

PASCAL USERS GROUP

Pascal News

Communications about the Programming Language Pascal by Pascalers

Number 24

- Pascal Standards: Progress Report
- Status Report on Version 3.0
- WRITENUM — A Routine to Output Real Numbers
- TREEPRINT — A Package to Print Trees on Character Printers
- Three Proposals for Extending Pascal
- Announcements

Number

24

JANUARY 83

EX LIBRIS: David T. Craig
736 Edgewater
[#] Wichita, Kansas 67230 (USA)

POLICY: PASCAL NEWS

(Jan. 83)

- *Pascal News* is the official but *informal* publication of the User's Group.

Purpose: The Pascal User's Group (PUG) promotes the use of the programming language Pascal as well as the ideas behind Pascal through the vehicle of *Pascal News*. PUG is intentionally designed to be non political, and as such, it is not an "entity" which takes stands on issues or support causes or other efforts however well-intentioned. Informality is our guiding principle; there are no officers or meetings of PUG.

The increasing availability of Pascal makes it a viable alternative for software production and justifies its further use. We all strive to make using Pascal a respectable activity.

Membership: Anyone can join PUG, particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Memberships from libraries are also encouraged. See the ALL-PURPOSE COUPON for details.

- *Pascal News* is produced 3 or 4 times during a year; usually in March, June, September, and December.
- ALL THE NEWS THAT'S FIT, WE PRINT. Please send material (brevity is a virtue) for *Pascal News* single-spaced and camera-ready (use dark ribbon and 15.5 cm lines!)
- Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.
- *Pascal News* is divided into flexible sections:

POLICY — explains the way we do things (ALL-PURPOSE COUPON, etc.)

EDITOR'S CONTRIBUTION — passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.

HERE AND THERE WITH PASCAL — presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal in the news, history, membership rosters, etc.

APPLICATIONS — presents and documents source programs written in Pascal for various algorithms, and software tools for a Pascal environment; news of significant applications programs. Also critiques regarding program/algorithm certification, performance, standards conformance, style, output convenience, and general design.

ARTICLES — contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.).

OPEN FORUM FOR MEMBERS — contains short, informal correspondence among members which is of interest to the readership of *Pascal News*.

IMPLEMENTATION NOTES — reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Portable Pascals, Pascal Variants, Feature-Implementation Notes, and Machine-Dependent Implementations.

Pascal News

Communications about the Programming Language Pascal by Pascalers

JANUARY 1983

Number 24

2 COMPILERS NOTES

APPLICATIONS

- 3** A Pascal Bibliography *By Tony Hayes*

PASCAL STANDARDS

- 20** Pascal Standards: Progress Report *By Jim Miner*
20 Status Report on Version 3.0 of the Pascal Test Suite *By B.A. Wickmann*

ANNOUNCEMENTS

- 23** Distribution of the Edison System
23 Pascal Chosen as Sil
23 Pascal: A Problem Solving Approach
24 Modula-2

ARTICLES

- 25** WRITENUM — A Routine to Output Real Numbers
By Doug Grover and Ned Freed
27 TREEPRINT — A Package to Print Trees on any Character Printer
By Ned Freed and Kevin Carosso
32 Three Proposals for Extending Pascal *By R.D. Tennent*
32 The Where-Clause: A Proposed Extension to Pascal *By R.D. Tennent*
34 Proposals for Improved Exception Handling in Pascal *By R.D. Tennent*
37 The Definition Block: A Proposed Extension to Pascal *By R.D. Tennent*

40 OPEN FORUM

42 IMPLEMENTATION NOTES COUPON

45 SUBSCRIPTION COUPON

47 LICENSE APPLICATION

Hello

This is Pascal News and my name is Charlie Gaffney. Much has happened since I received my March #22-23 Issue. I am the publisher of USUS News. USUS is the UCSD p-System User Society. The p-system was developed to bring Pascal to micro computers. Our USUS News was modeled on Pascal News. We have a lot of information in USUS but it was a chore to read because of bad original and photo copy material used for printing.

I sought a typesetter and found we could typeset and print for only 10% increase in cost. This is a small premium cost to have a readable newsletter. We typeset in August and received many compliments so far.

I thought of our model Pascal News and called Rick Shaw to explain our (USUS) improvement and ask if he needed help.

But Rick had his own story to tell. The work at Pascal Users Group was not performed by a group but by one man, Rick Shaw. He was hard pressed to keep up with the business of PUG.

An offer had been made by the "Journal of Pascal & Ada" to take all pending articles and publish them.

I made a counter offer to maintain PUG as it is under new management. Rick thought that was a nice idea, but the problems would persist and PUG would fail either now or later. After three phone calls Rick decided to let me try.

The News will be typeset and I hope you approve of our new appearance. The articles

you submit may be in any format because they will now be typeset. It is possible to enlarge the program listings if they are submitted in a narrow format of 15.5 cm wide.

Business

I have decided to pay a small business to update:

1. the member list
2. new and renew members
3. banking records

Membership costs have gone up but if you pay for two years the third year is free.

Back issues have tied up a great deal of money. We have articles and programs just waiting for you. Buy a set. Buy a complete set. Buy a set for your friends.

A little about me

I am an electrician, and I work for Chevrolet in Parma, Ohio. I have no college education and no formal computer training. My experience with computers involved the purchase of a Western Digital microengine, 16 bit computer. The computer uses p-code as defined by UCSD p-System and directly implements the code without an interpreter. Pascal News and USUS News, and 25 text books, have been my teachers. I thank them and each of you.

Charlie

A Pascal Bibliography

By Tony Heyes
Blind Mobility Research Unit,
Department of Psychology,
University of Nottingham
England

Introduction

The Pascal Bibliography is a package of programs written in standard Pascal and should therefore be easily transported. It enables users to store references and to retrieve them either by AUTHOR name or by KEYWORD; or logical combinations of AUTHORS and KEYWORDS. The bibliography is designed for human use; it uses very explicit prompts.

Design Philosophy

The bibliography consists of a collection of ITEMS. Each ITEM takes the form of:-

One line devoted to AUTHOR or ADDRESSEE names.

Two lines devoted to TITLE or ADDRESS.

Two lines devoted to LOCATION.

DATE ITEM NUMBER.

Two lines devoted to KEYWORDS.

For example:-

HEYES A.D.,FERRIS A.J.,ORLOWSKI R.J.
COMPARISON BETWEEN TWO METHODS
OF RESPONSE FOR
AUDITORY LOCALISATION IN THE AZI-
MUTH PLANE.
J. ACOST. SOC. AMER., 58; 1336-1339

1975 260
DEAFNESS,LOCALISATION,AUDITORY
DISPLAYS
STEREOPHONIC SOUNDS,KINAESTHESIS

If ITEMS are addresses the convention is to store the address on the two lines of title.

For example:-

BLOGGS J.B.
Mr.J.B.Bloggs\13 Fishpond Rd.\ Beeston,
Nottingham.\ NG7 2RD\ U.K.
Tel 0602-251234

1980 27
ADDRESS,CIRCULATION LIST,XMAS
CARD

Note the use of the backslash [\] to indicate the start of a new line. Note also that additional information

such as the telephone number can be stored on the location lines. Note, finally, the date has little meaning in this context.

Items may be located by running the program "bibout".

Items may be APPENDED or CHANGED by running the program "bibin".

Both programs are well supplied with prompts and are very simple to use.

Since additions and changes require that the current DICTIONARY be recompiled and this takes time, the actual changes take place during the night. The instructions to implement the changes reside in a PENDING TRAY until the night time run. The user will remain unaware of this slight restriction unless he tries to locate an ITEM during the day on which the ITEM was loaded.

Method of Use

The following assumed the use of the UNIX operating system. Login with your user name, give your password, respond to the first system prompt "%" with "cd bib", ie. change directory to "bib". In answer to the next system prompt, "%", you may select any one of the programs from within the package.

These are:-

- a) "bibbin" to enter new items or to change an ITEM.
- b) "bibout" to search the bibliography for an ITEM.
- c) "outdict" to produce a hard copy of the current DICTIONARY.
- d) "cat scratch lpr" to output a hard copy of the SCRATCH FILE.

NEW USERS SHOULD ASK IF THEY MAY HAVE ACCESS TO AN ESTABLISHED BIBLIOGRAPHY AND THEN TRY USING "bibout" TO LOCATE ITEMS OF INTEREST.

To logout respond to the system prompt "%" by typing "control Z".

The Programs

- a) "bibin"
The opening prompt allows the selection of one of the following options:-
APPEND

The prompts should be sufficiently explicit, but note:-

- (1) Authors and keywords should be separated by commas. Since they are used in the dictionary they should not spill over the end of a line. They can be any length but only the first 20 characters are significant.
- (2) The terminal will probably be set to produce lower case letters. The program will automatically convert them to upper case. If you wish to override this, begin each line of text with a backslash [\].
- (3) The date must be a single integer e.g. 1980.
- (4) If addresses are to be stored use the two title lines, close pack but indicate new lines with a backslash [\].
- (5) A personal local storage reference may be kept on the second location line. It should be enclosed in square brackets; e.g. [BM760] means that a copy of this ITEM is in the BM library, entry number 760.

