(* *************************************************************)
(* PROGRAM : Pair_odd_men (39)                                *)
(* DATE : May 2                                               *)
(* PURPOSE : This module pairs any odd man from above and, if *)
(* necessary, removes an odd man from the current            *)
(* score group.                                               *)
(* ***************************************************************

PROCEDURE
   Pair_odd_men;

VAR
   Dummy_character : Varying [1] of Char;

(* DECLAre EXTERNAL REFERENCES *)
(* INCLUDE '47'                     *)
(* INCLUDE '48'                     *)

Begin
   Writeln(' Pairing_odd_men ...');
   If Odd_man_pending
      Then Begin
         Pair_odd_man_from_above;
         Odd_man_pending := False;
      End;
   If ( ((Number_in_group REM 2) = 1) )
      Then Begin
         Remove_odd_man_from_group;
         Odd_man_pending := True;
      End;
   End;
(* ********************************************************************)
(* PROGRAM : Pair_score_group (3a)                          DATE: April 9 *)
(* PURPOSE: This module pairs the current score group. *)
(* ********************************************************************)

PROCEDURE
   Pair_score_group;

VAR
   Dummy_character : Varying [1] of Char;

(*
(* DECLARE EXTERNAL REFERENCES *)
(*

XInclude '49'
XInclude '4a'
XInclude '4b'

*)

Begin
   Writeln(' Pairing score group. ...');
   Legal_pairings_exist := True;
   Match_top_with_bottom;
   Eliminate_illegal_pairings;
   If Legal_pairings_exist
      Then Allocate_colors
   Else Begin
      Lib$Erase_page(Row:=1, Col:=1);
      Lib$Set_cursor(Row:=10, Col:=1);
      writeln(' Help!! A situation has arisen in one of the score groups');
      writeln(' that I am unable to resolve (given the current status of my');
      writeln(' software!). Take a look at the pairing cards and resolve this');
      writeln(' problem. Hit <RETURN> to continue : ');
      Readln (Dummy_character);
      End;
   End;

End;

PROGRAM: Print_pairing_list (3b) DATE: May 9

PURPOSE: This module prints the newly-made pairing list.

PROCEDURE

Print_pairing_list;

VAR

Dummy_character : Varying [1] of Char;
Index : Integer;
Print_selection : Char;
Desired_queue : Varying [10] of Char;
Table_number : Packed array [1..2] of Char;
Player1_name / Player2_name : Packed array [1..30] of Char;
Player1 / Player2 : Card_pointer;

Function Character(Digit : Integer) : Char;
Begin
Case Digit of
  1 : Character := '1';
  2 : Character := '2';
  3 : Character := '3';
  4 : Character := '4';
  5 : Character := '5';
  6 : Character := '6';
  7 : Character := '7';
  8 : Character := '8';
  9 : Character := '9';
End;
End;

Procedure Get_next_table_number;

Var Digit1 / Digit2 : Char;
Begin
ready(Table_number, Digit1, Digit2);

if Digit2 < '9'
then Table_number := Digit1 + succ(Digit2)
else if Digit1 = '
then Table_number := '1' + succ(Digit2)
else Table_number := succ(Digit1) + succ(Digit2);

procedure Write_header;
begin
writeln(Pairing_list);
writeln(Pairing_list);
writeln(Pairing_list, 'TOURNAMENT : ' + Tournament_name);
writeln(Pairing_list);
writeln(Pairing_list, 'ROUND : ' + Character(Round_number));
writeln(Pairing_list);
writeln(Pairing_list, 'TABLE WHITE BLACK');
end;

procedure Write_pairing_lines;
begin
Table_number := '1';
Player1 := Paired_pointer;
while Player1 <> nil do
begin
Findk(Tournament_roster, Player1^, Pairing_number);
Player1_name := Tournament_roster^, Name;
if Player1^, Next_card = nil
then begin
Player2_name := 'Wy';
Player1 := Player1^, Next_card;
end
else begin
Player2 := Player1^, Next_card;
Findk(Tournament_roster, Player2^, Pairing_number);
Player2_name := Tournament_roster^, Name;
Player1 := Player2^, Next_card;
end;
Writeln(Pairing_list,"*"); + Table_number + "+ Player1_name + "+ Player2_name );

Get_next_table_number;
End;

End;

Begin
Writeln("Preparing pairing list file... ");

(** CREATE THE PAIRING LIST **) Rewrite(Pairing_list);
Write_header;
Open (Tournament_roster
, History := Old
, Organization := Indexed
, Access_method := Keyed
); Resetk(Tournament_roster,0)
Write_pairing_lines;
Close(Pairing_list);
Close(Tournament_roster);

(** PRINT THE PAIRING LIST **) Lib$Erase_page(Row:=16,Col:=1);
Lib$Set_cursor(Row:=16,Col:=1);
Writeln(1 Print pairing list to attached printer
);
Writeln(2 Print pairing list on selected queue
);
Writeln(3 Return to main menu
);
Lib$Set_cursor(Row:=16,Col:=1);
Write (Enter your selection : ");
Readln(Print_selection);
While Not ( Print_selection in [ '1', '2', '3' ] ) Do
Begin
Lib$Erase_page(Row:=15,Col:=1);
Lib$Set_cursor(Row:=17,Col:=1);
Writeln(1 INVALID SELECTION !!");
Writeln(2 Enter your selection : ");
Readln(Print_selection);
End;

Case Print_selection of
'1' : For Index := 1 to 3 Do Lib$Spawn("Prt Pairing_list.dat");
'2' : Begin
 Lib$Erase_page(Row:=16,Col:=1)
;
Lib$Set_cursor(Row:=17, Col:=1);
Write (' Enter desired queue : ');
Readln (Desired_queue);
Lib$Spawn('Print' + Desired_queue + ' Pairing_List.Dat/Copies=3');
End;

Lib$Set_cursor(Row:=19, Col:=1);
Write (' Hit <RETURN> to continue : ');
Readln (Dummy_character);
End;
**PROGRAM Update_pairing_cards**  
**DATE : May 12**  

**PURPOSE : This module updates the player records and the pairing cards for the new pairings.**

---

**PROCEDURE**

Update_pairing_cards;

Var

  Curr, Prev : Card_pointer;
  Score_value : Real;

Begin

  (** Prepare screen display **)  
  LibSpeak(Row:=10, Col:=1);  
  Write(' Updating the pairing_cards');

