8] & mask[x % 8])
#define setbit(x,y)     (page[y][x/8] |= mask[x % 8])

#define START   0
#define SUR     1
#define FIRST   2
#define ALT     3
#define TEXT    4
#define QUAL    5
#define LOC     6
#define DATE    7
#define FORMAT  8
#define SHELF   9

#define HEADER  10      /* this state is specially for finding the three */
/* letter headers */
#define TRAINING 11
/* adding 'intermediate states' 4/91 GPA */
#define FIR_QUAL 12             /* state between FIRST and QUAL */

/* SETS added 15/1/1991 GPA */

#define SETS    8               /* the no. of sets of templates */
/* changed from 6 to 8, 6/2/1991 */
#define STATES  13              /* the no. of states possible */
```

```
typedef struct {
  char name[5];
  char font;
  int width;
  int height;
  int perimeter;
  int area;
}

st;

typedef struct {
  unsigned char bitmap[TEMPLATESIZE][TEMPLATESIZE / 8];
  st stats;

}

tmplt;

typedef struct {
  unsigned char surname[NAMESIZE];
  unsigned char firstname[NAMESIZE];
  unsigned char filename[NAMESIZE];
}

pentry;

typedef struct {
  unsigned char surname[NAMESIZE];
  unsigned char firstname[NAMESIZE];
  int x, y;
  int byte_offset;
}

entry;

typedef struct {
  long int score;
  char name[5];
  char font;
  long int percent;
}

scr;

typedef struct {
  int xlow, xhigh, ylow, yhigh, perimeter, area;
}

blobtype;

struct list {
  blobtype blob;
  struct list *next;
  struct list *prev;
};

struct space_structure {
  int character;
  int size;
};

extern int order[STATES][SETS]; /**/
extern int no_of_sets[STATES];  /**/
extern tmplt *alpha[SETS];
extern no_of_tmplts[SETS];
extern entry entries[AUTHORSIZE];
extern pentry pageindex[PINDEXSIZE];
extern unsigned char *page[SPOTLENGTH];
extern scr scores[SETS][ALPHABETSIZE];
extern unsigned char mask[8];
```



```
extern unsigned char templatefilename[NAMESIZE], spotfilename[NAMESIZE];
extern unsigned char entryfilename[NAMESIZE], lastentry[NAMESIZE];
extern unsigned char firstentry[NAMESIZE];
extern unsigned char known[NAMESIZE];
extern unsigned char c, strng[200], guess[5];
extern unsigned char character[TEMPLATESIZE][TEMPLATESIZE / 8];
extern int x, y, loopcount, no_of_pages, left_text, temp;
extern int run, found, no_of_entries, length, size, traceflag;
extern int startorigx;
extern int toplimit, bottomlimit, firstchar, gap;
extern int complete;
extern int startorigy, lastxhigh, startright, origx, origy, scroll_on;
extern int starttop, c_line;
extern void *image, *chunk;
extern FILE *templatefile, *spotfile, *entryfile, *pageindexfile, *outfile;
extern int endp, startp, start, end, method, printer_on, thresh, s_blobs;
extern blobtype blob, tempblob, firstblob;
extern int no_of_blobs;
extern blobtype blobs[100];
extern int bufstart;
extern int bufend, bottom_text, state;
extern struct space_structure space;
extern char left_header[NAMESIZE], centre_header[NAMESIZE], right_header[NAMESIZE];
extern char prev_surname[NAMESIZE];
extern int count_error_1, count_error_2, count_total;    /* 5/6/91 GA */
extern int file_on;            /* 11/6/91 */
```

```
/* tocr.c       */
/* This module contains the main function. Main consists largely of a    */
/* large switch statement which interprets keys pressed. The switch       */
/* statement calls the main program functions located in tocr_an1.c and */
/* tocr_an2.c. Both single and double-character key presses are handled */
/* by the switch statement. Variables for the whole program are          */
/* declared in this module. When compiled, this module (tocr.c) is       */
/* linked with tocr_an1, tocr_an2, tocr_util, tocr_io, tocr_help,        */
/* tocr_scan and graphics. All '#defines' are defined in tocr_defs.h     */
/* which is needed by each of the modules. Types are defined in          */
/* tocr_var.h as well as extern declarations of the variables in this    */
/* module.      */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <timer.h>                /* exclusive to transputer */

#include "tocr_defs.h"
#include "tocr_var.h"
#include "graphics.h"
#include "tocr_scan.h"
#include "tocr_io.h"
#include "tocr_an1.h"
#include "tocr_an2.h"
#include "tocr_util.h"
#include "tocr_help.h"

tmplt *alpha[SETS];
int no_of_tmplts[SETS];

/* order added 22/1/91,12/2/91 GPA       */
int order[STATES][SETS] =
{0, 1, 0, 0, 0, 0, 0, 0,          /* START */
 0, 1, 0, 0, 0, 0, 0, 0,          /* SUR   */
 0, 1, 2, 0, 0, 0, 0, 0,          /* FIRST */
 0, 1, 2, 3, 0, 0, 0, 0,          /* ALT   */
 0, 1, 2, 3, 4, 5, 0, 0,          /* TEXT */
 0, 1, 2, 0, 0, 0, 0, 0,          /* QUAL */
 0, 4, 5, 0, 0, 0, 0, 0,          /* LOC   */
 0, 5, 0, 0, 0, 0, 0, 0,          /* DATE */
 0, 2, 5, 0, 0, 0, 0, 0,          /* FORMAT */
 4, 7, 0, 0, 0, 0, 0, 0,          /* SHELF */
 6, 0, 0, 0, 0, 0, 0, 0,          /* HEADER */
 0, 1, 2, 3, 4, 5, 6, 7,          /* TRAINING */
 0, 1, 2, 0, 0, 0, 0, 0           /* FIR_QUAL */
};

/* Sets: 0:specials,1:caps,2:lower,3:small,4:italics,5:digits,6:header */
/*       7:shelf         */

int no_of_sets[STATES] =
{2, 2, 3, 4, 6, 3, 3, 2, 3, 2, 1, 8, 3};          /* 22/1/91,12/2/91 GPA */
entry entries[AUTHORSIZE];
unsigned char *page[SPOTLENGTH];
pentry pageindex[PINDEXSIZE];
scr scores[SETS][ALPHABETSIZE];
unsigned char mask[8], templatefilename[NAMESIZE], spotfilename[NAMESIZE];
unsigned char entryfilename[NAMESIZE], lastentry[NAMESIZE];
unsigned char firstentry[NAMESIZE], known[NAMESIZE];
unsigned char c, strng[1000], guess[5],
        character[TEMPLATESIZE][TEMPLATESIZE / 8];
int x, y, loopcount, no_of_pages, left_text, temp, run;
int found = FALSE, no_of_entries = 0, length = 0, size, traceflag;
int startorigx;
int toplimit, bottomlimit, firstchar, gap;
int complete, startorigy, lastxhigh, startright;
int origx = -1, origy = -1, scroll_on = FALSE, starttop;
void *image, *chunk;
FILE *templatefile, *spotfile, *entryfile, *pageindexfile, *outfile;
```



```
int endp, startp, start, end, method = 3, printer_on = FALSE, thresh = 30;
int s_blobs = TRUE, c_line = 0;
blobtype blob, tempblob, firstblob;
int no_of_blobs = 0;
blobtype blobs[BLOBNUM];
int bufstart = 0;
int bufend = 0, bottom_text, state = 0;
struct space_structure space;
char left_header[NAMESIZE], centre_header[NAMESIZE], right_header[NAMESIZE];
char prev_surname[NAMESIZE];
int count_error_1, count_error_2, count_total, file_on = FALSE;

void main (void)
{

  int finished, num = 0, terminator, loop, loop2, found, value;
  int a, b, c, d;

  initialise ();
  showscreen (0, 0);
  finished = FALSE;
  while (!finished) {
    printf ("(%d,%d)\n", origx, origy);
    c = getch ();
    /* Program runs until the user types escape to quit */
    if (c >= 48 && c <= 58)       /* a numeral? */
      num = num * 10 + (int) c - 48;      /* convert to integer */
    if (c == 0) {
      c = getch ();
      switch (c) {
      case 59:                    /* scan entire page and log words with F1 */
        /* found */
        startp = timer_now ();
        for (loopcount = 0; loopcount < SETS; loopcount++)
          if (no_of_tmplts[loopcount] == 0) {
            printf ("No templates stored.\n");
            break;
          }
        state = TRAINING;
        printf ("Starting full page scan.\n");
        /* find start of text at left and hence centre line */
        if (c_line == 0)
          find_cline ();
        no_of_entries = 0;
        if (scroll_on)
          showscreen (0, 0);      /* start at beginning! */
        /* find_last_entry();*/
        startorigx = SPOTWIDTH;
        strng[0] = '\0';
        /* found = findblob(0,SPOTWIDTH-1,0,SPOTLENGTH-1);*/
        found = find_next_entry (0, left_text, left_text + 25, &tempblob);
        startright = tempblob.xlow + 1100;
        starttop = tempblob.ylow;
        /* startright is the estimated x coordinate of */
        /* the start of text on the right hand side of */
        /* the page */
        while (found) {
          guessauthor (entries[no_of_entries].surname,
                       entries[no_of_entries].firstname);
          entries[no_of_entries].x = firstblob.xlow;     /* record location */
          entries[no_of_entries].y = firstblob.ylow;
          if (strcmp (entries[no_of_entries].surname, "_") == 0)
            entries[no_of_entries].y -= 20;
          /* 'KNOWN' code will go back here */
          no_of_entries++;
          if (scroll_on)
            if (startorigx != origx || startorigy != origy)
              showscreen (startorigx, startorigy);
          startorigx = SPOTWIDTH;
          found = find_next_entry (firstblob.yhigh, firstblob.xlow,
                                   firstblob.xlow + 15, &tempblob);
```

```
    }                          /* end while more words */
    fflush (stdin);
    endp = timer_now ();
    printf ("Time taken: %f seconds.\n", (endp - startp) / TRANS_TICKS);
    break;
  case 60:                     /* find and guess a word with F2 */
    for (loopcount = 0; loopcount < SETS; loopcount++)
      if (no_of_tmplts[loopcount] == 0) {
        printf ("No templates stored.\n");
        break;
      }
    state = TRAINING;
    printf ("Starting training scan on word.\n");
    if (c_line == 0)
      find_cline ();
    scroll_on = TRUE;
    length = 0;
    firstblob.xlow = SPOTWIDTH;
    startorigx = SPOTWIDTH;
    strng[0] = '\0';
    found = findblob (origx, origx + XMAX, origy, origy + YMAX, &tempblob);
    strcpy (guess, " ");
    lastxhigh = SPOTWIDTH;
    firstchar = TRUE;
    terminator = FALSE;
    toplimit = tempblob.ylow;
    /*bottomlimit = tempblob.yhigh;*/
    bottomlimit = min (tempblob.yhigh, tempblob.ylow + LINE_HEIGHT);
    while (found && (strcmp (guess, "_") || !terminator)) {
      start = timer_now ();
      found = old_findsplit (guess, &blob);
      end = timer_now ();
      c = train (blob, guess);
      if (!strcmp (guess, ")") ||
          !strcmp (guess, ",") ||
          !strcmp (guess, "."))
        terminator = TRUE;
      if ((!strcmp (guess, "_") && firstchar) || (c == ESC))
        break;                 /* we don't want to go past */
      /* a '-' if it's the first   */
      /* character in an entry     */
      if (firstchar)
        firstchar = FALSE;
    }                          /* end while */
    printf ("%s\n", strng);
    strng[0] = '\0';
    break;
  case 61:                     /* find and guess a full text line with F3 */
    timestamp ("Very start of F3", timer_now ());
    for (loopcount = 0; loopcount < SETS; loopcount++)
      if (no_of_tmplts[loopcount] == 0) {
        printf ("No templates stored.\n");
        break;
      }
    state = TRAINING;
    printf ("Starting scan of full text line.\n");
    timestamp ("Before findblob in F3", timer_now ());
    if (c_line == 0)
      find_cline ();
    found = findblob (origx, origx + XMAX, origy, origy + YMAX, &tempblob);
    timestamp ("Right after findblob in F3", timer_now ());
    firstblob.xlow = SPOTWIDTH;
    startorigx = SPOTWIDTH;
    strcpy (guess, " ");
    lastxhigh = SPOTWIDTH;
    firstchar = TRUE;
    toplimit = tempblob.ylow;
    /*bottomlimit = tempblob.yhigh;*/
    bottomlimit = min (tempblob.yhigh, tempblob.ylow + LINE_HEIGHT);
    timestamp ("Before while loop in F3", timer_now ());
    while (found) {
```



```
      timestamp ("Before old_findsplit in F3", timer_now ());
      found = old_findsplit (guess, &blob);
      timestamp ("Right after old_findsplit in F3", timer_now ());
      if (firstchar) {
        firstchar = FALSE;
        firstblob = blob;
        startorigx = origx;
        startorigy = origy;
      }
      /* look for a ' ' between chars */
      if ((blob.xlow - lastxhigh - 1) > SPACE) {
        printf ("%c", ' ');
        if (file_on)
          fprintf (outfile, "%c", ' ');
        fflush (stdout);
      }
      printf ("%s", guess);
      if (file_on)
        fprintf (outfile, "%s", guess);
      fflush (stdout);
      lastxhigh = blob.xhigh;
    }                          /* end while */
    timestamp ("Right after while loop in F3", timer_now ());
    printf ("\n");
    if (file_on)
      fprintf (outfile, "\n");
    break;
  case 62:                     /* find and guess a character with F4 */
    printf ("Starting training scan on single character.\n");
    state = TRAINING;
    firstblob.xlow = SPOTWIDTH;
    startorigx = SPOTWIDTH;
    strng[0] = '\0';
    if (c_line == 0)
      find_cline ();
    if (findblob (origx, origx + XMAX, origy, origy + YMAX, &tempblob)) {
      guess_char (&blob, guess);
      train (blob, guess);
    }
    break;
  case 63:             /* test of classification of whole page with F5 */
    printf ("Do you really want to clasify a whole page? (y/n)\n");
    if (getch () != 'y')
      break;
    state = TRAINING;
    old_guesspage ();
    break;
  case 64:                     /* remove templates with F6 */
    printf ("Remove all templates (y/n)?\n");
    if (getch () == 'y') {
      printf ("Are you sure (y/n)\n");
      if (getch () == 'y') {
        printf ("Removing templates.\n");
        for (loop = 0; loop < SETS; loop++) {
          printf ("Remove set %d?\n", loop);
          if (getch () != 'y')
            continue;
          no_of_tmplts[loop] = 0;
          for (loop2 = 0; loop2 < ALPHABETSIZE; loop2++)
            for (y = 0; y < TEMPLATESIZE; y++)         /* clear templates */
              for (x = 0; x < TEMPLATESIZE / 8; x++)
                alpha[loop][loop2].bitmap[y][x] = 0;
        }
      }
    } else {
      printf ("Remove which template?\n");
      length = 0;
      while (length < 4 && ((c = getch ()) != RET)) {
        strng[length] = c;
        printf ("%c", strng[length++]);
        fflush (stdout);
```

```
      }                       /* end while */
      strng[length] = '\0';
      printf ("\nFrom which set?\n");
      do {
        scanf ("%d", &value);
        if (num < 0 || value >= SETS)
          printf ("Set must be between 0 and %d.\n", SETS - 1);
      }
      while (value < 0 || value >= SETS);
      printf ("\nRemove '%s'. Are you sure?\n", strng);
      if (getch () == 'y')
        for (loopcount = 0; loopcount < no_of_tmplts[value]; loopcount++)
          if (!strcmp (alpha[value][loopcount].stats.name, strng)) {
            printf ("Remove template no.%d (Y/N)?\n", loopcount);
            if (getch () == 'y') {
              for (x = loopcount; x < no_of_tmplts[value] - 1; x++)
                alpha[value][x] = alpha[value][x + 1];
              no_of_tmplts[value]--;
            }
          }
    }                         /* end else */
    strng[0] = '\0';
    break;
  case 65:                    /* display info on an entry with F7 */
    if (no_of_entries == 0)
      printf ("No entries found yet.\n");
    else {
      do {
        fflush (stdin);
        printf ("Please type in the name of the author\n");
      }
      while (scanf ("%s", strng) != 1);
      found = FALSE;
      for (loopcount = 0; loopcount < no_of_entries; loopcount++)
        if (strcmp (entries[loopcount].surname, strng) == 0) {
          found = TRUE;
          showscreen (entries[loopcount].x, entries[loopcount].y);
          pause ();
        } else
          found = FALSE;
      if (!found)
        printf ("Author not found.\n");
    }                         /* end else */
    break;
  case 66:                    /* display the bitmap of a template with F8 */
    printf ("Display which set?\n");
    scanf ("%d", &value);
    if (value >= SETS || value < 0) {
      printf ("Illegal set number.\n");
      break;
    }
    printf ("Set %d selected.\n", value);
    display_tmplt (value);
    break;
  case 67:                    /* change spot file with F9 */
    printf ("Do you really want to change page? (y/n)\n");
    if (getch () != 'y')
      break;
    fclose (spotfile);        /* close all current files */
    fclose (entryfile);
    fclose (pageindexfile);
    fclose (templatefile);
    getspotfile ();
    printf ("Loading page...\n");
    start = timer_now ();
    getpage ();
    end = timer_now ();
    printf ("Page loaded.\n");
    printf ("Time taken : %d ticks, %f seconds.\n", end - start,
            (end - start) / TRANS_TICKS);
    no_of_entries = 0;
```



```
    origx = -1;
    origy = -1;
    find_cline ();
    showscreen (0, 0);
    break;
  case 68:                    /* save individual entries with F10 */
    if (no_of_entries == 0) {
      printf ("No entries found yet. (use F1)\n");
      break;
    }
    printf ("Do you really want to save the entries?\n");
    if (getch () != 'y')
      break;
    printf ("Saving...\n");
    split_entries ();
    break;
  case 71:                    /* go to the top of the page with <home> */
    showscreen (origx, 0);
    break;
  case UP:                    /* go 100 pixels up with <uparrow> */
    if (num != 0)
      showscreen (origx, origy - num);
    else
      showscreen (origx, origy - 100);
    num = 0;
    break;
  case 73:                    /* go up 1 screen size with <pageup> */
    showscreen (origx, origy - YMAX);
    break;
  case LFT:           /* go one screen size to the left with <leftarrow> */
    if (num != 0)
      showscreen (origx - num, origy);
    else
      showscreen (origx - XMAX, origy);
    num = 0;
    break;
  case RGT:           /* go one screen size to the right with <rightarrow> */
    if (num != 0)
      showscreen (origx + num, origy);
    else
      showscreen (origx + XMAX, origy);
    num = 0;
    break;
  case 79:                    /* go to the end of the page with <end> */
    showscreen (origx, SPOTLENGTH - 1);
    break;
  case DWN:           /* go 100 pixels down with <downarrow> */
    if (num != 0)
      showscreen (origx, origy + num);
    else
      showscreen (origx, origy + 100);
    num = 0;
    break;
  case 81:                    /* go down 1 screen size with <pagedown> */
    showscreen (origx, origy + YMAX);
    break;
  case 84:            /* Mark a block and delete or keep with <ctrl>F1 */
    printf ("Mark a block of text.\n");
    mark_block ();
    break;
  case 86:                    /* guess page (test) with <shift>F3 */
    printf ("Really guess all the entries.(y/n)\n");
    if (getch () == 'y') {
      start = timer_now ();
      guesspage ();
      end = timer_now ();
    }
    printf ("Time taken : %d ticks, %f seconds.\n", end - start,
            (end - start) / TRANS_TICKS);
    break;
  case 89:            /* change template name or font with <shift> F6 */
```

```
      printf ("Change template name or font.\n");
      printf ("From which set?\n");
      do {
        scanf ("%d", &num);
        if (num < 0 || num >= SETS)
          printf ("Set num must be in the range 0..%d\n", SETS - 1);
      }
      while (num < 0 || num >= SETS);

      printf ("Change which template? (no.)\n");
      scanf ("%d", &value);
      if ((value >= no_of_tmplts[num]) || value < 0)
        printf ("Illegal value.\n");
      else {
        printf ("Change (n)ame or (f)ont?\n");
        if ((c = getch ()) == 'n') {
          printf ("Old name: %s\n", alpha[num][value].stats.name);
          printf ("Enter new name : ");
          scanf ("%s", strng);
          if (strlen (strng) < 5) {
            strcpy (alpha[num][value].stats.name, strng);
            printf ("Name changed to %s.\n", strng);
          } else {
            printf ("Name should be 4 chars or less.\n");
            printf ("Name not changed.\n");
          }
        } else {
          if (c == 'f') {
            printf ("Old font: %c\n", alpha[num][value].stats.font);
            printf ("Enter new font : ");
            fflush (stdin);
            scanf ("%c\n", &c);
            if ((c != 'A') && (c != 'B') && (c != 'I')) {
              printf ("Illegal font.\n");
              printf ("Font not changed.\n");
            } else {
              alpha[num][value].stats.font = c;
              printf ("Font changed to %c\n", c);
            }
          }
        }
      }
      num = 0;
      break;
    }                          /* end function key switch */
  }
  /* end 'if function key pressed' */
  else
    switch (c) {
    case ESC:
      printf ("Really quit? (Y/N)\n");
      c = getch ();
      if (c == 'y' || c == 'Y')
        finished = TRUE;
      break;
    case 10:                   /* spawn a dos shell after <ctrl RET>*/
      printf ("Spawning...\n");
      if (system ("\\command.com") == -1) {
        perror ("Error spawning.");
        printf ("Any key to continue...\n");
        pause ();
      }
      break;
    case 'a':
      show_entries ();
      break;
    case 'g':          /* go to a point on the library page */
      do {
        fflush (stdin);
        printf ("Please type in the coordinates of the screen\n");
      }
```



```
      while (scanf ("%d,%d", &x, &y) != 2);
      /* get coordinates of top left corner of screen to be viewed */
      start = timer_now ();
      showscreen (x, y);
      end = timer_now ();
      printf ("Time taken : %d ticks, %f seconds.\n", end - start,
              (end - start) / TRANS_TICKS);
      break;
    case 'h':            /* help menu */
      show_help ();
      break;
    case 'l':            /* load entries or templates */
      printf ("Load Authors or Templates (a/t)?\n");
      c = getch ();
      if (c == 'a' && (load_entries () == -1)) {
        printf ("Authors not found.\n");
      } else if (c == 't') {
        printf ("Load templates?\n");
        c = getch ();
        if (c == 'y') {
          strcpy (templatefilename, "tmplts");
          if (load_templates (templatefilename) == -1)
            printf ("Templates not found.\n");
          else
            printf ("Templates loaded.\n");
        }
      }
      break;
    case 'p':            /* print all or some of the templates */
      printf ("Do you really want to print the templates? (y/n)\n");
      if (getch () == 'y') {
        printf ("Print all templates? (y/n)\n");
        if (getch () == 'y')
          for (value = 0; value < SETS; value++)
            for (loop = 0; loop < no_of_tmplts[value]; loop++)
              print_tmplt (value, loop);
        else {
          printf ("From what set?\n");
          do {
            scanf ("%d", &value);
            if (value < 0 || value >= SETS)
              printf ("Illegal set no.\n");
          } while (value < 0 || value >= SETS);
          printf ("Type the number of the template you want to print.\n");
          scanf ("%d", &loop);
          if (loop < no_of_tmplts[value]) {
            print_tmplt (value, loop);
          } else
            printf ("Illegal template number.\n");
        }
      }
      break;
    case 's':            /* save entries or templates or page */
      printf ("Save Authors or Templates or Page (a/t/p)?\n");
      c = getch ();
      switch (c) {
      case 'a':
        save_entries ();
        break;
      case 't':
        if (save_templates (templatefilename) == -1)
          printf ("Error saving templates.\n");
        else
          printf ("Templates saved.\n");
        break;
      case 'p':
        printf ("Do you really want to save the page?\n");
        if (getch () == 'y') {
          printf ("Do you want a text header at the top of the ");
          printf ("file?\n");
          if (getch () == 'y')
```

