

APPENDIX

**APPENDIX I: FORM FOR RECORDING OF DATA COLLECTED DURING
CLINICAL EXAMINATION OF POSSUMS**

POSSUM CARD

DATE OBSERVER

POSSUM No.

TRAP No. Cage (M,D,E)

TATTOO EAR NOTCHED

COLOUR: BBlack,BrownBlack,BBrown,GreyBrown,GRey

WEIGHT (kg)

LENGTH (cm) Tail Total

CONDITION: Good, Average, Poor

Male, Female Mature, Immature

TESTIS WIDTH (mm)

POUCH YOUNG: Present, Absent Lactating (Yes/No)

Tag No. Sex

Head length (mm) Weight (g)

PALPATION: (+, -)

ANAESTHETIC: Ketamin ml Other: ml
(specify)

BLOOD: + o - ml

TOOTH WEAR CLASS: (based on upper left first molar)



COMMENT:

APPENDIX III: FORM FOR RECORDING OF POSSUM NECROPSY DATA

POSSUM POST MORTEM CARD

DATE OBSERVER

POSSUM No.

LOCATION (trap in vicinity or other)

CAPTURE (C), FOUND (F)

Male, Female Mature, Immature

POUCH YOUNG: Present, Absent Tag No.

Head length (mm) Weight (g)

LENGTH (cm) Tail Total

WEIGHT (kg)

TESTIS WIDTH (mm)

COLOUR: Black,BrownBlack,Brown,GreyBrown,GRay


CONDITION: Good, Average, Poor

CAUSE OF DEATH: Natural, Euthanasia

CONDITION OF CARCASS (Fresh,Blooded,Decay,DRy)

TOOTH WEAR CLASS: (based on upper left first molar)

1 2 3 4 5 6 7



RIGHT MANDIBLE (and upper left first molar) Yes/No

POST MORTEM OBSERVATIONS: not possible

n.p.f. TB suspect lesions other

SAMPLES TAKEN Yes/No Specify under comments

COMMENTS:

DETAILS ON POST MORTEM FINDINGS

1. EXTERNAL EXAMINATION
Open lesions present? Yes/No

Describe:

2. ABDOMEN

Renal Lnn.:

Mesenterical Lnn.:

Liver:

Spleen:

Kidney:

Other:

3. THORAX

Mediastinal Lnn.:

Lung:

4. RETROPHARYNGEAL LNN.:

5. SUPERFICIAL LNN.:

Axillary Lnn.:

Inguinal Lnn.:

CODES:

0	= no pathological findings				
1	= few military lesions				
2	= many 1 - 2 mm lesions				
3	= up to 5 mm lesions				
4	= more than 5 mm lesions				
Consistency		F	= fluid, mucous		
		C	= caseous		
Position of organ		L	= left		
		R	= right		
		B	= both		
Colour		Y	= yellowish		
		G	= greenish		

**APPENDIX V: QUESTIONNAIRE FOR CASE FARMS INCLUDED IN
TUBERCULOSIS CASE-CONTROL STUDY**

**QUESTIONNAIRE FOR WAIKATO-WEST
TUBERCULOSIS CASE-CONTROL STUDY**

PROJECT MAFQual - MASSEY UNIVERSITY

VERSION FOR CASE HERDS

QUESTIONNAIRE NO. []

General Information

1. Date of interview: _____
2. Name of Interviewer: _____
3. Name of interviewee:
(note: where possible this should be the person directly managing the herd at the time of TB breakdown)

4. Address and location of farm:

5. Address of interviewee *(if different from farm address)*:

Telephone No.: _____

6. Name of **farm** owner (if not interviewee):

7. Address of farm owner *(if different from farm address)*:

8. Name of **herd** owner (if not interviewee):

9. Address of herd owner *(if different from farm address)*:

Farm Specific Information

We would now like to get some general information about your farm. We are interested in the situation at the time when the TB breakdown occurred.