  (** Prepare file for reading **)  
  Open ( Tournament_roster = Old,  
       History := Old,  
       Organization := Indexed,  
       Access_method := Keyed );  
  Reset(Tournament_roster = 0 );

  (** Initialize pointers **)  
  Prev := Paired_pointer;
  Curr := Prev^.Next_card;

  (** Perform update for each card **)  
  While ( Prev <> Nil ) and ( Curr <> Nil ) Do  
    Begin  
      (* Update color due status on pairing cards *)  
      Case Prev^.Color_due_status of  
        +3 ,  

PrevA.Color_due_status := +2;
0  := PrevA.Color_due_status := +1;
-1  := PrevA.Color_due_status := +1;
-2  := PrevA.Color_due_status := -1;
-3  := PrevA.Color_due_status := +1;
End;

Case CurrA.Color_due_status of
+3  := CurrA.Color_due_status := -1;
+2  := CurrA.Color_due_status := +1;
+1  := CurrA.Color_due_status := -1;
 0  := CurrA.Color_due_status := -1;
-1  := CurrA.Color_due_status := -1;
-2  := CurrA.Color_due_status := -2;
-3  := CurrA.Color_due_status := -2;
End;

(* Update opponent on pairing cards *)
CurrA.Opponent[Round_number] := PrevA.Pairing_number;
PrevA.Opponent[Round_number] := CurrA.Pairing_number;

(* Update player records *)
Findk (Tournament_roster, 0, CurrA.Pairing_number);
Tournament_rosterA.Opponent[Round_number] := PrevA.Pairing_number;
Tournament_rosterA.Color[Round_number] := 'B';
Update(Tournament_roster);
Findk (Tournament_roster, 0, PrevA.Pairing_number);
Tournament_rosterA.Opponent[Round_number] := CurrA.Pairing_number;
Tournament_rosterA.Color[Round_number] := 'W';
Update(Tournament_roster);

(* Advance pointers *)
Prev := CurrA.Next_card;
If Prev <> Nil
Then Curr := PrevA.Next_card;
End;

(** Update records for byed player **) If Prev <> Nil
Then Begin
(* Assign byed player's score *)
Case Scoring_byes of
 0  := Score_value := 1.0;
1  := Score_value := 0.5;
End;

(* Update pairing card *)
Prev*.Score := Prev*.Score + Score_value;
Prev*.Bye_given_status := 'Y';
Prev*.Opponent[Round_number] := 'BYE';

(*) Update player record *)

Findk (Tournament_roster / 0, Prev*.Pairing_number);
Tournament_roster*.Opponent[Round_number] := 'BYE';
Tournament_roster*.Color[Round_number] := * ;
Tournament_roster*.Result[Round_number] := Score_value;
Tournament_roster*.Score := Tournament_roster*.Score + Score_value;
Update( Tournament_roster );

(*) Locate second-to-last card in paired list *)

Curr := Paired_pointer;

(*) Remove byed player from paired list and place in scored list *)

Curr*.Next_card := Nil;
Prev*.Next_card := Nil;
Scored_pointer := Prev;
End;

(** Close file **)  
Close(Tournament_roster);

End;
PROGRAM Create_crosstable (3d)  DATE: April 27

PURPOSE: This module serves as the control procedure for creation of a new crosstable.

PROCEDURE

Create_crosstable:

DECLARE EXTERNAL REFERENCES
XIInclude '4c'
XIInclude '4d'
XIInclude '4e'

Begin

Lib$Erase_page(Row:=10,Col:=1);
Lib$Set_cursor(Row:=10,Col:=1);
WriteIn('1) Crosstable in wallchart order');
WriteIn('2) Crosstable in game-score order');
WriteIn('3) Crosstable in tiebreak order');
WriteIn('4) Return to the main menu');

Lib$Set_cursor(Row:=16,Col:=1);
Write ('Enter your selection: ');
Readln(Order_selection);

While Not (Order_selection in ['1', '2', '3', '4'] ) Do Begin
Lib$Erase_page(Row:=16,Col:=1);
Lib$Set_cursor(Row:=17,Col:=1);
WriteIn(INVALID SELECTION !!);
Write('Enter your selection: ');
Readln(Order_selection);
End;

If Order_selection <> '4' Then Begin

ACTUAL PREPARATION OF CROSSTABLE

If Order_selection = '3' Then Eliminate_tied_scores;
Rewrite(Crosstable);
Lib$Erase_page(Row:=7, Col:=1);
Write_header_Items;
Write_detail_lines;
Close(Crosstable);
End;

End;
PROGRAM : Output_crosstable (3e)  DATE : May 16

PURPOSE : This module serves as the control procedure that allows the user to see the current crosstable.

PROCEDURE

Output_crosstable;

VAR

Dummy_character : Varying [1] of Char;
Desired_queue : Varying [10] of Char;

Begin

Lib$Erase_page(Row:=10,Col:=1);
Lib$Set_cursor(Row:=10,Col:=1);
Write(' (1) Print to attached printer');
Write(' (2) Print crosstable on selected queue');
Write(' (3) Return to main menu');

Lib$Set_cursor(Row:=16,Col:=1);
Write ('Enter your selection: ');
Readln(Print_selection);

While Not (Print_selection in [ '1' , '2' , '3' ]) Do Begin
Lib$Erase_page(Row:=15,Col:=1);
Lib$Set_cursor(Row:=17,Col:=1);
Write(' INVALID SELECTION !!!');
Write (' Enter your selection: ');
Readln(Print_selection);
End;

Case Print_selection of

'1' : Begin
Lib$Erase_page(Row:=1,Col:=1);
Lib$Set_cursor(Row:=1,Col:=1);
Lib$Spawn("Prt_crosstable.Dat");
Write(' End of crosstable file.');
Write (' Hit <RETURN> to continue: ');
Readln (Dummy_character);
End;

'2' : Begin

End.
Lib$Erase_page(Row:=16,Col:=1);
Lib$Set_cursor(Row:=17,Col:=1);
Write ('Enter desired queue : ');
Readln (Desired_queue);
Lib$Spawn('Print' + Desired_queue + ' Crosstable.Dat');
Lib$Set_cursor(Row:=19,Col:=1);
Write ('Hit <RETURN> to continue : ');
Readln (Dummy_character);
End;