```
          save_page (TRUE);
        else
          save_page (FALSE);
      }
      break;
    }                         /* end switch */
    break;
  case '*':           /* show status */
    show_status ();
    break;
  case 'S':           /* set parameters */
    printf ("Which parameter do you want to set?\n");
    printf ("(s)croll, (p)rinter, (t)raceflag, (f)ile\n");
    printf ("t(h)reshold, (m)ethod , sub_(b)lobs\n");
    switch (getch ()) {
    case 'b':                 /* toggle sub_blobs parameter */
      s_blobs = !s_blobs;
      printf ("Sub_blobs scanning is %s\n", s_blobs ? "ON" : "OFF");
      break;
    case 'f':                 /* open a file for output */
      file_out ();
      break;
    case 'h':                 /* set threshold */
      printf ("Threshold %d\n", thresh);
      printf ("Type in a threshold value.\n");
      scanf ("%d", &thresh);
      break;
    case 'm':                 /* change method */
      printf ("Method is now %d\n", method);
      printf ("Type in new method no. between 1 and 5.\n");
      scanf ("%d", &value);
      if (value < 1 || value > 5)
        printf ("Illegal value.\n");
      else {
        method = value;
        printf ("Method is now %d\n", method);
        switch (method) {
        case 1:
          thresh = 30;
          break;
        case 2:
          thresh = 95;
          break;
        case 3:
          thresh = 30;
          break;
        case 4:
          thresh = 95;
          break;
        case 5:
          thresh = -60;
          break;
        default:
          thresh = 30;
          break;
        }
      }
      printf ("Threshold %d\n", thresh);
      break;
    case 'p':                 /* turn printer on and off */
      printer ();
      break;
    case 's':                 /* toggle scroll_on with 's' */
      if (scroll_on)
        printf ("Scrolling is OFF\n");
      else
        printf ("Scrolling is ON\n");
      scroll_on = !scroll_on;
      break;
    case 't':                 /* toggle traceflag with F7 */
      traceflag = !traceflag;
```



```
      printf ("Traceflag %s\n", traceflag ? "ON" : "OFF");
      break;
    }                         /* end switch */
    break;
  case '#':           /* display the entire page, scaled onto one screen */
    show_page ();
    break;
  case '?':           /* show times stamps */
    show_times ();
    break;
  case '+':           /* digitise page */
    printf ("Are you sure you want to digitise a page?\n");
    if (getch () == 'y') {
      printf ("Do you want to check alignment?\n");
      if (getch () == 'y')
        digitise (TRUE);
      else
        digitise (FALSE);
    }
    /* true indicates a quality test will be done. */
    break;
  case '>':           /* test of 'find_last_entry' */
    printf ("Searching for last entry.\n");
    find_cline ();
    find_last_entry ();
    break;
  case '<':           /* test of 'split_page' */
    printf ("Do you really want to split the page?\n");
    if (getch () == 'y') {
      printf ("Saving split page.\n");
      split_page ();
    }
    break;
  case '|':           /* test of 'find_header' */
    find_header (&a, &b, &c, &d);
    break;
  case '!':           /* test of segment line */
    printf ("Really segment a line? (y/n)\n");
    if (getch () == 'y')
      if (!segment_line (origx, origx + XMAX, origy, origy + YMAX))
        printf ("No line found.\n");
    break;
  case '@':           /* test of mark entries */
    mark_entries ();
    break;
  case '~':           /* test of guessentry */
    if (no_of_entries == 0)
      mark_entries ();
    printf ("Guess which entry?\n");
    do {
      fflush (stdin);
      scanf ("%d", &value);
      if (value >= no_of_entries || value < 0)
        printf ("%d is an illegal value - Try again.\n", value);
    }
    while (value >= no_of_entries || value < 0);
    /*display_entry(value);*/
    guessentry (value);
    /*showscreen(origx,origy);*/
    break;
  case '[':           /* full save */
    full_save ();
    break;
  case ']':           /* full read */
    full_read ();
    break;
  case ':':           /* test of first_and_last */
    first_and_last (known);
    break;
  case '{':           /* verify template statistics */
    v_templates ();
```

```
            break;
        }                               /* end ordinary key switch */
    }                                   /* end while */
    fclose (spotfile);
    fclose (entryfile);
    fclose (pageindexfile);
    fclose (templatefile);
    fclose (outfile);
    video_mode (MONO_80COL_TEXT_MODE);

}/* end it all */
```

```
int old_findsplit (char guess[], blobtype * blob);
int inc (int x);
int dec (int x);
void guessauthor (char surname[], char firstname[]);
char train (blobtype blob, char guess[]);
int find_next_entry (int starty, int startx, int stopx, blobtype * tempblob);
int find_last_entry (void);
int find_header (int *x1, int *y1, int *x2, int *y2);/* revised 13/2/91 GPA */
void guessword (char word[]);
void old_guesspage (void);
void find_cline (void);

/* 'f_s_b' altered 18/7/1990, 4/10/1990, 4/6/1991 */
int find_sub_blobs (blobtype * tempblob, blobtype last_blob);

/* 10/10/1990,16/10/90 GPA */
int segment_line (int startx, int stopx, int starty, int stopy);
void guessentry (int e_num);     /* 16/10/90 GPA */
void mark_entries (void);        /* 16/10/90 GPA */
void guesspage (void);           /* 22/10/90 GPA */
int delimit_fields (char guess[], int set, int num, int *state, int newline,
                        int *end, int pos);

/* delimit_fields -> 24/10/90, 12/12/90, 9/1/91,16/1/91 GPA */
/* check_space -> 8/11/90, 23/1/91, 6/5/91 GPA */
void check_space (blobtype blobb, blobtype bloba,
                        struct space_structure *space);

/* findsplit altered 16/1/91,12/3/91 */
int findsplit (blobtype * blob, int *num, int *set, char *nomatch);
int check_slope (int x1, int y1, int x2, int y2);       /* 13/2/91 GPA */

/* guess_char 21/2/91 gpa */
int guess_char (blobtype * blob, char guess[]);
void first_and_last (char known[]);     /* 22/2/91 GPA */
```

```c
/* tocr_an1.c */

/* This module contains high level character recognition functions      */
/* which rely on low level OCR functions found in tocr_an2.c. The       */
/* functions in this module are generally called by tocr.c while the    */
/* functions in tocr_an1.c generally call functions in tocr_an2.c.      */
/* Function prototypes are declared in tocr_an1.h.       */

#include <stdio.h>
#include <time.h>
#include <timer.h>                  /* exclusive to transputer */
#include <string.h>
#include <stdlib.h>
#include <math.h>                   /* used for finding slope with atan */

#include "tocr_defs.h"
#include "tocr_vars.h"
#include "graphics.h"
#include "tocr_io.h"
#include "tocr_an1.h"
#include "tocr_an2.h"
#include "tocr_util.h"

#define MAX_ANGLE 0.1

static struct list bloblist, *blob_ptr;

/* The 'old_findsplit' function puts sucessive blobs into a circular    */
/* buffer until the next traced blob would either make the combined     */
/* blob too large or is too far(large gap) from the previous blob. The  */
/* function then tries to find a good match by matching against a       */
/* combined version of the n blobs in the buffer. If this does not      */
/* produce a 'good' match, a match is then done against n-1 blobs,      */
/* excluding the one furthest to the right. This excluded blob stays in */
/* the buffer to avoid having to re-trace it. When a good match is      */
/* found or there is only one blob involved in the match, the x blobs   */
/* involved in the match are removed from the buffer and the function   */
/* terminates. Blobs remaining in the buffer are matched the next time  */
/* 'old_findsplit' is called. */

int old_findsplit (char guess[], blobtype * blob)
{

  static blobtype blobs[24];
  static int bufend = 0, bufstart = 0, no_of_blobs = 0, blobfound = TRUE;
  blobtype largest;
  int split = TRUE, temp, loop, top, bottom;
  int finished, dot = FALSE, num, set;
  char nomatch = FALSE;
  long score;

  blob->perimeter = 0;
  /* reset buffer in case of irregular escape last time */
  if (firstchar) {
    bufstart = bufend = 0;
    no_of_blobs = 0;
    blobfound = TRUE;
  }
  if (blobfound)                /* if a temporary blob has been scanned */
    /* while there is a split between chars, add to buffer */
    while (split)
      if ((no_of_blobs == 0) || /* if (a || (b && c)) */
          ((tempblob.xlow - (blobs[dec (bufend)].xhigh) < SPACE) &&
         /* check gap */
           (tempblob.xhigh - (blobs[bufstart].xlow) - 1 <= TEMPLATESIZE))) {     /* check o
verall width */
        blobs[bufend] = tempblob;       /* add blob to buffer */
        bufend = inc (bufend);
        no_of_blobs++;
        split = TRUE;
```



```c
        /* find new blob */
        temp = dec (bufend);
        /* the following lines reset the vertical limits within which a */
        /* character is searched for on a line */
        if (bottomlimit - toplimit < LINE_HEIGHT) {
          if (tempblob.ylow < toplimit &&
              bottomlimit - tempblob.ylow <= LINE_HEIGHT)
            toplimit = tempblob.ylow;
          if (tempblob.yhigh > bottomlimit &&
              tempblob.yhigh - toplimit <= LINE_HEIGHT)
            bottomlimit = tempblob.yhigh;
        }
        if (!(blobfound = findblob (blobs[temp].xhigh + 1,
                                    blobs[temp].xhigh + 100,
                                    toplimit,
                                    bottomlimit,
                                    &tempblob)))
          break;
        /* s_blobs 'bit' moved to before guessletter 5/10/90 */
        /*if(s_blobs)
              dot = find_sub_blobs(&tempblob);*/
      } else
        split = FALSE;
  /* now do guess */
  finished = FALSE;
  temp = bufend;
  do {
    largest.xhigh = -1; /* use these four to find the bounding box of a */
    largest.yhigh = -1;           /* combined blob */
    largest.xlow = SPOTWIDTH;
    largest.ylow = SPOTLENGTH;
    temp = dec (temp);
    loop = dec (bufstart);
    largest.perimeter = 0;
    do {
      loop = inc (loop);
      if (blobs[loop].xlow < largest.xlow)       /* find bounding box of */
        largest.xlow = blobs[loop].xlow;         /* combined blobs */
      if (blobs[loop].ylow < largest.ylow)
        largest.ylow = blobs[loop].ylow;
      if (blobs[loop].xhigh > largest.xhigh)
        largest.xhigh = blobs[loop].xhigh;
      if (blobs[loop].yhigh > largest.yhigh)
        largest.yhigh = blobs[loop].yhigh;
      largest.perimeter += blobs[loop].perimeter;       /* approx.! */
    }
    while (loop != temp);
    if (s_blobs)
      dot = find_sub_blobs (&largest, blobs[loop - 1]);
    strcpy (guess, guessletter (&largest, &score, &num, &set, &nomatch));

    if (score > thresh || temp == bufstart) {    /* thresh is a parameter */
      *blob = largest;
      finished = TRUE;
      if (dot && !strcmp (guess, "l")) {
        printf ("%c", ' ');       /* char to indicate a change */
        if (file_on)
          fprintf (outfile, "%c", ' '); /*   is 175 ascii */
        strcpy (guess, "i");
      }
      do {
        bufstart = inc (bufstart);       /* remove blobs from buffer */
        no_of_blobs--;
      }
      while (bufstart != inc (temp));
    }
  }                                 /* end while */
  while (temp != bufstart && !finished);
  if (no_of_blobs != 0)
    return (TRUE);
  else
```

```
    return (blobfound);

}                                   /* end old_findsplit */

/* the 'inc' function is called by old_findsplit to increment a buffer  */
/* pointer */

int inc (int x)
{

  if (x == 9)
    return (0);
  else
    return (++x);

}                                   /* end inc */

/* the 'dec' function is called by old_findsplit to decrement a buffer  */
/* pointer */

int dec (int x)
{

  if (x == 0)
    return (9);
  else
    return (--x);

}                                   /* end dec */

/* The 'train' procedure presumes the bounding box of a character has    */
/* been found and displays that character on the screen in reverse       */
/* video. The function is used by the OCR sytem to store instantiations */
/* of characters as templates, in one of several template arrays. */

char train (blobtype blob, char guess[])
{

  int i, j, size, width, height;
  char c;

  width = (blob.xhigh - blob.xlow) - 1;
  height = (blob.yhigh - blob.ylow) - 1;
  if (firstchar) {
    /* toplimit and bottom limit mark the */
    /* bounds within which all following  */
    /* characters are searched for.*/
    firstblob = blob;

    toplimit = blob.ylow;
    bottomlimit = blob.yhigh;
    startorigx = origx;
    startorigy = origy;
  }
  size = imagesize (blob.xlow, blob.ylow, blob.xhigh, blob.yhigh);
  if ((chunk = (void *) malloc (size)) == NULL) {
    perror ("Malloc failed in 'train'.");
    exit (1);
  }
  getimage (blob.xlow - origx, blob.ylow - origy, blob.xhigh - origx,
              blob.yhigh - origy, chunk);
  putimage (blob.xlow - origx, blob.ylow - origy, chunk, NOT_PUT);
  cga_update ();
  printf ("Char: %s Area: %d Per: %d Width: %d Height: %d\n", guess, blob.area,
              blob.perimeter, width, height);
  c = getch ();
  putimage (blob.xlow - origx, blob.ylow - origy, chunk, COPY_PUT);
  cga_update ();
  if ((c != ' ') && (c != ESC))
    if (c == '?') {
      printf ("%s in %d ticks,%f seconds\n", guess, end - start,
```



```
                (end - start) / TRANS_TICKS);
        /* look for a ' ' between chars */
        if ((blob.xlow - lastxhigh - 1) > SPACE)
          strcat (strng, " ");
        strcat (strng, guess);
        lastxhigh = blob.xhigh;
        /* show scores here */
        if (getch () == '*')
          show_scores ();
    }
  /* end then part */
    else if (c == 0)
      if ((c = getch ()) == 59) {/* F1 pressed so display expanded char */
        gwindow (OPEN, 0, 0, width * 5 + 1, height * 5 + 1);
        for (j = 0; j < height; j++)     /* read character from */
          for (i = 0; i < width; i++)    /* screen */
            if (j < TEMPLATESIZE && i < TEMPLATESIZE)
              if (page[blob.ylow + 1 + j][(blob.xlow + 1 + i) / 8] &
                mask[(blob.xlow + 1 + i) % 8])
                /*if(getbit(blob.xlow+1+i,blob.ylow+1+j))*/
                bar (i * 5 + 2, j * 5 + 2, i * 5 + 5, j * 5 + 5, 1);
        /*for(j = 0;j < height*7;j += 7)
                line(0,j,width*7,j);
             for(i = 0;i < width*7;i += 7)
                line(i,0,i,height*7);*/
        cga_update ();
        pause ();
        gwindow (CLOSE);
      } else if (c == 60)        /* F2 key pressed so try shifted match */
        shift_match (blob);
      else
        fflush (stdin);
    else if (c == 's')
      store_template (blob);
  putimage (blob.xlow - origx, blob.ylow - origy, chunk, COPY_PUT);
  cga_update ();
  free (chunk);
  return (c);

}                                   /* end train */

/* the 'guessauthor' function returns the surname and firstname of an   */
/* entry. It presumes a blob has been found by 'findblob' and then      */
/* calls  'old_findsplit' to merge and guess characters. It also        */
/* records the coordinates of the first character found, which are used */
/* by 'find_next_entry' to find the start of the next entry name.       */

void guessauthor (char surname[], char firstname[])
{

  int charfound = TRUE, lastxhigh = SPOTWIDTH, terminator = FALSE;
  char c[5];

  surname[0] = '\0';
  firstname[0] = '\0';
  firstchar = TRUE;
  toplimit = tempblob.ylow;
  bottomlimit = tempblob.yhigh;
  charfound = TRUE;
  terminator = FALSE;
  while (charfound && !terminator) {
    charfound = old_findsplit (c, &blob);

    if (firstchar) {
      /* These limits are the limits within */
      /* which the next character is searched for */
      toplimit = blob.ylow;
      bottomlimit = blob.yhigh;
      /* firstblob records the position of the first letter in the */
      /* author. This is used later on to find the start of the next */
      /* author. */
```

```
      firstblob = blob;
      startorigx = origx;
      startorigy = origy;
    }
    /* 'KNOWN' code was here */
    if (strcmp (c, ".") == 0 || strcmp (c, ",") == 0 || strcmp (c, "(") == 0)
      terminator = TRUE;
    else {
      if ((blob.xlow - lastxhigh - 1) > SPACE)
        /* look for a ' ' between chars */
        strcat (surname, " ");
      strcat (surname, c);
    }
    lastxhigh = blob.xhigh;
    if (!strcmp (c, "_") && firstchar)  /* don't go past leading hyphens */
      break;
    if (firstchar)
      firstchar = FALSE;
  }                              /* end while */

  firstname[0] = '\0';
  terminator = FALSE;
  while (charfound) {
    if (strcmp (c, "_") == 0)
      break;
    if (strcmp (c, ".") != 0 && strcmp (c, ",") != 0 && strcmp (c, "_") != 0) {
      if ((blob.xlow - lastxhigh - 1) > SPACE)
        /* look for a ' ' between chars */
        strcat (firstname, " ");
      strcat (firstname, c);
    }
    lastxhigh = blob.xhigh;
    charfound = old_findsplit (c, &blob);
  }                              /* end while */

}                                /* end guessauthor */

/* The 'find_next_entry' function scans the page for a significant blob */
/* in the area delineated by the three paramaters. If no blob is found, */
/* FALSE is returned, else TRUE. If no blob is found on the left hand   */
/* side of the page, the search is continued on the right.              */

int find_next_entry (int starty, int startx, int stopx, blobtype * tempblob)
{

  int x, y, wordfound = FALSE;
  int col, tempperim;
  static int left_hand_side = TRUE;

  y = starty;
  while (!wordfound) {
    while (y < SPOTLENGTH && !wordfound) {
      x = startx;
      while (x <= stopx && !wordfound) {
        if (scroll_on && traceflag) {
          col = gethirespixel (x - origx, y - origy);
          putdirectpixel (x - origx, y - origy, !col);
          /*for(loopcount=0;loopcount<300;loopcount++);*/
          putdirectpixel (x - origx, y - origy, col);
        }
        /*if(gethirespixel(x,y))*/
        if (page[y][x >> 3] & mask[x & 7]) {
          /*if( getbit(x,y)){*/
          if ((tempperim = trace_perim (x, y, tempblob)) > NOISE)
            wordfound = TRUE;
          else if (tempperim == -1) {
            wordfound = -1;
            break;
          }
          x = startx;
          y = tempblob->yhigh;
```



```
        } else
          x += 2;                /* scan every second column */
      }                          /* end while */
      if (wordfound == -1) {
        wordfound = FALSE;
        break;
      } else
        y += 2;                  /* scan every second row */
    }                            /* end while */
    if (!wordfound) {
      if (!left_hand_side) {
        wordfound = FALSE;
        left_hand_side = TRUE;
        break;
      } else {
        left_hand_side = FALSE;
        /* showscreen(startright,0);*/
        /* startright is the estimated x */
        /* coordinate of the start of */
        /* text on the right hand side */
        /* of the screen */
        firstblob.xlow = SPOTWIDTH;
        wordfound = findblob (startright, SPOTWIDTH - 1, starttop - 20,
                              starttop + 30, tempblob);
      }
    }                            /* end 'if !wordfound' */
  }                              /*end while */
  return (wordfound);

}                                /* end find_next_entry */

/* the 'find_last_entry' function searches for the last entry on the    */
/* page and copies the surname of this last entry into the global       */
/* variable lastentry. This is later compared with the first entry on   */
/* the page to provide high level info. on the alphabetical order of    */
/* the entries.             */

int find_last_entry (void)
{

  int x, y, wordfound, startx, stopx, col, tempperim;
  static unsigned char cname[NAMESIZE];
  int hyphen = TRUE;             /**/

  firstblob.xlow = SPOTWIDTH;
  y = SPOTLENGTH - 1;
  startx = left_text - 7 + 1100;
  stopx = left_text + 7 + 1100;
  while (hyphen) {               /* while the entry found is a single hyphen */
    wordfound = FALSE;
    while (y > 0 && !wordfound) {
      x = startx;
      while (x <= stopx && !wordfound) {
        if (scroll_on && traceflag) {
          col = gethirespixel (x - origx, y - origy);
          putdirectpixel (x - origx, y - origy, !col);
          /*for(loopcount=0;loopcount<300;loopcount++);*/
          putdirectpixel (x - origx, y - origy, col);
        }
        if (page[y][x >> 3] & mask[x & 7]) {
          /*if( getbit(x,y) ){*/
          if ((tempperim = trace_perim (x, y, &tempblob)) > NOISE)
            wordfound = TRUE;
          x = startx;
          y = tempblob.yhigh - 2;       /* to compensate for pending +2 */
        } else
          x += 2;                /* scan every second column */
      }                          /* end while */
      y -= 2;                    /* scan every second row */
    }                            /* end while */
    if (wordfound)
```

```
      guessauthor (lastentry, cname);
    else
      break;
    if (strcmp (lastentry, "_") != 0)
      hyphen = FALSE;
    else {
      startx = firstblob.xlow;
      stopx = firstblob.xhigh;
      y = firstblob.ylow;
    }
  }                              /* end 'while hyphen' */
  printf ("Lastentry is %s\n", lastentry);
  return (wordfound);

}                                /* end find_last_entry */

/* the 'find_header' function finds and recognises the two three-letter */
/* indicators and the page number at the top of the page. coordinates   */
/* of a line are returned which can be used to find the slope of the    */
/* header */

int find_header (int *x1, int *y1, int *x2, int *y2)
{

  int ok;

  state = HEADER;
  if (ok = findblob (0, SPOTWIDTH, 0, 100, &tempblob)) {
    guessword (left_header);
    printf ("Left: %s\n", left_header);
    if (file_on)
      fprintf (outfile, "Left: %s\n", left_header);
    *x1 = firstblob.xlow + 1;
    *y1 = firstblob.ylow + 1;
    if (ok = findblob (blob.xhigh + 1, SPOTWIDTH, 0, 100, &tempblob)) {
      guessword (centre_header);
      printf ("Centre: %s\n", centre_header);
      if (file_on)
        fprintf (outfile, "Centre: %s\n", centre_header);
      if (ok = findblob (blob.xhigh + 1, SPOTWIDTH, 0, 100, &tempblob)) {
        guessword (right_header);
        printf ("Right: %s\n", right_header);
        if (file_on)
          fprintf (outfile, "Right: %s\n", right_header);
        *x2 = blob.xhigh - 1;
        *y2 = blob.ylow + 1;
      }
    }
  }
  return (ok);

