```
program BGA2019a;


 // Version 2019a
                                                    01-June-2019


//-------------------------------------------------------------------------
-------------------------------------------------------------

 // Written by: Paul Candler   email: paul.candler@hotmail.com

 //

 //******   IMPORTANT: FOR 2019 CHANGES MADE TO METHOD OF ENTRY OF CONTEST
WIND,   ******

 //******   DHT HANDICAPPED DISTANCE, AND HANDLING OF WITHDRAWN PILOTS
   ******

 //******   PLEASE READ THE FOLLOWING NOTES, ESPECIALLY 2.1 AND 2.3
   ******

 //

 // Based on original scripts written by Tim Scott for Fixed Course Tasks and
AATs

 //

 // Scores a Fixed Course Task, Distance Handicapped Task, or AAT in accordance
with "The 2019 Rules for BGA Rated Competitions"

 //

 // If Task Time is zero then Fixed Course Task, unless string starting "H="
found in contest day properties tag box then

 //    scores as 'Distance Handicapped Task' task, all speed indices=100, no
windicap.

 //

 // If Task Time is non-zero, then scores as AAT.

 //

 //

 // If you have any problems with this script, or queries regarding using it
with SeeYou, please email me at the above address
```

```
 //

 // Notes:

 // 1.  Before importing into SeeYou Competition:

 //

 //     1.1 Check on BGA website for latest version of this script

 //

 //     1.2 Ensure 'CompClass' is set (in the table below) to the correct
Competition Class.

 //         Remove the double slash from in front of the line containing the
required Competition class and

 //         add a double slash in front of the line with no double slash
(CompClass = '?') before importing

 //         into SeeYou Competition.

 //

const     // <<do not alter this line>>


//#########################################################################
########################################

            CompClass = '?',                            ClassVariables = '',

          //CompClass = 'Open Nationals',               ClassVariables =
'1,50,100000,200000,40,150000,7200,1.18,250,200,0',

          //CompClass = '18m Nationals',                ClassVariables =
'1,50,090000,180000,36,150000,7200,1.10,250,200,0',

          //CompClass = '15m Nationals',                ClassVariables =
'1,50,090000,180000,36,150000,7200,1.04,250,200,0',

          //CompClass = '15m&Std Nationals',            ClassVariables =
'1,50,090000,180000,36,150000,7200,1.04,250,200,1',

          //CompClass = 'Std Nationals',                ClassVariables =
'1,50,080000,160000,32,150000,7200,1.00,250,200,0',

          //CompClass = '20m Two Seater Championship',  ClassVariables =
'1,50,090000,180000,36,150000,7200,1.04,250,200,1',
```

```
          //CompClass = 'Club Nationals',               ClassVariables =
'1,50,080000,160000,32,120000,7200,1.00,250,200,1',

          //CompClass = 'Handicapped Nationals',        ClassVariables =
'1,50,080000,160000,32,150000,7200,1.00,250,200,1',

          //CompClass = 'Junior Nationals',             ClassVariables =
'0,40,060000,120000,30,080000,7200,1.00,000,000,1',

          //CompClass = 'Regionals',                    ClassVariables =
'0,40,060000,120000,30,080000,7200,1.00,000,000,1',


//##############################################################################
##########################################


 //     1.3 Ensure "Need task leg data in scoring script" is checked in Contest
Properties in SeeYou Competition

 //

 // 2.  Each day:

 //

 //     2.1 For tasks other than Distance Handicapped Tasks, you will need to
put wind velocity in SeeYou Competition,

 //         in the Contest day properties dialog, in the Tag box on the General
Tab, in the

 //         format W=ddd/ss where ddd is the wind direction and ss in the wind
speed in knots

 //

 //         For Distance Handicapped Tasks, you must enter 'H=ddd.d' (where
ddd.d is the 'Handicapped Task Distance', this

 //         disables handicapping and windicapping in distance calculations.

 //

 //         The 'Handicapped Task Distance' for a DHT is the distance in
kilometres that the Distance Handicapped task setting software

 //         has calculated for a glider with speed index 100. If this number is
given as zero, then the Handicapped Task Distance is

 //         calculated as 'Declared task distance' * (100/max_HCap) where
```

max_HCap is the maximum speed index of any glider not withdrawn

 //   from the competition.

 //

 //   2.2 Check that 'Racing Task' or 'Assigned Areas Task' is selected as appropriate in the Contest day properties dialog,

 //   in the Type box on the Task Options Tab

 //   'Racing Task' for Fixed Course and DHT, 'Assigned Areas Task' for AAT

 //

 //

 //   2.3 If any competitors withdraw from the competition, for correct points calculations, you MUST enter the number of withdrawn

 //   competitors before or after the contest wind or Handicapped Distance, separated by a semi-colon, in the Day Tag box.