End;
(******************************************************************************)
(*  PROGRAM:  Assign_bye_to_odd_man (45)  DATE:  April 30 *)
(*  PURPOSE:  This module locates the lowest-ranked eligible *)
(*           player and assigns him a bye.*)
(*  *)
(* ******************************************************************************)

PROCEDURE
Assign_bye_to_odd_man;

VAR
   Current ;
   Previous ;
   Byed_player ;
   Before_byed : Card_pointer;

Procedure Locate_lowest_eligible_player;
   Begin
      Byed_player := Nil;
      Before_byed := Nil;
      Previous := Scored_pointer;
      Current := Scored_pointer;

      While Current <> Nil Do
         If Current^.Bye_given_status = 'N'
            Then Begin
                     Before_byed := Previous;
                     Byed_player := Current;
                     End;
         Previous := Current;
         Current := Current^.Next_card;
      'End
   End;

   Begin
      Locate_lowest_eligible_player;
      Byed_player^.Bye_given_status := 'Y';
      If Byed_player = Scored_pointer
         Then Scored_pointer := Byed_player^.Next_card
         Else Before_byed^.next_card := Byed_player^.Next_card;
      Byed_player^.Next_card := Nil;

   End;}
Paired_pointer := Byed_player;
End_of_paired := Byed_player;
---Ends---
*******

(* PURPOSE: This module segregates a score group from the scored pointer list. *)

PROCEDURE

Create_a_score_group;

VAR

Current, Previous: Card_pointer;
Group_score: Real;

Begin

WriteInt(' Creating a score group. . .');

If Not Odd_man_pending

Then Number_in_group := 0

Else If Odd_man^Next_card = Nil

Then Number_in_group := 1

Else Number_in_group := 2;

Previous := Scored_pointer;
Current := Scored_pointer;

If Current <> Nil

Then Begin

Group_score := Current^Score;
Current := Current^Next_card;
Number_in_group := Number_in_group + 1;
End;

While (Current <> Nil) and (Current^Score = Group_score) Do

Begin

Previous := Current;
Current := Current^Next_card;
Number_in_group := Number_in_group + 1;
End;

Score_group := Scored_pointer;
Scored_pointer := Current;
Previous^Next_card := Nil;

End;
(* PROGRAM: Pair_odd_man_from_above (47)   DATE: May 2 *)

(* PURPOSE: This module pairs the odd man removed from the previous score group with the highest-ranked eligible player from the current group. *)

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

PROCEDURE Pair_odd_man_from_above;

VAR

Index : Integer;
Eligible ;
Elig_previous ;
Current ;
Current2 ;
Previous : Card_pointer;
No_eligible_players ;
They_have_not_played ;
Player_is_eligible : Boolean;

Procedure Locate_an_eligible_player;

Begin

Player_is_eligible := False;
Previous := score_group;
Current := score_group;

Repeat

They_have_not_played := True;
For Index := 1 to Round_number - 1 Do
  If Current^.Component[Index] = Odd_man^.Pairing_number
    Then They_have_not_played := False;

If They_have_not_played
  Then Player_is_eligible := True
Else Begin
  Previous := Current;
  Current ^= Current^.Next_card;
End;

Until ( Player_is_eligible ) or ( Current = Nil );

If Current = Nil Then No_eligible_players := True;
End;

(* *******************************************************)
Procedure Remove_players_from_lists
Begin
If Current = Previous
    Then Score_group := Current^.Next_card
Else Previous^.Next_card := Current^.Next_card;
Current2 := Odd_man;
Odd_man := Odd_man^.Next_card;
End;

Procedure Insert(Var Pointer : Card_pointer);
Var Curr, Prev : Card_pointer;
Begin
Prev := Paired_pointer;
Curr := Paired_pointer;
While Curr <> End_of_paired Do
Begin
    Prev := Curr;
    Curr := Curr^.Next_card;
End;
Pointer^.Next_card := End_of_paired;
If Paired_pointer = End_of_paired
Then Paired_pointer := Pointer
Else Prev^.Next_card := Pointer;
End;

Procedure Insert_players_in_paired_list;
Var Odd_color : Char;
Begin
Case Current^.Color_due_status of
+3 : Odd_color := 'b';
+2 : Odd_color := 'h';
+1 : If Current^.Color_due_status > +1
    Then Odd_color := 'w'
    Else Odd_color := 'b';
0  : If Current^.Color_due_status < 0
    Then Odd_color := 'h'
    Else Odd_color := 'w';
-1 : If Current^.Color_due_status < -1
    Then Odd_color := 'b';
Else Odd_color := 'w';
-2 : Odd_color := 'w';
-3 : Odd_color := 'b';
End;

If Odd_color = 'w'
Then Begin
  Insert(Current2);
  Insert(Current);
End
Else Begin
  Insert(Current);
  Insert(Current2);
End;

Current.A.Odd_man_status := Current.A.Odd_man_status + 1;
Current.Z.A.Odd_man_status := Current.Z.A.Odd_man_status - 1;
End;

Begin
  Repeat
    Locate_an_eligible_player;
    If No_eligible_players
      Then Number_in_group := Number_in_group - 1
      Else Begin
        Remove_players_from_lists;
        Insert_players_in_paired_list;
        Number_in_group := Number_in_group - 2;
      End;
    Until Odd_man = Nil;
  End;
*******************************************************************************
(* PROGRAM : Remove_odd_man_from_group (48) DATE : May 3 *)
(* PURPOSE : This module removes the odd man from a score *)
(* )
(* )
*******************************************************************************

PROCEDURE

Remove_odd_man_from_group;

VAR

Current : Card_pointer;
Previous : Card_pointer;

Begin

(** Find last card of score_group -- Current points to it **) 
Current := Score_group;
Previous := Score_group;

While Current^.Next_card <> Nil Do
Begin
  Previous := Current;
  Current := Current^.Next_card;
End;

(** Remove this card from the score group list **) 
If Current = Previous
Then Score_group := Nil
Else Previous^.Next_card := Nil;
Current^.Next_card := Nil;

(** Place card in odd man list **) 
If Odd_man = Nil
Then Odd_man := Current
Else Begin

(** Find last card in odd man list -- Previous points to it **) 
Previous := Odd_man;

While Previous^.Next_card <> Nil Do
  Previous := Previous^.Next_card;

(** Place odd-man card **) 

pointed to by Current in the odd-man list, after the card pointed to by Previous **)

Previous^.Next_card := Current;
End;