}                                /* end find_header */

/* the 'check_slope' function checks the slope of a line defined by     */
/* four parameters, x1,y1,x2,y2. if the slope is within an accepted     */
/* tolerance, the function returns as TRUE, else FALSE.                 */

int check_slope (int x1, int y1, int x2, int y2)
{

  double tilt;
  int ok = TRUE;

  printf ("(x1,y1) (x2,y2)\n");
  printf ("(%d,%d) (%d,%d)\n", x1, y1, x2, y2);
  tilt = atan (((double) (y2 - y1)) / (x2 - x1)) * 180.0 / 3.14159;
  printf ("Page is sloping at %.4f degrees.\n", tilt);
  if (fabs (tilt) >= MAX_ANGLE) {
    printf ("This is unacceptable and the page is rejected.\n");
    ok = FALSE;
  } else
```



```
    printf ("This is within the accepted tolerance.\n");
  return (ok);

}                                /* end check_slope */

/* the 'guessword function' presumes an initial blob has been found and */
/* traced and continues finding, combining and guessing blobs until no  */
/* more blobs have been found. 'Guessed' blobs are added to the 'word'  */
/* array                                                                */

void guessword (char word[])
{

  int charfound = TRUE, lastxhigh = SPOTWIDTH;
  char c[5];

  word[0] = '\0';
  firstchar = TRUE;
  toplimit = tempblob.ylow;
  bottomlimit = tempblob.yhigh;
  while (charfound) {
    charfound = old_findsplit (c, &blob);
    if (firstchar) {
      /* firstblob records the position of the first letter in the word. */
      /* This is used later on to find the start of the next word */
      firstblob = blob;
      toplimit = blob.ylow;
      bottomlimit = blob.yhigh;
      /* These limits are the limits within */
      /* which the next character is searched for */
      startorigx = origx;
      startorigy = origy;
    }
    if ((blob.xlow - lastxhigh - 1) > SPACE)/* look for a ' ' between chars */
      strcat (word, " ");
    strcat (word, c);
    lastxhigh = blob.xhigh;
    if (firstchar)
      firstchar = FALSE;
  }                              /* end while */

}                                /* end guessword */

/* The 'old_guesspage' function classifies the entire page, one line at */
/* a time. The start of each entry is found with the findblob function. */
/* Characters are segmented and classified with the old_findsplit       */
/* function.    */

void old_guesspage (void)
{

  int found, screen = TRUE, ok = FALSE, count = 0, loop;
  char c;
  FILE *textfile;
  char filename[NAMESIZE];

  printf ("Output from a full page scan can be sent to a file or to the\n");
  printf ("screen. Output can also be sent the printer in parallel.\n");
  printf ("\nOutput to (s)creen or (f)ile?\n");
  c = getch ();
  if (c == 'f' || c == 'F') {
    screen = FALSE;
    do {
      printf ("Enter the filename in full.\n");
      fflush (stdin);
      scanf ("%s", filename);
      if ((textfile = fopen (filename, "r")) != NULL)
        printf ("File already exists.\n");
      else {
        ok = TRUE;
        fclose (textfile);
```

```
        if ((textfile = fopen (filename, "wt")) == NULL) {
          perror ("Unable to open file for writing\n");
          exit (1);
        }
      }
    }                           /* end do */
    while (!ok);
  }
  printf ("Starting scan of full page.\n");
  printf ("Method is %d\n", method);
  printf ("Threshold is %d\n", thresh);
  if (c_line == 0)
    find_cline ();
  found = find_next_entry (0, left_text, left_text + 25, &tempblob);
  while (found) {
    firstblob.xlow = SPOTWIDTH;
    startorigx = SPOTWIDTH;
    strcpy (guess, " ");
    lastxhigh = SPOTWIDTH;
    firstchar = TRUE;
    toplimit = tempblob.ylow;
    bottomlimit = tempblob.yhigh;
    while (found) {
      found = old_findsplit (guess, &blob);
      if (firstchar) {
        firstchar = FALSE;
        firstblob = blob;
        toplimit = blob.ylow;
        bottomlimit = blob.yhigh;
        startorigx = origx;
        startorigy = origy;
      }
      /* look for a ' ' between chars */
      if ((blob.xlow - lastxhigh - 1) > SPACE) {
        if (screen)
          printf ("%c", ' ');
        else
          fprintf (textfile, "%c", ' ');
        if (file_on)
          fprintf (outfile, "%c", ' ');
        fflush (stdout);
      }
      if (screen)
        printf ("%s", guess);
      else
        fprintf (textfile, "%s", guess);
      if (file_on)
        fprintf (outfile, "%s", guess);
      fflush (stdout);
      lastxhigh = blob.xhigh;
    }                           /* end while */
    if (screen)
      printf ("\n");
    else {
      fprintf (textfile, "\n");
      printf ("Line %d.\n", ++count);
    }
    if (file_on)
      fprintf (outfile, "\n\n");

    /* now search for next line, allowing one blank line between */

    if (!(found = findblob (left_text, c_line, firstblob.yhigh + 20,
                            firstblob.yhigh + 40, &tempblob)))
      found = findblob (left_text, c_line, firstblob.yhigh + 41,
                        firstblob.yhigh + 61, &tempblob);
  }                             /* end 'while there is another line */
  if (!screen)
    fclose (textfile);

}                               /* end old_guesspage */
```



```
/* the 'find_cline' procedure does a vertical signature analysis of the */
/* entire page and then finds the start of text on the left hand side   */
/* by looking for a sharp rise in the signature from one column to the  */
/* next. The center line is then found by adding a known displacement   */
/* to the leftmost position of the text. */

void find_cline (void)
{

  int signature[SPOTWIDTH / 6], loopy, loop, loopx, text_found;

  /* clear signatures */
  for (loop = 0; loop < SPOTWIDTH / 6; loop++)
    signature[loop] = 0;

  timestamp ("before sig. setup", timer_now ());
  for (loopy = 0; loopy < SPOTLENGTH; loopy += 4)
    for (loopx = 0; loopx < SPOTWIDTH / 6; loopx++)
      if (page[loopy][loopx >> 3] & mask[loopx & 7])
        /*if(getbit(loopx,loopy))*/
        signature[loopx]++;
  timestamp ("after sig. setup & before search", timer_now ());
  printf ("Signature table set up.\n");
  text_found = FALSE;
  loopx = 0;
  for (loop = 1; loop < SPOTWIDTH / 6 && !text_found; loop++) {
    if (signature[loop] > signature[loop - 1] + 5) {
      left_text = loop;
      text_found = TRUE;
    }
  }                             /* end for */
  c_line = left_text + 1080;
  timestamp ("After search", timer_now ());

}                               /* end find cline */

/* the 'find_sub_blobs' function takes a blob and scans directly above  */
/* and below to find blobs that may be related to the primary blob. a   */
/* check is made on the last blob to ensure that it is not mistaken for */
/* a sub blob as can happen if it overhangs the current character       */

int find_sub_blobs (blobtype * tempblob, blobtype last_blob)
{

  blobtype newblob;
  int found = FALSE;


  if (tempblob->ylow > toplimit)
    if (findblob ((tempblob->xlow + tempblob->xhigh) / 2,
                  (tempblob->xlow + tempblob->xhigh) / 2,
                  toplimit, tempblob->ylow, &newblob))
      if (newblob.ylow >= toplimit - 10 &&
          newblob.yhigh <= bottomlimit + 10 &&
          newblob.xlow != last_blob.xlow) {     /* could be any of the four! */

        tempblob->xlow = min (tempblob->xlow, newblob.xlow);
        tempblob->ylow = min (tempblob->ylow, newblob.ylow);
        tempblob->xhigh = max (tempblob->xhigh, newblob.xhigh);
        tempblob->yhigh = max (tempblob->yhigh, newblob.yhigh);
        tempblob->perimeter += newblob.perimeter;
        found = TRUE;
      }
  if (tempblob->yhigh < bottomlimit)
    if (findblob ((tempblob->xlow + tempblob->xhigh) / 2,
                  (tempblob->xlow + tempblob->xhigh) / 2,
                  tempblob->yhigh, bottomlimit, &newblob))
      if (newblob.ylow > toplimit - 10 &&
          newblob.yhigh < bottomlimit + 10 &&
          newblob.xlow != last_blob.xlow) {     /* could be any of the four! */
```

```c
        tempblob->xlow = min (tempblob->xlow, newblob.xlow);
        tempblob->ylow = min (tempblob->ylow, newblob.ylow);
        tempblob->xhigh = max (tempblob->xhigh, newblob.xhigh);
        tempblob->yhigh = max (tempblob->yhigh, newblob.yhigh);
        tempblob->perimeter += newblob.perimeter;
        found = TRUE;
      }
  return (found);

}                                /* end find_sub_blobs */

/* the 'segment_line' function finds a line within the coordinates      */
/* provided by its parameters using findblob and find_sub_blobs and     */
/* proceeds to find blobs along that line. Each blob is added to a      */
/* global list. This list is later read by findsplit.    */

int segment_line (int startx, int stopx, int starty, int stopy)
{

  blobtype blob, tempblob, lastblob;
  static int left = TRUE;
  int dot;

  /*    if(no_of_templates == 0){
      printf("No templates stored.\n");
      return(FALSE);
    }*/
  lastblob.xlow = lastblob.xhigh = lastblob.ylow = lastblob.yhigh = -1;
  if (startx < c_line)
    left = TRUE;
  else
    left = FALSE;

  no_of_blobs = 0;
  /* scroll_on = TRUE;*/
  if (c_line == 0)
    find_cline ();
  firstblob.xlow = SPOTWIDTH;
  if (!(found = findblob (startx, stopx, starty, stopy, &blob)))
    return (FALSE);
  /* changed from findblob to find_next_entry 27/2/91 GPA */
  /* change back 1/3/91 */
  /*if( !(found = find_next_entry(starty,startx,stopx,&blob)))
      return(FALSE);*/
  toplimit = blob.ylow;
  /*bottomlimit = blob.yhigh;*/
  bottomlimit = min (blob.yhigh, blob.ylow + LINE_HEIGHT);
  firstblob = blob;
  while (found) {
    /*rectangle(blob.xlow-origx,blob.ylow-origy,
      blob.xhigh-origx,blob.yhigh-origy);
      cga_update();*/
    if (s_blobs)
      dot = find_sub_blobs (&blob, lastblob);
    /* add blob to buffer */
    list_add (&bloblist, blob);
    /*      if(no_of_blobs < BLOBNUM)
         blobs[no_of_blobs++] = blob;
      else{
         printf("ERROR - blob array overflow.\n");
         found = FALSE;
         break;
      }*/
    if (bottomlimit - toplimit < LINE_HEIGHT) {
      if (blob.ylow < toplimit &&
          bottomlimit - blob.ylow <= LINE_HEIGHT)
        toplimit = blob.ylow;
      if (blob.yhigh > bottomlimit &&
          blob.yhigh - toplimit <= LINE_HEIGHT)
        bottomlimit = blob.yhigh;
    }
```



```c
    lastblob = blob;
    /*do*/
    found = findblob (blob.xhigh + 1, stopx, toplimit, bottomlimit, &tempblob);
    /*while(found && ((tempblob.ylow < toplimit-20)
                  ||  (tempblob.yhigh > bottomlimit+20)) );*/
    /* 20 changed to 10, 5/6/91 GA. and back again 6/6/91 */
    if (found)                    /* loop here */
      if ((tempblob.ylow < toplimit - 20))
        found = findblob (blob.xhigh + 1, stopx, tempblob.yhigh, bottomlimit,
                          &blob);
      else if ((tempblob.yhigh > bottomlimit + 20))
        found = findblob (blob.xhigh + 1, stopx, toplimit, tempblob.ylow,
                          &blob);
      else
        blob = tempblob;
  }                               /* end 'while found' */
  /*do
      found = findblob(blob.xhigh+1,stopx,tempblob.yhigh,bottomlimit,&blob);
   while(tempblob.ylow < toplimit-20) );*/
  /* now cycle through list of blobs and remove any blobs that should not */
  /* be on this line */

  blob_ptr = bloblist.next;
  while (blob_ptr != NULL) {
    /* if (bad blob on line above OR bad blob on line below) */
    if (((blob_ptr->blob.ylow < toplimit - 20) &&
         (blob_ptr->blob.yhigh < bottomlimit)) ||

        ((blob_ptr->blob.yhigh > bottomlimit + 20) &&
         (blob_ptr->blob.ylow > toplimit))) {

      printf ("+\n");
      if (traceflag) {
        highlight_blob (blob_ptr->blob);/* highlight blob to be removed */
        c = getch ();
        unhighlight_blob (blob_ptr->blob);
      }
      blob_ptr = list_remove (blob_ptr);
    } else
      blob_ptr = blob_ptr->next;
  }
  blob_ptr = bloblist.next;     /**********/
  return (TRUE);

}                               /* end segment_line */

/* the 'guessentry' function takes the number of an entry as a          */
/* parameter and then classifies that entry line by line. Lines within  */
/* the entry are segmented with segment_line. Characters within a line  */
/* are segemented and classified with findsplit. Classified characters  */
/* are passed on to the state machine via delimit_fields which          */
/* which state/field the program is in from the character      */

void guessentry (int e_num)
{

  int loop, num, set, finished = FALSE;
  char guess[5];
  blobtype blob;
  int firstchar;
  char error = FALSE;

  if (file_on)
    fprintf (outfile, " \n");   /* end of entry symbol */
  if (!segment_line (entries[e_num].x, entries[e_num].x + 1060,
                     entries[e_num].y, entries[e_num].y + 20)) {
    printf ("No line found.\n");
    return;
  }
  state = START;                /* start in first state :-) */
  while (!finished) {
```

```
    firstchar = TRUE;
    while (findsplit (&blob, &num, &set, &error)) {
      /* error added 12/3/91 */
      if (error != 0) {
        guess[0] = error, guess[1] = '\0';
        if (error == ' ')
          count_error_1++;
        else
          count_error_2++;
      } else
        strcpy (guess, alpha[set][num].stats.name);
      count_total += strlen (guess);
      /* this code turns an apostrophe into a comma if necessary */
      if (strcmp (guess, "'") == 0) {   /* if an apostrophe */
        if (blob.ylow > (bottomlimit + toplimit) / 2) {
          printf ("\7");          /* bell */
          strcpy (guess, ",");
        }
      } else
      /* bot. cgd. to (bot.+top./2) 6/5/92 GA */ if (strcmp (guess, ",") == 0)  /* if a co
mma */
        if (blob.yhigh < (bottomlimit + toplimit) / 2) {
          printf ("\7\7");        /* bell */
          strcpy (guess, "'");
        }
      /*if(state == FORMAT)
          printf("%s",guess);
        else*/
      delimit_fields (guess, set, num, &state, firstchar, &finished,
                      blob.xlow - entries[e_num].x);
      /*if(finished)
          break;          TEMP */
      /*          if(file_on)
          fprintf(outfile,"%s",guess);*/
      fflush (stdout);
      if (firstchar)
        firstchar = FALSE;
    }                             /* end 'while characters left on the line */
    list_delete (&bloblist);
    /*if(finished)
        break;    TEMP */
    /* if(firstblob.yhigh+40 > bottom_text || */


    if ((e_num + 1 < no_of_entries) &&
        ((firstblob.yhigh + 50) > entries[e_num + 1].y) &&
        (entries[e_num].y < entries[e_num + 1].y)) {
      finished = TRUE;
      delimit_fields (guess, set, num, &state, firstchar, &finished,
                      blob.xlow - entries[e_num].x);
      printf ("\nEnd because of collision.\n");
    } else if (!segment_line (entries[e_num].x, entries[e_num].x + 1060,
                              firstblob.yhigh + 20, firstblob.yhigh + 35)) {
      printf ("No line found.\n");
      finished = TRUE;
      delimit_fields (guess, set, num, &state, firstchar, &finished,
                      blob.xlow - entries[e_num].x);
      printf ("\nEnd because of blank line.\n");
    }
  }                               /* end 'while not finished' */
  if (file_on)
    fprintf (outfile, "\n");
  printf ("End of entry.\n");
  if (file_on)
    fprintf (outfile, "\n");
  /* adding code to read shelf number in reverse here! */

}                                 /* end guessentry */

/* the 'mark_entries' function scans the page and fills the entries     */
/* array with the start coordinates of each entry on the page. The      */
```



```
/* first character in each entry is found with the find_next_entry      */
/* function.    */

void mark_entries (void)
{

  int found;

  if (c_line == 0)
    find_cline ();
  no_of_entries = 0;
  state = SUR;
  if (scroll_on)
    showscreen (0, 0);             /* start at beginning! */
  /* found = findblob(0,SPOTWIDTH-1,0,SPOTLENGTH-1);*/
  found = find_next_entry (0, left_text, left_text + 25, &tempblob);
  startright = tempblob.xlow + 1100;
  starttop = tempblob.ylow;
  /* startright is the estimated x coordinate of */
  /* the start of text on the right hand side of */
  /* the page */
  while (found) {
    guess_char (&blob, guess);
    strcpy (entries[no_of_entries].surname, guess);
    firstblob = blob;
    printf ("%s ", guess);
    fflush (stdout);
    entries[no_of_entries].x = blob.xlow;        /* record location */
    entries[no_of_entries].y = blob.ylow;
    /* if its a hyphen (22/2/91) */
    /*if(!strcmp(guess,"_"))
        entries[no_of_entries].y -= 20;*/
    no_of_entries++;
    found = find_next_entry (firstblob.yhigh, firstblob.xlow,
                             firstblob.xlow + 15, &tempblob);
  }                          /* end 'while found' */
  printf ("\n");
  bottom_text = entries[0].y + 3900;
  printf ("Bottom text at %d.\n", bottom_text);
  printf ("%d entries found.\n", no_of_entries);
  if (file_on)
    fprintf (outfile, "%d entries found.\n\n", no_of_entries);

}                            /* end mark_entries */

/* the 'guesspage' function classifies the entire page, one entry at a  */
/* time and writes the output to screen and, if required, printer. The  */
/* beginning of each entry is marked with the mark_entry function. The  */
/* page header is first found with find_header. The function then loops */
/* through each entry and calls guessentry to class each entry.         */

void guesspage (void)
{

  char c, common[NAMESIZE];
  int loop;
  int w, x, y, z, count1 = 0, count2 = 0;

  /*   if(no_of_templates == 0){
     printf("No templates stored.\n");
     return;
   }*/

  if (file_on)
    fprintf (outfile, "Page %s\n\n", spotfilename);
  if (no_of_entries == 0)
    mark_entries ();
  find_header (&w, &x, &y, &z);
  /* now deduce KNOWN element of surname from header */
  for (loop = 0; loop < strlen (right_header); loop++)
    if (left_header[loop] != right_header[loop])
```

```
    break;
  strncpy (known, right_header, loop);
  known[loop] = '\0';
  strcpy (prev_surname, known);
  printf ("KNOWN element of surname is(from header): %s\n", known);
  first_and_last (common);
  printf ("KNOWN element of surname is(from first & last): %s\n", common);
  printf ("Scroll on? (y/n)\n");
  scroll_on = (getch () == 'y');
  count_error_1 = count_error_2 = count_total = 0;
  for (loop = 0; loop < no_of_entries; loop++)
    guessentry (loop);
  printf ("No. of   errors (size error) : %d\n", count_error_1);
  printf ("No. of   errors (below threshold for recognition) : %d\n",
              count_error_2);
  printf ("Total no. of characters classified : %d\n", count_total);
  printf ("Recognition rate : %.2f%%\n",
              100 - 100.0 * (count_error_1 + count_error_2) / count_total);
  if (file_on) {
    fprintf (outfile, " \n");   /* terminate symbol */
    fprintf (outfile, "No. of   errors (size error) : %d\n", count_error_1);
    fprintf (outfile, "No. of   errors (below threshold for recognition) :
 %d\n", count_error_2);
    fprintf (outfile, "Total no. of characters classified : %d\n",
              count_total);
    fprintf (outfile, "Recognition rate : %.2f%%\n",
              100 - 100.0 * (count_error_1 + count_error_2) / count_total);
    fflush (outfile);
  }
}                                 /* end guesspage */

/* The 'delimit_fields' function is composed of a large switch          */
/* statement which switch condition is the most recently classified     */
/* character, passed in as a parameter. The switch statement implements */
/* a state machine. The machine changes state depending on the current  */
/* character and current state. */

/* state 0 : start, 1 : surname, 2 : firstname, 3 : alt, 4 : text */
/* 5 : qual */

int delimit_fields (char guess[], int set, int num, int *state, int newline,
                     int *end, int pos)
{

  char temp[5];
  static int bracket = 0;
  int count, count2, mnm;

  if (!strcmp (guess, "[") || !strcmp (guess, "("))
    bracket++;
  else if (!strcmp (guess, "]") || !strcmp (guess, ")"))
    bracket--;

  switch (*state) {
  case START:
    if (!strcmp (guess, "_")) {
      *state = TEXT;
      /* *end = TRUE;     TEMP */
    } else
      *state = SUR;
    strng[0] = '\0';
    break;
  case SUR:
    if (!strcmp (guess, ",") || !strcmp (guess, "(") ||
        !strcmp (guess, "_") || !strcmp (guess, "[")) {
      if (!strcmp (guess, ",") || !strcmp (guess, "[")) {
        *state = ALT;
        /* *end = TRUE;     TEMP */
      } else if (!strcmp (guess, "(")) {
        *state = FIRST;
        /* *end = TRUE;  TEMP */
```