10. What type of operation was your farm at the time of the breakdown?

(if more than one answer, give approximate percentage **total** farm income)

- cattle _____
- sheep _____
- deer _____
- goats _____
- pigs _____
- other (specify: _____)

11. What type of operation was the cattle related part of your farm?

(if more than one answer, give approximate percentage **total** farm income)

- dairy _____
- factory supply _____
- town supply _____
- beef _____
- breeding _____
- dry stock - fattening _____
- dry stock - dairy type _____
- other (specify: _____)

12. Did the ownership of the **farm** change during the 5 years prior to the date of the breakdown?

- no
- yes (specify how many times: _____)

13. Did the ownership of the **herd** change during the 5 years prior to the date of the breakdown?

- no
- yes (specify how many times: _____)

14. Did the management system of the farm change in a major way during the 5 years prior to the date of the breakdown?

- no
- yes (specify: _____)

15. What was the size of the farm at the time of the breakdown?
(specify vegetation areas in hectares)

Total farm size _____

crops _____ (specify type)

forest _____

willows _____

scrub/bush _____

gorse _____

water _____

pasture _____

non- grazeable area (not included above) _____

other, specify: _____

16. How would you describe the predominant topography of your farm?

flat

rolling

steep

17. Do you have any of the listed surface waters on your farm?

(if possible, give rough estimate of length or area; include boundary)

river/canal _____

creek _____

lake _____

swamp _____

open drain _____

flood prone area _____

18. What is the average rainfall per year on your farm? _____

General Information on Interviewee

We would now like to ask you some questions about your background and working situation.

19. Sex:

male

female

20. In what year were you born? _____

21. What is your ethnic background? _____

22. Do you live on the farm?

no

yes

23. Do you have any other employment commitments?
 no
 yes (specify: _____)
24. Can you cover your living expenses from your farm income?
 no
 yes
25. What is your relation to the property?
 owner
 share milker
 50%
 39%
 29%
 manager
 lessee
 other (specify: _____)

If interviewee is the owner, go to question no. 27.

26. What is your relation to the owner?
 parent
 relative
 none of the above
27. Which persons are involved in decisions concerning herd management?
 only interviewee
 others (specify: _____)

If interviewee is the owner, go to question no.31.

28. How long have you been working on this farm? _____
29. Does the owner live on the farm?
 no
 yes
30. How frequent is contact between the owner and you?
 daily
 at least once a week
 at least once month
 less than once a month
31. How many years have you worked in farming? _____
32. How many years have you worked in non-farm jobs? _____
(note: total of question 31 and 32 should equal years of work)

33. What is your farming background?
- started farm job without having any farming background
 - brought up on farm
 - other (specify: _____)
34. What level of formal education do you have?
- lower than school certificate
 - school certificate
 - university entrance
 - technical institute
 - university
35. What kind of farm-specific qualification do you have?
- none
 - technical agricultural institute
 - agricultural diploma
 - bachelor of agriculture
 - other (specify: _____)
36. Where do you seek advice concerning the herd management on your farm?
(give ratings in order of importance)
- none
 - family, relatives
 - owner or other workers on farm
 - neighbours
 - farm discussion group
 - stock agent
 - salesmen
 - MAF consultant
 - private consultant
 - veterinarian
 - magazines
 - books
 - other (specify: _____)
37. How many farm workers do you have on the farm?
(give approximate labour units according to the following categories including yourself)
- none
 - permanent
 - temporary
 - family

38. What kind of aids do you use to muster your animals?

- none
 motor bike
 tractor
 4 wheel drive vehicle
 dog
 horse
 other (specify: _____)

39. Have any capital investments been made on your farm during last two years?

- none
 minor (< \$ 5000)
 major

General Stock Information

We would now like to get some general information on your stock especially numbers, purchases, sales and agistments. We are interested in the situation at the present time.

