***************
* DATA FILES *
***************

Tournament-roster

COMPOSITION ... \{ Player-record \}

Tournament-statistics

COMPOSITION ... Tournament-identification
CREATE - TOURNAMENT - ROSTER

RECORD - GAME - RESULTS

FAIR - PLAYERS - FOR - PAIR

COMPILE - CROSSTABLE

CONTROL - TRAP

TD
FUTURE IMPROVEMENTS AND ADDITIONS (classified by type)

PROGRAMMING STYLE IMPROVEMENTS

1. Trap end-of-file character on keyboard input.
2. Allow user to exit from any entry menu without completing the data entry.
3. Default input values to most recently entered values.
4. Standardize the GET_AFFIRMATION module and copy into the main module; eliminate the individual versions.
5. Clean up the format of the crosstable report -- header style, detail line style, and paging format.

"SMALL" SYSTEM DETAILS

1. Verify that all game scores have been recorded, whether by adjudication or by completion, before proceeding with pairings for the round.
2. Inform the user when all game scores have been recorded for the preceding round.
3. Implement the following status flags:
   Odd_man_status
   Unfinished_game_status
   Pairing_odd_men
   Alternating_colors
   Scoring_unfinished_games
   Ranking_unrated_players
SYSTEM MODIFICATIONS

1. Improve Record_round_result.
   (a) require user to enter round number of result
   (b) verify that the players were paired and that
       the result has not yet been recorded

2. Provide for late entry of a player.
   (a) assign a pairing number and update Highest_pairing_number if necessary.
   (b) create a new pairing limitation list and
       update the lists thereby affected

3. Provide for the early withdrawal of a player.

4. Allow adjourned games, with either automatic or
   manual assignment of adjudicated scores.

5. Incorporate requested pairing limitations.
   (a) create new pairing lists after assigning
       pairing numbers
   (b) set a point total/round number threshold to
       limit application to non-critical pairings

6. Code modules to break tied scores for crosstable
   generation.

7. Provide for system restoration (linked lists,
   tournament variables, etc.) to allow for multiple-
   day tournaments and to safeguard against system
   failure.

8. Allow choice of current tournament files for editing
   and adding tournament information, so that concurrent
   tournaments can be maintained.

9. Incorporate the accelerated pairing variation of
   the Swiss System into the TD.
FILE LAYOUT FOR Tournament_statistics

<table>
<thead>
<tr>
<th>Record #</th>
<th>Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tournament_name</td>
<td>Packed Char</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>Sponsor</td>
<td>Packed Char</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Director</td>
<td>Packed Char</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Site</td>
<td>Packed Char</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>Starting_date</td>
<td>Packed Char</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Ending_date</td>
<td>Packed Char</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Number_of_rounds</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Ranking_unrated_players</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Scoring_byes</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Scoring_unfinished_games</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Pairing_odd_men</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Alternating_colors</td>
<td>Char</td>
<td>1</td>
</tr>
</tbody>
</table>
FILE_LAYOUT FOR Tournament_roster

REPETITIONS OF Player_record

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairing_number</td>
<td>3</td>
<td>Packed Char</td>
<td>1- 3</td>
</tr>
<tr>
<td>Name</td>
<td>30</td>
<td>Packed Char</td>
<td>4- 33</td>
</tr>
<tr>
<td>Rating</td>
<td>4</td>
<td>Packed Char</td>
<td>34- 37</td>
</tr>
<tr>
<td>USCF_ID</td>
<td>8</td>
<td>Packed Char</td>
<td>38- 45</td>
</tr>
<tr>
<td>Opponent()</td>
<td>9x3^2</td>
<td>Packed Char</td>
<td>46- 72</td>
</tr>
<tr>
<td>Color()</td>
<td>9x1^2</td>
<td>Char</td>
<td>73- 81</td>
</tr>
<tr>
<td>Result()</td>
<td>9x8^2</td>
<td>Real</td>
<td>82-153</td>
</tr>
<tr>
<td>Score</td>
<td>8</td>
<td>Real</td>
<td>154-161</td>
</tr>
</tbody>
</table>

1 See Tournament_roster.Fdl for parameters used to index this file.

2 Repetitions occur Max_rounds times, a constant currently defined as 9 in TD's main module.
TITLE "Tournament_roster.FDL"
IDENT "20-APR-1986 15:39:51 VAX-11 FDL Editor"

SYSTEM SOURCE VAX/VMS.

FILE NAME "Tournament_roster.DAT"
ORGANIZATION indexed

RECORD CARRIAGE_CONTROL carriage_return
FORMAT fixed
SIZE 121

AREA 0
ALLOCATION 42
BEST_TRY_CONTIGUOUS yes
BUCKET_SIZE 3
EXTENSION 9

AREA 1
ALLOCATION 9
BEST_TRY_CONTIGUOUS yes
BUCKET_SIZE 3
EXTENSION 3

KEY 0
CHANGES no
DATA_AREA 3
DATA_FILL 100
DATA_KEY_COMPRESSION yes
DATA_RECORD_COMPRESSION yes
DUPLICATES no
INDEX_AREA 1
INDEX_COMPRESSION yes
INDEX_FILL 100
LEVEL1_INDEX_AREA 1
NAME "Pairing_number"
PROLOG 3
SEGO_LENGTH 3
SEGO_POSITION 0
TYPE string
TOURNAMENT: Test Tournament #2

ROUND: 4

<table>
<thead>
<tr>
<th>TABLE</th>
<th>WHITE</th>
<th>BLACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Adams</td>
<td>Lester VanMeter</td>
</tr>
<tr>
<td>2</td>
<td>Ron Powell</td>
<td>Jim Mills</td>
</tr>
<tr>
<td>3</td>
<td>Lou Kontos</td>
<td>Joshua Bousum</td>
</tr>
<tr>
<td>4</td>
<td>John Dowling</td>
<td>John Roush</td>
</tr>
<tr>
<td>5</td>
<td>Mike Sheaf</td>
<td>Tom Harris</td>
</tr>
</tbody>
</table>
## TOURNAMENT: Test Tournament #2