CHANGE

Answer the prompts but please take note of the following:-

- 1) You must know in advance the ITEM number of the ITEMS you require to change.
- 2) You have to retrieve the ITEMS from the bibliography so CHANGE is relatively slow; be patient. It saves time, if you are changing more than one ITEM to make the changes in numerical order of ITEM number.
- 3) You retrieve the ITEM to be changed from the bibliography, the changed ITEM goes into the PENDING TRAY. If you change the same ITEM more than once in a single day only the last version will survive.

SPECIAL FACILITY

This option moves the contents of the SCRATCH file into the PENDING tray. It can be used for moving ITEMS from one bibliography to another. Since SCRATCH is a text file, ITEMS may be changed using an editor and then loaded back into the PENDING tray. (Clever stuff!!).

b) "bibout"

The computer will count the ITEMS in the bibliography and then offer the option of producing a HARD COPY of the dictionary or doing a SEARCH for ITEMS.

SEARCH

You may either search by NUMBER or, more usually by using the DICTIONARY.

You may opt to send the results either to the TERMINAL or to the SCRATCH FILE for subsequent printing.

SEARCH by NUMBER

The search is terminated by asking to search for item number zero [0].

A block of ITEMS may be searched for by asking to search for item number minus one [-1]. You will then be asked for the lowest and the highest item numbers of the block.

SEARCH by DICTIONARY

You will be asked for a word i.e. an AUTHOR

name or a KEYWORD. The computer will look this up in the DICTIONARY and list the ITEM numbers of all ITEMS containing this word in their AUTHOR or KEYWORD string. If you are doing a single word search answer the next prompt will a full stop [.], and then the instruction to LOOK UP. If, however, it is a multiple word search give the next word. Once again the corresponding ITEM number list will be printed out. The answer to the prompt "AND, OR or NOT" enables you to combine the current ITEM number list with the previous ITEM number list. For instance:-

AND Only numbers present in both lists are retained.

OR All numbers from both lists are retained.

NOT Numbers present in the current list are deleted from the previous list.

A new current list is printed out showing the results of the selection. The search sequence may be continued for any number of logical combinations of words. At any time a search for the ITEMS in the current list may be initiated by giving a full stop [.]. After which you may either LOOK UP the selected ITEMS or, if you have made a mistake in your list combinations simply RESTART. There is one special word, namely ***, this word will match all the dictionary.

c) "outdict"

No prompts and no option, simply type "outdict" in answer to the system prompt "%" to obtain a hard copy of the current DICTIONARY.

Note, you must have first prepared a copy of the DICTIONARY by running the appropriate HARD COPY option of "bibout".

d) "opr scratch"

This program is run to obtain the printed output from "bibout", provided the option had been chosen to send the output to the SCRATCH FILE.

No prompts and no options, simply type "opr scratch" in answer to the system prompt "%" to obtain a hard copy of the contents of the SCRATCH FILE.

N.B. If you would like to list the SCRATCH FILE to the terminal to check the contents then run "cat scratch".

Acknowledgements

I gratefully acknowledge the encouragement and support I have received from Roger Henry and Chris Blunsdon.

The bibliography was originally intended for use by the members of the BLIND MOBILITY RESEARCH UNIT it is however available to any members of the Pascal Users Group. Would anyone wishing to take up this offer please contact Tony Heyes to arrange medium of transportation.

NOTES FOR IMPLEMENTORS

The following notes outline the steps the imple-

menter should take in order to establish a new bibliography. After this groundwork, the user can use the shell commands *bibin*, *bibout*, and *outdict* to build and manipulate the bibliography.

1. The bibliography system requires 6 workfiles named b1 to b6. The recommended practice is for the user to devote a directory to the bibliography, say 'user/bib'. The workfiles can be created easily using the cat command. E.g

```
cat > b1      Z
```

File b3 requires a link named scratch. This can be created by the command —

```
ln b3 scratch
```

2. b6 is used as a temporary scratch file during the overnight run. It grows to be as large as b1. If there is insufficient room on the user's disc b6 may be coerced on to another disc.
3. The bib directory must contain the following shell commands:-

```
bibin      Bibin.out b1 b2 b3 b4 b5
bibout     Bibout.out b1 b2 b3 b4 b5
bibupdate  Bibupdate.out b1 b2 b3 b4 b5 b6
outdict    (1pr b4;rm b4;>b4)&
```

4. Finally, an entry must be made in the UNIX table 'crontab' so that bibupdate will be executed during the night.

```
program Bibin(input,output,bank,dict,scratch,dlist,PendingTray);
(* To ADD, CHANGE or REMOVE items,
instructions left in a PendingTray file 'pending',
actual changes made by running "Bibupdate.p" *)
(* written by Tony Heyes, Blind Mobility Research Unit,
Department of Psychology, The University,
Nottingham, U.K. *)

label 10;

const   LineLn = 70;
        RowLn  = 20;
        HiTag  = 10000;
        NonDate = -1066;

type    string = packed array [1..LineLn] of char;
        item = record
            authors,title1,title2,
            place1,place2 : string;
            date          : integer;
            key1,key2     : string
        end;
        word = packed array [1..20] of char;
        row = array [1..RowLn] of integer;
        dic = record
            name      : word;
            numbers   : row;
            cont      : boolean
        end;
        TagItem = record
            tag : integer;
            entry : item
        end;

var     empty,entry : item;
        bank : file of item;
        PendingTray,TempPendingTray : file of TagItem;
        dlist,scratch : text;
        dict : file of uic;
        TagEntry : TagItem;
        ch,AppendOption,ChangeOption,MainOption,HelpOption,
        SpecialOption : char;|chge : boolean;
        a,n,nn,count : integer;

procedure InlChar (var ch : char);
(* to read the first character of a word typed into the terminal *)
begin
    ch := input^;
    while not (ch in ['A'..'Z','a'..'z']) do
```

```
begin (* skips along until first character found *)
    get(input);
    if eoln(input)
    then
        begin
            writeln;
            write('ERROR: character required .... ')
        end;
        ch := input^
    end;
    while not eoln(input) do (* skips over rest of line *)
        get(input)
    end; (* of InlChar *)

procedure InlInt (var int : integer);
(* to read an integer and not cause a fatal error if a
character is given *)
var ch : char;
    a,OrdZero : integer;
    NegFound : boolean;
begin
    repeat (* skips along until integer is found *)
        get(input);
        if eoln(input)
        then
            begin
                writeln;
                write('ERROR: digit required .... ')
            end;
            ch := input^
        until ch in ['-','+', '0'..'9'];
        if ch = '-'
        then
            begin
                NegFound := true;
                get(input);
                ch := input^
            end
        else
            begin
                NegFound := false;
                if ch = '+'
                then
                    begin
                        get(input);
                        ch := input^
                    end
                end;
            end;
            a := 0;
            OrdZero := ord('0');
            repeat
                a := 10*a+ord(ch)-OrdZero;
                get(input);
                ch := input^
            until not (ch in ['0'..'9']);
            while not eoln(input) do (* skips over rest of line *)
                get(input);
            if NegFound
            then
                int := -a
            else
                int := a
            end; (* of InlInt *)

procedure VDUinString(var str : string);
(* to input from terminal *)

var i,n : integer;
    ch : char;
    AllCaps : boolean;
begin
    n := 0;
    AllCaps := true;
    repeat
        n := n+1;
        read(ch);
        if (n=1) and (ch = ' ')
        then
            n := 0;
        if (n=1) and (ch = '\')
        then
            begin (* defeat automatic shift with '\ ' *)
                AllCaps := false;
                n := 0
            end;
        if n<0
        then
            begin
                if AllCaps
                then
                    if ch in ['a'..'z']
```

```

        then
            ch := chr(ord(ch)-32);
        str[n] := ch
    end
until eoln(input);
for i:=n+1 to LineLn do
    str[i] := ' '
end; (* of VDUinString *)

procedure ScratchInStr(var str : string);
(* input from file scratch *)
var n,i : integer;
    ch : char;
begin
    if not eof(scratch)
    then
        begin
            n := 0;
            repeat
                read(scratch,ch);
            until (ch=';') or (eof(scratch));
            while (not eoln(scratch)) do
                begin
                    read(scratch,ch);
                    n := n+1;
                    str[n] := ch;
                end;
                if n+1<=LineLn
                then
                    for i:=n+1 to LineLn do
                        str[i] := ' ';
                    end
                end;
            end;
        end; (* of ScratchInStr *)

function ScratHoldsItems : boolean;
(* to inspect the SCRATCH FILE and check that ITEMS are complete *)
var count,LineNo : integer;
    FaultFound,HeadingError,NegFound : boolean;
procedure CheckLine;
var CharCount : integer;
    LineTooLong,BadLine : boolean;
begin
    LineNo := LineNo + 1;
    CharCount := 1;
    BadLine := false;
    LineTooLong := false;
    get(scratch);
    while (not eoln(scratch)) and (CharCount < LineLn + 9) do
        begin
            get(scratch);
            CharCount := CharCount + 1;
            if (CharCount = 9) and (scratch^ <> ':') then
                BadLine := true;
            end;
            if CharCount < 9 then BadLine := true;
            while not eoln(scratch) do
                begin
                    get(scratch);
                    if scratch^ <> ' ' then LineTooLong := true
                end;
            end;
            if BadLine then
                begin
                    FaultFound := true;
                    writeln('Line',LineNo : 4,' bad line ':' missing.')