 //   e.g. W=270/12;Nw=1 or Nw=1;H=310.6

 //

 //

 //*****************************

 // Changes since version 2014a

 //

 // 2014b For Distance Handicapped Tasks, find the highest handicap of any competing glider (max_Hcap) and use to calculate DHT_hc_factor = 100/max_Hcap

 //   All distances will then be adjusted by multiplying by DHT_hc_factor, which will result in more representative handicapped distances and

 //   speeds being reported

 //

 // 2014c Allow 'Handicapped Task Distance' (distance calculated in task setting for glider with speed index 100) to be entered in Day Tag box after the 'H'

 //   If distance is entered, max_HCap calculation is ignored and DHT_hc_factor = Handicapped Task Distance/Total declared task distance

```
 //       This value will include wind adjustment and be more accurate and
reliable than 100/max_HCap

 //       Y distance is also calculated using the Handicapped Task Distance
instead of the declared task distance.

 //

 // 2015a Corrected ClassVariables for 20m Two Seater Championship

 //

 // 2016a Removed code specific to Overseas Championships and weathercapping.

 //       Removed leg timeout distance calculations for AATs and use greatest
distance instead of timeout distance for devaluations.

 //       Restructured some of the code to improve modularity and to make it
easier to follow, understand and maintain.

 //       Added a test to see if "Need task leg data in scoring script" has
been checked in SeeYou Competition.

 //

 // 2017a Added interim support for trial of Designated Starts (Approval from
Comps Committee required  for use in 2017 comps)

 //       Minor mods to ParseDayTag procedure to support Designated Starts

 //       Added new class type to ClassVariables, '15m&Std Nationals' for
combined 15m and Standard competitions

 //

 // 2017b Set Pilot.sdis to hold marking distance to left of decimal place and
actual distance to right of decimal place, to make actual distance

 //       available to Soaring Spot API  (used by OnGlide)

 //

 // 2018  Minor changes to reflect name of 'Junior Nationals' and AAT
Designated Time. No code changes

 // 2018b Change method of marking a pilot withdrawn due to loss of Pilot.Tag
field in >SeeYou 9.1 (and unreliable in SeeYou 8)

 //

 // 2019  Removed code for 2017 trial of Designated Starts
```

```
 //       Changed way wind and handicapped distance are entered in DayTag (see
note 2.1 above)

 //       Changed handling of withdrawn pilots (see note 2.3 above)

 //       Changed method of manual scoring (for fixed course tasks only) - see
separate notes

 //

 //*****************************

 //


//-------------------------------------------------------------------------------
-------------------------------------------------------------------

 //conversion factors

  ms2kmh  = 3.6;

  deg2rad = Pi / 180;



 var

                          // Class variable parameters set up in procedure
SetUpCompParameters, held in string ClassVariables

  Qualifying_distance_factor,      // Percentage, used to calculate Y.
                   {

  Ymin,                            // Minimum Qualifying distance in metres
                   { See Rules Table 7.21

  Ymax,                            // Maximum Qualifying distance in metres
                   {



  MinTaskDistance, MinTaskTime,    // Minimum task distance/time set at
briefing in metres          {

  Qualifying_Time_Factor,          // Used in AATs to calculate Y
                   {

  Da, Ta :                         // Devaluation distance & Time adjustments.
                   { See Rules Table 7.22
```

```
    integer;                        //
                    {

  Contest_Wind_division_factor,     // Wind strength is divided by contest
dependant factor.         {

  Wdiv :                            // Division constant used in wind increment
calculation.         {

    double;




 // Numbers

  N,                        // Number of participating gliders

  Nl,                       // Number of participating gliders launched

  Ny,                       // Number of participating gliders reaching or
exceeding Y

  Nv,                       // Number of participating gliders equalling or
exceeding 2/3rds Vh

  Nf,                       // Number of participating gliders who complete the
task and finish

  Nw,                       // Number of pilots withdrawn

  TaskLegs,                 // Number of legs in task (from Task data)

  LegDataCount,             // Number of traces with leg data

  ManualCount,              // Number of manually scored pilots (takeoff =
00:00:01)

  i :                       // Array index used in main procedure

  integer ;



 // Distance

  DTask,                    // Task distance as reported at briefing (not AATs)

  Hcap_task_dist,           // Handicapped Task Distance (actual distance in km
that glider with speed index 100 must fly, given in DayTag string)
```

```
  Y,                    // Qualifying distance

  Dmax,                 // Greatest marking distance of any glider

  Dw,                   // Winners Marking Distance



 // Speed

  Vh,                   // Greatest handicapped speed



 // Handicapping

  W,                    // Contest Wind in knots

  Windkts,              // Actual wind strength in knots

  Winddeg,              // Direction of the Contest Wind in degrees relative
to True North

  Radwind,              // Direction of the Contest Wind in radians relative
to True North

  max_HCap,             // Highest handicap of competing gliders

  DHT_hc_factor,        // Multiplication factor to convert actual distance
reported by SeeYou to handicapped distance



 // Points

  F,                    // Day total points

  Fd,                   // Day distance points

  Fv,                   // Day speed points



 // Times

  Winners_Tg,           // Fixed Course Task winner's Tg (time to complete
the course) in seconds

  Td,                   // Minimum Time for Assigned Area Tasks in seconds
```

```
 //Calculated Factors

  Ff,                     // Day points reduction Factor

  WindiCapTaskDist :      // Task distance adjusted for windicapping