**SPONSOR:** Honors College  
**DATE:** May 14, 1986  
**SITE:** Muncie, Indiana  
**DIRECTOR:** Eugene Wallingford

<table>
<thead>
<tr>
<th>No.</th>
<th>Player</th>
<th>Rating</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>Lester Vanmeter</td>
<td>2331</td>
<td>W</td>
<td>O60 W</td>
<td>320 B</td>
<td>W</td>
<td>090 W</td>
</tr>
<tr>
<td>020</td>
<td>Jim Mills</td>
<td>2223</td>
<td>W</td>
<td>C70 B</td>
<td>L</td>
<td>O10 W</td>
<td>W</td>
</tr>
<tr>
<td>090</td>
<td>Ron Powell</td>
<td>1911</td>
<td>W</td>
<td>C40 W</td>
<td>W</td>
<td>030 B</td>
<td>L</td>
</tr>
<tr>
<td>040</td>
<td>Nick Adams</td>
<td>2173</td>
<td>L</td>
<td>090 B</td>
<td>W</td>
<td>100 B</td>
<td>W</td>
</tr>
<tr>
<td>100</td>
<td>Lou Kontos</td>
<td>1644</td>
<td>D</td>
<td>050 B</td>
<td>L</td>
<td>340 W</td>
<td>W</td>
</tr>
<tr>
<td>030</td>
<td>Joshua Bousum</td>
<td>2197</td>
<td>D</td>
<td>080 W</td>
<td>L</td>
<td>090 W</td>
<td>D</td>
</tr>
<tr>
<td>050</td>
<td>John Bowling</td>
<td>2111</td>
<td>D</td>
<td>100 W</td>
<td>D</td>
<td>050 B</td>
<td>L</td>
</tr>
<tr>
<td>070</td>
<td>John Roush</td>
<td>1997</td>
<td>L</td>
<td>020 W</td>
<td>D</td>
<td>060 B</td>
<td>D</td>
</tr>
<tr>
<td>080</td>
<td>Mike Sheaf</td>
<td>1970</td>
<td>D</td>
<td>030 B</td>
<td>D</td>
<td>050 W</td>
<td>L</td>
</tr>
<tr>
<td>060</td>
<td>Tom Harris</td>
<td>2043</td>
<td>L</td>
<td>C10 B</td>
<td>D</td>
<td>070 W</td>
<td>L</td>
</tr>
</tbody>
</table>
COLOR-DUE MATRIX
(Used in color allocation algorithm, Module 4b)

<table>
<thead>
<tr>
<th>H/L</th>
<th>+3</th>
<th>+2</th>
<th>+1</th>
<th>0</th>
<th>-1</th>
<th>-2</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>+2</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>+1</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>0</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>-1</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>-2</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>-3</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>

Y-axis: Higher-ranked player's Color_due_status
X-axis: Lower-ranked player's Color_due_status

Matrix values: "H" means higher-ranked player gets white, and "L" means lower-ranked player gets white.
# Module Index

<table>
<thead>
<tr>
<th>Module</th>
<th>Name</th>
<th>Page</th>
<th>Done?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tournament director</td>
<td>63</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>Control_trap</td>
<td>64</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>Create_tournament_roster</td>
<td>69</td>
<td>Y</td>
</tr>
<tr>
<td>21</td>
<td>Record_game_results</td>
<td>71</td>
<td>Y</td>
</tr>
<tr>
<td>22</td>
<td>Pair_players_for_round</td>
<td>73</td>
<td>Y</td>
</tr>
<tr>
<td>23</td>
<td>Compile_crosstable</td>
<td>77</td>
<td>Y</td>
</tr>
<tr>
<td>30</td>
<td>Enter_tournament_statistics</td>
<td>79</td>
<td>Y</td>
</tr>
<tr>
<td>31</td>
<td>Enter_a_player</td>
<td>84</td>
<td>Y</td>
</tr>
<tr>
<td>32</td>
<td>Prepare_late_entry</td>
<td>87</td>
<td>Y</td>
</tr>
<tr>
<td>33</td>
<td>Prepare_for_round_one</td>
<td>90</td>
<td>Y</td>
</tr>
<tr>
<td>34</td>
<td>Place_cards_in_scored_list</td>
<td>92</td>
<td>Y</td>
</tr>
<tr>
<td>35</td>
<td>Enter_round_result</td>
<td>95</td>
<td>Y</td>
</tr>
<tr>
<td>36</td>
<td>Record_round_results</td>
<td>97</td>
<td>Y</td>
</tr>
<tr>
<td>37</td>
<td>Locate_pairing_cards</td>
<td>99</td>
<td>Y</td>
</tr>
<tr>
<td>38</td>
<td>Prepare_for_pairing</td>
<td>100</td>
<td>Y</td>
</tr>
<tr>
<td>39</td>
<td>Pair_odd_men</td>
<td>101</td>
<td>Y</td>
</tr>
<tr>
<td>3a</td>
<td>Pair_score_group</td>
<td>102</td>
<td>Y</td>
</tr>
<tr>
<td>3b</td>
<td>Print_pairing_list</td>
<td>106</td>
<td>Y</td>
</tr>
<tr>
<td>3c</td>
<td>Update_pairing_cards</td>
<td>109</td>
<td>Y</td>
</tr>
<tr>
<td>3d</td>
<td>Create_crosstable</td>
<td>111</td>
<td>Y</td>
</tr>
<tr>
<td>3e</td>
<td>Output_crosstable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Create_limitation_list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Update_limitation_lists</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Clear_unfinished_game_flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Enter_adjudicated_score</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Assign_unfinished_score</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Assign_bye_to_odd_man</td>
<td>113</td>
<td>Y</td>
</tr>
<tr>
<td>46</td>
<td>Create_a_score_group</td>
<td>115</td>
<td>Y</td>
</tr>
<tr>
<td>47</td>
<td>Pair_odd_man_from_above</td>
<td>116</td>
<td>Y</td>
</tr>
<tr>
<td>48</td>
<td>Remove_odd_man_from_group</td>
<td>119</td>
<td>Y</td>
</tr>
<tr>
<td>49</td>
<td>Match_top_with_bottom</td>
<td>121</td>
<td>Y</td>
</tr>
<tr>
<td>4a</td>
<td>Eliminate_illegal_pairings</td>
<td>122</td>
<td>Y</td>
</tr>
<tr>
<td>4b</td>
<td>Allocate_colors</td>
<td>126</td>
<td>Y</td>
</tr>
<tr>
<td>4c</td>
<td>Eliminate_tied_scores</td>
<td>131</td>
<td>N</td>
</tr>
<tr>
<td>4d</td>
<td>Write_header_items</td>
<td>132</td>
<td>Y</td>
</tr>
<tr>
<td>4e</td>
<td>Write_detail_lines</td>
<td>134</td>
<td>Y</td>
</tr>
<tr>
<td>**</td>
<td>Prt.Com</td>
<td>137</td>
<td>Y</td>
</tr>
</tbody>
</table>
(******************************************************************************)
(*    PROGRAM : Tournament_director (10)  DATE : April 9, 1986    *)
(*    PURPOSE : This is the top-level control module for the          *)
(*        Tournament Director system.                               *)
(*    ******************************************************************************)