```

```

begin
    if not NegFound then (* no ITEMS present *)
        begin
            HeadingError := true;
            writeln('SCRATCH does not contain ITEMS.')

```



```

        writeln(placel);
        writeln(place2);
        writeln(date:8,'          Item number :',n :5);
        writeln(key1);
        writeln(key2)
    end
end; (* of OutRecord *)

procedure GetReference(n : integer);
(* to count through bank to find an ITEM *)
begin
    if n<count
    then
        begin
            reset(bank);
            count := 1
        end;
    while (count < n) and (not eof(bank)) do
        begin
            count := count+1;
            get(bank)
        end;
    if eof(bank)
    then
        begin
            writeln;
            writeln(' You have only got',count -1,' Items.');
            writeln;
            goto l0
        end
    else
        OutRecord(bank^,n)
    end; (* of GetReference *)

procedure change(var entry : item; m : integer);
(* to change the mth. ITEM *)
var line : integer;
    DhOption,LineOption : char;
    str : string;
begin
    writeln;
    writeln;
    repeat
        write('Do you wish to DELETE or MODIFY .... ');
        InlChar(DhOption)
        until DhOption in ['D','d','H','h','m'];
        if DhOption in ['D','d']
        then
            begin
                empty;
                entry := empty
            end
        else
            begin
                writeln;
                writeln('You may REPLACE a line,');
                writeln('move to the NEXT line,');
                writeln('or SKIP to the end of the item. ');
                writeln;
                line := 0;
                repeat
                    line := line+1;
                until
                    write('Do you wish to APPEND, to CHANGE, ');
                    writeln('to use the SPECIAL facility, ');
                    write('or to FINISH .... ');
                    InlChar(MainOption)
                    until MainOption in ['A','a','C','c','S','s','F','f'];

                (* MainOption= S is a special facility,
                used for loading from 'scratch' to 'PendingTray' *)

                case MainOption of
                    'A','a': (* TO APPEND *)
                        begin
                            writeln;
                            repeat
                                write('Do you need help
                                [YES or NO] .... ');
                                InlChar(HelpOption)
                                until HelpOption in ['Y','y','N','n'];
                                if HelpOption in ['Y','y']
                                then
                                    begin
                                        writeln;
                                        writeln('NOTES.');
                                        write('(a) Authors and keywords separated');
                                        writeln(' by a comma ",,");
                                        write('(b) To remove the automatic conversion to ');
                                        writeln('upper case letters');
                                        write(' begin a line of text with');
                                        writeln(' a backslash "\.');
                end
            end
        end;
    with entry do
        case line of
            1: str := authors;
            2: str := title1;
            3: str := title2;
            4: str := placel;
            5: str := place2;
            6: ;
            7: str := key1;
            8: str := key2
        end; (* of case *)
        if line>6
        then
            begin
                writeln;
                writeln(str);
                writeln(output);
                repeat
                    write('REPLACE, NEXT line or SKIP to end .... ');
                    InlChar(LineOption)
                until LineOption in ['R','r','N','n','S','s'];
                writeln;
                if LineOption in ['R','r']
                then
                    begin
                        writeln('Type replacement line :');
                        writeln;
                        VDUinString(str);
                        with entry do

```

```

        case line of
            1: authors := str;
            2: title1 := str;
            3: title2 := str;
            4: placel := str;
            5: place2 := str;
            7: key1 := str;
            8: key2 := str
        end; (* of case *)
    end
end
else
    begin
        writeln('Date ',entry.date :4);
        writeln;
        repeat
            write('REPLACE, NEXT line or SKIP to end .... ');
            InlChar(LineOption)
        until LineOption in ['R','r','N','n','S','s'];
        if LineOption in ['R','r']
        then
            begin
                writeln('Type replacement date ');
                write(': ');
                InlInt(entry.date)
            end
        end;
        until ((line=8) or (LineOption in ['S','s']));
    end;
    writeln;
    writeln('Modified item reads : ');
    writeln;
    OutRecord(entry,m);
    writeln;
end; (* of change *)

begin (* MAIN PROGRAM *)
    count := HiTag;
    n := 1;
    reset(PendingTray);
    rewrite(TempPendingTray);
    while not eof(PendingTray) do
        begin (* copy down existing contents of file
        'PendingTray' *)
            TempPendingTray^ := PendingTray^;
            put(TempPendingTray);
            get(PendingTray)
        end;
        rewrite(PendingTray);
        reset(TempPendingTray);
        while not eof(TempPendingTray) do
            begin (* copy back 'PendingTray' and count contents *)
                PendingTray^ := TempPendingTray^;
                put(PendingTray);
                get(TempPendingTray);
                n := n+1
            end;
            rewrite(TempPendingTray);
        repeat
            writeln;
            repeat
                write('Do you wish to APPEND, to CHANGE, ');
                writeln('to use the SPECIAL facility, ');
                write('or to FINISH .... ');
                InlChar(MainOption)
                until MainOption in ['A','a','C','c','S','s','F','f'];

                (* MainOption= S is a special facility,
                used for loading from 'scratch' to 'PendingTray' *)

                case MainOption of
                    'A','a': (* TO APPEND *)
                        begin
                            writeln;
                            repeat
                                write('Do you need help
                                [YES or NO] .... ');
                                InlChar(HelpOption)
                                until HelpOption in ['Y','y','N','n'];
                                if HelpOption in ['Y','y']
                                then
                                    begin
                                        writeln;
                                        writeln('NOTES.');
                                        write('(a) Authors and keywords separated');
                                        writeln(' by a comma ",,");
                                        write('(b) To remove the automatic conversion to ');
                                        writeln('upper case letters');
                                        write(' begin a line of text with');
                                        writeln(' a backslash "\.');
                end
            end
        end;
    with entry do
        case line of
            1: str := authors;
            2: str := title1;
            3: str := title2;
            4: str := placel;
            5: str := place2;
            6: ;
            7: str := key1;
            8: str := key2
        end; (* of case *)
        if line>6
        then
            begin
                writeln;
                writeln(str);
                writeln(output);
                repeat
                    write('REPLACE, NEXT line or SKIP to end .... ');
                    InlChar(LineOption)
                until LineOption in ['R','r','N','n','S','s'];
                writeln;
                if LineOption in ['R','r']
                then
                    begin
                        writeln('Type replacement line :');
                        writeln;
                        VDUinString(str);
                        with entry do

```

```

write('c) Date must be a single integer number');
writeln(' eg. 1980. ');
write('d) If addresses are to be entered use the two');
writeln(' title lines; ');
write(' close pack but indicate new');
writeln(' lines with a backslash "\. ');
write('e) A personal local storage reference');
writeln(' may be kept on the 2nd. location line');
write(' but should be enclosed in square brackets; ');
writeln(' for example: [B:360]. ');
end;
repeat
  writeln;
  writeln('New item:- ');
  writeln;
  for a:=1 to 7 do
    write('-----I');
  writeln;
  with entry do
    begin
      writeln( 'Line of author names, or name of addressee : ');
      VDUinString(authors);
      writeln('First line of title or address : ');
      VDUinString(title1);
      writeln('Second line of title or address : ');
      VDUinString(title2);
      writeln('First line of reference location : ');
      VDUinString(place1);
      writeln('Second line of reference location : ');
      VDUinString(place2);
      writeln('Date - just the year - : ');
      InlInt(date);
      writeln('First line of keywords : ');
      VDUinString(key1);
      writeln('Second line of keywords : ');
      VDUinString(key2);
    end;
  writeln;
  OutRecord(entry,n);
  repeat
    writeln;
    repeat
      write( 'Do you wish to make a change [YES or NO] .... ');
      InlChar(ChangeOption)
      until ChangeOption in ['Y','y','N','n'];
      if ChangeOption in ['Y','y']
      then
        change(entry,n)
      until ChangeOption in ['N','n'];
      if entry.date <> NonDate
      then
        begin
          TagEntry.tag := HiTag;
          TagEntry.entry := entry;
          PendingTray^ := TagEntry;
          put(PendingTray);
          n := n+1
        end
      else
        begin
          writeln;
          writeln('Item withdrawn. ');
          writeln
        end;
      writeln;
    repeat
      write( 'Do you wish to append more items [YES or NO] .... ');
      InlChar(AppendOption)
      until AppendOption in ['Y','y','N','n'];
      until AppendOption in ['N','n']
    end; (* of Append option *)

'C','c': (* TO CHANGE *)
begin;
  writeln;
  repeat
    write('Do you need help [YES or NO] .... ');
    InlChar(HelpOption)
    until HelpOption in ['Y','y','N','n'];
    if HelpOption in ['Y','y']
    then
      begin
        writeln;
        writeln( 'You MUST know the ITEM NUMBERS of the ITEMS you wish to
        change.' );
        writeln( 'If you do not, leave this program and run "bibout" to
        find them.' );
        writeln( 'Changes do not take place immediately, they stay in the
        PENDING' );
        writeln('tray until the "update" program is run. ');
      end;
    end;
  end;
  repeat
    writeln( 'If an ITEM is changed more than once only the last
    version survives.'
    end;
    repeat
      10: writeln;
      chge := false;
      writeln('Type 0 if no ITEM needs changing, otherwise
      type');
      write('the ITEM number... ');
      InlInt(nn);
      if nn<0
      then
        begin
          writeln;
          writeln('No negative numbered ITEMS')
        end;
      if nn > 0
      then
        begin
          writeln;
          GetReference(nn);
          if not eof(bank)
          then
            begin
              entry := bank^;
              repeat
                writeln;
                repeat
                  write('Do you wish to change this item [YES or NO] .... ');
                  InlChar(ChangeOption)
                  until ChangeOption in ['Y','y','N','n'];
                  if ChangeOption in ['Y','y']
                  then
                    begin
                      change(entry,nn);
                      chge := true
                    end
                  until ChangeOption in ['N','n'];
                  TagEntry.tag := nn;
                  TagEntry.entry := entry;
                  if chge
                  then
                    begin
                      PendingTray^ := TagEntry;
                      put(PendingTray);
                      n := n+1
                    end
                  end
                end;
                writeln;
                until nn = 0
              end; (* of Change option *)

'S','s': (* To move from text file 'scratch' to 'PendingTray' *)
begin
  writeln;
  write('This option moves the contents of the ');
  writeln('SCRATCH file into the PENDING tray. ');
  write('It can be used to copy selected ITEMS from one');
  writeln(' bibliography to another. ');
  write('OR, it can be used to reinstate ITEMS ');
  writeln('which have been changed by the editor. ');
  writeln;
  repeat
    write('Do you wish these items to be APPENDED, REINSTATED or
    NO ACTION .... ');
    InlChar(SpecialOption)
    until SpecialOption in ['A','a','N','n','R','r'];
    if SpecialOption in ['A','a','R','r']
    then
      begin
        reset(scratch);
        writeln;
        (* now check that scratch holds ITEMS in
        the correct form *)
        if (not eof(scratch)) and
        ScratchHoldsItems
        then
          begin
            while not eof(scratch) do
              begin
                with entry do
                  begin
                    ScratchInStr(authors);
                    ScratchInStr(title1);
                    ScratchInStr(title2);
                    ScratchInStr(place1);
                    ScratchInStr(place2);
                    read(scratch,date);