  double;



 // Strings

  Devalued,               // string to hold reason for day devaluation

  debuginfo,              // string for debug info which may be output with
ShowMessage(debuginfo)

  Hi_string :             // string to hold Hi values for optional output in
Info2 or User_str1

  string;



 // Booleans

  Nationals_Y,            //  if TRUE, y is %age of wind adjusted task
distance, if FALSE y is %age of unhandicapped task distance

  handicapped_comp,       //  if TRUE, glider speed indices to be used, if
FALSE use speed index of 100 for all gliders

  Fixed_Course,           //  determined at run-time, set TRUE if value of
Task.Time=0

  Distance_Handicapped,   //  determined at run-time, set TRUE if first
character of Contest day properties='H'

  Assigned_Area,          //  determined at run-time, set TRUE if value of
Task.Time>0

  Manual_Scoring,         //  set TRUE if current pilot being manually scored

  Debug_Mode :            //  will be set TRUE if '?' precedes wind string in
Day Tag box

  Boolean ;



//-------------------------------------------------------------------------
-------------------------------------------------------------
```

```
// Functions and Procedures. (Main Program is at end of script)

//---------------------------------------------------------------------------
-------------------------------------------------------------

// return the minimum value of a,b,c

function MinValue (a,b,c : double) : double;

var

  m : double;


begin

    m := a;

    If b < m then m := b;

    If c < m then m := c;

    Result := m;

end;


//---------------------------------------------------------------------------
-------------------------------------------------------------

// return the maximum value of a,b

function MaxValue(a,b : double) : double;


begin

    If a > b then result := a else result := b;

end;


//---------------------------------------------------------------------------
-------------------------------------------------------------

// use values in string ClassVariables to set the correct values for various
parameters
```

```
// for each class, to ensure correct rules used in CalculateY and
CalculateDaypoints etc.

procedure SetUpCompParameters;



begin



    handicapped_comp := false;

    Nationals_Y      := false;



// ClassVariables =
Nationals_Y,Qualifying_distance_factor,Ymin,Ymax,Qualifying_Time_Factor,MinTask
Distance,MinTaskTime,Contest_Wind_division_factor,Da,Ta,handicapped_comp



    Qualifying_distance_factor   := StrToInt  (copy(ClassVariables, 3,2),0);

    Ymin                         := StrToInt  (copy(ClassVariables, 6,6),0);

    Ymax                         := StrToInt  (copy(ClassVariables,13,6),0);

    Qualifying_Time_Factor       := StrToInt  (copy(ClassVariables,20,2),0);

    MinTaskDistance              := StrToInt  (copy(ClassVariables,23,6),0);

    MinTaskTime                  := StrToInt  (copy(ClassVariables,30,4),0);

    Contest_Wind_division_factor := StrToFloat(copy(ClassVariables,35,4)  );

    Da                           := StrToInt  (copy(ClassVariables,40,3),0);

    Ta                           := StrToInt  (copy(ClassVariables,44,3),0);



    Wdiv                         := 46.0;     // constant for all comps except
Overseas Championships which are no longer supported



    If copy(ClassVariables, 1,1) = '1' then Nationals_Y      := true;

    If copy(ClassVariables,48,1) = '1' then handicapped_comp := true;
```

```
    ManualCount := 0;




end;




//-------------------------------------------------------------------------
-------------------------------------------------------------

// function to check for gliders with invalid speed index in Pilot Details and
find max speed index

function CheckHcaps : boolean;



var

  x : integer;



begin

    Result := false;

    max_HCap := 0;



    for x := low(Pilots) to high(Pilots) do

      begin

        If (Pilots[x].Hcap > max_HCap)  then max_HCap :=  Pilots[x].Hcap;

        If Pilots[x].Hcap = 0 then

          begin

            Result := true;

            pilots[x].warning :=  'Glider ' + Pilots[x].CompID + ' has speed
index = 0 !!?';
```

```
            debuginfo := Pilots[x].CompID + '  speed index = 0';

            showmessage(debuginfo);

          end;

    end;

end;



//-------------------------------------------------------------------------------
-------------------------------------------------------------

// calculate the number of participating gliders, N by stripping out HC and
withdrawn pilots.

function GetNumberGliders : integer;



var

  x : integer;



begin

    for x := low(Pilots) to high(Pilots) do

    begin

      If PilotCompeting(Pilots[x]) then                      // ignore Hors
Concours pilots.

       Result := Result + 1;

    end;



    Result := Result - Nw;                                   // Reduce by
number given as withdrawn in DayTag



end;
```

```
//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate the number of competing pilots launched, Nl

function GetNumberPilotsLaunched : integer;



var

  x : integer;

begin

    for x := low(Pilots) to high(Pilots) do

    begin

      If PilotCompeting(Pilots[x]) then                      // ignore Hors
Concours pilots

        If Pilots[x].takeoff > 0 then Result := Result + 1;  // takeoff > 0
implies launched, withdrawn pilots will be DNF

    end;

end;



//-------------------------------------------------------------------------
-------------------------------------------------------------

// Get task info and set global variables

procedure GetTaskInfo;



begin



    Fixed_Course  := false;

    Distance_Handicapped := false;

    Assigned_Area := false;
```

```
    If Task.TaskTime > 0 then Assigned_Area := true;

    If Task.TaskTime = 0 then Fixed_Course  := true;       // assume Fixed
Course to start with, may be modified by content of DayTag string



    TaskLegs := GetArrayLength(Task.Point) - 1;



end;

//------------------------------------------------------------------------
------------------------------------------------------------

// parse the Contest Day Tag string for wind info etc. and put results in