Number_in_group := Number_in_group - 1;
End;
(*  PROGRAM : Match_top_with_bottom (49)   DATE : April 9 *)
(*  PURPOSE : This module matches the top half of the score group with the bottom half. *)
(*  ------------------------------------------------------------------)*)

PROCEDURE
Match_top_with_bottom;

VAR
Dummy_character   : Varying [1] of Char;
Current            : Card_pointer;
Index1, Index2     : Integer;

Begin

  Current := Score_group;

  For Index2 := 1 to 2  Do
    For Index1 := 1 to Number_in_group DIV 2  Do
      Begin
        Pairing[Index1, Index2] := Current;
        Current := Current^.Next_card;
      End;
    End;

End;
(* PROGRAM : Eliminate Illegal pairings (4a) DATE : May 12 *)

(* PURPOSE : This module eliminates illegal pairs. *)

PROCEDURE

Eliminate_illegal_pairings;

VAR

Dummy_character : Varying [1] of Char;
Pairing_is_invalid
Switch_is_bad
More_to_check
No_more_above
No_more_below : Boolean;
Highest_pairing
Index
Round
Offset : Integer;
Hold_card : Card_pointer;

Procedure Determine_if_pairing_is_valid;

Begin
Pairing_is_invalid := False;
For Round := 1 to Round_number - 1 Do
   If Pairing[Index,1]^Pairing_number = Pairing[Index,2]^Opponent[Round]
      Then Pairing_is_invalid := True;
End;

Procedure Resolve_pairing_problem;

Var Switch_is_valid : Boolean;

Begin

More_to_check := True;
Switch_is_bad := True;
No_more_below := False;
No_more_above := False;
Offset := 1;

While Switch_is_bad and More_to_check Do
   Begin
If Not No_more_below Then If Index + Offset <= Highest_pairing
Then Begin
  Switch_is_valid := True;
  For Round := 1 to Round_number - 1 Do
  Begin
    If Pairing[Index,1] = Pairing[Index+Offset,2].Opponent[Round] Then Switch_is_valid := False;
    If Pairing[Index+Offset,1] = Pairing[Index,2].Opponent[Round] Then Switch_is_valid := False;
  End;
  If Switch_is_valid Then Begin
    Hold_card := Pairing[Index,2];
    Pairing[Index,2] := Pairing[Index+Offset,2];
    Pairing[Index+Offset,2] := Hold_card;
    Switch_is_bad := False;
  End;
End;
Else No_more_below := True;

If Not No_more_above Then If ( Index - Offset > 0 ) and Switch_is_bad
Then Begin
  Switch_is_valid := True;
  For Round := 1 to Round_number - 1 Do
  Begin
    If Pairing[Index,1] = Pairing[Index-Offset,2].Opponent[Round] Then Switch_is_valid := False;
    If Pairing[Index-Offset,1] = Pairing[Index,2].Opponent[Round] Then Switch_is_valid := False;
  End;
  If Switch_is_valid Then Begin
    Hold_card := Pairing[Index,2];
    Pairing[Index,2] := Pairing[Index-Offset,2];
    Pairing[Index-Offset,2] := Hold_card;
    Switch_is_bad := False;
  End;
End;
Else No_more_above := True;

If No_more_above and No_more_below Then More_to_check := False
Else Offset := Offset + 1;
End;

Procedure Resolve_on_right_side;
Var
  Switch_is_valid : Boolean;
Special_mark := Index + 1;

Begin

While ( Switch_is_bad ) and ( Special_mark <= Highest_pairing ) Do 
Begin
Switch_is_valid := True;
For Round := 1 to Round_number - 1 Do 
Begin
If Pairing[Index,1]^.Pairing_number = Pairing[Special_mark,2]^.Opponent[Round] 
Then Switch_is_valid := False;
End;
End;
If Switch_is_valid
Then Begin
Hold_card := Pairing[Index,2];
Pairing[Index,2] := Pairing[Special_mark,2];
Pairing[Special_mark,?] := Hold_card;
Switch_is_bad := False;
End
Else Special_mark := Special_mark + 1;
End;
End;

Procedure Sort_remaining_pointers;
Var
I , J : Integer;
Begin
For I := Index + 1 to Highest_pairing - 1 Do 
For J := Index + 1 to Highest_pairing - 1 Do 
If Pairing[I,1]^.Pairing_number > Pairing[I+1,1]^.Pairing_number 
Then Begin
Hold_card := Pairing[I,1];
Pairing[I,1] := Pairing[I+1,1];
Pairing[I+1,1] := Hold_card;
End;
End;
Procedure Resolve_on_left_side;

Var
Switch_is_valid : Boolean;
Special_mark : Integer;

Begin
Special_mark := Highest_pairing;
While (Switch_is_bad) and (Special_mark > Index) Do
Begin
Switch_is_valid := True;
For Round := 1 to Round_number - 1 Do
Begin
If Pairing[Index,1]^Pairing_number = Pairing[Special_mark,1]^Opponent[Round]
Then Switch_is_valid := False;
End;
If Switch_is_valid
Then Begin
Hold_card := Pairing[Index,2];
Pairing[Index,2] := Pairing[Special_mark,1];
Pairing[Special_mark,1] := Hold_card;
Switch_is_bad := False;
Sort_remaining_pointers;
End
Else Special_mark := Special_mark - 1;
End;
End;

Begin
Highest_pairing := Number_in_group DIV 2;
For Index := 1 to Highest_pairing Do
Begin
Determine_if_pairing_is_valid;
If Pairing_is_invalid Then Resolve_pairing_problem;
If Switch_is_bad Then Resolve_on_right_side;
If Switch_is_bad Then Resolve_on_left_side;
If Switch_is_bad Then Legal_pairings_exist := False;
End;
End;
PROCEDURE
Allocate_colors;

TYPE
Priority = ( Okay, Should, Must );

VAR
Dummy_character : Varying [1] of Char;
Switch_is_made /
More_to_check /
No_more_above /
No_more_below : Boolean;
Highest_pairing /
Index /
Round /
Offset : Integer;
Color_change : Priority;
Color_matrix : Array [ -3..+3 , -3..+3 ] of Integer;
Hold_card : Card_pointer;
Seed : Real;

Function For$Secnds(Value:Real):Real; External;
Function Mth$Random(Seed:Real):Real; External;