```
      } else if (!strcmp (guess, "_")) {
        *state = TEXT;
        /* *end = TRUE;     TEMP */
      }
      printf ("SUR : %s\n", strng);
      /*   is ascii 240 */
      if (file_on)
        fprintf (outfile, " S %s\n", strng);
      for (count = 0; count < strlen (known); count++)
        strng[count] = known[count];
      /*mnm = min(strlen(prev_surname),strlen(strng));
                                    for(count2 = count;count2 < mnm;count2++){
                                      if(strng[count2] > prev_surname[count2])
                                         break;
                                      if(strng[count2] < prev_surname[count2])
                                         strng[count2] = prev_surname[count2];
                                    }*/
      strcpy (prev_surname, strng);
      strng[0] = '\0';
    }
    break;
  case FIRST:
    if (!strcmp (guess, ")") && bracket == 0)
      *state = FIR_QUAL;
    else if (!strcmp (guess, "_")) {
      *state = TEXT;
      printf ("FIRST : %s\n", strng);
      if (file_on)
        fprintf (outfile, " F %s\n", strng);
      strng[0] = '\0';
    }
    break;
  case FIR_QUAL:
    printf ("FIRST : %s\n", strng);
    if (file_on)
      fprintf (outfile, " F %s\n", strng);
    strng[0] = '\0';
    if (!strcmp (guess, "_"))
      *state = TEXT;
    else
      *state = QUAL;
    break;
  case ALT:
    if (!strcmp (guess, "(")) {
      *state = FIRST;
      printf ("ALT : %s\n", strng);
      if (file_on)
        fprintf (outfile, " A %s\n", strng);
      strng[0] = '\0';
    }
    if (!strcmp (guess, "_")) {
      *state = TEXT;
      printf ("ALT : %s\n", strng);
      if (file_on)
        fprintf (outfile, " A %s\n", strng);
      strng[0] = '\0';
    }
    break;
  case TEXT:
    if (newline)
      if (alpha[set][num].stats.font == 'I') {
        *state = LOC;
        /* now turn off s_blobs because of italics */
        s_blobs = FALSE;
        printf ("TEXT : %s\n", strng);
        if (file_on)
          fprintf (outfile, " T %s\n", strng);
        strng[0] = '\0';
      } else if (pos > 500) {
        *state = SHELF;
        printf ("TEXT : %s\n", strng);
```

```
      if (file_on)
        fprintf (outfile, " T %s\n", strng);
      strng[0] = '\0';
    } else {
      strcpy (temp, "\n");
      strcpy (guess, strcat (temp, guess));
  } else
  /************/ if (space.character && (space.size > 7 * SPACE)) {
    *state = SHELF;
    printf ("TEXT : %s\n", strng);
    if (file_on)
      fprintf (outfile, " T %s\n", strng);
    strng[0] = '\0';
  } else if (*end) {
    bracket = 0;
    printf ("Text : %s\n", strng);
    if (file_on)
      fprintf (outfile, " T %s\n", strng);
    strng[0] = '\0';
  }
  break;
case QUAL:
  if (!strcmp (guess, "_")) {
    *state = TEXT;
    printf ("QUAL : %s\n", strng);
    if (file_on)
      fprintf (outfile, " Q %s\n", strng);
    strng[0] = '\0';
  }
  break;
case LOC:                       /* if it's a digit or a left square bracket */
  if (set == 5 || !strcmp (guess, "[")) {
    *state = DATE;
    printf ("LOC : %s\n", strng);
    if (file_on)
      fprintf (outfile, " L %s\n", strng);
    strng[0] = '\0';
    /* turn on s_blobs cause no more italics */
    s_blobs = TRUE;
  }
  break;
case DATE:
  if (!strcmp (guess, ".") ||
      !strcmp (guess, "]")) {
    *state = FORMAT;
    strcat (strng, guess);
    printf ("DATE : %s\n", strng);
    if (file_on)
      fprintf (outfile, " D %s\n", strng);
    strng[0] = '\0';
    return (0);
  } else
  /* change o into 0 if necessary */ if (strcmp (guess, "o") == 0)
    strcpy (guess, "0");
  break;
case FORMAT:
  if (!strcmp (guess, ".")) {
    *state = SHELF;
    strcat (strng, guess);
    printf ("FORMAT : %s\n", strng);
    if (file_on)
      fprintf (outfile, " f %s\n", strng);
    strng[0] = '\0';
    return (0);
  }
  break;
case SHELF:
  if (*end) {
    bracket = 0;
    printf ("SHELF : %s\n", strng);
    if (file_on)
```



```
        fprintf (outfile, " S %s\n", strng);
      strng[0] = '\0';
    } else if (newline)
      if (alpha[set][num].stats.font == 'I') {
        *state = LOC;
        /* now turn off s_blobs because of italics */
        s_blobs = FALSE;
        printf ("SHELF : %s\n", strng);
        if (file_on)
          fprintf (outfile, " S %s\n", strng);
        strng[0] = '\0';
      } else if (pos > 500) {
        *state = SHELF;           /* new shelf mark */
        printf ("SHELF : %s\n", strng);
        if (file_on)
          fprintf (outfile, " S %s\n", strng);
        strng[0] = '\0';
      } else {
        *state = FORMAT;
        printf ("SHELF : %s\n", strng);
        if (file_on)
          fprintf (outfile, " S %s\n", strng);
        strng[0] = '\0';
      }
    break;
  }                               /* end switch state */
  if (space.character)            /* if there is a space between words */
    strcat (strng, " ");
  strcat (strng, guess);
  return (0);

}                                 /* end delimit_fields */

/* the 'check_space' function takes in two blobs as parameters and      */
/* compares the space between them with the SPACE constant. a boolean   */
/* is returned */

void check_space (blobtype blobb, blobtype bloba,
                  struct space_structure *space)
{

  space->size = blobb.xlow - bloba.xhigh;
  space->character = (space->size > SPACE);

}                                 /* end checkspace */

/* the 'findsplit' function works similarly to the 'old_findsplit'      */
/* function in that it works with a buffer of blobs. unlike             */
/* 'old_findsplit', the buffer has been filled, BEFORE the function is  */
/* called. the 'segment line' function fills a global buffer with a     */
/* line full of blobs and 'findsplit' works through this buffer,        */
/* checking for splits and returning the (possibly) combined blob and   */
/* the guess for that blob */

int findsplit (blobtype * blob, int *num, int *set, char *nomatch)
{

  blobtype tempblob;
  static int guessed = FALSE;
  static int score[3];
  static int nm[3];
  static int s_t[3];
  static char error[3];
  int count, dot;

  /*   if(bufstart == bufend)
      return(FALSE);      no blobs in buffer */
  if (blob_ptr == NULL)
    return (FALSE);
  /* detect space between characters */
  /*   if(bufstart != 0)
```

```
    check_space(blobs[bufstart],blobs[bufstart-1],&space);
  else{
    space.character = FALSE;
    guessed = FALSE;
  }*/
  if (blob_ptr->prev->prev != NULL)
    check_space (blob_ptr->blob, blob_ptr->prev->blob, &space);
  else {
    space.character = FALSE;     /* first blob so no space before it */
    guessed = FALSE;
  }
  if (!guessed) {
    guessletter (&(blob_ptr->blob), &score[0], &nm[0], &s_t[0], &error[0]);
  }
  /*   if(bufend-bufstart == 1){          if only 1 blob in buffer */
  if (blob_ptr->next == NULL) {
    *num = nm[0];
    *set = s_t[0];
    *nomatch = error[0];
    /* *blob = blobs[bufstart];
      bufstart++;*/
    *blob = blob_ptr->blob;
    blob_ptr = blob_ptr->next;   /* move on to next blob */
    guessed = FALSE;
    return (TRUE);
  }
  /* now check gap between blobs and overall width of combined blobs */
  /*   if( (blobs[bufstart+1].xlow-blobs[bufstart].xhigh < 5) &&
        (blobs[bufstart+1].xhigh-blobs[bufstart].xlow <= TEMPLATESIZE)){*/
  if ((blob_ptr->next->blob.xlow - blob_ptr->blob.xhigh < 5) &&
      (blob_ptr->next->blob.xhigh - blob_ptr->blob.xlow <= TEMPLATESIZE)) {
    guessletter (&(blob_ptr->next->blob), &score[1],
                 &nm[1], &s_t[1], &error[1]);

    tempblob.ylow = min (blob_ptr->blob.ylow, blob_ptr->next->blob.ylow);
    tempblob.yhigh = max (blob_ptr->blob.yhigh, blob_ptr->next->blob.yhigh);
    tempblob.xlow = min (blob_ptr->blob.xlow, blob_ptr->next->blob.xlow);
    tempblob.xhigh = max (blob_ptr->blob.xhigh, blob_ptr->next->blob.xhigh);
    tempblob.perimeter = blob_ptr->blob.perimeter +
      blob_ptr->next->blob.perimeter;   /* approx! */
    guessletter (&tempblob, &score[2], &nm[2], &s_t[2], &error[2]);
    if (score[2] > ((score[0] + score[1]) / 2)) {
      *num = nm[2];
      *set = s_t[2];
      *nomatch = error[2];
      *blob = tempblob;
      /*bufstart += 2;*/
      blob_ptr = blob_ptr->next->next;  /* move forward two blobs */
      guessed = FALSE;
    } else {
      *num = nm[0];
      *set = s_t[0];
      *nomatch = error[0];
      /*          *blob = blobs[bufstart];
         bufstart++;*/
      *blob = blob_ptr->blob;
      blob_ptr = blob_ptr->next;
      guessed = TRUE;
      score[0] = score[1];
      nm[0] = nm[1];
      s_t[0] = s_t[1];
      error[0] = error[1];
    }
  }
  /* end 'if with gap and with conditions */
  else {
    *num = nm[0];
    *set = s_t[0];
    *nomatch = error[0];
    /*      *blob = blobs[bufstart];
      bufstart++;*/
```



```
    *blob = blob_ptr->blob;
    blob_ptr = blob_ptr->next;
    guessed = FALSE;
  }
  return (TRUE);

}                                /* end findsplit */

/* the 'guess_char' function segments and guesses a single character.   */
/* the character is found by the findblob function and classified by    */
/* the findsplit function. TRUE is returned if a character has been      */
/* found, else FALSE.   */

int guess_char (blobtype * blob, char guess[])
{

  int num, set;
  char error = 0;

  firstchar = TRUE;
  toplimit = SPOTLENGTH;
  bottomlimit = -1;

  /*   no_of_blobs = 0;
   blobs[no_of_blobs++] = tempblob;*/
  list_add (&bloblist, tempblob);
  if (findblob (tempblob.xhigh + 1, tempblob.xhigh + 20,
                tempblob.ylow, tempblob.yhigh, &tempblob))
    /*blobs[no_of_blobs++] = tempblob;*/
    list_add (&bloblist, tempblob);
  /*bufstart = 0;bufend = no_of_blobs;*/
  blob_ptr = bloblist.next;
  findsplit (blob, &num, &set, &error);
  list_delete (&bloblist);
  if (error != 0)
    guess[0] = error, guess[1] = '\0';
  else
    strcpy (guess, alpha[set][num].stats.name);
  return (found);

}                                /* end guess_char */

/* the 'first_and_last' function finds the surname of the first entry   */
/* on the page and the surname of the last entry on the page and finds  */
/* what is common to both. findblob and guessauthor are used to find    */
/* the names. */

void first_and_last (char known[])
{

  int count, found = FALSE;
  char fname[NAMESIZE], lname[NAMESIZE], dummy[NAMESIZE];

  if (no_of_entries == 0) {
    printf ("No entries found yet.\n");
    return;
  }
  state = SUR;                   /* use only templates for surname */
  /* now find last entry from bottom */
  for (count = no_of_entries - 1; count >= 0; count--) {
    printf ("Count: %d.\n", count);
    if (strcmp (entries[count].surname, "_")) { /* if its not a hyphen */
      found = findblob (entries[count].x, entries[count].x + 20,
                        entries[count].y, entries[count].y + 20, &tempblob);
      if (found)
        guessauthor (lname, dummy);
      break;
    }
  }
  /* first find first entry from top */
  for (count = 0; count < no_of_entries; count++)
```

```
    if (strcmp (entries[count].surname, "_")) { /* if its not a hyphen */
      found = findblob (entries[count].x, entries[count].x + 20,
                        entries[count].y, entries[count].y + 20, &tempblob);
      if (found)
        guessauthor (fname, dummy);
      break;
    }
  /* now deduce common element */

  for (count = 0; count < strlen (fname); count++)
    if (lname[count] != fname[count])
      break;
  strncpy (known, lname, count);
  known[count] = '\0';
  printf ("First: %s\n", fname);
  if (file_on)
    fprintf (outfile, "First: %s\n", fname);
  printf ("Last: %s\n", lname);
  if (file_on)
    fprintf (outfile, "Last: %s\n", lname);
  printf ("String common to first & last is %s\n", known);
  if (file_on)
    fprintf (outfile, "known: %s\n", known);

}                              /* end first & last */
```

```
/* tocr_an2.* created 23/10/90 GPA */

void set_lut (void);
unsigned char countbits (unsigned char no);
int findblob (int startx, int stopx, int starty, int stopy,
                blobtype * tempblob);
int trace_perim (int xstart, int ystart, blobtype * tempblob);
int scanone (int dir, int xtemp, int ytemp);
void go (int dir, int *x, int *y);
void checklimits (int x, int y, blobtype * tempblob);

/* guessletter altered 16/10/90, 16/1/91,12/3/91 GPA */
char *guessletter (blobtype * blob, long *maxscore, int *num, int *set,
                        char *error);
void shift_match (blobtype blob);
```

```
/* tocr_an2.c */

/* This module contains low level character recognition functions. These  */
/* functions are called by higher level OCR functions found in tocr_an1.c */

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>                /* used for finding slope with atan */

#include "tocr_defs.h"
#include "tocr_vars.h"
#include "graphics.h"
#include "tocr_io.h"
#include "tocr_an1.h"
#include "tocr_an2.h"
#include "tocr_util.h"

unsigned char lut[256] =
{0};                            /*  for decalibration  */

static int chain[4] =
{0, 0, 0, 0};                   /* used for calculating area */
static int count[4] =
{0, 0, 0, 0};

/* the 'countbits' function is called by the set_lut function.  */
/* countbits returns the number of bits in a byte as an unsigned char   */

unsigned char countbits (unsigned char no)
{

  unsigned char i, count, bit;

  count = 0;
  bit = 1;
  for (i = 0; i < 8; i++, bit += bit)
    if (no & bit)
      count++;
  return count;

}                               /* end countbits */

/* the 'set_lut' function is used to set up a look_up table which is    */
/* used to speed up the guessletter function. There are 256 entries in  */
/* the look_up table. Each entry stores the number of bits of the index */
/* number for that entry. Thus entry 255 is 8 (bits), entry 0 is 0,     */
/* entry 24 is 2 (bits) */

void set_lut (void)
{

  unsigned char i;

  i = 0;
  do
    lut[i] = countbits (i);
  while (i++ != 255);

}                               /* end set_lut */

/* The 'findblob' function scans a portion of the screen, denoted by    */
/* its four parameters, until it finds a pixel. On finding a pixel,     */
/* findblob does a perimiter trace on the potential blob. If the blob's */
/* perimeter is less than 'NOISE', it is presumed to be noise and       */
/* ignored. If no substantial blob is found on the screen FALSE is      */
/* returned else TRUE */

int findblob (int startx, int stopx, int starty, int stopy,
                blobtype * tempblob)
```



```
{

  int x = startx, y = starty, found = FALSE;
  int col, tempperim;

  while (x <= stopx && !found) {
    /* must check x is on screen and page */
    if (x >= SPOTWIDTH) {
      found = FALSE;
      break;
    }
    if (scroll_on && (x - origx > XMAX))
      showscreen (origx + XMAX / 2, origy);
    /* end of validating x */
    y = starty;
    while (y <= stopy && !found) {
      if (scroll_on && traceflag) {
        /* if user has set the traceflag then */
        /* show character being scanned */
        col = gethirespixel (x - origx, y - origy);
        putdirectpixel (x - origx, y - origy, !col);/* 'direct' for speed */
        /*for(loopcount=0;loopcount<3000;loopcount++);  pause */
        putdirectpixel (x - origx, y - origy, col);
      }
      if (x - firstblob.xlow > 1060) {
        found = -1;
        break;                  /* don't hit the center line */
      }
      /*if (gethirespixel(x,y)){*/
      /*if (page[y][x/8] & mask[x % 8]){*/
      if (page[y][x >> 3] & mask[x & 7]) {
        /*if(getbit(x,y)){*/
        if ((tempperim = trace_perim (x, y, tempblob)) > NOISE)
          found = TRUE;
        else if (tempperim == -1) {
          found = -1;
          break;
        }
        y = starty;
        x = tempblob->xhigh;
      } else
        y += 2;                 /* sample every second pixel */
    }                           /* end while */
    if (found == -1) {
      found = FALSE;
      break;
    } else
      x += 2;                   /* sample every second pixel */
  }                             /* end while */
  return (found);

}                               /* end findblob */

/* The 'scanone' procedure tests one pixel in the 'dir' direction and   */
/* and returns TRUE if it is on or FALSE if it is off. The screen will  */
/* be scrolled if the coordinates (xtemp,ytemp) are off the screen and  */
/* scroll_on is TRUE.*/

int scanone (int dir, int xtemp, int ytemp)
{

  int blocked = FALSE;

  switch (dir) {
  case 0:
    ytemp++;
    if (ytemp >= SPOTLENGTH) {
      blocked = TRUE;
      break;
    }
    if (scroll_on && (ytemp - origy > YMAX))
```

```
      showscreen (origx, origy + YMAX / 2);
      break;
    case 1:
      xtemp++;
      if (xtemp >= SPOTWIDTH) {
        blocked = TRUE;
        break;
      }
      if (scroll_on && (xtemp - origx > XMAX))
        showscreen (origx + XMAX / 2, origy);
      break;
    case 2:
      ytemp--;
      if (ytemp < 0) {
        blocked = TRUE;
        break;
      }
      if (scroll_on && (ytemp - origy < 0))
        showscreen (origx, origy - YMAX / 2);
      break;
    case 3:
      xtemp--;
      if (xtemp < 0) {
        blocked = TRUE;
        break;
      }
      if (scroll_on && (xtemp - origx < 0))
        showscreen (origx - XMAX / 2, origy);
      break;
    default:
      printf ("Direction overflow in scanone.\n");
      exit (1);
    }                               /* end switch */
    if (blocked)
      return (-1);          /* return an error 'cause we're at edge of page */
    /* return( page[ytemp][xtemp/8] & mask[xtemp % 8] );*/
    return (page[ytemp][xtemp >> 3] & mask[xtemp & 7]);
    /*return( getbit(xtemp,ytemp) );*/

}                                   /* end scanone */

/* the 'go' procedure is used by the 'trace_perim' procedure to move     */
/* the coordinates (x,y) one pixel in the 'dir' direction */

void go (int dir, int *x, int *y)
{

  int col;

  /* left : 3, down : 0, right : 1, up : 2 */
  if (scroll_on && traceflag) {
    col = gethirespixel (*x - origx, *y - origy);
    putdirectpixel (*x - origx, *y - origy, !col);/* 'direct' for speed */
    /*   for(loopcount=0;loopcount<3000;loopcount++);*/
    putdirectpixel (*x - origx, *y - origy, col);
  }
  switch (dir) {
  case 0:
    (*y)++;
    break;
  case 1:
    (*x)++;
    break;
  case 2:
    (*y)--;
    break;
  case 3:
    (*x)--;
    break;
  }                                 /* end switch */
```



```
  /* now add to chain */
  chain[dir] += ((dir % 2) == 0) ? *x : *y;
  count[dir]++;

}                                   /* end go */

/* The 'checklimits' function is responsible for recording the bounding */
/* box of a character */

void checklimits (int x, int y, blobtype * tempblob)
{

  if (x < tempblob->xlow)
    tempblob->xlow = x;
  if (x > tempblob->xhigh)
    tempblob->xhigh = x;
  if (y < tempblob->ylow)
    tempblob->ylow = y;
  if (y > tempblob->yhigh)
    tempblob->yhigh = y;

}                                   /* end checklimits */

/* The 'guessletter' function compares the character in the current      */
/* bounding box, with its list of templates. The name of the best match */
/* is returned as the function, as well as the score for the template,  */
/* the number of the template, and the set from which the template      */
/* came. Initially, the character is copied from the screen into the    */
/* temporary character array. A switch statement then implements one of */
/* several alternative template scoring algorithms.     */

char *guessletter (blobtype * blob, long *maxscore, int *num,
                        int *st, char *error)
{

  int i, j, loop1, loop2, set, width, height, nomatch, area;
  register unsigned char *ptr, *ptt;
  static char dummy[5];

  /* int score;*/
  long score;
  char c;

  timestamp ("At start of guessletter", timer_now ());
  nomatch = TRUE;
  /* 13/3/91, error added. error indicates no match or low score */
  *error = 0;
  width = (blob->xhigh - blob->xlow) - 1;
  height = (blob->yhigh - blob->ylow) - 1;
  *maxscore = -10000;
  gap = 1;
  /* 'gap' can be incremented to increase the speed but will result in a */
  /* loss of accuracy */
  if ((height < 10) || (width < 10))
    gap = 1;/*    if width or height small then high gap could miss things */
  /* use pointers to optimize for speed */
  for (j = 0; j < TEMPLATESIZE; j += gap) {
    ptr = character[j];
    *ptr++ = 0;
    *ptr++ = 0;
    *ptr++ = 0;
    *ptr++ = 0;
    *ptr++ = 0;
    *ptr++ = 0;
  }
  area = 0;
  timestamp ("In guessletter before character copy", timer_now ());
  for (j = 0; j < height; j += gap)      /* this bit puts the character on */
    for (i = 0; i < width; i++) /* the page into a temporary       */
      if (j < TEMPLATESIZE && i < TEMPLATESIZE) /* character array in memory */
        /*if(gethirespixel(lowx+1+i,lowy+1+j)){*/
```

```
        if (page[blob->ylow + 1 + j][(blob->xlow + 1 + i) / 8] &
                mask[(blob->xlow + 1 + i) % 8]) {
          character[j][i >> 3] |= mask[i & 7];
          area++;
        }
    timestamp ("In guessletter after character copy", timer_now ());
    area *= gap;                    /* approx. calc. */
    /* NB >> 3 equals / 8, & 7 equals % 8 */
    timestamp ("guessletter before while nomatch", timer_now ());
    /*do{  do while no match */
    timestamp ("guessletter before template loop", timer_now ());
    for (loop1 = 0; loop1 < no_of_sets[state]; loop1++) {
      set = order[state][loop1];
      for (loop2 = 0; loop2 < no_of_tmplts[set]; loop2++) {
        if ((abs (alpha[set][loop2].stats.width - width) < max (5, width / 7)) &&
            (abs (alpha[set][loop2].stats.height - height) <
                max (5, height / 7))) {
          nomatch = FALSE;
          score = 0;
          timestamp ("guessletter before match loops", timer_now ());
          switch (method) {
          case 1:          /* 1pt for black pixels */
            /* this does the actual match */
            for (j = 0; j < TEMPLATESIZE; j += gap) {
              ptr = character[j];
              ptt = alpha[set][loop2].bitmap[j];
              for (i = 0; i < TEMPLATESIZE / 8; i++) {
                score += lut[*ptr & *ptt];
                ptr++;
                ptt++;
              }
            }
            scores[set][loop2].score = score;
            score = (score << 7) / area;
            break;
          case 2:          /* 1pt for black & white pixels */
            /* this does the actual match */
            for (j = 0; j < TEMPLATESIZE; j += gap) {
              ptr = character[j];
              ptt = alpha[set][loop2].bitmap[j];
              for (i = 0; i < TEMPLATESIZE / 8; i++)
                score += lut[*ptr++ ^ *ptt++];
            }
            score = TEMPLATESIZE * TEMPLATESIZE - score;
            scores[set][loop2].score = score;
            score = (score << 7) / (TEMPLATESIZE * TEMPLATESIZE);
            break;
          case 3:          /* 1 pt for black pixels, -1 for no match */
            /* this does the actual match */
            for (j = 0; j < TEMPLATESIZE; j += gap) {
              ptr = character[j];
              ptt = alpha[set][loop2].bitmap[j];
              for (i = 0; i < TEMPLATESIZE / 8; i++)
                score += lut[*ptr++ & *ptt++];
            }
            /* next line gives XOR */
            /*score = 3*score-alpha[set][loop2].stats.area-area;*/
            score = (score << 1) + score - alpha[set][loop2].stats.area - area;
            scores[set][loop2].score = score;
            score = (score << 7) / area;
            break;
          case 4:          /* 1pt for black & white pixels, -1 pt for no match */
            /* this does the actual match */
            for (j = 0; j < TEMPLATESIZE; j += gap) {
              ptr = character[j];
              ptt = alpha[set][loop2].bitmap[j];
              for (i = 0; i < TEMPLATESIZE / 8; i++)
                score += lut[*ptr++ ^ *ptt++];
            }
            score = TEMPLATESIZE * TEMPLATESIZE - 2 * score;
            scores[set][loop2].score = score;
```