global variables

procedure ParseDayTag(TagString : string);



var

    WindString, DHTString, WithdrawnString,

    taskdis,strength,direction,DHTdist,taskstr,windinfo : string;

    xoff : integer;



begin

    debug_mode := false;                                    // start with
debug flag off

    Windkts    := 0;                                        // and zero wind
parameters

    Winddeg    := 0;

    Radwind    := 0;

    windinfo := ' ';



    If Fixed_course then taskstr := 'Fixed Course Task' else taskstr := 'AAT';
// based on Task.Time being zero or non-zero
```

```
    If copy(TagString,1,1) = '?' then                          // If first
character of TagString is question mark

      begin

        debug_mode := true;                                    // set flag for
debug mode

        TagString := copy(TagString,2,99);                     // and strip
question mark off

      end;



   TagString       := Uppercase(TagString);                     // convert any
lowercase chars to uppercase



   WindString      := ExtractString(Tagstring,'W=');            // look for 'W='
and extract wind info

   DHTString       := ExtractString(TagString,'H=');            // look for 'H='
and extract Handicapped Task Distance

   WithdrawnString := ExtractString(TagString,'NW=');           // look for 'NW='
and extract number withdrawn




   If length(WithdrawnString) > 0 then

        Nw := StrToInt(WithdrawnString,0) else Nw := 0;        //  set number
withdrawn



   If length(DHTString) > 0 then

      begin

        Distance_Handicapped := true;                          // set flag for
Distance Handicapped Task, but leave Fixed_Course true

        handicapped_comp := false;                             // and switch off
handicapping
```

```
                                                        // wind speed and
direction will default to 0

        Hcap_task_dist := 0;

        Hcap_task_dist := StrToFloat(DHTString);            // get DHT task
distance for speed index 100 in km, will remain 0.0 if no distance given

        taskstr := 'Distance Handicapped Task';

      end;



    xoff  := pos('/',WindString);                            // check if wind
string is present and formatted correctly

    If xoff > 0 then

      begin

        direction := copy(WindString,1,xoff-1);              // extract the
wind and direction from the wind string

        strength  := copy(WindString,xoff + 1,10);

        Windkts   := StrToInt(strength,0);                  // store results
in global variables

        Winddeg   := strtoint(direction,0);

        Radwind   := Winddeg * Deg2Rad;                     // convert degrees
to radians



        W := Minvalue( (Windkts/Contest_Wind_division_factor), 30,30 );   //
get value for W, must not exceed 30 (Rule 5.22.5)




        windinfo :=  ',   Contest Wind ' + floattostr(Winddeg) + ' degs/' +
floattostr(Windkts) + ' kts';

      end;



      If (xoff = 0) and not Distance_Handicapped then windinfo := ', Wind not
set or format incorrect';
```

```
      Info1 :=  CompClass + ' / ' + taskstr + windinfo;


end;




//-------------------------------------------------------------------------------
-------------------------------------------------------------------

// find specified substring in xstring and return characters to right of '=' up
to semi-colon or end of string

Function ExtractString (xstring,substring : string) : string;



var

  x : integer;



begin



    Result := '';                                          // return null string if
string not found



    x := pos(substring,xstring);                           // locate string

    if x > 0 then

      begin

        Result :=  copy(xstring,x,99);                     // copy from start of
string and everything to the right



        x := pos('=',Result);                              // locate '=' sign

        if x > 0 then Result :=  copy(Result,x+1,99);  // copy from right of
'=' sign
```

```
        x := pos(';',Result);                           // check for trailing
semi-colon

        if x > 0 then Result := copy(Result,1,x-1);    // strip off semi-colon
and everything to its right

    end;



end;




//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate DTask

procedure SetDTask;



begin



    If Fixed_course then DTask := Task.TotalDis;                          // Get
total task distance from Task record



    If Distance_Handicapped then

      begin

        If Hcap_task_dist = 0.0 then DHT_hc_factor := 100/Max_HCap else // no
distance entered, use max handicap to calculate DHT_hc_factor

          DHT_hc_factor := (Hcap_task_dist * 1000)/Task.TotalDis;       //
convert distance to metres and divide by full task distance



        DTask := Task.TotalDis * DHT_hc_factor;                         //
Adjust task distance to representative handicapped task distance

      end;
```

```
end;



//-------------------------------------------------------------------------
-------------------------------------------------------------------

// calculate the Hi value and store in Task.Point.td1 for each leg of a Fixed
Course Task

procedure CalculateTaskHiValues;



var

    x : integer;

  sep : string;



begin

    sep := ' ';

    Hi_string     := 'Leg Hi values =';
 // initialize string for Hi values



    for x := 1 to TaskLegs do

      begin

        Task.Point[x].td1 := CalcLegHi(Task.Point[x].crs);
 // calculate the wind handicap increment for each leg

        Hi_string := Hi_string + sep + formatfloat('0.000',Task.Point[x].td1);
 // save in Hi_string for possible output in debug mode

        sep := ',  ';

      end;



end;
```

```
//-------------------------------------------------------------------------
-------------------------------------------------------------

// Calculate Windicapped task distance for given Hcap

Function CalculateWindicapTaskDist (Hcap : Double) : Double;



var

  x : integer;

  LegHl : double;



begin

    Result := 0;



    For x := 1 to TaskLegs do

    begin
// Task.Point[x].td1 already holds Hi for that leg

      LegHl := CalcLegHl(Hcap,Task.Point[x].td1);
// get LegHl