```

```

repeat
  read(scratch,ch)
until ch = ':';
readln(scratch,TagEntry.tag);
writeln(n,' Dated ',date,' Item number ',TagEntry.tag);
ScratchInStr(key1);
ScratchInStr(key2);
end;
if SpecialOption in ['A','a'] then
  TagEntry.tag := HiTag;
  TagEntry.entry := entry;
  PendingTray^ := TagEntry;
  put(PendingTray);
  n := n+1;
  if not eof(scratch)
  then
    get(scratch)
  end;
  rewrite(scratch)
end
end; (* of Special option *)

'F','f': begin
  writeln;
  writeln('Number of ITEMS now in Pending
  Tray =',n-1 :5);
  writeln
end
end (* of case "MainOption" *)
until MainOption in ['F','f']
end. (* end of program Bibin.p *)

```

```

program Bibout(input,output,bank,dict,scratch,dlist,PendingTray);
(* To call down items from the bibliography *)
(* written by Tony Heyes, Blind Mobility Research Unit,
Department of Psychology, The University,
Nottingham, U.K.. *)

```

```
label 10;
```

```

cc      LineLn = 70;
      RowLn = 20;
      HiTag = 10000;
      LinesPerPage = 64 ;
      VDULinesPerPage = 24;

```

```

type  string = packed array [1..LineLn] of char;
      item = record
        authors,title1,title2,
        place1,place2 : string;
        date          : integer;
        key1,key2     : string
      end;
      word = packed array [1..20] of char;
      row = array [1..RowLn] of integer;
      dic = record
        name      : word;
        numbers   : row;
        cont      : boolean
      end;
      link = ^DicLine;
      DicLine = record
        val : integer;
        next : link
      end;

```

```

var FileAssigned : boolean;
    Bank,PendingTray : file of item;
    dlist,AddressFile,scratch : text;
    dict : file of dic;
    FirstLink,SecondLink,ThirdLink,pt1,here : link;
    low,high,n,NumSoFar,
    LineNo,AddLineNo,count,TopItem,NFromDict,NumI : integer;
    device,FileStyle,MainOpt,NDOption,LogicAction : char;

```

```

procedure InlChar (var ch : char);
(* to read the first character of a word typed into the terminal *)
begin
  ch := input^;
  while not (ch in ['A'..'Z','a'..'z']) do
    begin
      (* skips along until first character found *)
      get(input);
      if eoln(input)
      then
        begin

```

```

        writeln;
        write('ERROR: character required .... ')
      end;
      ch := input^
    end;
    while not eoln(input) do (* skips over rest of line *)
      get(input)
    end; (* of InlChar *)

procedure InlInt (var f : text; var int : integer);
(* to read an integer and not cause a fatal error if a character
is given *)

var ch : char;
    a,OrdZero : integer;
    NegFound : boolean;
begin
  repeat (* skips along until integer is found *)
    get(f);
    if eoln(f)
    then
      begin
        writeln;
        write('ERROR: digit required .... ')
      end;
      ch := f^
    until ch in ['-','+','0'..'9'];
    if ch='-+'
    then
      begin
        NegFound := true;
        get(f);
        ch := f^
      end
    else
      begin
        NegFound := false;
        if ch='+'
        then
          begin
            get(f);
            ch := f^
          end
        end;
      a := 0;
      OrdZero := ord('0');
      repeat
        a := 10*a+ord(ch)-OrdZero;
        get(f);
        ch := f^
      until not (ch in {'0'..'9'});
      while not eoln(f) do (* skips over rest of line *)
        get(f);
      if NegFound
      then
        int := -a
      else
        int := a
      end; (* of InlInt *)

procedure SkipToEndOfPage(PageLines : integer;
var where : text);
begin
  while LineNo < PageLines do
    begin
      writeln(where);
      LineNo := LineNo+1
    end;
    LineNo := 0
  end; (* of SkipToEndOfPage *)

procedure GetRef(n : integer; destination : char);
var a,CharCount,LineInQuestion,NOfCommas,WordLength : integer;
    line : string;
    DoubleSpace,InBrackets,KeepNextCap,
    something,KeepAllCaps,woops : boolean;
    ch,LastCh : char;
begin
  if n<count
  then
    begin
      reset(bank);
      count := 1
    end;
    while (count < n) and (not eof(bank)) do
      begin
        count := count+1;
        get(bank)
      end;
    if eof(bank)

```

```

then
  begin
    writeln;
    writeln(' You have only got',count -1,' Items.');
```

```

    writeln;
    goto 10
  end
else
  with bank^ do
    begin
      case destination of
        'T','t': (* Output to terminal *)
          begin
            if (VDULinesPerPage-LineNo < 9)
              then
                SkipToEndOfPage(VDULinesPerPage,output);
            for a:=1 to 7 do
              write('-----I');
              writeln;
              writeln(authors);
              writeln(title1);
              writeln(title2);
              writeln(place1);
              writeln(place2);
              writeln(date:8,'          Item number :',n :5);
              writeln(key1);
              writeln(key2);
              LineNo := LineNo + 9
            end; (* of 'T' *)
          'I','i': (* Output to scratch file *)
          begin
            if LinesPerPage-LineNo < 9
              then
                SkipToEndOfPage(LinesPerPage,scratch);
            for a:=1 to 7 do
              write(scratch,'-----I');
              writeln(scratch,'-----');
              writeln(scratch,'Names      :',authors);
              writeln(scratch,'Details  :',title1);
              writeln(scratch,'         :',title2);
              writeln(scratch,'         :',place1);
              writeln(scratch,'         :',place2);
              writeln(scratch,date:14,'          Item number:',n :5);
              writeln(scratch,'Keywords:',key1);
              writeln(scratch,'         :',key2);
              LineNo := LineNo + 9
            end; (* of 'I' *)
          'E','e': (* Output to scratch file in envelope label format.
                    Only for addresses. *)
          begin
            writeln(AddressFile);
            AddLineNo := AddLineNo +1;
            woops := true;
            for LineInQuestion:=1 to 2 do
              begin
                DoubleSpace := false;
                LastCh := ':'; (* initail value *)
                CharCount := 0;
                writeln(AddressFile);
                AddLineNo := AddLineNo +1;
                write(AddressFile,' ');
                if LineInQuestion=1
                  then
                    line := title1
                  else
                    line := title2;
                while (CharCount<LineLn) and not DoubleSpace do
                  begin
                    CharCount := CharCount+1;
                    ch := line[CharCount];
                    if ch='\ '
                      then
                        begin
                          woops := false;
                          writeln(AddressFile);
                          AddLineNo := AddLineNo +1;
                          write(AddressFile,' ');
                        end
                      else
                        write(AddressFile,ch);
                        DoubleSpace := (ch=' ') and (LastCh=' ');
                        LastCh := ch
                      end
                    end;
                    while (AddLineNo mod 8) <> 0 do
                      begin
                        writeln(AddressFile);
                        AddLineNo := AddLineNo + 1
                      end;
                    if woops
                      then
                        begin
                          writeln;
                          writeln;
                          write('An attempt to output a reference');
                          writeln(' in address format.');
```

```

                          writeln;
                          writeln;
                          write( 'Addresses must be close-packed on the two' );
                          writeln(' title lines.');
```

```

                          writeln( 'Use the backslash "\" as line separator.' );
                          writeln;
                          rewrite(scratch);
                          FileAssigned := false;
                          goto 10
                        end
                      end; (* of 'E' *)
                    'R','r': (* Output in format for wordprocessor NROFF *)
                    begin (* firstly the author line *)
                      writeln(scratch,'nr');
                      (* this is an NROFF macro *)
                      write(scratch,'\ ');
                      (* bold lettering command *)
                      DoubleSpace := false;
                      KeepAllCaps := false;
                      woops := false;
                      LastCh := ':'; (* initial value *)
                      CharCount := 0;
                      NOfCommas := 0;
                      if authors[1]='\ '
                        then
                          begin
                            KeepAllCaps := true;
                            CharCount := CharCount+1
                          end;
                          while (CharCount<LineLn)
                            and not DoubleSpace do
                              begin
                                CharCount := CharCount+1;
                                ch := authors[CharCount];
                                if ch=', '
                                  then
                                    NOfCommas := NOfCommas+1;
                                    DoubleSpace := (ch=' ') and (LastCh=' ');
                                    LastCh := ch
                                end;
                                DoubleSpace := false;
                                LastCh := ':';
                                CharCount := 0;
                                while (CharCount<LineLn) and not DoubleSpace do
                                  begin
                                    CharCount := CharCount+1;
                                    ch := authors[CharCount];
                                    if (ch in ['A'..'Z']) and (LastCh in ['A'..'Z'])
                                      and not KeepAllCaps
                                      then
                                        write(scratch,chr((ord(ch)+32)))
                                      else
                                        if ch=', '
                                          then
                                            begin
                                              if NOfCommas=1
                                                then
                                                  write(scratch,' & ')
                                                else
                                                  write(scratch,' ');
                                              NOfCommas := NOfCommas-1
                                            end
                                          else
                                            write(scratch,ch);
                                            DoubleSpace := (ch=' ') and (LastCh=' ');
                                            LastCh := ch
                                          end;
                                        writeln(scratch,'(,date : 4,)\:');
                                        for LineInQuestion :=1 to 4 do
                                          begin
                                            (* title and place lines *)
                                            KeepNextCap := true;
                                            KeepAllCaps := false;
                                            case LineInQuestion of
                                              1: line := title1;
                                              2: begin
                                                  line := title2;
                                                  KeepNextCap := false
                                                end;
                                              3: line := place1;
                                              4: begin
                                                  line := place2;
                                                  CharCount := 0;
                                                  InBrackets := false;
                                                  repeat
                                                    CharCount := CharCount+1;
                                                    if line[CharCount]='['
                                                      then