```
            score = (score << 7) / (TEMPLATESIZE * TEMPLATESIZE);
            break;
          case 5:          /* just do XOR -: -1 pt for no match */
            /* this does the actual match */
            for (j = 0; j < TEMPLATESIZE; j += gap) {
              ptr = character[j];
              ptt = alpha[set][loop2].bitmap[j];
              for (i = 0; i < TEMPLATESIZE / 8; i++) {
                score -= lut[*ptr ^ *ptt];
                ptr++;
                ptt++;              /* note CAN NOT do this on above line! */
              }
            }
            scores[set][loop2].score = score;
            score = 128 - (-128 * score / (TEMPLATESIZE * TEMPLATESIZE));
            break;
          default:
            printf ("Method overflow in match switch.\n");
            exit (1);
          }                         /* end switch */

          timestamp ("guessletter after match loops", timer_now ());
          scores[set][loop2].percent = score;
          strcpy (scores[set][loop2].name, alpha[set][loop2].stats.name);
          scores[set][loop2].font = alpha[set][loop2].stats.font;
          if (score > *maxscore) {
            *maxscore = score;
            *num = loop2;           /* return the number of the template */
            *st = set;              /* return the set containing the template */
          }                         /* end if */
        }
        /* end 'if wrong size' */
        else {
          scores[set][loop2].score = -10000;/* indicate template not used */
          strcpy (scores[set][loop2].name, alpha[set][loop2].stats.name);
          scores[set][loop2].font = alpha[set][loop2].stats.font;
        }                           /* end else */
      }                             /* end for loop */
    }                               /* end for loop */
    blob->area = area;
    timestamp ("guessletter after template loop", timer_now ());
    timestamp ("guessletter after while nomatch", timer_now ());
    if (scroll_on && traceflag) {
      highlight_blob (*blob);
      printf ("Area: %d Per: %d Width: %d Height: %d\n", blob->area,
              blob->perimeter, width, height);
      printf ("Score: %d,char: %s,set: %d\n", *maxscore,
              alpha[*st][*num].stats.name, *st);
      c = getch ();
      unhighlight_blob (*blob);
      if (c == 's')                 /* store a character */
        store_template (*blob);
    }
    timestamp ("At end of guessletter", timer_now ());

    if (nomatch) {
      *error = ' ';
      return (" ");
    } else if (*maxscore < -100) {/* this figure is arbitrary */
      *error = ' ';
      return (" ");
    } else
      return (alpha[*st][*num].stats.name);

}                                   /* end guessletter */

/* the 'shift_match' function is a trial fuction to see what benefits   */
/* may be reaped from matching a blob in several 'shifted' positions    */
/* the blob specified by the parameter is matched in 8 extra positions, */
/* radiating from the center. Matching is achieved with the guessletter */
/* function.    */
```

```
/* 0 1 2 */
/* 3 * 4 */
/* 5 6 7 */

void shift_match (blobtype blob)
{

  blobtype tempblob;
  int loop, score, num, set;
  char guess[5], error;

  printf ("Trying shifted match...\n\n");
  for (loop = 0; loop < 9; loop++) {
    tempblob = blob;
    switch (loop) {
    case 0:
      tempblob.xlow--;
      tempblob.ylow--;
      break;
    case 1:
      tempblob.ylow--;
      break;
    case 2:
      tempblob.xlow++;
      tempblob.ylow--;
      break;
    case 3:
      tempblob.xlow--;
      break;
    case 4:
      break;                      /* don't need to change center */
    case 5:
      tempblob.xlow++;
      break;
    case 6:
      tempblob.xlow--;
      tempblob.ylow++;
      break;
    case 7:
      tempblob.ylow++;
      break;
    case 8:
      tempblob.xlow++;
      tempblob.ylow++;
      break;
    }                             /* end switch */
    guessletter (&tempblob, &score, &num, &set, &error);
    strcpy (guess, alpha[set][num].stats.name);
    printf ("At pos %d: %s with score %d (set %d)\n", loop, guess, score, set);
  }

}                                 /* end shift_match */


/* the 'trace_perim' function traces the perimeter of a blob starting   */
/* at point (xstart, ystart). It returns the perimeter length or -1 if  */
/* the blob goes off the edge of the page. (note use of &3 instead of   */
/* %4 for speed optimization). We initially go in the 'dir' direction   */
/* but try to change to the next direction, that is (dir+1) mod 4       */

int trace_perim (int xstart, int ystart, blobtype * tempblob)
{

  int perimeter, temp, blocked = FALSE, loop;

  /* we initially go down and try to change to right */
  /* left : 3, down : 0, right : 1, up : 2 */
  int dir = 0;                    /* ie down */
  int x, y;
```



```
  perimeter = 0;
  tempblob->xlow = 10000;
  tempblob->xhigh = -1;
  tempblob->ylow = 10000;
  tempblob->yhigh = -1;
  x = xstart;
  y = ystart;
  /* the next bit moves leftwards to the edge of the character if necessary */

  while ((page[y][x / 8] & mask[x % 8]) && !blocked)
    if (scanone (3, x, y) == -1)
      blocked = TRUE;
    else {
      xstart--;
      go (3, &x, &y);
    }
  /* zero chains */
  /* note chains used only to calculate area */
  for (loop = 0; loop < 4; loop++)
    chain[loop] = 0, count[loop] = 0;

  /* do until we've traced right around the letter */
  do {
    checklimits (x, y, tempblob);
    if ((temp = scanone ((dir + 1) & 3, x, y)) == -1) {
      blocked = TRUE;
      break;
    } else if (temp)
      if ((temp = scanone (dir, x, y)) == -1) {
        blocked = TRUE;
        break;
      } else {
        if (temp)
          if ((temp = scanone ((dir + 3) & 3, x, y)) == -1) {
            blocked = TRUE;
            break;
          } else if (temp)        /* reverse direction */
            dir = (dir + 2) & 3;
          else                    /* go back to previous direction */
            dir = (dir + 3) & 3;
    } else                        /* go in the direction we want to change to */
      dir = (dir + 1) & 3;        /* go to next direction */
    perimeter++;
    go (dir, &x, &y);
  }
  while ((x != xstart) || (y != ystart));
  tempblob->perimeter = perimeter - 4;

  if (blocked)
    return (-1);                  /* Return an error 'cause we're at edge */
  else {                          /* of screen. */
    if (traceflag && scroll_on) {
      printf ("Per. in trace is: %d\n", perimeter - 4);
      /*for(loop=0;loop<4;loop++)
          printf("Chain[%d]: %d, count[%d]: %d.\n",loop,chain[loop],loop,
                 count[loop]);
       printf("chain[0]-chain[2] = %d, chain[1]-chain[3] = %d\n",
              abs(chain[0]-chain[2]),abs(chain[1]-chain[3]));*/
      printf ("Area in trace is: %d.\n",
              abs (chain[0] - chain[2]) - count[0] - count[1] + 1);
    }
    return (perimeter - 4);/* Each corner erroneously adds 1 to perimeter */
  }

}                                 /* end trace_perim */
```

```
void setpageindex (void);
void pause (void);
int getdisk (void);
int getch (void);
void gwindow (int action,...);
void show_entries (void);
void show_scores (void);
void show_page (void);
void timestamp (char *comment, int time_now);
void show_times (void);
void mark_block (void);
void box (int left, int top, int right, int bottom);
void show_status (void);
void store_template (blobtype blob);     /* 18/7/1990 */

/* converts integer num to string (base 10 only) */
char *itoa (int num);           /* 18/sep/1990 Glynn Anderson */

/* reverses string str */
char *strrev (char *str);       /* 18/sep/1990 Glynn Anderson */

/* list functions added, 26,28/apr/1991 Glynn Anderson */
void list_add (struct list *head, blobtype blob);
struct list *list_remove (struct list *elem);   /* remove an element */
void list_delete (struct list *head);   /* delete an entire list */
void v_templates (void);        /* verify templates 4/dec/1991 GA */
void highlight_blob (blobtype blob);    /* 6/may/1992 GA */
void unhighlight_blob (blobtype blob);  /* 6/may/1992 GA */
```

```
/* tocr_ytil.c*/

/* This module contains utility and user interface functions. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <time.h>

#include "graphics.h"
#include "tocr_defs.h"
#include "tocr_vars.h"

#include "tocr_io.h"
#include "tocr_util.h"

#define MAX_STAMPS 100

typedef struct {
  char comment[NAMESIZE];
  int time;
  int diff;
}

t_entry;

static t_entry tstamps[MAX_STAMPS];
static no_of_stamps = 0;

/* the 'setpageindex' function is used to set up an index of pages that */
/* have already been classified.         */

void setpageindex (void)
{

  char surname[NAMESIZE];
  char firstname[NAMESIZE];
  int nochange = TRUE, found = FALSE, alter, loop1, loop2;

  entryfile = fopen (entryfilename, "rb");
  if (entryfile != NULL)
    printf ("This page has been scanned already!\n");
  fclose (entryfile);
  pageindexfile = fopen ("pageindex", "rb");
  alter = FALSE;
  if (pageindexfile != NULL) {
    if ((fread (&no_of_pages, sizeof (no_of_pages), 1, pageindexfile)) != 1) {
      perror ("Error reading no_of_pages.\n");
      exit (1);
    }
    if ((fread (pageindex, sizeof (pentry), no_of_pages, pageindexfile)) !=
        no_of_pages) {
      perror ("Error reading pages from page index.\n");
      exit (1);
    }
    fclose (pageindexfile);
    loop1 = 0;
    do {
      if (strncmp (pageindex[loop1].filename, spotfilename,
               strlen (spotfilename) - 4) == 0) {
        found = TRUE;
        printf ("There is already an entry in the pageindex for this page.\n");
        printf ("Do you want to change the entry? (y/n).\n");
        if (getch () == 'y') {
          alter = TRUE;
          for (loop2 = loop1 + 1; loop2 < no_of_pages; loop2++)
            pageindex[loop2 - 1] = pageindex[loop2];
          no_of_pages--;
        }
      }
```



```
        loop1++;
      }                           /* end do loop */
      while (!found && loop1 < no_of_pages);
      if (!found)
        alter = TRUE;
    }
    /* end then part */
    else
      alter = TRUE;
    fclose (pageindexfile);       /* close for reading */
    if (alter) {
      printf ("Please enter the first surname on the page.\n");
      scanf ("%s", surname);
      printf ("Please enter the corresponding firstname.\n");
      scanf ("%s", firstname);
      if (no_of_pages == 0) {
        printf ("Creating a page index on disc.\n");
        strcpy (pageindex[no_of_pages].surname, surname);
        strcpy (pageindex[no_of_pages].firstname, firstname);
        strncpy (pageindex[no_of_pages].filename, spotfilename,
                 strlen (spotfilename) - 4);
        pageindex[no_of_pages].filename[strlen (spotfilename) - 4] = '\0';
        no_of_pages++;
      } else {
        loop1 = 0;
        while (loop1 < no_of_pages && nochange) {
          if ((temp = strcmp (pageindex[loop1].surname, surname)) > 0)
            nochange = FALSE;
          else if (temp == 0)
            if (strcmp (pageindex[loop1].firstname, firstname) > 0)
              nochange = FALSE;
          if (!nochange) {
            for (loop2 = no_of_pages; loop2 > loop1; loop2--)
              pageindex[loop2] = pageindex[loop2 - 1];
            strcpy (pageindex[loop1].surname, surname);
            strcpy (pageindex[loop1].firstname, firstname);
            strncpy (pageindex[loop1].filename, spotfilename,
                 strlen (spotfilename) - 4);
            pageindex[loop1].filename[strlen (spotfilename) - 4] = '\0';
            no_of_pages++;
          }
          loop1++;
        }                         /* end while */
        if (nochange) {
          strcpy (pageindex[no_of_pages].surname, surname);
          strcpy (pageindex[no_of_pages].firstname, firstname);
          strncpy (pageindex[no_of_pages].filename, spotfilename,
                 strlen (spotfilename) - 4);
          pageindex[no_of_pages].filename[strlen (spotfilename) - 4] = '\0';
          no_of_pages++;
        }
      }                           /* else part */
      pageindexfile = fopen ("pageindex", "wb");
      if (pageindexfile == NULL) {
        perror ("Error 1 writing page index.\n");
        exit (1);
      }
      if ((fwrite (&no_of_pages, sizeof (no_of_pages), 1, pageindexfile)) != 1) {
        perror ("Error 2 writing page index.\n");
        exit (1);
      }
      if ((fwrite (pageindex, sizeof (pentry), no_of_pages, pageindexfile)) !=
          no_of_pages) {
        perror ("Error 3 writing page index.\n");
        exit (1);
      }
      fclose (pageindexfile);
    }                             /* end 'if alter' */
    /* for(loop1 = 0;loop1 < no_of_pages;loop1++)
     printf("sur: %s first: %s file: %s\n",pageindex[loop1].surname,
                 pageindex[loop1].firstname,pageindex[loop1].filename);
```

```
 pause();*/
}                               /* end setpageindex */

/* the 'pause' function pauses until a key of any kind is pressed        */

void pause (void)
{

  if (getch () == 0)
    getch ();                   /* handle double characters */

}                               /* end pause */

/* the 'getdisk' function uses a bios call to find out the number of     */
/* currently actice disk drive. */

int getdisk (void)
{

  return (bdos (0x19, 0, 0));
  /* call dos system call $19 to get drive number : 0:A, 1:B... */

}                               /* end getdisk */

/* the 'getch' function returns a single character, typed at the         */
/* keyboard. There is no echoeing and special keys, such as function     */
/* and cursor keys are returned as two characters, the first being 0     */

int getch (void)
{

  union REGS regs;
  static int c = 0;
  static int special = FALSE;

  if (special) {
    special = FALSE;
    return (c);                 /* return extended keyboard code */
  }
  regs.x.ax = 0x00;             /* request character to be returned */
  /* now call BIOS interrupt 0x16 to get a character from the keyboard */
  int86 (0x16, &regs, &regs);
  if (regs.h.al == 0x00) {      /* if extended keyboard code */
    special = TRUE;
    c = regs.h.ah;
  }
  return (regs.h.al);

}                               /* end getch */

/* the 'gwindow' function just mallocs a chunk of memory specified by   */
/* its parameters, draws a box and clears it and returns. The top       */
/* lefthand corner of the box is stored in the static variables x & y   */
/* and used when the window is closed    */

void gwindow (int action, int left, int top, int right, int bottom)
{

  static int x, y, open = FALSE;
  static void *wndw;

  if (action == OPEN) {
    if (open == TRUE) {
      printf ("Attempt to open an open window.\n");
      exit (1);
    }
    open = TRUE;
    size = imagesize (left, top, right, bottom);
    if ((wndw = (void *) malloc (size)) == NULL) {
      perror ("\7Window malloc failed\n");
      exit (1);
```



```
    }
    getimage (left, top, right, bottom, wndw);
    bar (left, top, right, bottom, 0);  /* rub out what's there */
    rectangle (left, top, right, bottom);
    x = left;
    y = top;
  } else {
    if (open == FALSE) {
      printf ("Attempt to close a non-open window.\n");
      exit (1);
    }
    open = FALSE;
    putimage (x, y, wndw, COPY_PUT);
    free (wndw);
  }
  cga_update ();

}                               /* end gwindow */

/* The 'show_entries' function just displays the entries that have been */
/* found in a scan */

void show_entries (void)
{

  int temp, count = 0;

  if (no_of_entries == 0)
    printf ("No entry names stored.\n");
  else {
    printf ("Names Stored (Time taken: %f seconds).\n",
                   (endp - startp) / TRANS_TICKS);
    for (loopcount = 0; loopcount < no_of_entries; loopcount++) {
      if (strcmp (entries[loopcount].surname, "-") == 0)
        count++;
      else {
        if (count != 0)
          printf ("(%d entries).\n", count + 1);
        printf ("%s\n", entries[loopcount].surname);
        printf ("%s\n", entries[loopcount].firstname);
        printf ("x: %d y: %d b_o: %d\n", entries[loopcount].x,
                entries[loopcount].y,
                entries[loopcount].byte_offset);
        count = 0;
      }
    }                           /* end for loop */
    if (count != 0)
      printf ("(%d entries)\n", count + 1);
  }                             /* end else */

}                               /* end show_entries */

/* The 'show_scores' function displays the scores for all templates for */
/* the current match */

void show_scores (void)
{

  int swap, loop1, loop2;
  scr tempscore;
  FILE *scorefile;

  if ((scorefile = fopen ("scores", "wt")) == NULL) {
    printf ("Unable to open socrefile.\n");
    exit (1);
  }
  for (loop1 = 0; loop1 < SETS; loop1++) {
    printf ("Scores (sorted by 'percent', actually, 'per128')\n");
    printf ("SET %d.\n", loop1);
    printf ("\nName Score Percentage Font\n");
```

```
    for (loop2 = 0; loop2 < no_of_tmplts[loop1]; loop2++) {
      if (scores[loop1][loop2].score == -10000)
        continue;
      printf ("%s %d %ld%% (%c)\n", scores[loop1][loop2].name,
              scores[loop1][loop2].score, scores[loop1][loop2].percent,
              scores[loop1][loop2].font);
    }
    printf ("Save this set to file?\n");
    if (getch () == 'y') {
      fprintf (scorefile, "Method : %d\n", method);
      for (loop2 = 0; loop2 < no_of_tmplts[loop1]; loop2++) {
        if (scores[loop1][loop2].score == -10000)
          continue;
        fprintf (scorefile, "%s ", scores[loop1][loop2].name);
      }
      fprintf (scorefile, "\n");
      for (loop2 = 0; loop2 < no_of_tmplts[loop1]; loop2++) {
        if (scores[loop1][loop2].score == -10000)
          continue;
        fprintf (scorefile, "%d ", scores[loop1][loop2].score);
      }
      fprintf (scorefile, "\n");
      for (loop2 = 0; loop2 < no_of_tmplts[loop1]; loop2++) {
        if (scores[loop1][loop2].score == -10000)
          continue;
        fprintf (scorefile, "%ld%% ", scores[loop1][loop2].percent);
      }
      fprintf (scorefile, "\n");
    }
  }
  fflush (stdin);
  fclose (scorefile);

}                                 /* end show_scores */

/* the 'show_page' function, scales the entire page onto the graphics    */
/* screen. No attempt is made to correct distortion. */

void show_page (void)
{

  double scalex, scaley;
  int loopx, loopy, xindex, yindex;

  if ((image = malloc (imagesize (0, 0, XMAX, YMAX))) == NULL) {
    printf ("Malloc error in show_page. Not enough memory.\n");
    exit (1);
  }
  getimage (0, 0, XMAX, YMAX, image);
  cleardevice ();
  scalex = (double) SPOTWIDTH / (XMAX + 1);
  scaley = (double) SPOTLENGTH / (YMAX + 1);
  for (loopy = 0; loopy <= YMAX; loopy++)
    for (loopx = 0; loopx <= XMAX; loopx++) {
      xindex = loopx * scalex;
      yindex = loopy * scaley;
      /*if( page[yindex][xindex/8] & mask[xindex%8])*/
      if (page[yindex][xindex >> 3] & mask[xindex & 7])
        /*if(getbit(xindex,yindex))*/
        puthirespixel (loopx, loopy, 1);
    }
  cga_update ();
  printf ("Hit any key to return...\n");
  getch ();
  putimage (0, 0, image, COPY_PUT);
  cga_update ();

}                                 /* end showpage */

/* the 'timestamp' function allows a time instance to be recorded along */
/* with a comment. this time and comment is added to an array which can */
```



```
/* be reviewed later. */

void timestamp (char *comment, int time_now)
{

  if (strlen (comment) > 39) {
    printf ("Error: Timestamp comment too large.\n");
    exit (1);
  }
  if (no_of_stamps < MAX_STAMPS) {
    strcpy (tstamps[no_of_stamps].comment, comment);
    tstamps[no_of_stamps].time = time_now;
    if (no_of_stamps != 0)
      tstamps[no_of_stamps].diff = time_now - tstamps[no_of_stamps - 1].time;
    else
      tstamps[no_of_stamps].diff = time_now;
    no_of_stamps++;
  }
}                                 /* end timestamp */

/* the 'show_times' function displays an array of timestamps. */

void show_times (void)
{

  int loopcount;

  for (loopcount = 0; loopcount < no_of_stamps; loopcount++)
    printf ("%6d %6.4f seconds %s\n", tstamps[loopcount].diff,
        tstamps[loopcount].diff / TRANS_TICKS, tstamps[loopcount].comment);

}                                 /* end show_times */

/* the 'mark_block' function allows the user to delimit an area of the  */
/* screen by means of drawing a rectangle around the block by means of  */
/* a rectangle. */

void mark_block (void)
{

  int x, y, temp, marking = FALSE, left, top, loopx, loopy;
  int finished = FALSE;
  char c, *image;

  x = 0;
  y = 0;
  temp = gethirespixel (x, y);
  putdirectpixel (x, y, 1);
  while (!finished) {
    c = getch ();
    if (c == 0) {                 /* special key */
      c = getch ();
      switch (c) {
      case RGT:
        if (x + 5 <= XMAX) {
          if (marking) {
            box (left, top, x, y);
            box (left, top, x + 5, y);
            cga_update ();
          } else {
            putdirectpixel (x, y, temp);
            temp = gethirespixel (x + 5, y);
            putdirectpixel (x + 5, y, 1);
          }
          x += 5;
        }
        break;
      case LFT:
        if (x - 5 >= 0) {
          if (marking) {
            box (left, top, x, y);
```