PROGRAM

Tournament_director (Input, Output, 
   Crossstable, Pairing_list, 
   Temporary_roster, Tournament_roster, Tournament_statistics);

CONST

Max_rounds = 9;

TYPE

Opponent_pointer = ^Opponent_record;
Opponent_record = Record
   Opponent
   Next_record : Packed array [1..3] of Char;
   End;

Card_pointer = ^Pairing_card;
Pairing_card = Record
   Next_card : Card_pointer;
   Pairing_number : Packed array [1..3] of Char;
   Score : Real;
   Odd_even_status : Integer;
   Color_due_status : Integer;
   Unfinished_game_status : Packed array [1..2] of Char;
   Bye_given_status : Char;
   Opponent : Array [1..Max_rounds] of
              Packed array [1..3] of Char;
   End;

Limit_pointer = ^Pairing_limitation;
Pairing_limitation = Record
   Next_record : Limit_pointer;
   Player : Packed array [1..3] of Char;
   Limitation : Opponent_pointer;
   End;

Player_record = Record
   Pairing_number
   Name : Packed array [1..30] of Char;
   Rating : Packed array [1..4] of Char;
   USCF_ID : Packed array [1..8] of Char;
   Opponent : Array [1..Max_rounds] of
              Packed array [1..3] of Char;
   End;
VAR

Scored_pointer;
Paired_pointer;
End_of_scored;
End_of_paired = Card_pointer;
Pairing_limitations = Limit_pointer;
Not_yet_prepared_for_round_one := Boolean := True;
Tournament_name /
Sponsor /
Director := Packed array [1..30] of Char;
Site := Packed array [1..40] of Char;
Starting_date /
Ending_date := Packed array [1..30] of Char;
Ranking_unrated_players /
Scoring_byes /
Scoring_unfinished_games /
Pairing_odd_men /
Alternating_colors := Char;
Round_number := Integer;
Number_of_rounds := Integer;
Loop := Boolean;
Selection := Char;
Valid_selections := Menu_options;
Crosstable /
Pairing_list /
Tournament_statistics := Text;
Temporary_roster /
Tournament_roster := File of Player_record;
Number_of_players := Integer;
Highest_pairing_number := Packed array [1..3] of Char;

(*
* DECLARE EXTERNAL REFERENCES *)
(*
*)
Procedure Lib$Erase_page (Row,Col : Integer); External;
Procedure Lib$Set_cursor (Row,Col : Integer); External;
Procedure Lib$Spawn (IDescr Command : Command_line); External;
Procedure Trap_C; Fortran;
Procedure Ctrl_C_off; Fortran;
Procedure Ctrl_C_on; Fortran;
%Include '20'
%Include '21'
%Include '22'
%Include '23'

(* SET-UP INITIAL STATE OF SYSTEM *)
(* *)
Begin
Trap_C;
Ctrl_C_off;
Round_number := 0;
Scored_pointer := 'Nil';
Paired_pointer := 'Nil';
End_of_scored := 'Nil';
Pairing_limitations := 'Nil';
Loop := True;

(* MAIN CONTROL LOOP OF MODULE *)
(* *)
While Loop do
Begin
Valid_selections := ['1', '2', '3', '4', '5'];
Lib$Erase_page(Row:=1, Col:=1);
Lib$Set_cursor(Row:=1, Col:=1);
WriteIn("***********************************************************************");
WriteIn(" */ ");
WriteIn(" */ SWISS - SYSTEM TOURNAMENT DIRECTOR */ ");
WriteIn(" */ ");
Lib$Set_cursor(Row:=10, Col:=1);
WriteIn("(1) Enter tournament information;
WriteIn("(2) Record a game result;
WriteIn("(3) Pair players for the round;
WriteIn("(4) Prepare a current crosstable;
WriteIn("(5) Terminate the tournament director;
Lib$Set_cursor(Row:=16, Col:=1);
Write (" Enter your selection: ");
Readln(Selection);

While Not (Selection in Valid_selections) Do
  Begin
    Lib$Erase_page(Row:=16, Col:=1);
    Lib$Set_cursor(Row:=17, Col:=1);
    Writeln("INVALID SELECTION !!");
    Writeln;
    Write ("Enter your selection: ");
    Readln(Selection);
  End;

Case Selection of
  
  '1' : Create_tournament_roster;
  
  '2' : Record_game_results;
  
  '3' : Pair_players_for_round;
  
  '4' : Compile_crosstable;
  
  '5' : Loop := False;
  
  End;

End;

(* RESTORE SYSTEM STATE *)

Lib$Erase_page(Row:=1, Col:=1);
Ctrl_C_on;

End.
PROGRAM NAME: Trap_c (11)  DATE: April 1, 1986

PURPOSE: This Fortran program provides an asynchronous trap for Control-C and Control-Y characters.

This is the control subroutine to trap Control-Y and Control-C characters. This small routine calls the subroutine CAST.

SUBROUTINE TRAP_C
IMPLICIT INTEGER*4 (A-Z)
INTEGER*2 TTCHAN, IOS(4)
CHARACTER TERMINAL*11, 'SYS$COMMAND*', 'BUFFER*80
EXTERNAL CAST
INCLUDE '($IODEF)'
INCLUDE '($SSSDEF)'
STATUS = SYS$ASSIGN(TERMINAL,TTCHAN,)
IF (.NOT. STATUS) CALL LIBSSTOP(XVAL(STATUS))
FUNC = IOS_SETMODE OR IOSM_CTRLCAST
STATUS = SYS$ASSIGN(XVAL(TTCHAN),XVAL(FUNC),CAST,TTCHAN,)
IF (.NOT. STATUS) CALL LIBSSTOP(XVAL(STATUS))
RETURN
END

This routine provides the asynchronous trap for CTRL-C and CTRL-Y.
CTRLEXIT is called as a routine from this module.