Function Color_priority ( Left, Right: Integer ) : Priority;
  Begin
    If ( Abs(Pairing[Left,1]^, Color_due_status) >= ? ) and
        ( Abs(Pairing[Right,2]^, Color_due_status) >= 2 )
      Then Color_priority := Must
    Else If ( Abs(Pairing[Left,1]^, Color_due_status) > 0 ) and
        ( Abs(Pairing[Right,2]^, Color_due_status) > 0 )
      Then Color_priority := Should
    Else Color_priority := Okay;
  End;

(*---------------------------------------------------------------------*)
(*  PROGRAM : Allocate_colors (4b)  DATE : May 7  *)
(*---------------------------------------------------------------------*)
(*  PURPOSE : This module allocates colors to the pairings.  *)
(*---------------------------------------------------------------------*)

Procedure Init_color_matrix;
Begin
    Color_matrix[+3,+1] := 2;
    Color_matrix[+3,+2] := 1;
    Color_matrix[+3,+3] := 2;
    Color_matrix[+3, 0] := 2;
    Color_matrix[+3,-1] := 2;
    Color_matrix[+3,-2] := 2;
    Color_matrix[+3,-3] := 2;
    Color_matrix[+2,+3] := 2;
    Color_matrix[+2,+2] := 2;
    Color_matrix[+2,+1] := 2;
    Color_matrix[+2,-1] := 2;
    Color_matrix[+2,-2] := 2;
    Color_matrix[+2,-3] := 2;
    Color_matrix[+1,+3] := 2;
    Color_matrix[+1,+2] := 2;
    Color_matrix[+1,+1] := 2;
    Color_matrix[+1,+0] := 2;
    Color_matrix[+1,-1] := 2;
    Color_matrix[+1,-2] := 2;
    Color_matrix[+1,-3] := 2;
    Color_matrix[ 0,+3] := 1;
    Color_matrix[ 0,+2] := 1;
    Color_matrix[ 0,+1] := 1;
    Color_matrix[ 0, 0] := 1;
    Color_matrix[ 0,-1] := 1;
    Color_matrix[ 0,-2] := 1;
    Color_matrix[ 0,-3] := 1;
End;

Procedure Improve_color_scheme;
Var Switch_is_valid : Boolean;
Begin
    More_to_check := True;
    Switch_is_made := False;
    No_more_below := False;
    No_more_above := False;
    Offset := 1;
    While ( Not Switch_is_made ) and ( More_to_check ) Do
    Begin
        If Not No_more_below
        Then If ( Index + Offset <= Highest_pairing )
Then begin
    Switch_is_vali_ := True;
    For Round := 1 to Round_number - 1 Do
        Begin
            If Pairing[Index,1] * Pairing_number = Pairing[Index+Offset,2] * Opponent(Round)
                Then Switch_is_valid := False;
            If Pairing[Index,1] * Pairing_number = Pairing[Index+Offset,2] * Opponent(Round)
                Then Switch_is_valid := False;
            end;
        If Switch_is_valid
            Then If (Color_priority[Index,Index+Offset] < Color_change ) and
                (Color_priority[Index,Index+Offset] < Color_change )
                Then Begin
                    Hold_card := Pairing[Index,2];
                    Pairing[Index,2] := Pairing[Index+Offset,2];
                    Pairing[Index+Offset,2] := Hold_card;
                    Switch_is_made := True;
                end;
        end;
    End
Else No_more_below := True;
If Not No_more_above
    Then If (Index - Offset > 0 ) and (Not Switch_is_made)
        Then begin
            Switch_is_valid := True;
            For Round := 1 to Round_number - 1 Do
                Begin
                    If Pairing[Index,1] * Pairing_number = Pairing[Index-Offset,2] * Opponent(Round)
                        Then Switch_is_valid := False;
                    If Pairing[Index,1] * Pairing_number = Pairing[Index-Offset,2] * Opponent(Round)
                        Then Switch_is_valid := False;
                    end;
                If Switch_is_valid
                    Then If (Color_priority[Index,Index-Offset] < Color_change ) and
                        (Color_priority[Index,Index-Offset] < Color_change )
                        Then Begin
                            Hold_card := Pairing[Index,2];
                            Pairing[Index,2] := Pairing[Index-Offset,2];
                            Pairing[Index-Offset,2] := Hold_card;
                            Switch_is_made := True;
                        end;
        end;
Else No_more_above := True;
If No_more_above and No_more_below
    Then More_to_check := False
Else Offset := Offset + 1;
End;

procedure Switch_the_cards;
    Begin
        Hold_card := Pairing[Index,1];
        Pairing[Index,1] := Pairing[Index,2];
Pairing[Index] := Hole_cards
End;

Procedure Make_color_assignments;
Begin
If Round_number = 1
Then
  If ((Offset = 1) and (Index REM 2 = 0)) or
     ((Offset = 2) and (Index REM 2 = 1))
  Then Switch_the_cards
  Else Begin
   End
Else Begin
   (* Is player 2 "most" due for white? *)
   If Color_matrix[Pairing[Index]]\Color_due_status = Pairing[Index].2\Color_due_status = 2
   Then Switch_the_cards;
   (* ELSE cards are already arranged correctly *)
   End;
End;

Begin
Highest_pairing := Number_in_group DIV 2;
Init_color_matrix;
For Index := 1 to Highest_pairing Do
  Begin
   Color_color_change := Color_priority(Index, Index);
   If Color_change > Okay
     Then Improve_color_scheme
  End;
If Round_number = 1
Then Begin
  Seed := For$Secnds(Value := 0.0);
  If Math$Random(Seed) >= 0.5
    Then Offset := 1
    Else Offset := 2;
End;
For Index := 1 to Highest_pairing Do
...
Make_color_assignments;

End;
(*******************************************************************************)
(*  PROGRAM : Eliminate_tied_scores (4c)  DATE : April 27  *)
(*  PURPOSE : This module performs tiebreak procedures on the *)
(*            the list of scored pairing cards.                *)
(*  *******************************************************************************)

PROCEDURE

Eliminate_tied_scores;

VAR

    Dummy_character  : Varying [1] of Char;

(*******************************************************************************)
(*  DECLARE EXTERNAL REFERENCES *)
(*  *******************************************************************************)

Begin

    LibErase_page(Row:=1,Col:=1);
    Lib$Set_cursor(Row:=1,Col:=1);
    Writeln('This option has not been codified yet. Your crosstable will be');
    Writeln('tiebroken' on the basis of pairing card number (rating) only."
    Write ('Hit <RETURN> to continue : ');
    Readln(Dummy_character);
    Writeln;
    End;
PROGRAM: Write_header_items (4d)  DATE: April 27

PURPOSE: This module writes the header for the crossstable to the crossstable file.