```
          box (left, top, x - 5, y);
          cga_update ();
        } else {
          putdirectpixel (x, y, temp);
          temp = gethirespixel (x - 5, y);
          putdirectpixel (x - 5, y, 1);
        }
        x -= 5;
      }
      break;
    case UP:
      if (y - 5 >= 0) {
        if (marking) {
          box (left, top, x, y);
          box (left, top, x, y - 5);
          cga_update ();
        } else {
          putdirectpixel (x, y, temp);
          temp = gethirespixel (x, y - 5);
          putdirectpixel (x, y - 5, 1);
        }
        y -= 5;
      }
      break;
    case DWN:
      if (y + 5 <= YMAX) {
        if (marking) {
          box (left, top, x, y);
          box (left, top, x, y + 5);
          cga_update ();
        } else {
          putdirectpixel (x, y, temp);
          temp = gethirespixel (x, y + 5);
          putdirectpixel (x, y + 5, 1);
        }
        y += 5;
      }
      break;
    }                              /* end switch */
  }
  /* end 'if special key' */
  else if (c == RET)
    if (marking) {
      finished = TRUE;
      if ((image = malloc (imagesize (left, top, x, y))) == NULL) {
        printf ("Error doing malloc in 'mark_block'\n");
        exit (1);
      }
      box (left, top, x, y);
      getimage (left, top, x, y, image);
      putimage (left, top, image, NOT_PUT);
      cga_update ();
      printf ("Delete block (d) or keep block and delete rest(r)?\n");
      if ((c = getch ()) == 'd') {
        printf ("Deleting marked block...\n");
        for (loopy = top; loopy <= y; loopy++)
          for (loopx = left; loopx <= x; loopx++)
            page[origy + loopy][(origx + loopx) / 8] &=
                      ~mask[(origx + loopx) % 8];
        putimage (left, top, image, AND_PUT);
      } else if (c == 'r') {
        timestamp ("Start of remove", timer_now ());
        printf ("Deleting all but marked block...\n");
        for (loopy = 0; loopy < SPOTLENGTH; loopy++)
          if (loopy < origy + top || loopy > origy + y)
            memset (page[loopy], 0, SPOTWIDTH / 8);
          else
            for (loopx = 0; loopx < SPOTWIDTH; loopx++)
              if (loopx < origx + left || loopx > origx + x)
                page[loopy][loopx / 8] &= ~mask[loopx % 8];
        cleardevice ();
```



```
          putimage (left, top, image, COPY_PUT);
          timestamp ("End of remove", timer_now ());
        }
        cga_update ();
        free (image);
      } else {
        marking = TRUE;
        putdirectpixel (x, y, temp);
        top = y;
        left = x;
        /* draw rectangle by 'NOT'ing pixels */
        box (left, top, x, y);
        cga_update ();
      }
  }                                /* end while */

}                                  /* end mark_block */

/* the 'box' function draws a box bounded by left,top,right,bottom by    */
/* reversing pixels on the screen. */

void box (int left, int top, int right, int bottom)
{

  int loop;

  for (loop = top; loop <= bottom; loop++) {
    puthirespixel (left, loop, !gethirespixel (left, loop));
    if (right != left)
      puthirespixel (right, loop, !gethirespixel (right, loop));
  }
  for (loop = left + 1; loop < right; loop++) {
    puthirespixel (loop, top, !gethirespixel (loop, top));
    if (bottom != top)
      puthirespixel (loop, bottom, !gethirespixel (loop, bottom));
  }

}                                  /* end 'box' */

/* the 'show_status' function displays the status of various system */
/* parameters, to the screen. */

void show_status (void)
{

  int count = 0, loop1, loop2;

  printf ("Status.\n");
  printf ("Filename: %s\n", spotfilename);
  printf ("Traceflag is %s.\n", traceflag ? "ON" : "OFF");
  printf ("Scrolling is %s.\n", scroll_on ? "ON" : "OFF");
  printf ("Printer is %s.\n", printer_on ? "ON" : "OFF");
  printf ("File output is %s.\n", file_on ? "ON" : "OFF");
  printf ("Threshold is %d.\n", thresh);
  printf ("Method is %d.\n", method);
  printf ("Sub blob scanning is %s,\n", s_blobs ? "ON" : "OFF");
  if (c_line == 0)
    printf ("Centre line not found yet.\n");
  else
    printf ("Centre line at %d\n", c_line);
  for (loop1 = 0; loop1 < SETS; loop1++) {
    printf ("SET %d.\n", loop1);
    printf ("There %s %d %s stored:\n",(no_of_tmplts[loop1] == 1)
              ? "is" : "are", no_of_tmplts[loop1], (no_of_tmplts[loop1] == 1)
              ? "template" : "templates");
    for (loop2 = 0; loop2 < no_of_tmplts[loop1]; loop2++) {
      printf ("%s ", alpha[loop1][loop2].stats.name);
      fflush (stdout);
    }
    printf ("\n");
  }
```

```c
  timestamp ("Before cline", timer_now ());
  find_cline ();                /***/
  timestamp ("After cline", timer_now ());
  printf ("Left text starts at %d\n", left_text);
  printf ("=> Right text at %d\n", left_text + 1100);

}                               /* show status */

/* 'strrev' created 18/sep/1990 Glynn Anderson for transputer. */
/* reverses a string. */

char *strrev (char *str)
{

  int count, length = strlen (str);
  char temp;

  for (count = 0; count < length / 2; count++) {
    temp = str[count];
    str[count] = str[length - 1 - count];
    str[length - 1 - count] = temp;
  }
  return (str);

}                               /* end strrev */

/* 'itoa' created 18/sep/1990 Glynn Anderson for transputer. */
/* changes an integer value into an ascii value. */

char *itoa (int num)
{

  static char string[20];
  int count;

  count = 0;
  while (num > 9) {
    string[count++] = (num % 10) + 48;
    num /= 10;
  }                             /* end while */
  string[count++] = num + 48;
  string[count] = '\0';
  strrev (string);
  return (string);

}                               /* end itoa */

/* the 'store_template' function stores a blob that is passed in as a    */
/* parameter. If there is no room for any more templates, a warning is   */
/* printed. */

void store_template (blobtype blob)
{

  char name[5], ch;
  int size, i, j, value, set, width, height;

  width = (blob.xhigh - blob.xlow) - 1;
  height = (blob.yhigh - blob.ylow) - 1;
  printf ("Store in which set? (0..%d)\n", SETS - 1);
  do {
    scanf ("%d", &set);
    if (set < 0 || set >= SETS)
      printf ("Illegal set no.\n");
  }
  while (set < 0 || set >= SETS);

  if (no_of_tmplts[set] == ALPHABETSIZE) {
    printf ("Template store %d is full at %d templates.\n", set, ALPHABETSIZE);
    printf ("To increase store size, change ALPHABETSIZE in the source");
    printf ("code.\n");
```



```c
  } else {
    printf ("Store template.\n");

    printf ("Store:");
    fflush (stdout);
    size = 0;
    while (size < 4 && ((ch = getch ()) != RET)) {
      name[size] = ch;
      printf ("%c", name[size]);
      fflush (stdout);
      size++;
    }                           /* end while */
    printf ("\n");
    name[size] = '\0';
    printf ("Enter the font type. (single character)\n");
    alpha[set][no_of_tmplts[set]].stats.font = getch ();
    printf ("Storing %s\n", name);
    for (j = 0; j < TEMPLATESIZE; j++)  /* clear template first */
      for (i = 0; i < TEMPLATESIZE / 8; i++)
        alpha[set][no_of_tmplts[set]].bitmap[j][i] = 0;
    for (j = 0; j < height; j++)
      for (i = 0; i < width; i++)
        if (j < TEMPLATESIZE && i < TEMPLATESIZE)
          if (page[blob.ylow + 1 + j][(blob.xlow + 1 + i) / 8] &
                mask[(blob.xlow + 1 + i) % 8])
            alpha[set][no_of_tmplts[set]].bitmap[j][i >> 3] |= mask[i & 7];
    strcpy (alpha[set][no_of_tmplts[set]].stats.name, name);
    alpha[set][no_of_tmplts[set]].stats.width = width;
    alpha[set][no_of_tmplts[set]].stats.height = height;
    alpha[set][no_of_tmplts[set]].stats.perimeter = blob.perimeter;
    alpha[set][no_of_tmplts[set]].stats.area = blob.area;
    no_of_tmplts[set]++;
  }                             /* end 'else' */

}                               /* end store_template */

/* the 'list_add' function adds an element to a list. it takes two      */
/* parameters, the element, and the head of the list */

void list_add (struct list *head, blobtype blob)
{

  struct list *temp;

  if (head->next == NULL) {
    if ((head->next = (struct list *) malloc (sizeof (struct list))) == NULL) {
      printf ("Error mallocing list element.\n");
      exit (1);
    }
    head->next->blob = blob;
    head->next->next = NULL;
    head->next->prev = head;
  } else
    list_add (head->next, blob);

}                               /* end list_add */

/* the 'list_remove' function takes one element, a sublist *elem, and    */
/* returns a new sublist with the first element removed. it is a FATAL   */
/* error to pass in a NULL list */

struct list *list_remove (struct list *elem)
{

  static struct list *temp;

  if (elem == NULL || elem->prev == NULL) {
    printf ("Attempt to delete from a NULL list.\n");
    exit (1);
  }
  elem->prev->next = elem->next;
```

```
  if (elem->next != NULL) {
    elem->next->prev = elem->prev;
  }
  temp = elem->prev;
  free (elem);
  return (temp);

}                                /* end *list_remove */

/* the 'list_delete' function deletes a list entirely, ie it removes      */
/* all the links and FREEs all memory used by the list */

void list_delete (struct list *head)
{

  struct list *temp, *list;

  list = head->next;
  head->next = NULL;
  while (list != NULL) {
    temp = list->next;
    free (list);
    list = temp;
  }

}                                /* end list_delete */

/* the 'v_templates' verifies that templates have been stored correctly */
/* This function was written when it was suspected that some of the     */
/* descriptive data stored with some of the templates, was incorrect.   */
/* 4/dec/1991 GA */

void v_templates (void)
{

  int set, loop, loop2, x, y, X, Y;
  int perim, area, width, height;
  blobtype blob;

  X = SPOTWIDTH / 2;
  Y = SPOTLENGTH / 2;

  for (set = 0; set < SETS; set++) {
    printf ("Set no. %d\n", set);
    for (loop = 0; loop < no_of_tmplts[set]; loop++) {
      printf ("No. %d\n", loop);
      area = perim = 0;
      /* clear page first */
      for (loop2 = 0; loop2 < SPOTLENGTH; loop2++)
        memset (page[loop2], 0, SPOTWIDTH / 8);
      for (y = 0; y < TEMPLATESIZE; y++)
        for (x = 0; x < TEMPLATESIZE; x++)
          if (alpha[set][loop].bitmap[y][x / 8] & mask[x % 8]) {
            page[Y + y][(X + x) / 8] |= mask[(X + x) % 8];
            area++;
          }
      firstblob.xlow = 10000;
      if (!findblob (X, X + 1, Y, Y + TEMPLATESIZE, &blob)) {
        printf ("Trace error on template %d, set %d: %s\n", loop, set,
                alpha[set][loop].stats.name);
        printf ("Skipping to next template.\n");
        continue;
      }
      width = blob.xhigh - blob.xlow - 1;
      height = blob.yhigh - blob.ylow - 1;
      if (area != alpha[set][loop].stats.area)
        printf ("Area mismatch %d %d\n", area, alpha[set][loop].stats.area);
      if (blob.perimeter != alpha[set][loop].stats.perimeter)
        printf ("Perimeter mismatch %d %d\n", blob.perimeter,
                alpha[set][loop].stats.perimeter);
      if (width != alpha[set][loop].stats.width)
```



```
        printf ("Width mismatch %d %d\n", width, alpha[set][loop].stats.width);
      if (height != alpha[set][loop].stats.height)
        printf ("Height mismatch %d %d\n", height,
                alpha[set][loop].stats.height);
    }
  }
}                                /* end v_templates */

/* the 'highlight_blob' function highlights a blob on the screen */

void highlight_blob (blobtype blob)
{

  if ((chunk = (void *) malloc (imagesize (blob.xlow, blob.ylow, blob.xhigh,
        blob.yhigh))) == NULL) {
    perror ("Malloc failed in 'highlight_blob'.");
    exit (1);
  }
  if ((blob.xlow - origx < 0) || (blob.xlow - origx > XMAX) ||
      (blob.ylow - origy < 0) || (blob.ylow - origy > YMAX))
    showscreen (blob.xlow, blob.ylow);
  getimage (blob.xlow - origx, blob.ylow - origy, blob.xhigh - origx,
            blob.yhigh - origy, chunk);
  rectangle (blob.xlow - origx, blob.ylow - origy, blob.xhigh - origx,
            blob.yhigh - origy);
  cga_update ();

}                                /* end highlight_blob */

/* the 'unhighlight_blob' function returns a 'highlighted' blob to      */
/* normal video */

void unhighlight_blob (blobtype blob)
{

  putimage (blob.xlow - origx, blob.ylow - origy, chunk, COPY_PUT);
  cga_update ();
  free (chunk);

}                                /* end unhihglight_blob */
```

```
void initialise (void);
short int getrun (int *run);
void validate (int *x, int *y);
void getpage (void);
void showscreen (int x, int y);
void getspotfile (void);
int load_entries (void);
void save_entries (void);
void save_page (int header);    /* altered 30/7/1990 Glynn Anderson */
int writerun (int run, FILE * imagefile);/* altered 21/9/90 Glynn Anderson */
void split_page (void);
int load_templates (char templatefilename[]);    /* altered 16/1/1991 */
int save_templates (char templatefilename[]);    /* altered 16/1/1991 */
void printer (void);
void print_tmplt (int set, int no);     /* altered 16/1/1991 */
void split_entries (void);
void display_entry (int num);
void display_tmplt (int n);
void full_save ();              /* 31/1/91 gpa */
void full_read ();              /* 31/1/91 gpa */
void file_out (void);           /* 11/6/91 gpa */
```

```
/* tocr_io.c */

/* This module contains input/output functions which are concerned with */
/* reading and writing pages to disk and decoding and encoding them,    */
/* displaying portions of the page on screen, loading and saving        */
/* templates and writing ASCII output to disk.                          */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#include "graphics.h"
#include "tocr_def.h"
#include "tocr_vars.h"
#include "tocr_an1.h"
#include "tocr_an2.h"
#include "tocr_io.h"
#include "tocr_util.h"

/* the 'initialise' function is the first called by the main function   */
/* it prints and introduction, sets up and clears some variables, and   */
/* loads the template file.     */

void initialise (void)
{

  int loopcount, loop;

  gographics ();                /* switch default adaptor to cga */
  video_mode (CGA_HIRES_GRAPHICS_MODE);
  gomono ();                    /* switch default adaptor to monochrome */

  printf ("Optical Character Recognition System.\n");
  printf ("For Trinity College Old Printed Library Catalogues.\n");
  printf ("Copyright 1989,1990,1991 Glynn Anderson.\n");
  printf ("Dept. of Computer Science.\n");
  printf ("Trinity College Dublin, Ireland.\n");
  printf ("\nInitilialising data structures...\n");
  /* now allocate memory for array of sets of templates */
  for (loopcount = 0; loopcount < SETS; loopcount++) {
    if ((alpha[loopcount] = (tmplt *) calloc (ALPHABETSIZE, sizeof (tmplt)))
        == NULL) {
      printf ("Fatal error: Alpha calloc failed at %d.\n", loopcount);
      exit (1);
    }
    no_of_tmplts[loopcount] = 0;
  }
  /* now allocate memory for a single catalogue page (about 1.2M) */
  /* and 'zero' it! */

  for (loopcount = 0; loopcount < SPOTLENGTH; loopcount++)
    if ((page[loopcount] = (char *) calloc (SPOTWIDTH / 8, 1)) == NULL) {
      printf ("Error callocing page memory at line %d.\n", loopcount);
      exit (1);
    }
  for (x = 0; x <= 7; x++)
    mask[x] |= 1 << (7 - x);
  /* set up look-up table for masking bits */
  for (loop = 0; loop < SETS; loop++)
    for (y = 0; y < TEMPLATESIZE; y++)  /* zero the template array */
      for (x = 0; x < TEMPLATESIZE / 8; x++)
        for (loopcount = 0; loopcount < ALPHABETSIZE; loopcount++) {
          alpha[loop][loopcount].bitmap[y][x] = 0;
        }
  set_lut ();                   /* set up the look up table for recognition */
  if ((image = (void *) malloc (imagesize (0, 0, XMAX, YMAX))) == NULL) {
    perror ("Error with image malloc.");
    exit (1);
  }
  printf ("\nLoading templates...\n");
```



```
  strcpy (templatefilename, "tmplts");
  if (load_templates (templatefilename) != 0)
    printf ("Templates not loaded successfully.\n");

}                               /* end initialise */

/* The 'getrun' procedure gets a run from the spotfile. If the msb of   */
/* the first byte read is 1 then a second byte is read. The count of    */
/* the number of bytes read is returned as the function. */

short int getrun (int *run)
{

  int count = 0;

  *run = getc (spotfile);
  count++;
  if (*run & 128) {             /* test ms bit */
    *run &= 127;                /* mask off ms bit */
    *run = (*run << 8) + getc (spotfile);       /* calculate actual run */
    count++;
  }
  return (count);

}                               /* end get run */

/* The 'validate' procedure checks that the coordinates x and y or on   */
/* the page and changes them if necessary. */

void validate (int *x, int *y)
{

  if (*x >= (SPOTWIDTH - XMAX)) /*\                             */
    *x = SPOTWIDTH - 1 - XMAX;  /* \                            */
  if (*x < 0)                   /*  \                           */
    *x = 0;                     /*   \ Make sure x and y are on screen */
  if (*y >= (SPOTLENGTH - YMAX))/*   /                          */
    *y = SPOTLENGTH - 1 - YMAX; /*  /                           */
  if (*y < 0)                   /* /                            */
    *y = 0;                     /*/                             */

}                               /* end validate */

/* the 'getpage' function loads a catalogue page from disk. Runs are    */
/* read from disk and decoded and bits are then written to the page     */
/* array. */

void getpage (void)
{

  int loopcount, loopx, loopy, width, white, run;
  unsigned char *line;

  /* clear page first */
  for (loopcount = 0; loopcount < SPOTLENGTH; loopcount++)
    memset (page[loopcount], 0, SPOTWIDTH / 8);
  /* now read in page */
  if (fseek (spotfile, SPOTHEADER, 0) != 0) {   /* hop over the header */
    perror ("\7Error doing fseek in getpage.\n");
    exit (1);
  }
  for (loopy = 0; loopy < SPOTLENGTH; loopy++) {
    width = 0;
    white = TRUE;
    line = page[loopy];         /**/
    do {                        /* while width != SPOTWIDTH */
      getrun (&run);
      if (!white)
        for (loopx = width; loopx < width + run; loopx++)
          /*page[loopy][loopx/8] |= mask[loopx % 8];*/
          /*page[loopy][loopx >> 3] |= mask[loopx & 7];*/
```

```
        /*page[loopy][loopx >> 3] |= 1 << (7- (loopx & 7));*/
        *(line + (loopx >> 3)) |= 1 << (7 - (loopx & 7));
      /* like this for speed */
      width += run;
      white = !white;
    }
  while (width != SPOTWIDTH);
  }

}                               /* end getpage */

/* the 'showscreen' function displays a section of the catalogue page   */
/* delimited by the coordinates px,py */

void showscreen (int px, int py)
{

  int sx, sy;

  validate (&px, &py);
  /* check that px and py are on the page. If they are not, alter them */
  if (px != origx || py != origy) {
    origx = px;
    origy = py;
    cleardevice ();
    for (sy = 0; sy <= YMAX; sy++)
      for (sx = 0; sx <= XMAX; sx++)
        /*if( page[py+sy][(px+sx)/8] & mask[(px+sx)%8])*/
        if (page[py + sy][(px + sx) >> 3] & (1 << 7 - ((px + sx) & 7)))
          /*if(getbit( px+sx,py+sy ))*/
          puthirespixel (sx, sy, 1);
    cga_update ();
  }
}                               /* end showscreen */

/* The 'getspotfile' procedure gets the name of the spot file from the  */
/* user and validates it. It also creates the entry file names from the */
/* spot file name. */

void getspotfile (void)
{

  int ok;

  fflush (stdin);
  printf ("Please enter the name of the spot file\n");
  do {
    fflush (stdin);
    ok = TRUE;
    scanf ("%s", spotfilename);
    /*    strlwr(spotfilename); convert to lower case */
    if (strcmp (&spotfilename[strlen (spotfilename) - 4], ".spt") != 0)
        /* if no .spt extension */
      if (strchr (spotfilename, '.') == NULL)   /* if no extension at all */
        strcat (spotfilename, ".spt");  /* add extension */
      else {
        ok = FALSE;
        perror ("\7Illegal filename, please re-enter\n");
      }
    if (ok) {
      spotfile = fopen (spotfilename, "rb");    /* open digitised file */
      if (spotfile == NULL) {
        perror ("\7File does not exist. Please re-enter.\n");
        ok = FALSE;
      }
      fclose (spotfile);
    }
  }                             /* end do */
  while (!ok);
  /* The lines below creates the entry file name by appendening '.aut' */
  /* to the spot file name without the .spt extension */
```



```
  strncpy (entryfilename, spotfilename, strlen (spotfilename) - 4);
  entryfilename[strlen (spotfilename) - 4] = '\0';
  strcat (entryfilename, ".aut\0");
  setpageindex ();
  printf ("Please ensure the disc containing the SPOT file is resident.\n");
  pause ();
  spotfile = fopen (spotfilename, "rb");        /* open digitised file */
  if (spotfile == NULL) {
    perror ("\7Error opening spotfile\n");
    exit (1);
  }
}                               /* end getspotfile */

/* The 'load_entries' function loads an entry list, from drive c, into  */
/* the entry array, returning -1 on error. */

int load_entries (void)
{

  if ((entryfile = fopen (entryfilename, "rb")) == NULL) {
    fclose (entryfile);
    return (-1);
  }
  printf ("Loading Authors.\n");
  if ((fread (&no_of_entries, sizeof (no_of_entries), 1, entryfile)) != 1) {
    fclose (entryfile);
    return (-1);
  }
  if ((fread (entries, sizeof (entry), no_of_entries, entryfile)) !=
        no_of_entries) {
    fclose (entryfile);
    return (-1);
  }
  fclose (entryfile);
  return (0);

}                               /* end load_entries */

/* the 'save_entries' function just saves the entry array in full, to   */
/* disc. */

void save_entries (void)
{

  printf ("Saving Authors.\n");
  if ((entryfile = fopen (entryfilename, "wb")) == NULL) {
    perror ("\7Error creating entry file\n");
    exit (1);
  }
  if ((fwrite (&no_of_entries, sizeof (no_of_entries), 1, entryfile)) != 1) {
    perror ("\7Error writing output file\n");
    exit (1);
  }
  if ((fwrite (entries, sizeof (entry), no_of_entries, entryfile)) !=
        no_of_entries) {
    perror ("\7Error writing entry file\n");
    exit (1);
  }
  fclose (entryfile);

}                               /* end save_entries */

/* the 'save_page' function saves the entire page, in run length        */
/* encoded form to disk. */

void save_page (int header)
{                               /* header is a boolean */

  int ok, white, loop, pix, x, y, run;
  FILE *imagefile;
  char imagefilename[NAMESIZE], *im, *ma;
```