SUBROUTINE CAST(TTCHAN)
IMPLICIT INTEGER*4 (A-Z)
STATUS = SYS$ASSIGN(TTCHAN,)
IF (.NOT. STATUS) CALL LIBSSTOP(XVAL(STATUS))
CALL CTRLEXIT
RETURN
END

This is the module that actually disables the interrupt characters.

SUBROUTINE CTRL_C_OFF
INTEGER MASK
MASK= '02000000'x
CALL LIB$DISABLE_CTRL(MASK,MASK)
RETURN
END

This module restores the activity of CTRL-C and of CTRL-Y.

SUBROUTINE CTRL_C_ON
INTEGER MASK
MASK= '3460300B'

CALL LIBSENABLE_CTRL(MASK, MASK)
RETURN
END

This module invokes TRAP_C from CAST, which is itself called by
TRAP_C. Thus the asynchronous trap loop is completed.

SUBROUTINE CTRL_C_EXIT
CALL TRAP_C
CALL CTRL_C_OFF
TYPE* CHAR(7)
RETURN
END
CREATE TOURNAMENT ROSTER

DATE: April 9

PURPOSE: This module serves as the control menu for entry of tournament and player information.

PROCEDURE

CREATE_TOURNAMENT_ROSTER;

VAR

Selection : Char;
Loop : Boolean;
Dummy_character : Char;

DECLARE EXTERNAL REFERENCES

XInclude '30'
XInclude '31'
XInclude '32'
XInclude '33'

Valid_selections := ['1'..'6'];
Loop := True;

MAIN CONTROL LOOP OF MODULE

While Loop Do

Begin

Lib$Erase_page(Row:=1,Col:=1);
Lib$Set_cursor(Row:=1,Col:=1);
WriteLn('***********************************************************************************');
WriteLn(' ');
WriteLn(' Enter Tournament Information ');
WriteLn(' ');
WriteLn(' ');
WriteLn(' ');
Lib$Set_cursor(Row:=10,Col:=1);

End

Writeln('Enter specific tournament data');
Writeln('Enter a new player');
Writeln('Prepare for round one');
Writeln('Return to the main menu');

Library_cursor(Row:=16,Col:=1); Write ('Enter your selection : '); Readln(Selection);

While Not (Selection in Valid_selections) Do Begin
  Library_erase_page(Row:=16,Col:=1);
  Library_cursor(Row:=17,Col:=1);
  Writeln(' INVALID SELECTION !!');
  Writeln;
  Write ('Enter your selection : '); Readln(Selection);
End;

  Case Selection of
  '1' : Enter_tournament_statistics;
  '2' : Begin Enter_1_player;
  (*If Round_number > 0 Then Prepare_late_entry;*)
  End;
  '3' : If Not yet_prepared_for_round_one Then Begin Prepare_for_round_one;
  Not yet_prepared_for_round_one := False;
  End Else Begin
  Library_cursor(Row:=22,Col:=1);
  Write('You've already done this! Hit <RETURN> to continue...');
  Readln(Dummy_character);
  End;
  '4' : Loop := False;
End;
End;
PROGRAM: Record_game_results (21)  DATE: April 22, 1986

PURPOSE: This module provides for entry of game scores.

PROCEDURE

Record_game_results;

TYPE

Three_char = Packed array [1..3] of Char;

Round_result = Record

  Player : Array [1..2] of Three_char;
  Result : Char;
End;

Valid_error_codes = 0..41;

VAR

Game_result : Round_result;
Array_index : Integer;

Previous_card : Integer;
Desired_card : Array [1..2] of Card_pointer;

Error_code : Valid_error_codes;
Message : Varying [30] of Char;
Dummy_variable : Varying [1] of Char;

%Include '35'
%Include '37'
%Include '36'
%Include '34'

Begin

Enter_round_result;
Lib$Erase_page(Row:=7,Col:=1);
Writeln('Processing round result . . .');
Locate_pairing_cards;
If Error_code = 0
Then Begin
   Record_round_results;
   Place_cards_in_scored_list;
   End

Else Begin
   Case Error_code of
      1: Message := "Card was not found for player 1. Hit <RETURN> to continue."
      2: Message := "Card was not found for player 2. Hit <RETURN> to continue."
      3: Message := "Card was not found for either player. Hit <RETURN> to continue."
      4: Message := "Players were not paired to meet this round. Hit <RETURN> to continue."
   End;
   Write (Message);
   Readln(Dummy_variable);
End;

End;
PROCEDURE

Pair_players_for_rounds;

VAR

Dummy_character : Varying [1] of Char;
End_of_paired ;
Score_group ;
Odd_man ; Card_pointer ;
Pairing : Array [1..50,1..2] of Card_pointer ;
Legal_pairings_exist ;
Odd_man_pending ;
More_cards_to_pair : Boolean ;
Index ;
Highest_pairing ;
Number_in_group : Integer ;

Procedure Place_pairings_in_paired ;

Var

Curr , Prev : Card_pointer ;
Index1 , Index2 : Integer ;

Begin

For Index1 := 1 to Highest_pairing Do
For Index2 := 1 to 2 Do
  Begin
    (* Find next card to be moved to the paired list *)
    Prev := Score_group ;
    Curr := Score_group ;
    While Curr <> Pairing[Index1,Index2] Do
      Begin
        Prev := Curr ;
        Curr := Curr^..Next_card ;
  (* Find next card to be moved to the paired list *)
    Prev := Score_group ;
    Curr := Score_group ;
    While Curr <> Pairing[Index1,Index2] Do
      Begin
        Prev := Curr ;
        Curr := Curr^..Next_card ;
End;

(= Remove object card from score group list =)

If Curr = Prev Then Score_group := Curr^Next_card
Else Prev^Next_card := Curr^Next_card;

(= Place object card into the paired list =)

Prev := Paired_pointer;
Curr := Paired_pointer;

While Curr <> End_of_Paired Do
Begin

Prev := Curr;
Curr := Curr^Next_card;
End;

Pairing[Index1,Index2]^Next_card := End_of_Paired;

If Paired_pointer = End_of_Paired Then Paired_pointer := Pairing[Index1,Index2];
Else Prev^Next_card := Pairing[Index1,Index2];

End;

End;

(= Declare External References =)

(= Include '38' *)

(Include '46'

(Include '39'

(Include '3a'

(Include '3b'

(Include '3c'