```

```

InBrackets := true;
if InBrackets
then
  if line[CharCount]=']'
  then
    begin
      line[CharCount] := ' ';
      InBrackets := false
    end;
  if InBrackets
  then
    line[CharCount] := ' '
  until CharCount=LineLn
  end
end; (* of case LineInQuestion *)
CharCount := LineLn;
repeat
  CharCount := CharCount-1
until (CharCount=1) or (line[CharCount]<>' ');
if CharCount<LineLn
then
  line[CharCount+1] := '!'; (* a silly character '
(* placed at the end of the character string *)
WordLength := 0;
if CharCount>1
then
  repeat
    CharCount := CharCount-1;
    if line[CharCount]<>' '
    then
      begin
        if line[CharCount] in ['A'..'Z']
        then
          WordLength := WordLength+1
        end
      end
    else
      begin
        if not (WordLength in {2,3})
        then
          line[CharCount] := '-';
          (* another silly char fills up spaces
          before words which keep caps. *)
          WordLength := 0
        end
      end
    until CharCount=1;
    CharCount := 0;
    something := false;
    if line[1]='\'
    then
      begin
        KeepAllCaps := true;
        CharCount := CharCount+1
      end;
      ch := '!'; (* initial value *)
      while (CharCount < LineLn) and
        (line[CharCount+1] <> '!') do
        begin
          CharCount := CharCount+1;
          LastCh := ch;
          ch := line[CharCount];
          if not ((LastCh in ['-', '']) and
            (ch in ['-', '']))
          then
            begin
              if (ch in ['A'..'Z']) and not KeepNextCap
              then
                ch := chr((ord(ch)+32));
              if ch in ['A'..'Z']
              then
                KeepNextCap := false;
              if ch='\'
              then
                woops := true; (* its an address *)
              if ch='-'
              then
                begin
                  ch := ' ';
                  if (LineInQuestion in {3,4})
                  then
                    KeepNextCap := true
                  end;
                end;
              if (ch in ['1'..'9'])
              then
                KeepNextCap := false;
              if (ch<>' ') and (ch<>'!')
              then
                something := true;
              if something
              then
                write(scratch,ch)
              end
            end
          end
        end;
        if something
        then
          writeln(scratch)
        end;
        if woops
        then
          begin
            writeln;
            writeln;
            write('An attempt to output addresses in');
            writeln(' reference format.');
```

```

procedure join(var p1 : link; p2 : link; which : char);
var
  continue : boolean;
  q,qp,pt1,pt2,pt3 : link;
begin
  pt1 := p1;
  pt2 := p2;
  continue := (pt1<>nil) and (pt2<>nil);
  qp := nil;
  case which of
    'A','a':      (* AND *)
      begin
        while continue do
          begin
            if pt1^.val>pt2^.val
            then
              begin
                pt3 := pt1;
                pt1 := pt2;
                pt2 := pt3;
              end;
            if pt2^.val>pt1^.val
            then
              begin
                pt1 := pt1^.next;
                continue := pt1<>nil;
              end
            else
              if pt1^.val=pt2^.val
              then
                begin
                  new(q);
                  q^.val := pt1^.val;
                  q^.next := qp;
                  qp := q;
                  pt1 := pt1^.next;
                  pt2 := pt2^.next;
                  continue := (pt1<>nil) and
                    (pt2<>nil);
                end
              end
            end;
          end;
        end;
      end;
    (* of AND *)
    'O','o':      (* OR *)
      begin
        while continue do
          begin
            if pt1^.val>pt2^.val
            then
              begin
                pt3 := pt1;
                pt1 := pt2;
                pt2 := pt3;
              end;
            if pt1^.val<pt2^.val
            then
              begin
                new(q);
                q^.val := pt1^.val;
                q^.next := qp;
                qp := q;
                pt1 := pt1^.next;
                continue := pt1<>nil;
              end
            else
              if pt1^.val=pt2^.val
              then
                begin
                  new(q);
                  q^.val := pt1^.val;
                  q^.next := qp;
                  qp := q;
                  pt1 := pt1^.next;
                  pt2 := pt2^.next;
                  continue := (pt1<>nil) and
                    (pt2<>nil);
                end
              end
            end;
          end;
        end;
        if pt1=nil
        then
          pt1 := pt2;
        while pt1<>nil do
          begin
            new(q);
            q^.val := pt1^.val;
            q^.next := qp;
            qp := q;
            pt1 := pt1^.next;
          end
        end
      end;
    (* of OR *)
  end;
end;

```

```

'N','n':      (* NOT *)
begin
  while continue do
    begin
      if pt1^.val>pt2^.val
      then
        begin
          pt2 := pt2^.next;
          continue := pt2<>nil;
        end
      else
        if pt1^.val<pt2^.val
        then
          begin
            new(q);
            q^.val := pt1^.val;
            q^.next := qp;
            qp := q;
            pt1 := pt1^.next;
            continue := pt1<>nil;
          end
        else
          if pt1^.val=pt2^.val
          then
            begin
              pt1 := pt1^.next;
              pt2 := pt2^.next;
              continue := (pt1<>nil) and
                (pt2<>nil);
            end
          end;
        while pt1<>nil do
          begin
            new(q);
            q^.val := pt1^.val;
            q^.next := qp;
            qp := q;
            pt1 := pt1^.next;
          end
        end (* of NOT *)
      end;
    end;
  Rewind(qp);
  pl := qp;
end; (* of join *)

```

```

procedure OutList(ptr : link; var aa : integer);

```

```

var p : link;
begin
  p := ptr;
  aa := 0;
  writeln;
  while p<>nil do
    begin
      aa := aa+1;
      if aa mod 13 = 0
      then
        writeln(p^.val :5)
      else
        write(p^.val :5);
      p := p^.next;
    end;
  writeln;
  writeln;
end; (* of OutList *)

```

```

procedure DictList(var where : text);
(* TO LIST DICTIONARY *)

```

```

const NoOfLines = 64;
      WordsPerLine = 4; (* Change constants to suit page size *)
                          (* See also line 700 *)

```

```

type list = array[1..384] of word;

```

```

var
  num,i : integer;
  OldEntry : dic;
  WordList : list;
begin
  reset(dict);
  rewrite(dlist);
  i := 0;
  while not eof(dict) do
    begin
      for num:=1 to NoOfLines*WordsPerLine do
        begin
          OldEntry := dict^;
          while (dict^.cont=true)and(not eof(dict)) do
            get(dict);
            if not eof(dict)
            then