      Task.Point[x].td2 := (Task.Point[x].d * 100) / LegHl;
// calculate the windicap leg distance and store in Task.Point.td2

      Result := Result + Task.Point[x].td2;
// add windicap leg distance to Result

    end;

end;



//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate and return the Leg Handicap Increment (Hi)

Function CalcLegHi (legcrs : Double) : Double;
```

```
var

  Theta,Wind : Double;



begin

    Wind  := W/Wdiv;                    // Apply Wdiv constant

    Theta := Legcrs - Radwind;          // non-Reflexive angle between track and
direction of wind



    Result :=  100 * (Power(1 - (power(Wind,2)*power(sin(Theta),2)),0.5) - (1 +
(Wind)*cos(Theta)));



end;



//-------------------------------------------------------------------------------
-----------------------------------------------------------------

// calculate and return the Leg Wind Adjusted Speed Index (Hl)

Function CalcLegHl(hcap,Hi : Double) : Double;



var

  H : Double;



begin



    If not handicapped_comp then

      H := 100                                      // if not handicapped_comp
all gliders will have speed index of 100

    else
```

```
      H := hcap;



    Result := Maxvalue(25,H + Hi);                  // Return Hl, sum of speed
index and leg handicap increment (windicap). Must be >=25



end;



//-------------------------------------------------------------------------
-------------------------------------------------------------------

// Check check to see if trace data present or manually scored

// *** Manual scoring will only work for fixed course tasks ***

procedure CheckPilotData(var Pilot : TPilot);




begin

    pilot.user_str1 := '';

    pilot.user_str2 := '';

    Manual_scoring := False;




    If getarraylength(Pilot.Leg) > 0 then                          // no leg
data means no trace or possibly 'need task legs data' not set in contest
properties

                               LegDataCount := LegDataCount + 1;    // count
pilots with leg data in trace



    if round(Pilot.Takeoff) = 1 then                               // for
```

```
manual scoring Takeoff must be set to 00:00:01

      begin

        Manual_scoring := True;

        ManualCount := ManualCount + 1;                                  // Count
number of manually scored

        Pilot.User_Str2 := ' <<manually scored>>';                       // flag in
User_Str2

      end;




    // if there is takeoff > 00:00:01 and distance, but no leg data then either
"Need task legs data ..." has not been set, or manual scoring attempted

    // if other traces found with leg data then must be manual scoring



    If (LegDataCount > 0) and (getarraylength(Pilot.Leg) = 0) and (pilot.dis >
0) and (round(pilot.takeoff) <> 1) then

      begin

        pilot.warning := 'Glider ' + Pilot.CompID + ' has takefoff and distance
set, if manually scoring set takeoff to 00:00:01 for correct windicapping';

        showmessage(pilot.warning);

      end;




end;



//-------------------------------------------------------------------------------
-----------------------------------------------------------------

// use values derived from pilot's flight trace or Pilot.Tag string to
calculate pilots distance for each leg and

// store total handicap distance in Pilot.td1, windicapped speed will go in
```

```
Pilot.td2.

procedure CalcPilotVariables(var Pilot : TPilot);




var

    x, xlegs: integer;

    sum_of_legs, leg_Hi, Leg_Hl, leg_distance, windicapped_leg : double;


begin


    Pilot.td1 := 0;

    Pilot.td3 := 0;

    sum_of_legs := 0;




    If debug_mode and Assigned_Area then Pilot.user_str1 := 'Hi values = ';


    xlegs := getarraylength(Pilot.Leg);                                    //
get number of legs from trace


    for x := 1 to xlegs do                                                 //
for all task types calculate total marking distance on each leg

      begin

        If Fixed_Course then leg_hi := Task.Point[x].td1;                  // Hi
already calculated for Fixed Course and held in Task record


        If Assigned_Area then leg_hi := CalcLegHi(Pilot.Leg[x-1].crs);     //
Calculate Leg Hi for this leg using course from trace
```

```
        If debug_mode and Assigned_Area then

                Pilot.user_str1 := Pilot.user_str1 +
formatfloat('0.000',leg_hi) + '  ';                  // for AAT debugging



        If not Distance_Handicapped then Leg_Hl := CalcLegHl(Pilot.Hcap,leg_hi)
else Leg_Hl := 100;       // calculate Leg Hl



        If not Manual_Scoring and (xlegs > 0) then

                                leg_distance := Pilot.Leg[x-1].d;            //
Get leg distance from trace




        windicapped_leg := (leg_distance * 100) / Leg_Hl;                    //
full leg length windicapped



        sum_of_legs := sum_of_legs + leg_distance;                           //
add actual leg length to sum_of_legs

        Pilot.td1 := Pilot.td1 + windicapped_leg;                            //
add leg marking distance to total marking distance in Pilot.td1



      end;



    If Manual_Scoring and (Pilot.sdis > 0) then Pilot.td1 := Pilot.sdis;    //
use distance given

    If Manual_Scoring and (Pilot.sdis = 0) and (Pilot.finish > 0) then

                        Pilot.td1 := CalculateWindicapTaskDist(Pilot.Hcap); //
if no distance given the use windicapped task distance
```

```
    If Distance_Handicapped then Pilot.td1 := Pilot.td1 * DHT_hc_factor;    //
adjust actual distance reported by SeeYou to get a representative handicapped
distance



    CalculatePilotHandicapSpeed(Pilot);                                     //
Use result in Pilot.td1 to calculate pilot's windicap speed and store in
Pilot.td2



end;



//-------------------------------------------------------------------------
-------------------------------------------------------------------

// calculate Pilot handicapped speed and store in Pilot.td2 (note result in
m/s)

procedure CalculatePilotHandicapSpeed(var Pilot : TPilot);



var

    speed, TimeOnTask : double;



begin



    If Pilot.Finish <> -1 then

      begin                                                 // check if pilot
finished