40. Do you have any animals in the following categories?

(give approximate numbers)

- dairy cattle

weaners	_____
yearlings	_____
cows	_____
bulls	_____
- beef cattle

heifers	_____
cows	_____
steers	_____
bulls	_____
- deer

yearlings	_____
hinds	_____
stags	_____
- sheep (excluding lambs) _____
- goats (excluding kids) _____
- pigs (excluding unweaned piglets) _____

others, specify: _____

41. Have you introduced any animals on to your farm during the last 2 years?
(if not, go to question no.52)

yes

no

do not know

42. How many animals of the following stock categories have you bought during the last two years?

sheep _____

goats _____

pigs _____

others, specify: _____

43. Have you bought cattle during the last two years?
(if not, go to question no.48)

yes

no

do not know

44. What type of cattle did you buy in the last 2 years?
(give approximate numbers)

bobby calves

weaners

heifers

cows

bulls

steers

do not know

45. From how many different herds have you bought cattle during the last 2 years?

1 - 3 herds

more than 3 herds

46. What are your main reasons for deciding to buy cattle?
(give order of priority)

for trading purposes

need for breeding herd replacements

improvement of herd

breeding bull

increase herd size

other (specify: _____)

do not know

47. Where have you bought cattle?
(give rating according to importance)
- sale yards
 - through stock agent
 - private
 - clearing sale
 - other (specify: _____)
 - do not know
48. Have you bought or captured deer during the last two years?
(if not, go to question no.52)
- yes
 - no
 - do not know
49. Where have you bought or captured deer?
(give rating according to importance)
- sale yards
 - through stock agent
 - private
 - bush capture (specify location: _____)
 - other (specify: _____)
 - do not know
50. From how many properties have you bought or captured deer during the last two years?

51. What type of deer have you bought or captured?
(give numbers)
- fawns _____
 - hinds _____
 - stags _____
 - do not know
52. What measures do you take to avoid the introduction of infectious diseases through purchased stock?
(if more than one answer, give order of priority)
- none
 - do not buy
 - buy only from safe farms
 - buy only from farms after checking status with MAF
 - keep new stock separate
 - prior examination by veterinarian
 - get animals tested for (specify: _____)
 - other (specify: _____)

53. Have you grazed or leased any cattle belonging to other farmers on your farm during the last two years?

(if not, go to question no.58)

yes

no

do not know

54. From how many herds did you keep cattle belonging to other farmers on your farm during the last two years? _____

55. What type of cattle belonging to other farmers did you keep?

weaners

heifers

cows

bulls

steers

do not know

56. Where did the cattle belonging to other farmers come from?

Specify location:

57. Did you require any animal health tests prior to accepting the cattle?

yes (specify: _____)

no

58. Have you grazed or leased any deer belonging to other farmers on your farm during the last two years?

(if not, go to question no.62)

yes

no

do not know

59. From how many herds did you keep groups of deer belonging to other farmers on your farm during the last two years? _____

60. Where did the deer belonging to other farmers come from?

Specify location:

61. Did you require any animal health tests prior to accepting the deer?

yes (specify: _____)

no

Stock Management Information

We would now like to get some information about your methods for handling your stock. The questions are intended to relate to the time of the TB breakdown.

The first group of questions will be dealing with grazing management and the situation on the paddocks.

62. What system of cattle grazing management did you use?

(if more than one answer, give order of priority and reason)

- rotational grazing _____
- strip/block grazing _____
- set stocking _____
- open gate _____
- wintering pad/sacrifice paddock _____
- other (specify: _____)

63. What kind of fertilizer did you apply? (give order of priority and frequency of application per year)

- super phosphate _____
- lime _____
- organic fertilizer _____
- nitrogen _____
- other (specify: _____)
- none
- do not know

64. Did stock have access to the following water sources?

(give order of priority)

- river
- creek
- lake
- pond
- dam
- bore/well
- other (specify: _____)

65. Did you have problems with water supply for your stock during any period of the year?

- yes (give reasons: _____)
- no

66. Did you have a regular problem with flooding of pasture?

(if not, go to question no.68)