Begin

Lib$Er$ase_page(Row:=1,Col:=1);
Lib$Set_Cursor(Row:=1,Col:=1);
Writeln("***********";)
Writeln("*";)
Writeln("****";)
Writeln("***********";)
Writeln("PAIR PLAYERS FOR THE ROUND";)
Writeln("*";)
Writeln("**";)
Writeln("***********";)
Writeln("*";)
Writeln("****";)
Writeln("***********";)

Then Begin
WriteLn('Pairings have already been made for the final round.'); Write('Hit <RETURN> to continue the tournament: ');
Readln (Dummy_character);
End
Else Begin
Round_number := Round_number + 1;
End_of_pairs := Nil;
Write('Writeln(Round_number);
Prepare_for_pairings;
Odd_man_pending := False;
More_cards_to_pair := True;
Legal_pairings_exist := True;
Odd_man := Nil;
Score_group := Nil;
While More_cards_to_pair Do
Begin
...
Write ('Pairings completed for round');
WriteIn(Round_number);
WriteIn;
Write ('Hit <RETURN> to continue ...');
Readln(Dummy_character);
End;
End;
End;
End;
(* PROGRAM: Compile_crosstable (23) DATE: April 27 *)
(* PURPOSE: This module serves as the control menu for *)
(* the preparation and printing of a crosstable. *)

PROCEDURE
  Compile_crosstable;

VAR
  Selection /
  Order_selection /
  Print_selection : Char;
  Loop : Boolean;

(* DECLARE EXTERNAL REFERENCES *)
(* *)

%Include '3d'
%Include '3e'

Begin
  Loop := True;
  While Loop do
    Begin
      Lib$Erase_page(Row:=1,Col:=1);
      Lib$Set_cursor(Row:=1,Col:=1);
      Writeln('******************************************************************************
               *                           PREPARE A CURRENT CROSSTABLE                           *
               *                                                                                     *
               *                                                                                     *
               *****************************************************************************/
      Writeln('1) Create a new crosstable');
      Writeln('2) Output the most recent crosstable');
      Writeln('3) Return to the main menu');
      Lib$Set_cursor(Row:=16,Col:=1);
      Write('Enter your selection: ');
      Readln(Selection);
    End;
  End;

END Compile_crosstable;
While Not (Selection in [ '1' , '2' , '3' ] ) Do
Begin
 Lib$Erase_page(Row:=16,Col:=1);
 Lib$Set_cursor(Row:=17,Col:=1);
 Writeln('INVALID SELECTION !!!');
 Writeln;
 Write ('Enter your selection: ');
 Readln(Selection);
 End;

Case Selection of
 '1'  :  Create_crosstable;
 '2'  :  Output_crosstable;
 '3'  :  Loop := False;
End;

End;
(* PROGRAM: Enter_tournament_statistics (3D) DATE: April 18 *)

(* PURPOSE: This module serves as the entry menu for vital tournament information, such as name, dates, etc. *)

(* ******************************************************************************* *)

PROCEDURE

Enter_tournament_statistics;

VAR

Char_no_of_rounds : Char;
Valid_alternating_colors /
Valid_pairing_odd_men /
Valid_scoring_byes /
Valid_scoring_unfinisheds : Set of Char;
Valid_ranking_unrateds : Set of Char;
Valid_numbers ; Set of Char;
Desire_to_enter_options : Char;
Screen_is_okay : Boolean;

(* DECLARE INTERNAL REFERENCES *)

Procedure Get_data_fields;

Begin

Lib$Escape_page(Row:=7 / Col:=1);
Lib$Set_cursor(Row:=7 / Col:=1);
Write (' Tournament name : ');
Readln( Tournament_name );
Write (' Sponsor : ');
Readln( Sponsor );
Write (' Tournament Director : ');
Readln( Director );
Write (' Site : ');
Readln( Site );
Write (' Starting date : ');
Readln( Starting_date );
Write (' Ending date : ');

Readln( Ending_date );
Write ( ' Number of rounds : ' );
Repeat  
  Readln(Char_no_of_rounds)  
  Until Char_no_of_rounds in Valid_numbers;
Write ( ' Select variations ? (Y or N) : ' );
Repeat  
  Readln(Desire_to_enter_options)  
  Until Desire_to_enter_options in ['Y','N','y','n'];
If (Desire_to_enter_options = 'Y') or  
  (Desire_to_enter_options = 'y') then
Begin
Write ( ' Scoring byes : ' );
Repeat  
  Readln(Scoring_byes)  
  Until Scoring_byes in Valid_scoring_byes;
Write ( ' Ranking unrated players : ' );
Repeat  
  Readln(Ranking_unrated_players)  
  Until Ranking_unrated_players in Valid_ranking_unrateds;
Write ( ' Scoring unfinished games : ' );
Repeat  
  Readln(Scoring_unfinished_games)  
  Until Scoring_unfinished_games in Valid_scoring_unfinisheds;
Write ( ' Pairing odd men : ' );
Repeat  
  Readln(Pairing_odd_men)  
  Until Pairing_odd_men in Valid_pairing_odd_men;
Write ( ' Alternation of colors : ' );
Repeat  
  Readln(Alternating_colors)  
  Until Alternating_colors in Valid_alternating_colors;
End;
End;

Procedure Display_screen;
Begin
Lib$Erase_page(Row:=1,Col:=1);
Lib$Set_cursor(Row:=1,Col:=1);
Writeln("********************************************************************************");
Writeln(' ENTER TOURNAMENT DETAILS

********************************************************************************");
Writeln('};
 LibSSet_cursor(Row:=6, Col:=1);
 Writeln(' Tournament name : ' + Tournament_name);
 Writeln(' Sponsor : ' + Sponsor);
 Writeln(' Tournament Director : ' + Director);
 Writeln(' Site : ' + Site);
 Writeln(' Starting date : ' + Starting_date);
 Writeln(' Ending date : ' + Ending_date);
 Writeln(' Number of rounds : ' + Char_no_of_rounds);
 Writeln(' Options : ');
 Writeln(' Scoring byes : ' + Scoring_byes);
 Writeln(' Ranking unrated players : ' + Ranking_unrated_players);
 Writeln(' Scoring unfinished games : ' + Scoring_unfinished_games);
 Writeln(' Pairing odd men : ' + Pairing_odd_men);
 Writeln(' Alternation of colors : ' + Alternating_colors);

 End;