        TimeOnTask := Pilot.finish - Pilot.start;

                                                            // for Fixed Course
Tasks, Task.TaskTime will be 0,

                                                            // for AATs if
Task.TaskTime > TimeOnTask then use Task.TaskTime



        If Task.TaskTime > TimeOnTask then TimeOnTask := Task.TaskTime;
```

```
        speed := 0;

        if timeontask > 0 then speed := Pilot.td1 / TimeOnTask;  // pilot.td1
holds total windicapped distance flown, timeontask should be non-zero

                                                                 // but could
be 0 if manual scoring and finish time entered incorrectly,

                                                                 // so don't
let div by 0 fault crash script

      end

    else speed := -1;                                    //if not finished then
set speed to -1 as a flag



    Pilot.td2 := speed;                                  //store result in the
Pilot.td2 user variable



end;



//-------------------------------------------------------------------------------
-------------------------------------------------------------------

// function to check if pilot is competing

function PilotCompeting (Pilot : TPilot) : boolean;



begin

    result := true;



    If Pilot.isHC      then result := false;             //check if Hors
Concours



end;
```

```
//-------------------------------------------------------------------------
-------------------------------------------------------------

// find Dmax, the greatest marking distance flown by a participating glider

function GreatestDistanceAnyGlider : double;

var

  x  : integer;

  marking_distance : double;


begin

    Result := 0;


    If Nl > 0 then                  // if gliders have launched find greatest
Dm (in Pilot.td1)

      begin

        for x := low(Pilots) to high(Pilots) do

          If ((Pilots[x].td1 > result) and PilotCompeting(Pilots[x])) then

          Result := Pilots[x].td1;

      end;


    // Dmax will be set by Result, if no finishers then set Dw too.


    If Nf < 1 then Dw := Result;




end;


//-------------------------------------------------------------------------
-------------------------------------------------------------
```

```
// calculate the number of participating gliders finishing, Nf

function GetNumberFinishers : integer;

var

  x : integer;



begin

    Result := 0;

    for x := low(Pilots) to high(Pilots) do

    begin

      If ((Pilots[x].Finish > -1) and PilotCompeting(Pilots[x])) then

          Result := Result + 1;

    end;


end;


//------------------------------------------------------------------------
------------------------------------------------------------------

// calculate Y distance

function CalculateY : double;

begin

    If Fixed_Course then begin

      If Nationals_Y then

        Result := WindiCapTaskDist * (Qualifying_distance_factor / 100)    //
Nationals use %age of Windicapped Task Distance   (table 36.5)

      else

        Result := DTask * (Qualifying_distance_factor / 100)               //
```

```
Regionals use %age of Unhandicapped Task Distance (table 36.5)

      end

    else

     //  AAT: Y is Qualifying_Time_Factor multiplied by number of hours set for
task (divide Task.TaskTime by 3600 to get hours)



      Result := Qualifying_Time_Factor * (Task.TaskTime / 3600) * 1000;    //
Multiply result by 1000 to get metres



    If result < YMin then result := YMin;                                  //
Apply Min/Max (table 36.5)

    If result > YMax then result := YMax;

end;




//-------------------------------------------------------------------------
-----------------------------------------------------------------

// calculate NY, the number of pilots passing Y

function CalculateNY : integer;

var

  x : integer;

begin

    result := 0;

    for x := low(pilots) to high(Pilots) do

    begin

      If ((Pilots[x].td1 > Y) and PilotCompeting(Pilots[x])) then  //
Pilots[x].td1 hold pilot's marking distance (Dm)

        Result := Result + 1;

    end;
```

```
end;



//------------------------------------------------------------------------------
------------------------------------------------------------------

// calculate Vh, the fastest handicapped speed achieved by a participating
glider

function GreatestSpeed : double;



var

  x  : integer;

  finisher : boolean;



begin

    Vh := 0;

    Winners_Tg := 0;

    If Nf > 0 then

    begin

      for x := low(Pilots) to high(Pilots) do               // Pilot.td2 holds
Sh, finisher's speed or -1 for non-finishers

        If ((Pilots[x].td2 > Vh) and PilotCompeting(Pilots[x]) and
(Pilots[x].Finish <> -1)) then

          begin

            Vh         := Pilots[x].td2;                    // for Fixed Course
Task we have also found winner (not always true for AAT)

            Winners_Tg := Pilots[x].finish-Pilots[x].start; // Save Fixed
Course Task winner's time (not used in AAT)

            Dw         := Pilots[x].td1;                    // Save Fixed
Course Task winner's marking distance (not used in AAT)

          end;
```

```
     end;

     Result := Vh;                                          // Return Vh as
result

end;




//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate Nv, the number of participating gliders exceeding 2/3 winners
speed

function CalculateNv : integer;

var

  x : integer;

begin

    result := 0;

    If Nf > 0 then

      begin

        for x := low(Pilots) to high(Pilots) do

          If ((Pilots[x].td2 > (Vh*0.6667)) and PilotCompeting(Pilots[x])) then

            Result := Result + 1;

    end;

end;



//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate Day Factor Ff

function CalculateDayFactorFf : double;
```

```
begin

    Result := minvalue(1.0,1.0,(1.25 * (Ny / N)));  // result must not exceed
1.0

end;



//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate total day points

function CalculateDayPoints : double;

var

  FMaxDist, FMaxSpeed, FPilotsPastY,DayPoints : double;

  D, T : double;

  timestr, minstr : string;



begin

    Result := 0;



    If Assigned_Area then