- yes
- no

67. Give a rough estimate of hectares of pasture that was most likely to be affected by floods:

68. Did you have difficulties in achieving complete herd mustering?
 yes (give reasons: _____)
 no
69. How often did stock from neighbouring farms break into your property?
 (specify species)
 at least once or twice a month _____
 at least once or twice a year _____
 rarely, if ever _____
70. How quickly would you have moved neighbour's stock out?
 same day
 same week
 later
71. Did your cattle have access to bush or forest?
(if not, go to question no.74)
 yes
 no
72. Where was this area?
 on farm
 off farm
73. For what reasons did you graze your cattle in bush or forest?
 part of paddock
 difficult to prevent
 additional food supply when required
 other (specify: _____)
74. What type of fencing system did you use?
 (give order of priority, code **B** for farm boundaries and **O** otherwise)
 permanent electric fences _____
 wire and batten fences _____
 barbed wire fences _____
 hedges _____
 movable electric fences _____
 other (specify: _____)
75. If you found that your fences were not stock proof, when did you fix them?
 same day
 within same week
 later

76. Have you cleared any paddocks from bush/forest in the last 5 years?
(if not, go to question no.78)

- yes
 no
 do not know

77. What cover or debris remained after clearing?
(give order of importance)

- trees
 logs
 bush or scrub
 uncleared gullies
 pasture only
 other (specify: _____)

78. Where did your cattle graze during the 2 years before TB breakdown?
(give order of importance)

- main farm
 run-off
 graze on other farm
 graze in forest/bush off farm
 other, specify: _____

79. Did you graze the road frontage?

- frequently
 rarely
 never

If no run-off was used or cattle were not grazed on other farm, go to question no.84.

80. What class of cattle grazed on run-off or other farm during the 2 years before TB breakdown?

- weaners
 yearlings
 hold-over dry cows
 steers
 others (specify: _____)

81. What was the location of run-off or other farm?

82. Did stock grazing on the run-off or other farm have the chance for frequent contact with cattle or deer belonging to other farmers?

- yes (specify species: _____)
 no
 do not know

83. Describe the run-off or other farm in terms of the size of the following types of cover (in hectares):

Total size _____

crops _____

forest _____

willows _____

scrub _____

gorse _____

water _____

pasture _____

other non-grazeable area _____

do not know

84. Did you keep permanent separate mobs of animals in your cattle or deer herd?

yes

no

85. Did you graze different species in the same paddock?

yes, specify:

no

The next group of questions is related to herd management. It is divided into sub groups for the different types of cattle operations.

86. What were your reasons for culling cattle in order of importance?

age

health problems

fertility/empty cow

production

other (specify: _____)

do not cull

87. Where did you get your replacements for your cattle herd from in order of priority?

purchase

lease

rearing

88. What kind of individual animal records did you keep?

- diary/notebook
- individual record cards for each animal
- computerized herd records (e.g. Livestock Improvement Association)
- other (specify: _____)
- none

89. Did you identify your cattle individually?

(if not, go to question no.91)

- yes
- no

90. What was your method of identification?

- eartags
- earmarks
- hide brand
- tattoo
- other (specify: _____)

If farmer did not keep fattening cattle, go to question no.92.

91. What method did you use for assessing weight gain in your fattening cattle?

- scale
- weighband
- condition scoring (numerical)
- eye appraisal
- other (specify: _____)

If farmer did not keep dairy or beef breeding cattle, go to question no.100.

92. What kind of breeding information did you record regularly?

- calving dates
- dates on heat
- mating dates
- pregnancy diagnosis
- production
- animal health
- other (specify: _____)

93. Which of the listed breeding methods did you use?

- only artificial insemination
- artificial insemination followed by natural breeding for clean-up
- natural breeding

94. If a beef herd, what was the calving rate during 1987? _____

95. What method did you use for calf rearing?

(number in order of importance)

suckle mother

suckle cows other than mother

bulk milk (from herd)

milk replacer

other (specify: _____)

do not rear calves

96. Did you provide any supplementary feeding to your cattle?

hay

silage

crops (specify according to following categories)

lucerne

concentrates

others (specify: _____)

97. Did you buy any hay during the last 2 years?

yes (specify location: _____)

no

If farmer did not keep dairy cattle, go to question no.100.