```

```

begin
  WordList[num] := OldEntry.name;
  get(dict)
end
else
  WordList[num] := '          ';
end;
for num:=1 to NoOfLines do
  writeln(where,WordList[num],WordList[NoOfLines+num],
          WordList[2*NoOfLines+num],
          WordList[3*NoOfLines+num]);
  (* Extent this list for more words per line *)
  i := i+NoOfLines*WordsPerLine
end;
writeln;
write('Dictionary written to file. ');
writeln(' To obtain a hard copy run "outdict." ');
(* 'outdict' simply prints out the file 'dlist'. *)
writeln
end; (* of DictList *)

procedure TwoCols (var F,G : text);

const rows = 8;
      TwiceRows = 16;
      cols = 40;

type ChLink = ^chstack;
chstack = record
  ch : char;
  next : ChLink;
end;
lines = array[1..TwiceRows] of ChLink;

var pt,here : ChLink;
    lin,StartLin : lines;
    LineNo,CharNo : integer;
    ch : char;

procedure reverse(var ptr : ChLink);

var p,q,pt : ChLink;
begin
  p := ptr;
  pt := nil;
  while p <> nil do
    begin
      new(q);
      q^.ch := p^.ch;
      q^.next := pt;
      pt := q;
      p := p^.next
    end;
  ptr := pt
end; (* of reverse *)

begin
  reset(F);
  if not eof(F)
  then
    begin
      begin
        page(G);
        writeln;
        writeln('Output in two column "Xerox" label format. ');
        writeln
      end;
      while not eof(F) do
        begin
          mark(here);
          for LineNo := 1 to 2*rows do
            begin
              StartLin[LineNo] := nil;
              if not eof(F) then
                while not eoln(F) do
                  begin
                    read(F,ch);
                    new(lin[LineNo]);
                    lin[LineNo]^ch := ch;
                    lin[LineNo]^next := StartLin[LineNo];
                    StartLin[LineNo] := lin[LineNo]
                  end;
              if not eof(F)
              then
                readln(F);
                reverse(StartLin[LineNo]);
            end;
          end;
          for LineNo := 1 to rows do
            begin
              CharNo := 0;
              pt := StartLin[LineNo];
              while (pt <> nil) and (CharNo < cols) do
                begin
                  write(G,pt^.ch);
                  pt := pt^.next;
                  CharNo := CharNo + 1
                end;
              pt := StartLin[LineNo + rows];
              if pt <> nil
              then
                while CharNo < cols do
                  begin
                    write(G,' ');
                    CharNo := CharNo + 1
                  end;
                while pt <> nil do
                  begin
                    ch := pt^.ch;
                    write(G,ch);
                    pt := pt^.next
                  end;
                  writeln(G)
                end;
                release(here);
            end
          end; (* of TwoCols *)

procedure GetFromDict(var FirstWord,NumWords : integer);

var
  ch,action,option : char;
  n,ChCount,PointerNum,NumberFound : integer;
  name,signame : word;
  AllCaps : boolean;

begin
  writeln;
  AllCaps := true;
  ChCount := 0;
  write('Enter word required or [.] .... ');
  repeat
    read(ch)
  until ch<>' ';
  if ch='\ '
  then
    begin
      AllCaps := false;
      read(ch)
    end;
  if ch='.'
  then
    begin (* "action" *)
      while not eoln(input) do
        get(input);
        repeat
          writeln;
          writeln('Do you wish to LOOK UP the selected string,
                  to RESTART the ');
          write('selection or to QUIT the dictionary .... ');
          InlChar(action)
          until action in ['L','l','R','r','Q','q']
        end
      else
        begin (* word *)
          action := 'W';
          repeat
            ChCount := ChCount + 1;
            if ChCount > 1
            then
              read(ch);
              if AllCaps and (ch in ['a'..'z'])
              then
                name[ChCount] := chr(ord(ch)-32)
              else
                name[ChCount] := ch
              until eoln(input) or (ChCount = 20);
              if not eoln(input)
              then
                readln;
                for n:=ChCount+1 to 20 do
                  name[n] := ' '
                end;
              if action in ['L','l']
              then
                FirstWord := -1 (* look up *)
              else
                if action in ['R','r']
                then
                  FirstWord := -2 (* restart *)
                else
                  if action in ['Q','q']
                  then
                    FirstWord := 0 (* quit *)
                  else

```

```

if name='***
  (* special word *)
  then
  begin
  writeln;
  writeln('*** ALL ITEMS ***');
  writeln;
  repeat
  write('Is this correct [YES or NO] .... ');
  InlChar(option)
  until option in ['Y','y','N','n'];
  if option in ['Y','y']
  then
  FirstWord := HiTag
  else
  GetFromDict(FirstWord,NumWords)
  end
  else
  begin (* a real word *)
  reset(dict);
  NumberFound := 0;
  PointerNum := 0;
  writeln;
  signame := ' ';
  while (name >= signame) and not eof(dict) do
  begin
  if name=signame
  then
  begin
  writeln(dict^.name);
  NumberFound := NumberFound+1
  end;
  while (dict^.cont=true) do
  get(dict);
  if (PointerNum > 0) and not eof(dict)
  then
  get(dict);
  PointerNum := PointerNum+1;
  for n:=1 to ChCount do
  signame[n] := dict^.name[n];
  for n:=ChCount+1 to 20 do
  signame[n] := ' ';
  end;
  writeln;
  if NumberFound=0
  then
  begin
  writeln( 'Word not found in your dictionary; try again.' );
  writeln;
  GetFromDict(FirstWord,NumWords)
  end
  else
  begin
  repeat
  if NumberFound = 1
  then
  write( 'Is this word correct [YES or NO] .... ' )
  else
  write( 'Are ALL these words required [YES or NO] .... ' );
  InlChar(option)
  until option in ['Y','y','N','n'];
  if option in ['Y','y']
  then
  begin
  FirstWord := PointerNum -
                NumberFound;
  end
  else
  GetFromDict(FirstWord,NumWords)
  end
  end
  end;
end; (* of GetFromDict *)

begin (* MAIN PROGRAM *)
rewrite(scratch);
rewrite(AddressFile);
reset(bank);
count := HiTag;
LineNo := 0;
AddLineNo := 0;
FileAssigned := false;
writeln;
writeln('To retrieve ITEMS from the BIBLIOGRAPHY. ');
(* TO SEARCH BY AUTHORS and KEYWORDS *)
writeln;
reset(dlist);
if dlist^ = ''
then
  InlInt(dlist,TopItem)
else
  begin
  TopItem := 0;
  writeln('Counting, please wait. ');
  writeln;
  while not eof(bank) do
  begin
  TopItem := TopItem +1;
  get(bank)
  end;
  rewrite(dlist);
  writeln(dlist,'-',TopItem : 5);
  writeln(dlist);
  writeln(dlist);
  writeln(dlist,'Your DICTIONARY must first be compiled by running');
  writeln(dlist,' the HARD COPY option of ''bibout''. ');
  writeln(dlist);
  writeln(dlist);
  writeln('The BIBLIOGRAPHY currently holds ',TopItem,' ITEMS. ');
  repeat
  writeln;
  10: repeat
  writeln( 'Do you wish to obtain a HARD COPY of the current dictionary, '
  write('to SEARCH for items or to FINISH .... ');
  InlChar(MainOpt)
  until MainOpt in ['H','h','S','s','F','f'];
  writeln;
  if MainOpt in ['H','h']
  then
  begin
  DictList(dlist);
  MainOpt := 'F'
  end;
  if MainOpt in ['S','s']
  then
  begin
  repeat
  writeln;
  writeln('Do you wish to search by item NUMBER');
  write('or by use of the DICTIONARY .... ');
  InlChar(NDOption)
  until NDOption in ['N','n','D','d'];
  writeln;
  repeat
  writeln;
  write('Output to TERMINAL or to scratch FILE .... ');
  InlChar(device)
  until device in ['T','t','F','f','S','s'];
  writeln;
  if device in ['T','t']
  then
  FileStyle := 'T';
  if (device in ['F','f','S','s']) and not FileAssigned
  then
  repeat
  writeln('Is the desired output');
  write('an ITEM list, ');
  writeln(' ' 'the full item being given' ');
  write('a REFERENCE list, ');
  writeln(' ' 'only the reference part being given' ');
  write('or an address list suitable');
  write(' for ENVELOPE addressing .... ');
  InlChar(FileStyle);
  FileAssigned := true
  until FileStyle in ['I','i','R','r','E','e'];
  if FileStyle in ['R','r']
  then
  begin
  writeln(scratch,'.hy 0'); (* NROFF commands *)
  writeln(scratch,'.na');
  writeln(scratch,'.sp 2');
  writeln(scratch,'.de nr');
  writeln(scratch,'.sp');
  writeln(scratch,'.ne 6');
  writeln(scratch,'.ti -5');
  writeln(scratch,'..');
  writeln(scratch,'.ne 10');
  writeln(scratch,'\:References.\:');
  writeln(scratch,'.sp 2');
  writeln(scratch,'.in +5');
  end;
  writeln;
  case NDOption of
  'D','d' : begin
  writeln('Words are looked up in ');
  writeln('the dictionary and a list of reference numbers ');
  writeln('containing the given word is shown on the terminal. ');
  writeln;
  write( 'The special "word", [***] will match with all the words' );
  end;
  end;
end;