```
  int count = 0;

  fflush (stdin);
  printf ("Please enter the name of the page to be saved.\n");
  do {
    fflush (stdin);
    ok = TRUE;
    scanf ("%s", imagefilename);
    /*    strlwr(imagefilename); convert to lower case */
    if (strcmp (&imagefilename[strlen (imagefilename) - 4], ".spt") != 0)       /* if no .
spt extension */
      if (strchr (imagefilename, '.') == NULL)  /* if no extension at all */
        strcat (imagefilename, ".spt"); /* add extension */
      else {
        ok = FALSE;
        perror ("\7Illegal filename, please re-enter\n");
      }
    if (ok) {
      imagefile = fopen (imagefilename, "rb");  /* open digitised file */
      if (imagefile != NULL) {
        perror ("\7File exists. Please re-enter.\n");
        ok = FALSE;
      }
      fclose (imagefile);
    }
  }                              /* end do */
  while (!ok);
  if ((imagefile = fopen (imagefilename, "wb")) == NULL) {
    perror ("Error opening image file.\n\r");
    exit (1);
  }
  if (header)
    if (fprintf (imagefile, "%s%c%c%c", "Spot Glynn File", 0x1a, 0xaa, 0xbb)
        == EOF) {
      perror ("Error writing header.\n\r");
      exit (1);
    }
  /* compress by run-length encoding and write to disk. */
  white = TRUE;                  /* 'white' is the current run colour */
  /* 'y' is the y pixel coordinate of the image */
  pix = 0;              /* 'pix' is a temporary variable for legibility */
  for (y = 0; y < SPOTLENGTH; y++) {     /* for every scan line */
    white = TRUE;
    im = page[y];
    x = 0;              /* 'x' is the x pixel coordinate of the image */
    run = 0;            /* 'run' is the present size of the current run */
    do {
      ma = mask;
      if ((*im == 255) || (*im == 0))
        if ((white && (*im == 255)) || (!white && (*im == 0))) {
          writerun (run, imagefile);
          white = !white;
          run = 8;
        } else
          run += 8;
      else
        for (loop = 0; loop < 8; loop++) {      /* for every bit in the byte */
          pix = *im & *ma++;
          if ((pix && white) || (!pix && !white)) {     /* if at end of run */
            writerun (run, imagefile);
            white = !white;      /* reverse colour */
            run = 1;    /* start count again but we have one pix already */
          } else
            run++;                /* else continue counting this run */
        }                         /* end 'for each bit in byte' */
      x += 8;
      im++;
      if (x == SPOTWIDTH)
        count++;                  /**/
    }
    while (x != SPOTWIDTH);
```



```
    writerun (run, imagefile);
  }                              /* end 'for all scanlines' */
  fclose (imagefile);
  printf ("Count: %d\n", count);/**/

}                                /* end save_page */

/* the 'writerun' function writes a single run length to disk (used by  */
/* save_page). The no of bytes written. are returned.    */

int writerun (int run, FILE * imagefile)
{

  if (run < 128) {
    fputc ((char) run, imagefile);
    return (1);
  } else {
    fputc ((char) (run >> 8) | 128, imagefile); /* run / 256=run >> 8 */
    fputc ((char) (run & 255), imagefile);      /* run % 256 = run & 255 */
    return (2);
  }

}                                /* end writerun */

/* the 'split_page' function writes the two halves of the page to two   */
/* separate files. If the spot file name is eg. p222.spt, then the two  */
/* files will be lp222.spt and rp222.spt, representing left and right   */
/* respectively */

void split_page (void)
{

  int x, y, white, run, startx, stopx;
  FILE *splitfile;
  char filename[NAMESIZE];

  /* first save left hand side */
  strcpy (filename, "l");
  strcat (filename, spotfilename);
  if ((splitfile = fopen (filename, "wb")) == NULL) {
    printf ("Error opening left side file for writing.\n");
    exit (1);
  }
  if (fprintf (splitfile, "%s%c%c%c", "Spot Glynn File", 0x1a, 0xaa, 0xbb)
      == EOF) {
    perror ("Error writing header.\n\r");
    exit (1);
  }
  startx = left_text;
  stopx = startx + 1048;
  for (y = 0; y < SPOTLENGTH; y++) {
    white = TRUE;
    run = 0;
    for (x = startx; x < stopx; x++)
      /* if ( (a AND b) OR (!a AND !b) ) */
      if ((white && (page[y][x / 8] & mask[x % 8])) ||
          (!white && !(page[y][x / 8] & mask[x % 8]))) {
        writerun (run, splitfile);
        white = !white;
        run = 1;
      } else
        run++;
    writerun (run, splitfile);
  }
  fclose (splitfile);
  printf ("Left side written, now writing right side.\n");
  /* now save right hand side */
  strcpy (filename, "r");
  strcat (filename, spotfilename);
  if ((splitfile = fopen (filename, "wb")) == NULL) {
    printf ("Error opening right side file for writing.\n");
```

```
    exit (1);
  }
  if (fprintf (splitfile, "%s%c%c%c", "Spot Glynn File", 0x1a, 0xaa, 0xbb)
        == EOF) {
    perror ("Error writing header.\n\r");
    exit (1);
  }
  startx = left_text + 1100;
  stopx = startx + 1048;
  for (y = 0; y < SPOTLENGTH; y++) {
    white = TRUE;
    run = 0;
    for (x = startx; x < stopx; x++)
      /* if ( (a AND b) OR (!a AND !b) ) */
      if ((white && (page[y][x / 8] & mask[x % 8])) ||
          (!white && !(page[y][x / 8] & mask[x % 8]))) {
        writerun (run, splitfile);
        white = !white;
        run = 1;
      } else
        run++;
    writerun (run, splitfile);
  }
  fclose (splitfile);

}                               /* end split_page */

/* the 'printer' function opens or closes a stream to the printer        */

void printer (void)
{

  if (!printer_on) {
    if (file_on)
      file_out ();              /* turn off file output */
    if ((outfile = fopen ("prn", "wt")) == NULL) {
      perror ("Error opening printer.\n");
    }
  } else
    fclose (outfile);
  printer_on = !printer_on;

}                               /* end printer */

/* the 'print_tmplt' function prints a hard copy of each template in     */
/* each set.    */

void print_tmplt (int set, int no)
{

  int x, y;

  for (y = 0; y < alpha[set][no].stats.height; y++) {
    for (x = 0; x < alpha[set][no].stats.width; x++)
      if (alpha[set][no].bitmap[y][x / 8] & mask[x % 8])
        fprintf (outfile, "%c", 219);
      else
        fprintf (outfile, " ");
    fprintf (outfile, "\n");
  }
  fprintf (outfile, "No. %d ", no);
  fprintf (outfile, "Name %s ", alpha[set][no].stats.name);
  fprintf (outfile, "Width %d ", alpha[set][no].stats.width);
  fprintf (outfile, "Height %d ", alpha[set][no].stats.height);
  fprintf (outfile, "Area %d ", alpha[set][no].stats.area);
  fprintf (outfile, "Perimeter %d\n", alpha[set][no].stats.perimeter);

}                               /* end print_tmplt */

/* the 'split_entries' procedure examines each entry in the entry array */
/* and writes the bitmap, associated with the entry, to disk. */
```



```
void split_entries (void)
{

  int count, loop, white, pix, no_of_bytes = 0;
  char *im, *ma;
  char filename[NAMESIZE], dummy[20];
  FILE *file;

  entries[0].byte_offset = 0;
  strcpy (filename, spotfilename);
  *strchr (filename, '.') = '\0';
  strcat (filename, ".img");
  if ((file = fopen (filename, "wb")) == NULL) {
    perror ("Error opening image file.\n\r");
    exit (1);
  }
  if (fprintf (file, "%s%c%c%c", "Spot Glynn File", 0x1a, 0xaa, 0xbb) == EOF) {
    perror ("Error writing header.\n\r");
    exit (1);
  }
  for (count = 0; count < no_of_entries; count++) {

    printf ("Writing entry %d: %s\n", count, entries[count].surname);
    /* compress by run-length encoding and write to disk. */
    white = TRUE;             /* 'white' is the current run colour */
    /* 'y' is the y pixel coordinate of the image */
    pix = 0;                  /* 'pix' is a temporary variable for legibility */
    for (y = entries[count].y; y < entries[count + 1].y; y++) {
      /* for every line in entry */
      white = TRUE;
      x = entries[count].x;/* 'x' is the x pixel coordinate of the image */
      /* x -= (x % 8);  align to byte boundary */
      run = 0;                /* 'run' is the present size of the current run */
      do {
        if (page[y][x / 8] & mask[x % 8]) {
          if (white) {
            no_of_bytes += writerun (run, file);
            run = 1;
            white = !white;
          } else
            run++;
        } else {
          if (white)
            run++;
          else {
            no_of_bytes += writerun (run, file);
            run = 1;
            white = !white;
          }
        }
        x++;
      }
      while (x != entries[count].x + 1064);
      no_of_bytes += writerun (run, file);
    }                         /* end 'for all scanlines' */
    if ((count + 1) != no_of_entries)
      entries[count + 1].byte_offset = no_of_bytes;
  }                           /* end 'for each entry' */
  fclose (file);        /* close the image file when all entries written */

}                               /* end split_entries */

/* the 'display_entry' routine scales a entire entry onto the graphics   */
/* screen. */

void display_entry (int num)
{

  int screenx, screeny, n;
  int x, y, ent_size;
```

```
  float scale, countx, county;

  cleardevice ();
  validate (&x, &y);

  x = entries[num].x;
  y = entries[num].y;
  /*   x = 0;
   y = 0;*/
  /*if(strcmp(entries[num].surname,"-") == 0)
      y -= 20;*/

  /* now show screen */

  county = y;
  scale = 1064.0 / XMAX;
  if (num < no_of_entries - 1)
    ent_size = (entries[num + 1].y - entries[num].y) / (scale);
  else
    ent_size = YMAX;
  for (screeny = 0; screeny <= min (YMAX, ent_size); screeny++) {
    countx = x;
    for (screenx = 0; screenx <= XMAX; screenx++) {
      if ((countx < SPOTWIDTH) && (county < SPOTLENGTH))        /* on page? */
        if (page[(int) county][(int) countx / 8] & mask[(int) countx % 8])
          puthirespixel (screenx, screeny, 1);
      countx += scale;
    }
    county += scale;
  }
  cga_update ();

}                                 /* end display_entry */

/* the 'save_templates' function saves all sets of templates to disk.    */
/* it takes one parameter, the name of the file where the templates are */
/* to be saved */

int save_templates (char templatefilename[])
{

  char filename[NAMESIZE];
  int loop;

  filename[0] = getdisk () + 'A';
  filename[1] = ':';
  filename[2] = '\0';
  strcat (filename, templatefilename);
  printf ("Saving Templates.\n");
  if ((templatefile = fopen (templatefilename, "wb")) == NULL) {
    perror ("\7Error creating template file\n");
    return (-1);
  }
  for (loop = 0; loop < SETS; loop++) {
    if ((fwrite (&no_of_tmplts[loop], sizeof (no_of_tmplts[loop]), 1,
        templatefile)) != 1) {
      perror ("\7Error 1 writing template file\n");
      return (-1);
    }
    if ((fwrite (alpha[loop], sizeof (tmplt), no_of_tmplts[loop],
        templatefile)) != no_of_tmplts[loop]) {
      perror ("\7Error 2 writing template file\n");
      return (-1);
    }
  }
  fclose (templatefile);
  return (0);

}                                 /* end save_templates */

/* the 'load_templates' function searches for the template file on the  */
```



```
/* current disk and loads it if present or returns -1 on error. */

int load_templates (char templatefilename[])
{

  char filename[NAMESIZE];
  int loop;

  filename[0] = getdisk () + 'A';
  filename[1] = ':';
  filename[2] = '\0';
  strcat (filename, templatefilename);
  if ((templatefile = fopen (filename, "rb")) == NULL) {
    fclose (templatefile);
    return (-1);
    /*exit(1);*/
  }
  for (loop = 0; loop < SETS; loop++) {
    if ((fread (&no_of_tmplts[loop], sizeof (no_of_tmplts[loop]), 1,
        templatefile)) != 1) {
      fclose (templatefile);
      return (-1);
    }
    if ((fread (alpha[loop], sizeof (tmplt), no_of_tmplts[loop], templatefile))
        != no_of_tmplts[loop]) {
      fclose (templatefile);
      return (-1);
    }
  }

  fclose (templatefile);
  return (0);

}                                 /* end load_templates */

/* the 'display_tmplt' function displays the bitmap of a template on     */
/* the graphics screen. */

void display_tmplt (int n)
{

  int loopcount, x, y, value, length;
  char c;

  printf ("Displaying templates.\n");
  gwindow (OPEN, XMAX / 2 - 54, YMAX / 2 - 54, XMAX / 2 + 54, YMAX / 2 + 54);
  loopcount = 0;
  do {
    printf ("Set %d\n", n);
    printf ("No. %d\n", loopcount);
    printf ("%s\n", alpha[n][loopcount].stats.name);
    bar (XMAX / 2 - 53, YMAX / 2 - 53, XMAX / 2 + 53, YMAX / 2 + 53, 0);
    rectangle (XMAX / 2 - 52, YMAX / 2 - 52, XMAX / 2 - 1, YMAX / 2 - 1);
    rectangle (XMAX / 2 - 52, YMAX / 2 + 1, XMAX / 2 - 1, YMAX / 2 + 52);
    rectangle (XMAX / 2 + 1, YMAX / 2 - 52, XMAX / 2 + 52, YMAX / 2 - 1);
    for (y = 0; y < TEMPLATESIZE; y++)
      for (x = 0; x < TEMPLATESIZE; x++)
        if (alpha[n][loopcount].bitmap[y][x >> 3] & mask[x & 7])
          puthirespixel (x + XMAX / 2 - 50, y + YMAX / 2 - 50, 1);
    /* now produce signatures */
    for (y = 0; y < TEMPLATESIZE; y++) {
      value = 0;
      for (x = 0; x < TEMPLATESIZE; x++)
        if (alpha[n][loopcount].bitmap[y][x >> 3] & mask[x & 7])
          value++;
      for (x = 0; x < value; x++)
        puthirespixel (x + XMAX / 2 + 3, y + YMAX / 2 - 50, 1);
    }
    for (x = 0; x < TEMPLATESIZE; x++) {
      value = 0;
      for (y = 0; y < TEMPLATESIZE; y++)
```

```
      if (alpha[n][loopcount].bitmap[y][x >> 3] & mask[x & 7])
        value++;
    for (y = 0; y < value; y++)
      puthirespixel (x + XMAX / 2 - 50, y + YMAX / 2 + 3, 1);
  }

  cga_update ();
  printf ("Font: %c\n", alpha[n][loopcount].stats.font);
  printf ("Width:  %d\n", alpha[n][loopcount].stats.width);
  printf ("Height: %d\n", alpha[n][loopcount].stats.height);
  printf ("Perim: %d\n", alpha[n][loopcount].stats.perimeter);
  printf ("Area:   %d\n\n", alpha[n][loopcount].stats.area);
  if ((c = getch ()) == 0) {
    c = getch ();
    if (c == RGT)              /* right arrow */
      if (loopcount < no_of_tmplts[n] - 1)
        loopcount++;
      else
        loopcount = 0;
    else if (c == LFT)         /* left arrow */
      if (loopcount > 0)
        loopcount--;
      else
        loopcount = no_of_tmplts[n] - 1;
  }
  /* end if */
  else if (c != ESC) {
    /* find a template */
    /* template names are strings */
    length = 0;
    printf ("Find>");
    while (length < 4 && (c != RET)) {
      strng[length] = c;
      printf ("%c", strng[length]);
      fflush (stdout);
      if (length < 3)
        c = getch ();
      length++;
    }                          /* end while */
    printf ("\n");
    strng[length] = '\0';
    /* now find template */
    start = loopcount;
    do {
      if (loopcount < no_of_tmplts[n] - 1)
        loopcount++;
      else
        loopcount = 0;
      found = !strcmp (strng, alpha[n][loopcount].stats.name);
    }
    while (loopcount != start && !found);
  }                            /* end 'if c != ESC' */
 }                             /* end do loop */
 while (c != ESC);
 gwindow (CLOSE);

}                              /* display_tmplt */

/* the 'full_save' function saves a entire catalogue page to disk        */
/* without compression. */
/* 31/1/91 GPA */

void full_save ()
{

  FILE *file;
  int count;

  if ((file = fopen ("page.tst", "wb")) == NULL)
    perror ("\7Error creating 'full_save' file\n");
  else
```



```
    for (count = 0; count < SPOTLENGTH; count++)
      if ((fwrite (page[count], sizeof (char), SPOTWIDTH / 8, file))
        != SPOTWIDTH / 8) {
        perror ("\7Error 1 writing 'full_save file\n");
        break;
      }
  fclose (file);

}                                /* end full_save */

/* the 'full_read' function reads and entire uncompresssed page from    */
/* disc.       */
/* 31/1/91 GPA */

void full_read ()
{

  FILE *file;
  int count;

  if ((file = fopen ("page.tst", "rb")) == NULL)
    perror ("\7Error reading 'full_read' file\n");
  else
    for (count = 0; count < SPOTLENGTH; count++)
      if ((fread (page[count], sizeof (char), SPOTWIDTH / 8, file))
        != SPOTWIDTH / 8) {
        perror ("\7Error 1 reading 'full_read' file\n");
        break;
      }
  fclose (file);

}                                /* end full_read */

/* the 'file_out' function opens or closes a stream to the file */

void file_out (void)
{

  char filename[NAMESIZE];

  if (!file_on) {
    printf ("Please type in the name of the text file you wish to open.\n");
    scanf ("%s", filename);
    if (printer_on)
      printer ();                /* turn off printer output */
    if ((outfile = fopen (filename, "wt")) == NULL) {
      perror ("Error opening text file for writing.\n");
    }
  } else
    fclose (outfile);
  file_on = !file_on;

}                                /* end file_out */
```

```
int digitise (int test);
int init_scan (void);
```

```
/* tocr_scan.c */

/* This module contains functions which read a digitised page from the */
/* scanner. */

#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>

#include "tocr_defs.h"
#include "tocr_vars.h"

int testerror (int c, long x);
void writerun (int run);

static pcpointer crds;

/* the 'digitise' function intialises important variables, calls        */
/* init_scan and then scans in a page in sections. The page must be     */
/* scanned in sections because of the 64k limitation of the PC. */

int digitise (int test)
{

  pcpointer im, image;           /* pcpointer defined in transputer dos.h */
  union REGS r;
  struct SREGS s;
  int loop, y, x, error, ok;
  int line_width, count, lines_received, no_of_lines;
  int x1, y1, x2, y2;

  /* 65536 bytes is the max. contiguous PC memory allocatable from the */
  /* transputer. 'line_width' (in bytes) is calculated internally from the */
  /* the desired line width (in inches), which is set as one of the */
  /* transmission frame coordinates. 'no_of_lines' is then calculated */
  /* explicitly thus: (65536 DIV line_width) */

  if ((image = alloc86 (65536)) == NULL) {       /* 64k is max allowable */
    printf ("Error allocing PC memory from transputer.\n");
    return (-1);
  }
  if (init_scan () == -1)
    return (-1);

  printf ("Quality test will%sbe done.\n", test ? " " : " not ");
  printf ("Getting image.\n");
  ok = FALSE;
  while (!ok) {
    count = 0;  /* this is a count of the total no. of lines received */
    printf ("Starting Scan.\n");
    r.h.ah = 0x12;
    int86x (0x7e, &r, &r, &s);
    error = testerror (r.x.cflag, r.x.ax);
    if (r.x.cflag == 1) {
      printf ("Cannot send command or no record received.\n");
      printf ("Error status(al) is %x (hex).\n", r.h.al);
    }
    if (error) {
      printf ("An error has occured - aborting scan...\n");
      return (-1);
    }
    printf ("al is %x (hex) on exit.\n", r.h.al);
    line_width = r.x.cx;
    printf ("Line width in bytes is %d.\n", r.x.cx);
    no_of_lines = 65536 / line_width;
    printf ("Getting %d lines at a time.\n", no_of_lines);
    /* now get FIRST x lines */
    r.h.ah = 0x0f;
    r.h.al = 0x00;               /* memory page */
    r.x.cx = no_of_lines;        /* request 'no_of_lines' scan lines */
```



```
    s.es = image >> 16;           /* segment part */
    r.x.di = image & 0xffff;      /* offset part */
    /* getting image */
    int86x (0x7e, &r, &r, &s);
    if (r.x.cflag) {
      printf ("Link failure or fast and no DMA.\n");
      printf ("Error status is %x (hex).\n", r.h.al);
    }
    lines_received = r.x.cx;
    if (testerror (r.x.cflag, r.x.ax)) {
      printf ("Error while receiving data from scanner.\n");
      return (-1);
    }
    /* now get the 'no_of_lines' lines from PC memory */
    im = image;
    for (loop = 0; loop < lines_received; loop++) {
      if (from86 (line_width, im, page[loop + count]) != line_width) {
        printf ("Error getting image from PC.\n");
        return (-1);
      }
      im += line_width;
    }                             /* end loop */
    count += lines_received;      /* count is the TOTAL no. of lines so far */
    printf ("Total no. of lines received: %d\n", count);
    if (test) {                   /* if a quality test is to be done */
      find_header (&x1, &y1, &x2, &y2);
      if (!check_slope (x1, y1, x2, y2)) {
        /* stop scan */
        r.h.ah = 0x0a;
        int86x (0x7e, &r, &r, &s);
        if (testerror (r.x.cflag, r.x.ax)) {
          printf ("Error while stopping scan.\n");
          return (-1);
        }
        printf ("Please re-align the page and press a key.\n");
        pause ();
      } else
        ok = TRUE;
    } else
      ok = TRUE;
  }                               /* end 'while ok' */

  while (r.h.al != 0x03) {        /* while not finished */
    r.h.ah = 0x0f;
    r.h.al = 0x00;                /* memory page */
    r.x.cx = no_of_lines;         /* request 'no_of_lines' scan lines */
    s.es = image >> 16;           /* segment part *7
    r.x.di = image & 0xffff;      /* offset part */
    int86x (0x7e, &r, &r, &s);
    if (r.x.cflag) {
      printf ("Link failure or fast and no DMA.\n");
      printf ("Error status is %x (hex).\n", r.h.al);
    }
    lines_received = r.x.cx;
    if (testerror (r.x.cflag, r.x.ax)) {
      printf ("Error while receiving data from scanner.\n");
      return (-1);
    }
    /* now get the 'no_of_lines' lines from PC memory */
    im = image;
    for (loop = 0; loop < lines_received; loop++) {
      if (from86 (line_width, im, page[loop + count]) != line_width) {
        printf ("Error getting image from PC.\n");
        return (-1);
      }
      im += line_width;
    }                             /* end loop */
    count += lines_received;      /* count is the TOTAL no. of lines so far */
    printf ("Total no. of lines received: %d\n", count);
```

```
    printf ("Error occured\n");
    printf ("\a\n");
    return (TRUE);
  } else
    return (FALSE);

}                                /* end testerror */
```

```
void show_help (void);
```

```
/* tocr_help.c */

/* This module contains a function that displays help text on all options */
/* available. */

#include <stdio.h>
#include <time.h>

#include "tocr_defs.h"
#include "tocr_vars.h"
#include "tocr_util.h"
#include "tocr_help.h"