98. Did you record milk production?

yes

no

99. Which of the following methods did you use for mastitis control?

(if applicable, tick more than one)

none

teat spray

Individual Cow Somatic Cell Counts

dry cow therapy

other (specify: _____)

100. What method did you use for cattle effluent disposal?

nothing

settlement pond

spray on pasture

other (specify: _____)

The following questions are concerned with the animal health situation and management of your herd.

101. What are the four most important animal health problems in cattle and/or deer on your farm? (list in order of importance)

1. _____
2. _____
3. _____
4. _____

102. Do you drench your cattle herd regularly and why?
(give order of importance)

- no
- bloat
- facial eczema
- hypomagnesaemia
- internal parasites
- others (specify: _____)

103. What do you vaccinate your cattle for?

- leptospirosis
- blackleg/clostridial diseases
- salmonellosis
- Johne's disease
- other (specify: _____)
- no vaccination

104. How often does a private veterinarian visit your farm during calving and mating season?

- about once a week
- about once a month
- about once every 2-3 months
- rarely

105. Do you consider any animal diseases when purchasing or leasing animals?

- yes (specify: _____)
- no
- do not know

Tuberculosis Information

In this section we would like to ask you some questions seeking your opinion about and experiences with tuberculosis and tuberculosis control.

106. Do you think New Zealand should be willing to allow tuberculosis to be present permanently in the country, provided it is not on your farm? (give reasons)

yes _____

no _____

do not know

107. Would you be willing to allow TB to be present permanently on your farm? (give reasons)

yes _____

no _____

do not know

108. What methods does MAF use to control bovine TB?

1. _____

2. _____

3. _____

4. _____

do not know

109. Do you believe that overall these methods are effective?

yes, give reasons:

no, give reasons:

do not know

110. What do you think are the main reasons why tuberculosis has not been fully controlled in New Zealand?

1. _____

2. _____

3. _____

4. _____

do not know

111. If you were in charge of TB control, what approach would you use to control the disease?

1. _____
2. _____
3. _____
4. _____

do not know

112. What do you consider an appropriate percentage of animal value to use in compensating owners for TB infected cattle that are slaughtered?

If the government pays: _____

If farmer funds must be used: _____

do not know

113. Do you think that you personally can do anything to keep your herd clear from TB?

yes, specify:

no, give reasons:

do not know

114. Does tuberculosis affect any other species than cattle?

yes, specify:

no

do not know

115. Could you specify how the disease is spread from cattle to humans?

yes, specify in order of importance:

1. _____
2. _____
3. _____
4. _____
5. _____

no

116. Do you think you are working in a situation where you could catch the disease from your cattle, if they were infected?

yes, specify:

no

do not know

117. What do you think are the possible means for introduction of TB into a cattle herd? (give order of importance)

1. _____

2. _____

3. _____

4. _____

5. _____

do not know

118. Could you describe how the disease is spread between cattle?

yes, specify in order of importance:

1. _____

2. _____

3. _____

4. _____

5. _____

no

119. Are any wild animals important in the spread of TB?

yes, specify in order of importance:

no

do not know

120. What is the degree of contact between the following **wild** animals and your cattle? (code answer: 0=no, 1=possible, 2=very likely, 9=do not know)

possums

wild pigs

wild deer

wild cattle

wild goats

others (specify: _____)

121. Have there been TB infected wild or domesticated species identified in the neighbourhood of your farm during the last 5 years?