 Procedure Get_affirmation;
 Var
 Screen_response : Char;
 Begin
 Repeat
 Begin
 Lib$Erase_page(Row:=1, Col:=1);
 Lib$Set_cursor(Row:=23, Col:=1);
 Write (' Is all the information on this screen OK? (Y or N) : ');
 Readln (Screen_response);
 If (Screen_response = 'y') or (Screen_response = 'Y')
 Then Screen_is_okay := True;
 Until Screen_response In ['Y', 'N', 'y', 'n'];
 Case Screen_response of
 'y', 'Y' : Screen_is_okay := True;
 'n', 'N' : Screen_is_okay := False;
 End;

 End;
Begin

Char_no_of_rounds := 'O';
Ranking_unrated_players := 'O';
Scoring_byes := 'O';
Scoring_unfinished_games := 'O';
Pairing_odd_even := 'O';
Alternating_colors := 'O';

Valid_alternating_colors := ['0', '1'];
Valid_pairing_odd_even := ['0', '1'];
Valid_scoring_byes := ['0', '1'];
Valid_scoring_unfinisheds := ['0', '1'];
Valid_ranking_unrateds := ['0', '1', '2', '3'];
Valid_numbers := ['1', '2', '3', '4', '5', '6', '7', '8', '9'];

Lib$Erase_page(Row:=1, Col:=1);
Lib$Set_cursor(Row:=1, Col:=1);
WriteIn("*******************************************************");
WriteIn("* ENTER TOURNAMENT DETAILS *");
WriteIn("****************************************************");

(* GET GENERAL TOURNAMENT INFORMATION *)
(* WRITE DATA TO STATISTICS FILE *)

Repeat

Begin

Get_data_fields;
Get_affirmation
End

Until Screen_is_okay;

 Rewite(Tournament_statistics);

WriteIn(Tournament_statistics, Tournament_name);
WriteIn(Tournament_statistics, Sponsor);
WriteIn(Tournament_statistics, Director);
WriteIn(Tournament_statistics, Site);
WriteIn(Tournament_statistics, Start_date);
WriteIn(Tournament_statistics, Ending_date);
WriteIn(Tournament_statistics, char_no_of_rounds);
Writeln(Tournament_statistics, Ranking_unrated_players);
Writeln(Tournament_statistics, Scoring_byes);
Writeln(Tournament_statistics, Scoring_unfinished_games);
Writeln(Tournament_statistics, Pairing_odd_even);
Writeln(Tournament_statistics, Alternating_colors);
Close(Tournament_statistics);

(* CONVERT NUMBER OF ROUNDS TO INTEGER *)
(* *)

Case Char_no_of_rounds of
  '1' : Number_of_rounds := 1;
  '2' : Number_of_rounds := 2;
  '3' : Number_of_rounds := 3;
  '4' : Number_of_rounds := 4;
  '5' : Number_of_rounds := 5;
  '6' : Number_of_rounds := 6;
  '7' : Number_of_rounds := 7;
  '8' : Number_of_rounds := 8;
  '9' : Number_of_rounds := 9;
End;

(* RESET TOURNAMENT POSTER *)
Rewrite(Tournament_roster);
Close (tournament_roster);
End;
PROCEDURE
   Enter_a_player

VAR
   Player : Player_record;
   Screen_is_okay : Boolean;
   Loop_counter : Integer;

Procedure Get_data_fields;
Begin
   With Player Do
      Begin
         Lib$Erase_page(Row:=7, Col:=1);
         Lib$Set_cursor(Row:=7, Col:=1);
         Write( ' Name : ' );
         Readln( Name );
         Write( ' Rating : ' );
         Readln( Rating );
         Write( ' USCF ID number : ' );
         Readln( USCF_ID );
      End;
   End;

Procedure Display_screen;
Begin
   Lib$Erase_page(Row:=1, Col:=1);
   Lib$Set_cursor(Row:=1, Col:=1);

PROCEDURE
   Enter_a_player

VAR
   Player : Player_record;
   Screen_is_okay : Boolean;
   Loop_counter : Integer;

Procedure Get_data_fields;
Begin
   With Player Do
      Begin
         Lib$Erase_page(Row:=7, Col:=1);
         Lib$Set_cursor(Row:=7, Col:=1);
         Write( ' Name : ' );
         Readln( Name );
         Write( ' Rating : ' );
         Readln( Rating );
         Write( ' USCF ID number : ' );
         Readln( USCF_ID );
      End;
   End;

Procedure Display_screen;
Begin
   Lib$Erase_page(Row:=1, Col:=1);
   Lib$Set_cursor(Row:=1, Col:=1);
WriteIn("
***************************************************************************");
WriteIn("
Enter tournament details
");
WriteIn("
***************************************************************************");
Lib$Set_cursor(Row:=6,Col:=1);

With Player Do
    Begin
        WriteIn('Name');
        WriteIn('Rating');
        WriteIn('USCF ID number');
    End;
End;

Procedure Get_confirmation;

Var
    Screen_response : Char;

Begin
    Repeat
        Begin
            Lib$Erase_page(Row:=23,Col:=1);
            Lib$Set_cursor(Row:=23,Col:=1);
            Write('Is all the information on this screen OK? (Y or N):');
            Readln(Screen_response);
            If (Screen_response = 'R') or (Screen_response = 'r')
            Then Display_screen
        End
    Until Screen_response in ['Y', 'N', 'y', 'n'];
End;

Case Screen_response of
    'Y', 'y': Screen_is_okay := True;
    'N', 'n': Screen_is_okay := False;
End;

Begin
Lib$Erase_page(Row:=1,Col:=1);
Lib$Set_cursor(Row:=1,Col:=1);
Writeln('******************************************************************************');
Writeln('');
Writeln('');
Writeln('ENTER A PLAYER RECORD');
Writeln('');
Writeln('*************************************************************************************');

(* GET GENERAL TOURNAMENT INFORMATION *)

Repeat
Begin
Get_data_fields;
Get_affirmation
End
Until Screen_is_okay;

(* WRITE DATA TO TOURNAMENT ROSTER *)

With Player Do
Begin
Pairing_number := ' ';
For Loop_counter := 1 to max_rounds do
Begin
Opponent [Loop_counter] := ' ';
Color [Loop_counter] := 1;
Result [Loop_counter] := 0.0;
End;
Score := 0.0;
End;

Extend (Tournament_roster);
Write (Tournament_roster + Player);
Close (Tournament_roster);
Number_of_players := Number_of_players + 1;
End;


(* Program: Prepare for round one (33) Date: April 20, 1986 *)
(* Purpose: This module serves as the entry menu for vital tournament information, such as names, dates, etc. *)