PROCEDURE

Write_header_items;

VAR

Index : Integer;
Char_rounds : Char;
Date_string : Varying [100] of Char;
Header_string : Varying [100] of Char;

Begin

Writing header items to the Crossstable file...

Reset(Tournament_statistics);

Readln (Tournament_statistics, Tournament_name);
Readln (Tournament_statistics, Sponsor);
Readln (Tournament_statistics, Director);
Readln (Tournament_statistics, Site);
Readln (Tournament_statistics, Starting_date);
Readln (Tournament_statistics, Ending_date);
Readln (Tournament_statistics, Number_of_rounds);

Write(Crosstable);

Write(Crosstable, 'TOURNAメント: '+Tournament_name);
Write(Crosstable);
Write(Crosstable, 'SPONSOR: '+Sponsor);
Write(Crosstable, 'DIRECTOR: '+Director);

If Starting_date = Ending_date
Then Date_string := 'STARTS: '+Starting_date + 'ENDS: '+Ending_date;
Else Date_string := 'STARTS: '+Starting_date + 'ENDS: '+Ending_date;

Write(Crosstable, Date_string);
Write(Crosstable, 'SITE: '+Site);
Write(Crosstable);
Write(Crosstable);
Write(Crosstable);

Header_string := '*';
For Index := 1 to Number_of_rounds Do
Begin

Case Index of
  1 : Char_rounds := '*1*';
  2 : Char_rounds := '*2*';
  3 : Char_rounds := '*3*';
  4 : Char_rounds := '*4*';
  5 : Char_rounds := '*5*';
6 : Char_rounds := '6';
7 : Char_rounds := '7';
8 : Char_rounds := '8';
9 : Char_rounds := '9';

End;

Header_string := Header_string + ' |' + Char_rounds + ' |';
End;

Writeln(Crosstable, 'No. Player Rating ' + Header_string + ' Total');
Writeln(Crosstable);

End;
PROGRAM:  Write_detail_lines (4e)   DATE:  May 13

PURPOSE:  This module writes the player-entry detail lines to the crosstable file.

PROCEDURE

Write_detail_lines;

TYPE

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

VAR

Dummy_character : Varying [1] of Char;
Current           : Card_pointer;
Player            : Player_record;
Index             : Integer;
Result_char       : Char;
Result_list       : Varying (80) of Char;

Function Printable(Score : Real) : Three_char;
Begin
  If Score = 0.0 Then Printable := '0.0';
  Else If Score = 0.5 Then Printable := '0.5';
  Else If Score = 1.0 Then Printable := '1.0';
  Else If Score = 1.5 Then Printable := '1.5';
  Else If Score = 2.0 Then Printable := '2.0';
  Else If Score = 2.5 Then Printable := '2.5';
  Else If Score = 3.0 Then Printable := '3.0';
  Else If Score = 3.5 Then Printable := '3.5';
  Else If Score = 4.0 Then Printable := '4.0';
  Else If Score = 4.5 Then Printable := '4.5';
  Else If Score = 5.0 Then Printable := '5.0';
  Else If Score = 5.5 Then Printable := '5.5';
  Else If Score = 6.0 Then Printable := '6.0';
  Else If Score = 6.5 Then Printable := '6.5';
  Else If Score = 7.0 Then Printable := '7.0';
  Else If Score = 7.5 Then Printable := '7.5';
  Else If Score = 8.0 Then Printable := '8.0';
  Else If Score = 8.5 Then Printable := '8.5';
  Else If Score = 9.0 Then Printable := '9.0';
End;

Function Result_fxn (Cpp : Three_char ; Res : Real) : Char;
Begin
If Opp == ' ' ' ' Then Result_fxn := ' ' ' ' Else If Res == 1.0 Then Result_fxn := 'w' Else If Res == 0.5 Then Result_fxn := 'p' Else If Res == 0.0 Then Result_fxn := 'L';

Procedure Write_a_line; Begin
  If Order_selection == '1' Then With Player Do Begin
    Result_list := '';
    For Index := 1 to Number_of_rounds do
      Result_char := Result_fxn(Opponent[Index], Result[Index]);
    Result_list := Result_list + Result_char + Opponent[Index] + Color[Index] + ' ';
  End;
  Write (Crosstable / Pairing_number + ' ' + Name + ' ' + Rating + ' ' + Result_list);
  Writeln(Crosstable / Printable(Score));
  End;
  Else Begin
    Result_list := '';
    For Index := 1 to Number_of_rounds do
      Result_char := Result_fxn(Tournament_rosterA.Opponent[Index],
                                 Tournament_rosterA.Result[Index]);
    Result_list := Result_list + Result_char + Tournament_rosterA.Opponent[Index] + Tournament_rosterA.Color[Index] + ' ';
  End;
End;
Write (Crosstable, Tournament_roster^.Pairing_number + \\
Tournament_roster^.Name + \\
Tournament_roster^.Rating + \\
Result_list + \\

WriteIn(Crosstable, Printable(Tournament_roster^.Score));
End;

Begin
WriteIn("Writing detail lines to the Crosstable file ...");

If Order_selection = '1'
Then Begin
  Reset(Tournament_roster);
  While not Eof(Tournament_roster) do
    Begin
      Read(Tournament_roster,Player);
      Write_a_line;
    End;
End;
Else Begin
  Open (Tournament_roster, History := Old, \\
    Organization := Indexed, \\
    Access_method := Keyed);
  Current := scored_pointer;
  While Current <> Nil do
    Begin
      Findk(Tournament_roster,0,Current^.Pairing_number);
      Write_a_line;
      Current := Current^.Next_card;
    End;
  End;
End;

Close(Tournament_roster);

Write ('Hit <RETURN> : ');
Readln (Dummy_character);
End;
PROGRAM NAME: Prt.Con

DATE: April 1, 1936

AUTHORS: Rhonda Graft and Eugene Wallingford

PURPOSE: This module prints a file to the local printer.