(if not, go to question no.123)

yes, specify species and location:

no

do not know

122. If yes, did you take any precautions for your herd?

yes, specify:

no, give reasons:

123. Has there been any possum control on your farm over the last 5 years?

(if not, go to question no.129)

yes (specify the last time: _____)

no

do not know

124. Why has possum control been done?

specify:

do not know

125. Who did the possum control?

you

friend, relative

commercial trapper

part time trapper

MAF

pest destruction board

local/county ranger

do not know

126. What methods have been used?

- 1080 poisoning
 cyanide poisoning
 shooting
 trapping
 other (specify: _____)
 do not know

127. How many times has it been done during the last 5 years? _____

128. Has any **major** possum control operation been done before TB breakdown?

- yes (specify when: _____)
 no
 do not know

129. What are your views about the use of **large scale** possum control operations for TB?

130. Which particular form of possum control do you consider best?

- do not know

131. Do you have any other comments on TB control in general, or in relation to your farm?

Farmer's self concept

132. This is the final question and here we would like you to give us an assessment of your personal characteristics. For this purpose please try to describe yourself on the scale in relation to the following characteristics.

Not easy going	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Easy going
Meek	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Not meek
Patient	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Impatient
Unsociable	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Sociable
Not modest	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Modest
Persevering	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Giving up Easily
A worrier	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Not a worrier
Cheerful	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Grumpy
Talkative	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Not talkative
One who speaks one's mind	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who keeps quiet
Difficult to get on with	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Easy to get on with
Lacking confidence	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Confident
Liking change	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Suspicious of change
Forceful	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Giving in easily
One who prefers machinery	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who prefers cows
One who prefers buying a new machine	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who prefers choosing a new bull
Unwilling to learn	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Very willing to learn
Still learning	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Very knowledgeable
One who likes to avoid hard work	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who values hard work
One who dislikes using records	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who likes using records
One who values traditional ways	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who likes adopting new ideas
One who does not like to set targets	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who likes setting targets for him/herself
One who likes to look after his/her favourite animals a bit better than the rest	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	One who likes to strictly monitor performance of the herd

MAF Tuberculosis Test Data

The following information will be available through MAF.

133. Record Type: _____

134. Report number: _____

135. Herd number: _____

136. County: _____

137. Livestock officer area: _____

138. Case Control Status:

Case

Management control

Random control

139. TB possum risk area of farm location:

Endemic

Fringe

Non-endemic

Surveillance

140. Approximate elevation above sea level of farm:

141. Date of herd establishment

cattle _____

deer _____

142. Date of herd accreditation

cattle _____

deer _____

143. Date of TB breakdown: _____

144. Results of last four herd tests for cattle and deer:

SPECIES	DATE	TYPE OF TEST	NO ANIMALS TESTED	NO OF CF/CC REACTORS	PM RESULT	CULTURE

145. Have there been difficulties in complete herd mustering? _____

146. TB possum risk area of location given in answer to question no.56:

147. TB possum risk area of location given in answer to question no.60:

148. What is the TB test status of the herd where the introduced deer originated? (refers to question no.60)

149. TB possum risk area of location given in answer to question no.81:

150. TB possum risk area of location in answer to question no.97:

151. MAF information to question no.121:

152. MAF information about major possum control operations including location of farm: (refers to question no.128)

153. Comments of the interviewer:

Please attach copy of MAF assessment of TB breakdown investigation.