      D := Dmax / 1000 else                                          // Set D to
greatest distance flown (in km) for AATs

      D := Dw / 1000;                                                // Set D to
winner's distance (in km) for all other task types



    If Assigned_Area then

      T := Task.TaskTime / 3600                                      // set T to
task time (in hours) for AATs

    else T := Winners_Tg / 3600;                                     // else set
T to winner's time (in hours) for all other task types
```

```
    FPilotsPastY := Ff * 1000;                                  // Day
Factor determined by number past Y



    FMaxDist := Ff * ((5 * D) - Da);                            // Distance
devaluation factor



    If Nf < 1 then                                              // if no
finishers

      FMaxSpeed := 9999                                         // set
FMaxSpeed to obviously too high value

    else

      FMaxSpeed := Ff * ((400 * T) - Ta);                       // Speed
devaluation factor




    DayPoints := minvalue(FMaxDist,FMaxSpeed,FPilotsPastY);     // the
minimum of the above calculated values



    If Fixed_Course and (DTask < MinTaskDistance) then

      DayPoints := 0;                                           // Check
task setter has not underset Fixed Course Task



    If Assigned_Area and (Task.TaskTime < MinTaskTime) then

      DayPoints := 0;                                           // Check
task setter has not underset AAT



    timestr                       := 'Winners Time';

    If Assigned_Area then timestr := 'Task Time';

    minstr                        := 'Less than Minimum Task Distance';

    If Assigned_Area then minstr  := 'Less than Minimum Task Time';
```

```
   //set string Devalued to reflect what caused the day to be devalued

    If DayPoints = FMaxDist     then Devalued := 'Distance';

    If DayPoints = FMaxSpeed    then Devalued :=  timestr;

    If DayPoints = FPilotsPastY then Devalued := 'Number Past Y';

    If DayPoints = 0            then Devalued :=  minstr;

    If Ny = 0                   then Devalued := 'No  gliders past Y';



   // setup info3 to dump key parameters to info3 string to help debugging
(gets overwritten in main procedure if debug_mode is false)

    info3 := 'N=' + inttostr(N);

    if Nw > 0 then info3 := info3 + '(withdrawn=' + inttostr(Nw) + ')';

    info3 := info3 + ', Nl=' + inttostr (Nl) + ',  Ny=' + inttostr(ny)+ ',
Nf=' + inttostr(nf) + ',  Nv=' + inttostr(Nv);

    info3 := info3 + ', D=' + formatfloat('0.000',D) + ', T=' +
formatfloat('0.000',T)+ ',  F=Min(FF*1000=';

    info3 := info3 + formatfloat('0.000',Ff*1000)+ ', Fdist=' +
formatfloat('0.000',FMaxDist) ;

    If FMaxSpeed < 9999.0 then info3 := info3 + ', FSpeed=' +
formatfloat('0.000',FMaxspeed);

    info3 := info3 + ')';



    Result := DayPoints;  // Return result (F)

end;



//-------------------------------------------------------------------------
-------------------------------------------------------------

// calculate day speed points Fv

function CalculateDaySpeedPointsFv : double;
```

```
begin

    Result := 0.6667 * F * (Nv / Nl);

    info3 := info3 + ',  Fv=' + formatfloat('0.000',result);

end;


//---------------------------------------------------------------------------
-------------------------------------------------------------

// calculate day distance points

function CalculateDayDistancePoints : double;

begin

    Result := F - Fv;

    info3 := info3 + ',  Fd=' + formatfloat('0.000',result);

end;




//---------------------------------------------------------------------------
-------------------------------------------------------------

// calculate Day points for each pilot

procedure CalcPilotPoints(var Pilot : TPilot);

var

  Ps,Pd, Dm, Sh, ratio_dmax : double;

  exceeded_2thirds_Dmax,   finisher : Boolean;      // used below to try to aid
readability!


begin

    If Pilot.Start = -1 then                                  // check if pilot
started
```

```
    begin

      Pilot.Points := 0 - Pilot.penalty;                    // if not give 0
points (less any penalty) and exit procedure

      exit;

    end;



    Dm := Pilot.td1;                                        // Pilot.td1 holds
pilot's marking distance

    Sh := Pilot.td2;                                        // Pilot.td2 holds
the pilots marking speed



    exceeded_2thirds_Dmax := false;

    finisher := false;



    If pilot.finish <> -1 then finisher := true;



    If Dm > (0.6667*Dmax) then exceeded_2thirds_Dmax := true; // only used in
AAT



// speed points

    Ps := 0;                                                // speed points are
given in proportion to amount that Sh exceeds two-thirds Vh

    If Vh > 0 then Ps := 3 * Fv *((Sh / Vh) - 0.6667);      // calculate for
all finishers

    If Ps < 0 then Ps := 0;                                 // if result
negative then give zero speed points




// distance points

    ratio_dmax := 0;                                        // set ratio to
```

```
zero

    If Dmax > 0 then                                        // avoid div by 0

    begin

      If finisher and (Fixed_Course or exceeded_2thirds_Dmax) then     //
Finisher on Fixed Course Task or AAT finisher with Dm > 2/3 Dmax,

                            ratio_dmax := 1;                            //   so
set ratio to 1 to get full distance points



      If finisher and Assigned_Area and NOT exceeded_2thirds_Dmax then // AAT
finisher with Dm <= 2/3 * Dmax

                            ratio_dmax := Dm / (Dmax * 0.6667);         //   so
set ratio to Dm : 2/3Dmax



      If NOT finisher then ratio_dmax :=  (Dm/Dmax);                    //
non-finisher, set ratio to Dm : Dmax



    end;



    if ratio_dmax > 1 then Pd := Fd

      else Pd := Fd * ratio_dmax;                           // Glider Distance
Points are Day Distance points multiplied by appropriate ratio



    Pilot.points := round(Pd + Ps - Pilot.penalty);        // Total points are
the summed THEN rounded

end;



//-------------------------------------------------------------------------
-----------------------------------------------------------------