OPEN/READ/ERROR=BELLS FNAME 'P1'
CLOSE FNAME
ON CONTROL_Y THEN GOTO CTRLY
WRITE SYSS$OUTPUT "[s1"
WRITE SYSS$OUTPUT "[41"
WRITE SYSS$OUTPUT ""
EXIT.
CTRLY:
WRITE SYSS$OUTPUT "[41"
EXIT
BELLS:
WRITE SYSS$OUTPUT ""
EXIT
TESTING AND ANALYSIS OF THE SOFTWARE

If the world like it not, so much the worse for them.

Joseph Cowper
TESTING OF THE SYSTEM

Though considerably more testing was done on the system, I include here a sample run so that the reader can observe the system in action. The test data was taken from the Eastern Indiana Open (played on May 10-11, 1986) and simulates the final three rounds of play. The pairing list and a standard wallchart-crosstable is given for each round.

First-round pairings and results are uneventful. White apparently won the coin-toss to determine the initial color allocation. Powell's upset of Adams presents the first "color clash," since three players in the top score group are now due for white. Since VanMeter and Mills are the higher-ranked players, they should receive their due colors, and so they do! All other pairings are made in the optimal way as well.

Round three shows the functioning of the other precepts of the Swiss System and also reveals a small "bug" that has since been removed. The middle pack of players-- Bousum, Adams, Dowling, and Powell-- calls for an Adams-Powell match-up, but they already played each other in round one. A switch is appropriately made on the right side of the score group to produce legal pairings. Bousum has had black twice in a row and thus, under Swiss principle, must have white; Roush, in the zero score group, must likewise have
black. The TD recognizes this and makes all the switches necessary to produce an equitable color allocation.

The small bug manifests itself in the VanMeter-Mills pairing. VanMeter has had B-W in the first two rounds, while Mills has had W-B. Any self-respecting tournament director would therefore pair the final round as Mills-VanMeter, preserving the alternation of colors (Rule 21) in light of already equal colors (Rule 20). The program, as coded at the time of this test, only saw equal colors and gave the higher-ranked player-- VanMeter-- the preferred color-- white. Upon discovering this error, I quickly made the necessary change in the color allocation algorithm (which employs the color-due matrix given in section III). Now, alternation of colors is favored over the assignment of white to the higher-ranked player.

As a final note to this section I must discuss the color allocation algorithm that I designed for this system. As any avid tournament player (such as Dr. Owens) or TD (such as me) knows, color allocation is the most difficult element of the Swiss System to optimize. Other principles are easily followed. But, to distribute colors equitably (and in a way that satisfies all the players!) is often near to impossible. My algorithm represents one creative way to automate this process, and I think that the concept employed is very good. However, given the time constraints of a single quarter, I was not able to make it as sharp or as all-encompassing as I had hoped. My special request to
any reader of this document inclined to modify this system is this: Study the color allocation principles of the Swiss System, and then polish up my algorithm. I know that I am on the correct path, but I also know that some improvements could probably be made.
### Tournament: Test Tournament #1

**Round:** 1

<table>
<thead>
<tr>
<th>Table</th>
<th>White</th>
<th>Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lester VanMeter</td>
<td>John Dowling</td>
</tr>
<tr>
<td>2</td>
<td>Tom Harris</td>
<td>Jim Mills</td>
</tr>
<tr>
<td>3</td>
<td>Joshua Rousum</td>
<td>John Roush</td>
</tr>
<tr>
<td>4</td>
<td>Ron Powell</td>
<td>Nick Adams</td>
</tr>
</tbody>
</table>

---

**Tournament: Test Tournament #1**

**Sponsor:** Honors College

**Date:** May 14, 1986

**Site:** Muncie, Indiana

<table>
<thead>
<tr>
<th>No.</th>
<th>Player</th>
<th>Rating</th>
<th>1-</th>
<th>2-</th>
<th>3-</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>Lester VanMeter</td>
<td>2331</td>
<td>W 0</td>
<td>0</td>
<td>B</td>
<td>1.0</td>
</tr>
<tr>
<td>020</td>
<td>Jim Mills</td>
<td>2223</td>
<td>W 0</td>
<td>0</td>
<td>W</td>
<td>1.0</td>
</tr>
<tr>
<td>030</td>
<td>Joshua Rousum</td>
<td>2197</td>
<td>W 0</td>
<td>0</td>
<td>B</td>
<td>1.0</td>
</tr>
<tr>
<td>040</td>
<td>Nick Adams</td>
<td>2123</td>
<td>L 0</td>
<td>0</td>
<td>W</td>
<td>0.0</td>
</tr>
<tr>
<td>050</td>
<td>John Dowling</td>
<td>2111</td>
<td>L 0</td>
<td>0</td>
<td>W</td>
<td>0.0</td>
</tr>
<tr>
<td>060</td>
<td>Tom Harris</td>
<td>2043</td>
<td>L 0</td>
<td>0</td>
<td>B</td>
<td>0.0</td>
</tr>
<tr>
<td>070</td>
<td>John Roush</td>
<td>1997</td>
<td>L 0</td>
<td>0</td>
<td>W</td>
<td>0.0</td>
</tr>
<tr>
<td>080</td>
<td>Ron Powell</td>
<td>1911</td>
<td>W 0</td>
<td>0</td>
<td>B</td>
<td>1.0</td>
</tr>
<tr>
<td>No.</td>
<td>Player</td>
<td>Rating</td>
<td>- 1 -</td>
<td>- 2 -</td>
<td>- 3 -</td>
<td>Total</td>
</tr>
<tr>
<td>-----</td>
<td>----------------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>010</td>
<td>Lester VanMeter</td>
<td>2331</td>
<td>W 050</td>
<td>W 030</td>
<td>W 100</td>
<td>2.0</td>
</tr>
<tr>
<td>020</td>
<td>Jostan Mills</td>
<td>2223</td>
<td>W 060</td>
<td>W 080</td>
<td>W 010</td>
<td>2.0</td>
</tr>
<tr>
<td>030</td>
<td>Joshua Bousum</td>
<td>2197</td>
<td>W 070</td>
<td>L 010</td>
<td>W 010</td>
<td>1.0</td>
</tr>
<tr>
<td>040</td>
<td>Nick Adams</td>
<td>2123</td>
<td>L 030</td>
<td>W 060</td>
<td>W 010</td>
<td>1.0</td>
</tr>
<tr>
<td>050</td>
<td>John Dowling</td>
<td>2111</td>
<td>L 010</td>
<td>W 070</td>
<td>L 010</td>
<td>1.0</td>
</tr>
<tr>
<td>060</td>
<td>Tom Harris</td>
<td>2043</td>
<td>L 020</td>
<td>L 060</td>
<td>L 010</td>
<td>0.0</td>
</tr>
<tr>
<td>070</td>
<td>John Roush</td>
<td>1997</td>
<td>L 030</td>
<td>W 030</td>
<td>W 020</td>
<td>0.0</td>
</tr>
<tr>
<td>080</td>
<td>Ron Powell</td>
<td>1911</td>
<td>W 040</td>
<td>L 020</td>
<td>W 010</td>
<td>1.0</td>
</tr>
</tbody>
</table>
**TOURNAMENT : Test Tournament #1**