/* The 'show_help' function displays help windows on the hi-res screen. */

void show_help (void)
{

  int finished, size = 10, firstscr = TRUE;

  do {
    finished = TRUE;
    printf ("Help Menu.\n");
    if (firstscr) {
      printf ("F1  scan entire page and log words found.\n");
      printf ("F2  find and guess a word.\n");
      printf ("F3  find and guess a full text line.\n");
      printf ("F4  find and guess a letter.\n");
      printf ("F7  test of guesspage.\n");
      printf ("F6  remove templates from memory.\n");
      printf ("F7  display information on an entry.\n");
      printf ("F8  view the stored templates.\n");
      printf ("F9  change the spot file.\n");
      printf ("'a' show entries found.\n");
      printf ("'g'/'G' go to a point on the library page.\n");
      printf ("'h' get this help.\n");
      printf ("'l' load entries or templates.\n");
      printf ("'s' save entries or templates.\n");
      printf ("'S' toggle scrolling flag.\n");
      printf ("'T' toggle trace flag.\n");
    } else {
      printf ("'*' show system status.\n");
      printf ("'#' display entire page scaled onto screen.\n");
      printf ("<PgUp> go up one screen size.\n");
      printf ("<PgDn> go down one screen size.\n");
      printf ("<Home> go to the top of the page.\n");
      printf ("<End>  go to the bottom of the page.\n");
      printf ("<rght> go right 100 pixels.\n");
      printf ("<left> go left 100 pixels.\n");
      printf ("<down> go down 100 pixels.\n");
      printf ("<up>   go up 100 pixels.\n");
      printf ("<ctrl RET> Spawn a DOS shell.\n");
    }
    printf ("Press any key for help, Esc to return or RET for more.\n");
    fflush (stdin);
    while ((c = getch ()) != ESC) {
      /* Help until the user types escape to quit */
      if (c == RET) {
        finished = FALSE;
        firstscr = !firstscr;
        break;
      }
      if (c == 0) {
        c = getch ();
        switch (c) {
        case 59:
          printf ("                         F1.\n");
          printf ("F1 scans the entire catalogue page and logs the words\n");
          printf ("found. If there are no templates in memory, the\n");
          printf ("computer will respond 'No templates stored'. As the\n");
          printf ("authors' names are found, they are displayed on the\n");
```



```
          printf ("screen. Typing Esc will quit or any other character\n");
          printf ("will return.\n");
          finished = FALSE;
          break;
        case 60:
          printf ("                         F2.\n");
          printf ("F2 invokes the training system to segment and display\n");
          printf ("an entire author name, character by character. As\n");
          printf ("each character is found, it is displayed in reverse\n");
          printf ("video. The user then has 4 options, as follows:\n");
          printf ("1) Typing a '?' will make the computer guess at the\n");
          printf ("character. Typing a '*' after the '?' will cause the\n");
          printf ("computer to show the scores it has given to each\n");
          printf ("template match.\n");
          printf ("2) Typing 'S' followed by the name of the character\n");
          printf ("(max 4 chars.) stores that character as a template.\n");
          printf ("3) Typing F1 will cause the highlighted character to\n");
          printf ("be displayed in double size.\n");
          printf ("4) Typing anything else will skip the character.\n");
          finished = FALSE;
          break;
        case 61:
          printf ("                         F3.\n");
          printf ("F3 scans one line of text. Punctuation is ignored.\n");
          printf ("The line scanned is the first one on the current");
          printf ("screen.\n");
          finished = FALSE;
          break;
        case 62:
          printf ("                         F4.\n");
          printf ("F4 invokes the training system to segment and display\n");
          printf ("the first character it finds ON THE SCREEN. The\n");
          printf ("character is displayed in reverse video. The options\n");
          printf ("now open to the user can be found in the help for F2.\n");
          finished = FALSE;
          break;
        case 63:
          printf ("                         F5.\n");
          printf ("F5 tests the guesspage function.\n");
          finished = TRUE;
          break;
        case 64:
          printf ("                         F6.\n");
          printf ("F6 allows the user remove all or some of the\n");
          printf ("templates from memory. The user is invited to 'Remove\n");
          printf ("all templates(Y/N?' Type 'y' and the computer will\n");
          printf ("respond 'Are you sure(Y/N)?'. If the user is not\n");
          printf ("removing all the templates, the machine will promt\n");
          printf ("for the name of the template to be removed. If there\n");
          printf ("is more than one instance of the same template, the\n");
          printf ("machine will prompt for the number of the template to\n");
          printf ("be removed.\n");
          finished = FALSE;
          break;
        case 65:
          printf ("                         F7.\n");
          printf ("F7 invokes the demonstration index. If a page has not\n");
          printf ("yet been scanned (with F1), the computer will respond\n");
          printf ("'No entries found yet'. Otherwise, the user will be\n");
          printf ("prompted to type in the name of an author. If that\n");
          printf ("name is found, the corresponding section of the page\n");
          printf ("will be displayed on the screen. Typing any key will\n");
          printf ("display another author of the same name, if one\n");
          printf ("exists on that page. Otherwise the computer will\n");
          printf ("respond 'entry not found'. Type Esc to quit the\n");
          printf ("program or anything else to return.\n");
          finished = FALSE;
          break;
        case 66:
          printf ("                         F8.\n");
          printf ("F8 allows the user to view the templates as bitmaps.\n");
```

```
      printf ("After pressing F8, the first template will be\n");
      printf ("displayed in a window. The user can scroll backwards\n");
      printf ("and forwards through the templates using the cursor\n");
      printf ("left and cursor right keys. The name and number of\n");
      printf ("the template is also displayed in the window. Typing\n");
      printf ("Esc will return.\n");
      finished = FALSE;
      break;
    case 67:
      printf ("                         F9.\n");
      printf ("F9 allows the user to change the current spot file.\n");
      printf ("The user will be prompted for the new file name.\n");
      printf ("Authors recognised on the present page must be saved\n");
      printf ("as all data relating to the present page will be lost.\n");
      finished = FALSE;
      break;
    case 71:
      printf ("                         Home.\n");
      printf ("<home> allows the user to move directly to the top of\n");
      printf ("the page. The x coordinate is held constant.\n");
      finished = FALSE;
      break;
    case 72:
      printf ("                       Up Arrow.\n");
      printf ("Up arrow allows the user to scroll 100 pixels upwards.\n");
      finished = FALSE;
      break;
    case 73:
      printf ("                         PageUp.\n");
      printf ("PgUp allows the user to scroll up the screen 1 screen\n");
      printf ("length.\n");
      finished = FALSE;
      break;
    case 75:
      printf ("                       Left Arrow.\n");
      printf ("Left arrow allows the user to scroll 100 pixels to\n");
      printf ("the left.\n");
      finished = FALSE;
      break;
    case 77:
      printf ("                      Right Arrow.\n");
      printf ("Right arrow allows the user to scroll 100 pixels to\n");
      printf ("the right.\n");
      finished = FALSE;
      break;
    case 79:
      printf ("                         End.\n");
      printf ("<end> allows the user to move directly to the end of\n");
      printf ("the page. The x coordinate is held constant.\n");
      finished = FALSE;
      break;
    case 80:
      printf ("                       Down Arrow.\n");
      printf ("Down arrow allows the user to scroll 100 pixels\n");
      printf ("downwards.\n");
      finished = FALSE;
      break;
    case 81:
      printf ("                         PageDown.\n");
      printf ("PgDn allows the user to scroll down the screen 1\n");
      printf ("screen length.\n");
      finished = FALSE;
      break;

    }                       /* end switch */
  } else
    switch (c) {
    case 10:
      printf ("                    <ctrl RETURN>.\n");
      printf ("Spawn a DOS shell. Leave the DOS shell by typing\n");
      printf ("<exit> on the command line. Note, while in the\n");
```



```
      printf ("shell, to avoid losing the program, do not run\n");
      printf ("anything on the transputer!\n");
      finished = FALSE;
      break;
    case 'a':
      printf ("                          a.\n");
      printf ("'a' displays the names of the entries that have\n");
      printf ("been found. If a page scan has not yet been\n");
      printf ("done (with F1), the computer will respond with\n");
      printf ("'No names stored'.\n");
      finished = FALSE;
      break;
    case '*':
      printf ("                          *.\n");
      printf ("Typing '*' will display the system status.\n");
      printf ("The computer displays the names of any templates\n");
      printf ("stored. The computer will also display the status\n");
      printf ("of the tracflag, the scroll_on flag, the type of\n");
      printf ("graphics hardware present and the current mode\n");
      printf ("and resolution.\n");
      finished = FALSE;
      break;
    case 'g':
      printf ("                          g.\n");
      printf ("g allows the user to jump around the catalogue page.\n");
      printf ("The user is prompted for the coordinates of the new\n");
      printf ("section of the catalogue page. Type in the x and y\n");
      printf ("coordinates separated by  a comma. The coordinates of\n");
      printf ("the current section are displayed on the screen. If the\n");
      printf ("coordinates specified are not on the catalogue page,\n");
      printf ("they will be adjusted and the nearest section will be\n");
      printf ("displayed.\n");
      finished = FALSE;
      break;
    case 'h':
      printf ("                          h.\n");
      printf ("h invokes this help facility. The primary help screen\n");
      printf ("is a quick sysnopsis of what each keystroke does.\n");
      printf ("While on this screen, typing any key will display the\n");
      printf ("help screen for that keystroke. Typing Esc will\n");
      printf ("return to the primary help screen. Typing Esc on the\n");
      printf ("primary help screen will exit help.\n");
      finished = FALSE;
      break;
    case 'l':
      printf ("                          l.\n");
      printf ("l allows the user to load either a template set or a\n");
      printf ("list of pre-saved entries that were found in a\n");
      printf ("previous scan. The user is invited to 'Load Authors\n");
      printf ("or Templates(a/t)?'.\n");
      finished = FALSE;
      break;
    case 's':
      printf ("                          s.\n");
      printf ("s allows the user to save either a template set or a\n");
      printf ("list of entries that have been scanned. The user is\n");
      printf ("invited to 'Save Authors or Templates(a/t)?'.\n");
      finished = FALSE;
      break;
    case 'T':
      printf ("                          T.");
      printf ("T toggles the trace flag. When the trace flag is on,\n");
      printf ("the computer displays exactly where it is working on\n");
      printf ("the page by means of a pixel-wide dot. It will also\n");
      printf ("display how it decides to amalgamate two or more\n");
      printf ("blobs into a single character. NOTE. The trace flag\n");
      printf ("has no effect if the scroll_on flag is off.\n");
      finished = FALSE;
      break;
    case 'S':
      printf ("                          S.");
```

```
            printf ("S toggles the scroll_on flag. When the scroll_on flag\n");
            printf ("is on, the computer displays the section of the page\n");
            printf ("it is working on, at all times. This flag is\n");
            printf ("automatically turned on in training mode.\n");
            break;
        }                          /* end switch */
    }                              /* end 'while not ESC' */
  }                                /* end 'do while not finished' */
  while (!finished);
}                                  /* end show_help */
```

```
/***    graphics.h
 ***    A few constants for use with graphics.c
 ***/

#define MONO_80COL_TEXT_MODE        0x02
#define CGA_80COL_TEXT_MODE         0x03   /**/
#define CGA_LORES_GRAPHICS_MODE     0x05
#define CGA_HIRES_GRAPHICS_MODE     0x06
#define DONT_CLEAR_DISPLAY          0x80

#define XMAX            639
#define LORES_XMAX      319
#define YMAX            199

enum putimage_ops {             /* operators for putimage */
  COPY_PUT,                     /* MOV */
  XOR_PUT,                      /* XOR */
  OR_PUT,                       /* OR  */
  AND_PUT,                      /* AND */
  NOT_PUT                       /* NOT */
};

void cga_update (void);
void puthirespixel (int x, int y, int c);
int gethirespixel (int x, int y);
void putlorespixel (int x, int y, int colour);
int getlorespixel (int x, int y, int colour);
void cleardevice ();
void gographics ();
void gomono ();
void video_mode (int x);
void rectangle (int left, int top, int right, int bottom);
void bar (int left, int top, int right, int bottom, int col);
void putdirectpixel (int x, int y, int c);
void line (int x1, int y1, int x2, int y2);     /* Glynn Anderson Jan 1990 */
int getdirectpixel (int x, int y);      /* Glynn Anderson Jul 1990 */
unsigned int imagesize (int left, int top, int right, int bottom);
void getimage (int left, int top, int right, int bottom, char *bitmap);
void putimage (int left, int top, char *bitmap, int op);
```

```
/***   graphics.c
 ***
 ***   Primitives for CGA graphics (prelim)
 ***
 ***   Based on some by ADC
 ***
 ***   Modified Jan 1990, Glynn Anderson, Trinity College Dublin.
 ***
 ***   The method used by these very simple routines is to keep a complete
 ***   copy of the display in the transputer's memory.   This is set up the
 ***   first time video_mode is called.   Calls on putlorespixel and
 ***   puthirespixel alter this copy of the display.   When cga_update
 ***   is called, the whole buffer is sent off to the PC's display memory.
 ***/

#define NULL (0)
#define TRUE 1
#define FALSE 0

#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include "graphics.h"

#define BIOS_VIDEO_DRIVER 0x10
#define VIDEO_SET_MODE    0x00

/* CGA specific definitions */

#define CGA_BUFFER_SIZE 16384
#define BYTES_PER_LINE  80
#define CGA_BASE_ADDR   0xB8000000

#define CGA_HIRES_OFFSET(x,y)    ((x)>>3) + ((y)>>1)*0x50 + (((y)&1)*0x2000)
#define CGA_HIRES_BITMASK(x,y)   cga_hires_bitmask[(x)&7]/* independent of y */
static char cga_hires_bitmask[8] =
{0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1};

#define CGA_LORES_OFFSET(x,y)     ((x)>>2) + ((y)>>1)*0x50 + (((y)&1)*0x2000)
#define CGA_LORES_BITMASK(x,y,c)  (c << (cga_lores_shift[(x)&3]))
static char cga_lores_shift[4] =
{6, 4, 2, 0};

static char *buf = NULL;
static initialised = FALSE;

/* Send the whole buffer to the display */

void cga_update ()
{

  to86 ( /* num bytes */ CGA_BUFFER_SIZE,
         /* from */ buf,
         /* to */ CGA_BASE_ADDR);
}                                    /* end cga_update */

/* Display a point (hires) */
void puthirespixel (int x, int y, int c)
{

  if (!initialised) {
    printf ("Error: Graphics system not initilaised.\n");
    exit (1);
  }
  if (x >= 0 && x <= XMAX && y >= 0 && y <= YMAX)
    if (c)
      buf[CGA_HIRES_OFFSET (x, y)] |= CGA_HIRES_BITMASK (x, y);
    else
```



```
      buf[CGA_HIRES_OFFSET (x, y)] &= ~CGA_HIRES_BITMASK (x, y);
}                                       /* end puthirespixel */

/* Test a point (hires) */

int gethirespixel (int x, int y)
{

  return (buf[CGA_HIRES_OFFSET (x, y)] & CGA_HIRES_BITMASK (x, y));
}                                       /* end gethirespixel *//**/

/* Display a point (lores) */

void putlorespixel (int x, int y, int colour)
{

  if (!initialised) {
    printf ("Error: Graphics system not initilaised.\n");
    exit (1);
  }
  if (x >= 0 && x <= LORES_XMAX && y >= 0 && y <= YMAX)
    buf[CGA_LORES_OFFSET (x, y)] |= CGA_LORES_BITMASK (x, y, colour);
}                                       /* end putlorespixel */

/* Test a point (lores) */

int getlorespixel (int x, int y, int colour)
{

  return (buf[CGA_LORES_OFFSET (x, y)] & CGA_LORES_BITMASK (x, y, colour));
}                                       /* end getlores pixel */

/* the 'putdirectpixel' procedure ignores the copy of the screen in */
/* transputer memory and write the pixel DIRECTLY to the PC screen  */
/* memory. The function exists because plotting one pixel to the    */
/* transputer copy of the screen and the sending the whole screen to*/
/* the PC is very costly for just one pixel */

void putdirectpixel (int x, int y, int c)
{

  char temp;

  if (x >= 0 && x <= XMAX && y >= 0 && y <= YMAX) {
    from86 (1, CGA_BASE_ADDR + CGA_HIRES_OFFSET (x, y), &temp);
    if (c)
      temp |= CGA_HIRES_BITMASK (x, y);
    else
      temp &= ~CGA_HIRES_BITMASK (x, y);
    to86 (1, &temp, CGA_BASE_ADDR + CGA_HIRES_OFFSET (x, y));
  }
}                                       /* end putdirectpixel */

/* 'getdirectpixel' Glynn Anderson 18/jul/1990 */

int getdirectpixel (int x, int y)
{

  char temp;

  from86 (1, CGA_BASE_ADDR + CGA_HIRES_OFFSET (x, y), &temp);
  return (temp & CGA_HIRES_BITMASK (x, y));

}                                       /* end getdirectpixel */

/* 'cleardevice' cleard the screen buffer in transputer memory */
/* then updates the PC graphics screen */

void cleardevice ()
{
```

```
  if (!initialised) {
    printf ("Error: Graphics system not initilaised.\n");
    exit (1);
  }
  memset (buf, 0, CGA_BUFFER_SIZE);
  cga_update ();
}                                /* end cleardevice */

void gographics (void)
{

  system ("mode co80");
}                                /* end gographics */

void gomono (void)
{

  system ("mode mono");
}                                /* end gomono */

/* Set a video mode */

void video_mode (int x)
{

  union REGS regs;

  if (buf == NULL) {
    initialised = TRUE;
    if ((buf = calloc (1, CGA_BUFFER_SIZE)) == NULL) {
      printf ("Error callocing memory in video_mode.\n");
      exit (1);
    }
  }
  regs.h.ah = VIDEO_SET_MODE;
  regs.h.al = x;
  int86 (BIOS_VIDEO_DRIVER, &regs, &regs);
}                                /* end video_mode */

void line (int x1, int y1, int x2, int y2)
{                                /* Glynn Anderson Jan 1990 */

  int len, loopcount;
  float xinc, yinc, x, y;

  len = sqrt ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
  if (len == 0)
    len = 1;
  x = x1;
  y = y1;
  xinc = (float) (x2 - x1) / len;
  yinc = (float) (y2 - y1) / len;
  for (loopcount = 1; loopcount <= len; loopcount++) {
    puthirespixel ((int) x, (int) y, 1);
    x += xinc;
    y += yinc;
  }
  /*cga_hires_plot(x2,y2);*/
}                                /* end line */

/* 'imagesize returns the size(in bytes) required to hold an image */
/* specified by the coordinates left,top,right,bottom */

unsigned int imagesize (int left, int top, int right, int bottom)
{

  unsigned int size;

  size = (right - left + 1) * (bottom - top + 1);       /* in bits */
  if ((size % 8) != 0)
    size = size / 8 + 1;         /* add an extra byte for extra bits */
```



```
  else
    size /= 8;
  /* add space to put in the width and height */
  size += 3;                     /* two bytes for width, one for height */
  return (size);
}                                /* end imagesixe */

/* 'getimage' copies the designated part of the screen into memory */

void getimage (int left, int top, int right, int bottom, char *bitmap)
{

  unsigned int size;
  int loopcount, x, y, width, height;
  char *ptr;

  /* calculate size */
  size = imagesize (left, top, right, bottom);
  memset (bitmap, 0, size);      /* clear image buffer */
  width = right - left + 1;
  height = bottom - top + 1;
  /* now save the width and height of the image in the first few bytes */
  /* if the width is greater than a byte, then two bytes are used, the */
  /* first one being flagged. The height is never greater than 255(200)*/
  if (width < 128)
    bitmap[0] = width;
  else {
    bitmap[0] = (width >> 8) | 128;
    bitmap[1] = width & 255;
  }
  bitmap[2] = height;
  x = left;
  y = top;
  /* store image */
  for (ptr = &bitmap[3]; ptr < bitmap + size; ptr++)
    for (loopcount = 7; loopcount >= 0; loopcount--) {
      if (gethirespixel (x, y))
        *ptr |= (1 << loopcount);
      if (x == right) {
        x = left;
        y++;
      } else
        x++;
    }
}                                /* end getimage */

/* 'putimage' places a previously saved image at the coordinates x,y */

void putimage (int left, int top, char *bitmap, int op)
{

  int width, height, loopcount, x, y;
  unsigned int size;
  char *ptr;

  /* first get the width and height of the image */
  height = bitmap[2];
  if (bitmap[0] & 128)
    width = ((bitmap[0] & 127) << 8) + bitmap[1];
  else
    width = bitmap[0];
  /* now calculate size */
  size = imagesize (0, 0, width - 1, height - 1);
  /* now put image */
  x = left;
  y = top;
  for (ptr = &bitmap[3]; ptr < bitmap + size; ptr++)
    for (loopcount = 7; loopcount >= 0; loopcount--) {
      switch (op) {
      case COPY_PUT:
        if (*ptr & (1 << loopcount))
```

```
        puthirespixel (x, y, 1);
      else
        puthirespixel (x, y, 0);
      break;
    case XOR_PUT:
      if (gethirespixel (x, y) ^ (*ptr & (1 << loopcount)))
        puthirespixel (x, y, 1);
      else
        puthirespixel (x, y, 0);
      break;
    case OR_PUT:
      if (gethirespixel (x, y) | (*ptr & (1 << loopcount)))
        puthirespixel (x, y, 1);
      break;
    case AND_PUT:
      if (gethirespixel (x, y) & (*ptr & (1 << loopcount)))
        puthirespixel (x, y, 1);
      else
        puthirespixel (x, y, 0);
      break;
    case NOT_PUT:
      if (!(*ptr & (1 << loopcount)))
        puthirespixel (x, y, 1);
      else
        puthirespixel (x, y, 0);
      break;
    default:
      if (*ptr & (1 << loopcount))
        puthirespixel (x, y, 1);
      else
        puthirespixel (x, y, 0);
      break;
    }
    if (x == left + width - 1) {
      x = left;
      y++;
      if (y > top + height - 1)
        break;                /* remember there may be extra pixels */
    } else
      x++;
  }
}                             /* end putimage */

/* the 'rectangle' function draws a rectangle on the grpahics screen */
/* top left corner is (left,top), bottom right is (right,bottom)     */

void rectangle (int left, int top, int right, int bottom)
{

  int loopcount;

  for (loopcount = left; loopcount <= right; loopcount++) {
    puthirespixel (loopcount, top, 1);
    puthirespixel (loopcount, bottom, 1);
  }
  for (loopcount = top; loopcount <= bottom; loopcount++) {
    puthirespixel (left, loopcount, 1);
    puthirespixel (right, loopcount, 1);
  }
}                             /* end rectangle */

/* the 'bar' function just fills in a part of the screen, indicated by */
/* the parameters left, top, right and bottom, in the colour col.      */

void bar (int left, int top, int right, int bottom, int col)
{

  int x, y;

  for (y = top; y <= bottom; y++)
    for (x = left; x <= right; x++)
```



```
      puthirespixel (x, y, col);
}                             /* end bar */
```

TRINITY COLLEGE
10 MAR 1995
LIBRARY DUBLIN