```



```

writeln(' in the dictionary. ');
writeln;
write('Logical combination of ');
writeln('author and keywords continue until you wish ');
writeln('to terminated the search. ');
writeln;
writeln('To terminate a search answer the prompt with a full
stop [.]. ');
writeln;

repeat
writeln;
writeln('New sequence. ');
writeln;
NumSoFar := 0;
mark(here);
GetFromDict(NFromDict, NumW);
if NFromDict > 0 (* a real word *)
then
begin
GetDict(NFromDict, FirstLink);
if NumW > 1
then
repeat
NFromDict := NFromDict + 1;
GetDict(NFromDict, SecondLink);
join(FirstLink, SecondLink, 'O');
NumW := NumW - 1
until NumW = 1;
OutList(FirstLink, NumSoFar);
while NFromDict > 0 do
begin
GetFromDict(NFromDict, NumW);
if NFromDict > 0 (* a real word *)
then
begin
GetDict(NFromDict, SecondLink);
if NumW > 1
then
repeat
NFromDict := NFromDict + 1;
GetDict(NFromDict, ThirdLink);
join(SecondLink, ThirdLink, 'O');
NumW := NumW - 1
until NumW = 1;
OutList(SecondLink, NumSoFar);
repeat
? );
InlChar(LogicAction)
until LogicAction in ['A', 'a', 'O',
'o', 'N', 'n'];
join(FirstLink, SecondLink,
LogicAction);
OutList(FirstLink, NumSoFar)
end
end;
if ((NumSoFar > 0) and (NFromDict = -1))
then (* look up *)
begin
writeln;
writeln('Search in progress for', NumSoFar : 8, ' Items');
writeln;
ptl := FirstLink;
while ptl <> nil do
begin
GetRef(ptl^.val, FileStyle);
ptl := ptl^.next
end;
if FileStyle in ['I', 'i', 'R', 'r',
'E', 'e']
then
begin
writeln;
writeln('ITEMS written to SCRATCH FILE. ');
writeln;
end;
release(here)
end;
end
until NFromDict=0 (* quit *)
end;
'N', 'n' : begin (* TO SEARCH BY NUMBER *)
writeln;
writeln('ITEMS may be called by number. ');
writeln('A whole block of ITEMS may be called. ');
writeln('to do this answer this prompt with');
writeln(' minus one [-1]. ');
writeln;
writeln('To quit: answer prompt with a zero [0]. ');
repeat
writeln;
write('Number of ITEM to be referenced..... ');

```

```

InlInt(input, n);
writeln;
if n = -1
then
begin
writeln;
writeln('To output a block of
ITEMS. ');
writeln('Give the LOW ITEM number, then the HIGH number. ');
write('LOW number .... ');
InlInt(input, low);
write('HIGH number .... ');
InlInt(input, high);
if (low=0) or (high=0)
then
begin (* an escape *)
low := 1;
high := 0;
n := 0
end;
if low <= high
then
begin
writeln;
writeln('Search in progress');
writeln;
for n:=low to high do
GetRef(n, FileStyle)
end
end
else
if n > 0
then
begin
writeln;
writeln('Search in progress. ');
writeln;
GetRef(n, FileStyle)
end
until n=0
end (* of case NDOption *)
end
until MainOpt in ['F', 'f'];
if FileStyle in ['R', 'r']
then
begin
writeln(scratch, '.in -5');
writeln;
writeln('The output file 'scratch' contains the references and
the ');
writeln('instructions for the word processing program
'nroff''. ');
writeln;
writeln('An attempt has been made to reintroduce lower case
letters. ');
writeln('To obtain your output run 'nroff scratch' ');
writeln;
writeln('If all is not well edit scratch and run 'nroff
scratch' again. ');
writeln;
writeln('When all is correct get the hard copy output
by ');
writeln('running 'nroff scratch |lpr''. ');
writeln
end;
if FileStyle in ['E', 'e']
then
TwoCols(AddressFile, scratch);
writeln;
writeln;
writeln('FINISHED. ');
writeln
end. (* of program Bibout.p *)

```

```

program Bibupdate(input, output, bank, dict, scratch,
dlist, PendingTray, TempBank);
(* A non-interactive program which moves the contents of
'PendingTray' to the bibliography. Clever systems run this program
at night.
TempBank is made external because it grows to be as large as bank.
Diagnostics are written to 'scratch'.
Written by Tony Heyes, Blind Mobility Research Unit,
Department of Psychology, The University,
Nottingham, U.K.. *)
const LineLn = 70;
RowLn = 20;

```

```

heap = 200;
HiTag = 10000;
stack = 50;
NonDate = -1066;

type
  string = packed array [1..LineLn] of char;
  item = record
    authors,title1,title2,
    place1,place2 : string;
    date : integer;
    key1,key2 : string
  end;
  word = packed array [1..20] of char;
  row = array [1..RowLn] of integer;
  TagItem = record
    tag : integer;
    entry : item
  end;
  point = ^CoreTagItem;
  CoreTagItem = record
    TagEntry : TagItem;
    next : point
  end;
  dic = record
    name : word;
    numbers : row;
    cont : boolean
  end;
  link = ^dentry;
  dentry = record
    dline : dic;
    next : link
  end;

var
  bank,TempBank,addition : file of item;
  LastOne : item;
  PendingTray,correction : file of TagItem;
  first,here,p,pt,newp : link;
  efirst,now,ept,e,ewep : point;
  dlist,scratch : text;
  TempDict,dict : file of dic;
  GotFromCore,dlistOK,InitialBuild,continue,move,same : boolean;
  n,TopItem,m,corr,reps,add,OldTotal : integer;

procedure FromCore;

var p : link;
begin
  writeln(scratch,' FromCore');
  rewrite(dict);
  GotFromCore := true;
  p := first;
  while p<>nil do
    begin
      dict^ := p^.dline;
      put(dict);
      p := p^.next
    end
  end;
end; (* of FromCore *)

procedure build(entry : item;n : integer);
  (* TO BUILD THE DICTIONARY *)

var
  str : string;
  NewEntry,OldEntry : dic;
  l,let,line,i : integer;
  same,space,AlreadyHad,WordFound,LastWord : boolean;
begin
  for line:=1 to 3 do
    begin
      case line of
        1: str := entry.authors;
        2: str := entry.key1;
        3: str := entry.key2
      end;
      l := 0;
      let := 0;
      if not ((str[1]=' ')and(str[2]=' '))
        then
          repeat (* not empty line *)
            let := let+1;
            LastWord := (((str[let]=' ') and
              (str[let+1]=' '))
              or (let=LineLn-1));
            WordFound := ((str[let]='.') or LastWord);
            if not WordFound
              then
                begin
                  l := l+1;
                  if (l=1) and (str[let]=' ') then
                    l := 0
                end
            end;
          until LastWord
        end
      else
        begin
          if l<21 then
            NewEntry.name[l] := str[let]
          end
        end
      end
    end
  for i:=l+1 to 20 do
    NewEntry.name[i] := ' ';
    (* fill up with spaces *)
  end
  if InitialBuild
    then
      begin
        (* first entry *)
        NewEntry.numbers[1] := n;
        for i:=2 to RowLn do
          NewEntry.numbers[i] := 0;
        end
        NewEntry.cont := false;
        new(p);
        p^.dline := NewEntry;
        p^.next := nil;
        first := p;
        l := 0;
        InitialBuild := false
      end
    else
      begin
        OldEntry := first^.dline;
        pt := first;
        (* move pt past all words before the new entry *)
        while (pt^.next<>nil) and
          (NewEntry.name>pt^.next^.dline.name) do
          pt := pt^.next;
          OldEntry := pt^.dline;
          same := OldEntry.name=NewEntry.name;
          space := OldEntry.numbers[RowLn]=0;
          AlreadyHad := false;
          if same then
            begin
              i := RowLn;
              while OldEntry.numbers[i] = 0 do
                i := i-1;
              end
              if OldEntry.numbers[i] = n then
                AlreadyHad := true
              end;
            end
          if not AlreadyHad then
            begin (* if keyword has author name only
              'one dic'if (same and (not space))
              then
                begin
                  (* new entry already in dict but no space in the string *)
                  OldEntry.cont := true;
                  pt^.dline := OldEntry
                end;
              end
            end
          if same and space
            then
              begin
                (* new entry already in dict AND space in the number string *)
                i := 0;
                repeat
                  i := i+1
                until OldEntry.numbers[i]=0;
                OldEntry.numbers[i] := n;
                pt^.dline := OldEntry
              end
            else
              begin
                (* a new word for the dictionary OR a repeat of an old word *)
                NewEntry.numbers[1] := n;
                NewEntry.cont := false;
                for i:=2 to RowLn do
                  NewEntry.numbers[i] := 0;
                end
                new(newp);
                newp^.dline := NewEntry;
                if NewEntry.name<first^.dline.name
                  then
                    begin (* new head of the list *)
                      newp^.next := first;
                      first := newp;
                    end
                  else
                    begin (* slot entry into list *)
                      newp^.next := pt^.next;
                      pt^.next := newp
                    end
                end
              end
            end;
          until LastWord
        end
      end
    end
  l := 0
end