(* ********************************************************************* *)

PROCEDURE
Prepare_for_round_one;

TYPE
Three_char = Packed array [1..3] of Char;

VAR
Index : Integer;
Player : Player_record;
Current_number : Three_char;
New_pairing_card : Card pointer;

FUNCTION
Next(Current_number : Three_char) : Three_char;
VAR
Digit1, Digit2, Digit3 : Char;

Begin
Readv (Current_number, Digit1, Digit2, Digit3);

If Digit2 < '9'
Then Next := Digit1 + Succ(Digit2) + '0'
Else
If Digit1 < '9'
Then Next := Succ(Digit1) + '0' + '0'
Else Next := '000';

End;

PROCEDURE
Create_pairing_card;

Begin

New(New_pairing_card);

New_pairing_card^.Next_card := Nil;
New_pairing_card^.Pairing_number := Player.Pairing_number;
New_pairing_card^.Score := 0.0;
New_pairing_card^.Odd_man_status := 0;
New_pairing_card^.Color_due_status := 0;
New_pairing_card^.Unfinished_game_status := '1';
New_pairing_card^.Bye_given_status := 'N';
For Index := 1 To Max_rounds Do New_pairing_card\$-Opponent[Index] := ' ';
End;

PROCEDURE
Add_to_end_of_scored;
Begin
If End_of_scored = Nil
Then Scored_pointer := New_pairing_card
Else End_of_scored\$.Next_card := New_pairing_card;
End_of_scored := New_pairing_card;
End;

Begin
(* MAIN BODY OF MODULE *)
 Lib$Set_cursor(Row:=20,Col:=1);
 WriteIn(1 Preparing files and pairing cards for round one.*));
 Lib$Spawn(Command:='Sort/Key=(Position:34,Size:4,Descending) Tournament_roster,Dat Temporary_roster.Dat Temporary_roster.Dat' );
 Reset (Temporary_roster );
 Rewrite(Tournament_roster );
 Current_number := '010';
 While not (eof(Temporary_roster ) and not (Current_number = 'ono') Do
 Begin
 Read (Temporary_roster , Player);
 Player\$.Pairing_number := Current_number;
 Highest_pairing_number := Current_number;
 Create_pairing_card;
 Add_to_end_of_scored;
 Write(Tournament_roster ,Player);
 Current_number := Next(Current_number);
 End;
 Close (Temporary_roster );
 Close (Tournament_roster );
 Lib$Spawn(Command:='Delete/Noconfirm Temporary_roster,Dat;*');
 If Current_number = '000'
Then Begin
Lib$Set_cursor(Row:=22, Col:=1);  
Writeln(* Limit of 99 players reached... Rest ignored*);  
End;

Lib$Spawn(Command:='Convert/Fd=Roster Tournament_roster.Dat Tournament_roster.Dat');  
End;
PROGRAM: Place_cards_in_scored_list (34) DATE: April 25

PURPOSE: This module moves a newly-scored pairing card from the paired card list to the scored list.

PROCEDURE

Place_cards_in_scored_list;

Var

Current, Previous : Card_pointer;

Procedure Advance_one_cell;

Begin

  Previous := Current;
  Current := Current^.Next_card;

End;

Begin

  For Array_index := 1 to 2 do

  Begin

    (** REMOVE CARD FROM PAIRED LIST **)
    (** **)

    If Desired_card[Array_index] = Previous_card[Array_index]
      Then Paired_pointer := Desired_card[Array_index]^.Next_card
        Else Previous_card[Array_index]^.Next_card := Desired_card[Array_index]^.Next_card;

    If ( Array_index = 1 ) and
      ( Desired_card[1] = Previous_card[2] )
      Then If ( Previous_card[2] = Previous_card[1] )
        Then Previous_card[2] := Paired_pointer
       Else Previous_card[2] := Previous_card[1];

    (** PLACE CARD IN SCORED LIST AT THE PROPER LOCATION **)
    (** **)

    If Scored_pointer = Nil
      Then Begin

        (** PLACE CARD IN SCORED LIST *)

      End;

End;
Desired_card[Array_index]^Next_card := Nil;
Scored_pointer := Desired_card[Array_index];
End

Else Begin

(* LOCATE PROPER POSITION IN SCORED LIST *)

Previous := Scored_pointer;
Current := Scored_pointer;

While ( Current <> Nil ) and
  ( Current^Score > Desired_card[Array_index]^Score )
  do Advance_one_cell;

While ( Current <> Nil ) and
  ( Current^Score = Desired_card[Array_index]^Score ) and
  ( Current^Pairing_number < Desired_card[Array_index]^Pairing_number )
  do Advance_one_cell;

(* PLACE CARD IN SCORED LIST *)

Desired_card[Array_index]^Next_card := Current;

If Current = Previous
  Then Scored_pointer := Desired_card[Array_index]
  Else Previous^Next_card := Desired_card[Array_index];

End;

End;
(* *************************************************************** *)
(* PROGRAM : Enter_round_result (35)          DATE : April 21 *)
(* PURPOSE : This module allows entry of a game result. *)
(* *************************************************************** *)

PROCEDURE
Enter_round_result;

VAR

Screen_is_okay     : Boolean;

(* DECLARE INTERNAL REFERENCES *)
(*

Procedure Get_data_fields:

Var

Message1,Message2 : Packed array [1..38] of Char;
Pairing_number     : Varying [10] of Char;

Begin
Lib$Erase_page(Row:=7, Col:=1);
Lib$Set_cursor(Row:=7, Col:=1);
Write (' Result (W or D) : '); Repeat
Readln( Game_result.Result )
Until Game_result.Result in ['W','D','W','D'];

If ( Game_result.Result = 'W' ) or ( Game_result.Result = 'W' )
Then Begin
  Message1 := ' Pairing # of winner : ';
  Message2 := ' Pairing # of loser : ';
Else Begin
  Message1 := ' Pairing # of player 1 : ';
  Message2 := ' Pairing # of player 2 : ';
End;

Write ( Message1 );
Repeat
  Readln( Pairing_number )
Until ( Length(Pairing_number) = 3 ) and
  ( Pairing_number >= '001' ) and
  ( Pairing_number <= Highest_pairing_number );

(* *************************************************************** *)
Game_result.Player[1] := Pairing_number;