// display the result for each pilot for publication.

procedure DisplayResults;
```

```
var

  x1 : integer;



begin

    For x1 := 0 to GetArrayLength(Pilots) - 1 do

    begin



      If (length(Pilots[x1].User_Str2) > 0) or (length(Pilots[x1].User_Str3) >
0) then

          Pilots[x1].warning := Pilots[x1].CompId + Pilots[x1].User_Str2 +
Pilots[x1].User_Str3;




      Pilots[x1].sstart  := Pilots[x1].start;         // scored start is start
time from SeeYou

      Pilots[x1].sfinish := Pilots[x1].finish;        // scored finish is
finish time from SeeYou



      Pilots[x1].sspeed := 0;

      If (Pilots[x1].finish > 0) and (Pilots[x1].td2 > 0) then
Pilots[x1].sspeed := Pilots[x1].td2;



// td1 holds scored distance in metres, dis holds actual flown distance in
metres. Truncate both to integer metres,

// then add dis/10000000 to tdis, so that sdis will hold scored distance to
left of decimal place and actual distance to right.

// Rounding sdis will still display correct scored distance to nearest 10
metres (0.01 km) and actual flown distance is now available from Soaring Spot
API



      Pilots[x1].sdis := trunc(Pilots[x1].td1) +
```

```
(trunc(Pilots[x1].dis)/10000000) ;



    end;



end;




//Main Program starts
here--------------------------------------------------------------------------
-------------------------



begin




    If CompClass = '?' then begin                                          //
Check to see if CompClass has been set

      Info4 := 'No Comp Class selected in Scoring script !!';

      exit;

    end;



    SetUpCompParameters;                                                   // Sets
the factors that vary according to Contest and task type



    If handicapped_comp then begin                                         //
Check for invalid speed indices

      If CheckHcaps then begin

        Info4 := 'Glider found with invalid Speed Index !!?';

        exit;

      end;
```

```
    end;



    Nl := GetNumberPilotsLaunched;                                  // How
many launched?



    If Nl < 1 then begin

      Info4 := 'No Glider Launches Detected !!';

      exit;

    end;



    GetTaskInfo;                                                    // Get
task info and set up flags and arrays for leg data

    ParseDayTag(DayTag);                                            //
Parse the DayTag string for additional task info and contest wind



    N  := GetNumberGliders;                                         // Get
number of gliders excudling Hors Concours and withdrawn



    If not Assigned_Area then SetDtask;                             // Set
Dtask (task distance)



    If Fixed_Course then CalculateTaskHiValues;                     // get
Hi value for each leg of Fixed Course Task



    If Fixed_Course and Nationals_Y then CalculateWindicapTaskDist(100);  //
get windicap task distance for use in determining Y in Nationals



    for i := low(Pilots) to high(Pilots) do                         //
calculate windicap distance and speed for each pilot
```

```
      begin


        CheckPilotData(Pilots[i]);                                          //
check to see if trace data present or manually scored


        If Pilots[i].Takeoff > 0 then                                       // If
launch detected

          CalcPilotVariables(Pilots[i]);                                    //
calculate scoring distance and speed

      end;




    If (Nl - ManualCount > 0) and (LegDataCount = 0) then begin         // if
we have data derived from traces but no trace leg data

      Info4 := 'Please ensure that "Need task legs data in scoring script" is
checked in Contest Properties/Options';

      exit;

    end;



   //calculate scoring variables



    Nf := GetNumberFinishers;                                           // How
many finished?



    Dmax := GreatestDistanceAnyGlider;                                  // also
sets Dw if no finishers



    Y  := CalculateY;                                                   //
calculate the qualifying distance, Y

    NY := CalculateNY;                                                  // and
```

```
find out how many exceeded it

    Vh := GreatestSpeed;                                                    //
calculate greatest speed (and for Fixed Course set Winners_Tg and Dw)

    Nv := CalculateNv;                                                      // how
many finishers exceeded 2/3 Vh

    Ff := CalculateDayFactorFf;                                             //
calculate day factor

    F  := CalculateDayPoints;                                               //
calculate total day points

    Fv := CalculateDaySpeedPointsFv ;                                       //
calculate spilt of day points into speed points

    Fd := CalculateDayDistancePoints;                                       // and
distance points



    for i := low(Pilots) to high(Pilots) do                                 //
calculate points for each pilot

      begin

        CalcPilotPoints(Pilots[i]);

      end;



   DisplayResults;                                                          //
Loads fields in Pilot records for display (start, Finish, Speed, Distance)



   // information about task is displayed using strings Info1, Info2, Info3 and
Info4

   // Info1 is set in the ParseDayTag procedure.

    Info2 := 'Y distance = ' + formatfloat('0.0',Y / 1000) + ' km';



    If Distance_Handicapped          then Info2 := Info2 + ',  Handicapped Task
distance = '+ formatfloat('0.0',Hcap_task_dist) + ' km';

    If Distance_Handicapped and (Hcap_task_dist = 0)
```

```
                                   then Info2 := Info2 + ' (DTask=' +
formatfloat('0.0',DTask/1000) + ' km)';

    If Fixed_Course   and debug_mode then Info2 := Info2 + ',  ' + Hi_string;

    If not debug_mode                then Info3 := 'Day Speed Points = ' +
Inttostr(round(Fv)) + ',  Day Distance Points = '

                                                                          +
Inttostr(round(Fd+Fv)-round(Fv));

    If F < 1000                      then Info4 := 'Reason for Devaluation : '
+ Devalued;




end.
```