```

```

end
end; (* of build *)

procedure merge;
  (* to merge dict in core with existing dict on file *)

var  continue : boolean;
     j,jj : integer;
     NewEntry : dic;
begin
  writeln(scratch,' Merge');
  rewrite(TempDict);
  reset(dict);
  (* copy to scratch with additions *)
  pt := first;
  continue := (not eof(dict)) and (pt^.next<>nil);
  while continue do
    begin
      if dict^.name<pt^.dline.name
      then
        begin
          TempDict^ := dict^;
          put(TempDict);
          get(dict);
          continue := not eof(dict)
        end;
      if dict^.name>pt^.dline.name
      then
        begin
          TempDict^ := pt^.dline;
          put(TempDict);
          pt := pt^.next;
          continue := pt<>nil
        end;
      if dict^.name=pt^.dline.name
      then
        begin
          dict^.cont := true;
          TempDict^ := dict^;
          put(TempDict);
          get(dict);
          continue := not eof(dict)
        end
      end;
    while not eof(dict) do
      begin
        TempDict^ := dict^;
        put(TempDict);
        get(dict)
      end;
    while pt<>nil do
      begin
        TempDict^ := pt^.dline;
        put(TempDict);
        pt := pt^.next
      end;
    rewrite(dict);
    reset(TempDict);
    (* copy back to dict and squeeze *)
    while not eof(TempDict) do
      begin
        NewEntry := TempDict^;
        if (NewEntry.numbers[RowLn]>0) or (NewEntry.cont=false)
        then
          begin
            dict^ := NewEntry;
            put(dict);
            get(TempDict)
          end
        else
          begin
            get(TempDict);
            if not eof(TempDict)
            then
              begin
                for j:=2 to RowLn do
                  if NewEntry.numbers[j]=0
                  then
                    begin
                      NewEntry.numbers[j] := TempDict^.numbers[j];
                      for jj:=1 to RowLn-1 do
                        TempDict^.numbers[jj] := TempDict^.numbers[jj+1]
                      TempDict^.numbers[RowLn] := 0
                    end;
                  if TempDict^.numbers[1]=0
                  then
                    begin
                      NewEntry.cont := false;
                      get(TempDict);
                      dict^ := NewEntry;

```

```

          put(dict)
        end
      else
        begin
          dict^ := NewEntry;
          put(dict)
        end
      end
    end
  end;
  end;
  rewrite(TempDict)
end; (* of merge *)

begin (* MAIN PROGRAM *)
  reset(PendingTray);
  reset(bank);
  dlistOK := false;
  rewrite(scratch);
  writeln(scratch);
  writeln(scratch,'No new additions.');
```

```

  writeln(scratch);
  GotFromCore := false;
  corr := 0;
  reps := 0;
  add := 0;
  TopItem := 0;
  reset(dlist);
  if dlist^ = '' then dlistOK := true;
  if eof(PendingTray)
  then
    begin
      if not dlistOK then
        while not eof(bank) do
          begin
            TopItem := TopItem + 1;
            get(bank)
          end
        end
      else
        begin
          (* divide PendingTray into corrections and additions *)
          rewrite(correction);
          rewrite(additions);
          rewrite(dict);
          rewrite(scratch);
          dlistOK := false;
          while not eof(PendingTray) do
            if PendingTray^.tag<=!ITag
            then
              begin
                write(correction,PendingTray^);
                corr := corr+1;
                get(PendingTray)
              end
            else
              begin
                write(addition,PendingTray^.entry);
                add := add+1;
                get(PendingTray)
              end;
            reset(correction);
            writeln(scratch,'Corrections ',corr :5,' Additions ',add:5);
            while not eof(correction) do
              begin
                (* order correction into core in batches of 'stack' *)
                writeln(scratch,'To deal with corrections');
                mark(now);
                n := 1;
                new(e);
                e^.TagEntry := correction^;
                e^.next := nil;
                efirst := e;
                get(correction);
                while (not eof(correction)) and (n<stack) do
                  begin
                    n := n+1;
                    new(ewcp);
                    ewcp^.TagEntry := correction^;
                    if correction^.tag<efirst^.TagEntry.tag
                    then
                      begin (* new head of list *)
                        ewcp^.next := efirst;
                        efirst := ewcp
                      end
                    else
                      begin
                        (* move pointer ept to correct place, slot in new item *)
                        ept := efirst;
                        while (ept^.next<>nil) and
                          (correction^.tag>=ept^.next^.TagEntry.tag)

```

```

do
  ept := ept^.next;
  if correction^.tag=ept^.TagEntry.tag
  then
    ept^.TagEntry := correction^
(* replace with later correction, this is why items are sorted in
  this way *)
  else
    begin
      enewp^.next := ept^.next;
      ept^.next := enewp
    end
  end;
  get(correction)
end; (* n=stack or eof(correction) *)
write(scratch,'Corrections processed in ');
writeln(scratch,'this batch ',n :5);
(* first batch of items from 'correction' now in core and ordered *)

(* now read bank to TempBank making changes from core.
  Items are labelled for later extraction by making the
  date = NonDate.
  Replacement items are passed to join additions. *)
write(scratch,'Copy bank to TempBank ....');
rewrite(TempBank);
reset(bank);
OldTotal := 0;
ept := efirst;
while not eof(bank) do
  begin
    OldTotal := OldTotal+1;
    if (ept<>nil) and (ept^.TagEntry.tag=OldTotal)
    then (* we have found one to correct *)
      begin
        if ept^.TagEntry.entry.date<>NonDate
        then (* ie. it is not empty *)
          begin
            (* Replacement item written to addition file *)
            write(addition,ept^.TagEntry.entry);
            reps := reps+1
          end;
          bank^.date := NonDate;
          write(TempBank,bank^);
          get(bank);
(* Making the date = NonDate will remove the item when
  the last batch of corrections are processed *)
          ept := ept^.next;
          end
        else
          begin
            write(TempBank,bank^);
            get(bank)
          end
        end;
        release(now);
        writeln(scratch,' O.K. ');
(* read TempBank back to bank *)
        write(scratch,'Copy TempBank to bank ....');
        rewrite(bank);
        reset(TempBank);
        while not eof(TempBank) do
          if eof(correction) and (TempBank^.date=NonDate)
          then
            get(TempBank) (* removes corrected items *)
          else
            begin
              write(bank,TempBank^);
              get(TempBank);
              end; (* of reading back to bank *)
              writeln(scratch,' O.K. ');
              rewrite(TempBank)
            end; (* return for more corrections *)

            rewrite(correction);
            reset(addition);
            while not eof(addition) do
              begin
(* order additions alphabetically into core in batches of 'stack' *)
                writeln(scratch,'To deal with additions. ');
                if reps>0
                then
                  writeln(scratch,'These include ',reps :5,
                    ' replacements. ');
                mark(now);
                n := 1;
                new(e);
                e^.TagEntry.entry := addition^;
                e^.next := nil;
                efirst := e;
                get(addition);
                while not eof(addition) and (n<stack) do
                  begin
                    n := n+1;
                    new(enewp);
                    enewp^.TagEntry.entry := addition^;
                    move := ((enewp^.TagEntry.entry.authors
                      > efirst^.TagEntry.entry.authors) or
                      ((enewp^.TagEntry.entry.authors
                      = efirst^.TagEntry.entry.authors) and
                      (enewp^.TagEntry.entry.date
                      > efirst^.TagEntry.entry.date)));
                    if not move
                    then (* new head of list *)
                      begin
                        enewp^.next := efirst;
                        efirst := enewp
                      end
                    else
                      begin
(* move pointer ept to correct place, slot in new item *)
                        ept := efirst;
                        while (ept^.next<>nil) and
                          ((addition^.authors
                          > ept^.next^.TagEntry.entry.authors) or
                          ((addition^.authors
                          = ept^.next^.TagEntry.entry.authors) and
                          (addition^.date
                          > ept^.next^.TagEntry.entry.date))) do
                          ept := ept^.next;
                          enewp^.next := ept^.next;
                          ept^.next := enewp
                        end;
                        get(addition)
                      end; (* n=stack or eof(addition) *)
                      writeln(scratch,'Additions processed in this batch ',n :5);
(* now read bank to TempBank making additions from core *)
                      write(scratch,'Copy bank to TempBank ....');
                      reset(bank);
                      rewrite(TempBank);
                      ept := efirst;
                      continue := (not eof(bank)) and (ept<>nil);
                      while continue do
                        begin
                          if ((bank^.authors < ept^.TagEntry.entry.authors) or
                            ((bank^.authors = ept^.TagEntry.entry.authors) and
                            (bank^.date < ept^.TagEntry.entry.date)))
                            then
                              begin
                                write(TempBank,bank^);
                                get(bank);
                                continue := not eof(bank)
                              end
                            else
                              begin
                                write(TempBank,ept^.TagEntry.entry);
                                ept := ept^.next;
                                continue := ept<>nil
                              end
                            end; (* of the merging of the core and the file *)
                          while not eof(bank) do
                            begin
                              write(TempBank,bank^);
                              get(bank)
                            end;
                          while ept<>nil do
                            begin
                              write(TempBank,ept^.TagEntry.entry);
                              ept := ept^.next
                            end;
                          LastOne := bank^;
                          (* assigned to give LastOne a starting value *)
                          writeln(scratch,' O.K. ');
(* now copy back to bank *)
                          write(scratch,'Copy TempBank to bank ....');
                          reset(TempBank);
                          rewrite(bank);
                          release(now);
                          while not eof(TempBank) do
                            begin
                              same := ((TempBank^.authors=LastOne.authors)
                                and (TempBank^.title=LastOne.title)
                                and (TempBank^.title2=LastOne.title^
                                and (TempBank^.date=LastOne.date));
                              if not same
                              then
                                write(bank,TempBank^); (* rejects duplicates *)
                                LastOne := TempBank^;
                                get(TempBank)
                              end;

```