Write ( Message2 );
Repeat
   Readln( Pairing_number )
   Until ( Length(Pairing_number) = 3 ) and
   ( Pairing_number >= '001' ) and
   ( Pairing_number <= Highest_pairing_number ) and
   ( Pairing_number <> Game_result.Player[1] ) ;

Game_result.Player[2] := Pairing_number;
End;

Procedure Display_screen:
Begin
   LibSErase_page(Row:=1,Col:=1);
   LibSSet_cursor(Row:=1,Col:=1);
   Writeln('******************************************************************************');
   Writeln('*');
   Writeln('* ENTER A ROUND RESULT *');
   Writeln('*******************************************************************************');
   LibSSet_cursor(Row:=6,Col:=1);
   Writeln( 'Result : ' / Game_result.Result );
   If ( Game_result.Result = 'W' ) or ( Game_result.Result = 'w' )
   Then Begin
      Writeln( 'Pairing # of winner : ' / Game_result.Player[1] );
      Writeln( 'Pairing # of loser : ' / Game_result.Player[2] );
   End
   Else Begin
      Writeln( 'Pairing # of player 1: ' / Game_result.Player[1] );
      Writeln( 'Pairing # of player 2: ' / Game_result.Player[2] );
   End;
End;

Procedure Get_affirmation:
Var
   Screen_response : Char ;
Begin
   Repeat
      Begin
         LibSErase_page(Row:=73,Col:=1);
         LibSSet_cursor(Row:=23,Col:=1);
   End;
End;
Write ('* Is all the information on this screen OK? (Y or N) : ');
Readln (Screen_response);

If (Screen_response = 'R') or
  (Screen_response = 'r')
  Then Display_screen
End

Until Screen_response in [ 'Y' , 'N' , 'y' , 'n' ];

Case Screen_response of

  'R' , 'r' : Screen_is_okay := True;
  'N' , 'n' : Screen_is_okay := False;
End;

End;

Begin

 Lib$Erase_page(Row:=1,Col:=1);
 Lib$Set_cursor(Row:=1,Col:=1);
 Writeln(* ************************************************* *);
 Writeln(* ENTER A ROUND RESULT *);
 Writeln(* ************************************************* *);

(* GET GENERAL TCURNAMENT INFORMATION *)
(*

Repeat

  Get_data_fields;
  Get_affirmation

  Until Screen_is_okay;

End;
(* PROGRAM: Record_round_results (36) DATE: April 22 *)
(* PURPOSE: This module records a game result on a pairing card and in the appropriate roster record. *)

PROCEDURE

Record_round_results;

Begin

(* UPDATE THE PAIRING CARDS *)

WriteLn( "Updating the pairing_cards");

If ( Game_result.Result = 'W' ) or ( Game_result.Result = 'W' )
Then Desired_card[1].Score := Desired_card[1].Score + 1.0
Else Begin

Desired_card[1].Score := Desired_card[1].Score + 0.5;
Desired_card[2].Score := Desired_card[2].Score + 0.5;

End;

(* UPDATE THE ROSTER *)

WriteLn( "Updating the tournament_roster");

Open ( Tournament_roster /
History := Old /
Organization := Indexed /
Access_method := Keyed )

Reset( Tournament_roster /

If ( Game_result.Result = 'W' ) or ( Game_result.Result = 'W' )
Then Begin

Findk(Tournament_roster, Game_result.Player[1]);
Tournament_roster^.Result[Round_number] := 1.0;
Tournament_roster^.Score := Tournament_roster^.Score + 1.0;
Update(Tournament_roster);

Findk(Tournament_roster, Game_result.Player[2]);
Tournament_roster^.Result[Round_number] := 0.0;
Update(Tournament_roster);

End

Else Begin

Findk(Tournament_roster, Game_result.Player[1]);

End

End

End

End

End
Tournament_roster^Result[Round_number] := 0.5;
Tournament_roster^Score := Tournament_roster^Score + 0.5;
Update(Tournament_roster);

Findk(Tournament_roster,0,Game_result.Player(2));
Tournament_roster^Result[Round_number] := 0.5;
Tournament_roster^Score := Tournament_roster^Score + 0.5;
Update(Tournament_roster);

End;

Close(Tournament_roster);

End;
PROGRAM locate_pairing_cards (37) DATE: April 22

PURPOSE: This module locates a pairing card based on the pairing number in question.

PROCEDURE

Locate_pairing_cards:

VAR

Previous : Card_pointer;
Current_card : Card_pointer;
Found_one : Boolean;
Found_two : Boolean;
Card_pointer;
Card_pointer;

Begin

(* Locate the pairing cards *)

Writeln('Locating the pairing cards');

Found_one := False;
Found_two := False;

Previous := Paired_pointer;
Current_card := Paired_pointer;

While (Current_card <> Nil) And Not (Found_one and Found_two) Do

Begin

If Current_card^.Pairing_number = Game_result.Player[1]
Then Begin
  Previous_card[1] := Previous;
  Desired_card[1] := Current_card;
  Found_one := True;
End
Else If Current_card^.Pairing_number = Game_result.Player[2]
Then Begin
  Previous_card[2] := Previous;
  Found_two := True;
End;

Previous := Current_card;
Current_card := Current_card^.Next_card;

End;

(* Test for appropriate conditions *)
If not found_one
  then if not found_two
    then Error_code := 3
    else Error_code := 1
  else if not found_two
    then Error_code := 2
    else Error_code := 0;

End;

(*<--- This is where the test for *)
(* the players having been *)
(* paired is to be made -- in *)
(* the final else block. For *)
(* now assumed OK. *)
(* *** *)
Program: Prepare_for_pairing (38)  Date: April 30

Purpose: This module eliminates unfinished games and assigns a bye to the odd man (if there is one).

******************************************************************************

PROCEDURE

Prepare_for_pairing;

******************************************************************************

DECLARE EXTERNAL REFERENCES *)

*(Include '43' *)
 *(Include '44' *)
 *(Include '45' *)

Begin

WriteLn('Preparing for the pairings...');

(* The following code will be used to handle all unfinished games *)
(* when this feature of the system is ready to be introduced. *)
(*)
(* If Paired_pointer <> Nil *)
(* Then Case Scoring_unfinished_games of *)
(* '*1' : Enter_adjudicated_scores; *)
(* '*2' : Assign_score_to_unfinished_games; *)
(* End; *)

If ( (Number_of_players REM 2) = 1 ) Then Assign_bye_to_odd_man;

End;