**ROUND : 3**

<table>
<thead>
<tr>
<th>TABLE</th>
<th>WHITE</th>
<th>BLACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lester VanMeter</td>
<td>Jim Mills</td>
</tr>
<tr>
<td>2</td>
<td>Joshua Bousum</td>
<td>Ron Powell</td>
</tr>
<tr>
<td>3</td>
<td>Nick Adams</td>
<td>John Dowling</td>
</tr>
<tr>
<td>4</td>
<td>Tom Harris</td>
<td>John Roush</td>
</tr>
</tbody>
</table>

**TOURNAMENT : Test Tournament #1**

**SPONSOR : Honors College**

**DATE : May 14, 1986**

**SITE : Muncie, Indiana**

<table>
<thead>
<tr>
<th>No.</th>
<th>Player</th>
<th>Rating</th>
<th>1 -</th>
<th>2 -</th>
<th>3 -</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>Lester VanMeter</td>
<td>2331</td>
<td>B</td>
<td>W</td>
<td>W</td>
<td>3.0</td>
</tr>
<tr>
<td>020</td>
<td>Jim Mills</td>
<td>2277</td>
<td>W</td>
<td>0.5</td>
<td>0.5</td>
<td>2.0</td>
</tr>
<tr>
<td>030</td>
<td>Joshua Bousum</td>
<td>2197</td>
<td>W</td>
<td>0.5</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>040</td>
<td>Nick Adams</td>
<td>2123</td>
<td>L</td>
<td>0.5</td>
<td>0.5</td>
<td>2.0</td>
</tr>
<tr>
<td>050</td>
<td>John Dowling</td>
<td>2111</td>
<td>L</td>
<td>0.5</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>060</td>
<td>Tom Harris</td>
<td>2043</td>
<td>L</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>070</td>
<td>John Roush</td>
<td>1997</td>
<td>L</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>080</td>
<td>Ron Powell</td>
<td>1911</td>
<td>W</td>
<td>0.5</td>
<td>0.5</td>
<td>2.0</td>
</tr>
<tr>
<td>No.</td>
<td>Player</td>
<td>Rating</td>
<td>[-1-]</td>
<td>[-2-]</td>
<td>[-3-]</td>
<td>Total</td>
</tr>
<tr>
<td>-----</td>
<td>----------------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>010</td>
<td>Lester VanMeter</td>
<td>2331</td>
<td>W</td>
<td>050 B</td>
<td>W</td>
<td>030 W</td>
</tr>
<tr>
<td>020</td>
<td>Jim Mills</td>
<td>2223</td>
<td>W</td>
<td>060 W</td>
<td>W</td>
<td>080 B</td>
</tr>
<tr>
<td>040</td>
<td>Nick Adams</td>
<td>2123</td>
<td>L</td>
<td>060 W</td>
<td>W</td>
<td>060 B</td>
</tr>
<tr>
<td>080</td>
<td>Ron Powell</td>
<td>1911</td>
<td>W</td>
<td>040 B</td>
<td>L</td>
<td>020 W</td>
</tr>
<tr>
<td>030</td>
<td>Joshua Bousum</td>
<td>2197</td>
<td>W</td>
<td>070 B</td>
<td>L</td>
<td>010 B</td>
</tr>
<tr>
<td>050</td>
<td>John Dowling</td>
<td>2111</td>
<td>L</td>
<td>010 W</td>
<td>L</td>
<td>070 B</td>
</tr>
<tr>
<td>060</td>
<td>Tom Harris</td>
<td>2043</td>
<td>L</td>
<td>020 B</td>
<td>L</td>
<td>040 W</td>
</tr>
<tr>
<td>070</td>
<td>John Roush</td>
<td>1997</td>
<td>L</td>
<td>030 W</td>
<td>L</td>
<td>050 W</td>
</tr>
</tbody>
</table>
CONCLUSION

Failure isn't fatal,
and
success isn't final.

Attributed to
Don Shula
RETROSPECT

As I look back over this quarter, I am pleased with the ground that I covered. Though the resulting software does not do everything I had ambitiously hoped it would, I can see now that my goals were a bit lofty when I began this project. The Swiss System has proven difficult for me to master as a human TD; coding algorithms to model the system, as well as all the auxiliary duties performed by a tournament director, really stretched my commitment to this project to its limit. My hindsight in this respect only reaffirms my notion that the nature of this idea made it a highly appropriate topic for my thesis.

This quarter's research has provided me with a great understanding of the mechanics of the Swiss System. (As a result, I am now a better TD than I was before beginning the project!) The knowledge I gained allowed me to develop a strong theoretical framework for an automated model of the system. The data flow diagrams and data dictionary that were presented earlier in this document effectively describe the concepts of the method and are the stepping stones to an exhaustive system model. One added benefit of the project has been my exposure to structured analysis and design techniques in an environment in which I could steer the application of these tools in the direction most suitable to the
modeling process. My resultant experience with the DeMarco and Yourdon methods will be of great value to me in future software projects.

The most important benefit of developing this creative project lies in the realm of "research." The scope of my project was not sufficiently narrow for completion in one quarter, but I was able to abstract a fitting subset of the system for my study. Very often I found myself pressed by a self-imposed deadline; and, though I did fall behind, I was able to make up ground in the next phases of the project. I began the quarter with an idealistic view of individualized research, and now I have the more realistic view of a "hardened veteran."

As nice as all these ramblings sound, I have not been magically transformed into an expert researcher over the course of the last eleven weeks. But I am a better researcher than I was before, and the experience of this project will be invaluable as I pursue further academic studies. Though the results of my work are not what I had dreamed they might be, all in all the TD is a good software package. The day is not too distant when a later generation of this "creative project"-- perhaps "TD III Plus"-- stands alone as a practical tool for use in sanctioned tournaments.