APPENDIX VI: TURBO PASCAL FOR WINDOWS PROGRAM CODE FOR SIMULATION MODEL

```

program PossumPopulation;
{$D+}{$L+}{$R tmodel.res}{$G+}{$N+}{$F-}{$B-}{$W-}{$A+}{$S-}{$I-}{$V-}{$X+}
uses
  WObjects, WinProcs, WinCrt, WinDos, WinTypes, OPString, StringsListObj, OpDate, Ultrafx;
const
  Male    = 1;
  Female  = 0;
  MaxAge  = 6;
  pHealthy = 0;
  pInfected = 1;
  pClinical = 2;
  m = 2147483647;
  WindowSize: TPoint = (x:400;y:200);}
var
  DailyImmigrMale, DailyImmigrFemale : real;
  ControlInterval, ControlPeriod, NextControl, BackToNormal, code : integer;
  SplashRect: TRect;
  FirstSeed : longint;
  SurvAdultSeed, SurvImmatureSeed, MatingSeed, TBSurvAdultSeed, TBSurvImmSeed, TBImmigrSeed, SubToClinSeed,
  PopDensEmigrSeed, BirthSexSeed, TBPrevSeed, JoeySurvSeed, TBDenSeed, TBBufferSeed, DenRejectSeed, TB MatingSeed,
  AgeIndepSeed1, AgeIndepSeed2, AgeMatureSeed1, AgeMatureSeed2, DailyImmigrMaleSeed, DailyImmigrFemaleSeed,
  SurvAdultSeedOrg, SurvImmatureSeedOrg, MatingSeedOrg, TBSurvAdultSeedOrg, TBSurvImmSeedOrg, TBImmigrSeedOrg,
  SubToClinSeedOrg, PopDensEmigrSeedOrg, BirthSexSeedOrg, TBPrevSeedOrg, JoeySurvSeedOrg, TBDenSeedOrg,
  TBBufferSeedOrg, DenRejectSeedOrg, TB MatingSeedOrg, AgeIndepSeed1Org, AgeIndepSeed2Org, AgeMatureSeed1Org,
  AgeMatureSeed2Org, DailyImmigrMaleSeedOrg, DailyImmigrFemaleSeedOrg : extended;
  DenSummaryFile, NonDenSummaryFile : text;
  PopFile, DenMapFile, ControlFile, ParamFile, orN : string;
  PossumControl : boolean;
type
  SexType = byte;
  LocationType = longint;
TMyApplication = object(TApplication)
  FirstApp : Boolean;
procedure InitMainWindow; virtual;
end;

PMyWindow = TMyWindow;
TMyWindow = object(TWindow)
  MainFontRec : TLogFont;
  TheDC : HDC;
  procedure PaintPaintDC : HDC; var PaintInfo: TPaintStruct); virtual;
  procedure MakeFont; virtual;
  constructor Init(AParent: PWindowsObject; ATitle: PChar);
  procedure GetWindowClass(var WndClass : TWndClass); virtual;
  procedure About(var Msg: TMessage); virtual; m_First + 101;
  procedure RunModel(var Msg : TMessage); virtual; m_First + 201;
end;

ModelObj = object
  ParamFile : string;
  AdultSurvival : array [1..12, Female..Male] of real;
  ImmatureSurvival : array [1..12, Female..Male] of real;
  JoeySurvival : array [0..8] of real;
  Immigration : array [1..12, Female..Male] of integer;
  MatingProb : array [1..12] of real;
  MaleBirthProb : real;
  MatingBuffer : integer; Radius around female where to select mate from
  MatingTBProb : real;
  BackgroundTBProb : array [1..12] of real;
  ImmigrantTBProb : array [1..12] of real;
  ClinicalAdultSurvival : array [1..12] of real;
  ClinicalImmatureSurvival : array [1..12] of real;
  GoClinicalProb : array [1..12] of real;
  DenTBPeriod : integer; Number of days TB remains active in den
  DenTBProb : real; Probability of possum catching TB in a clinical den
  DenTBBuffer : integer; Radius of TB active buffer zone around clinical den
  BufferTBProb : real; Probability of catching TB in the buffer zone
  MaxDenTravel : integer;
  DenRejectProb : real;
  UseDenMemory : boolean;
  ResidentDenThreshold : integer;

```

```

NoDenMortality    : array [1..12] of real;
ImmigrantDenThreshold : integer;
ResidentDenWindow  : integer;
InitInfectProb : real;
constructorInit;
destructor Done; virtual;
procedure Edit;
procedure Go;
procedureWriteToFile(FileName : string);
procedureSetFromFile(FileName : string);
functionSaveChange :boolean;
procedure Load;
procedure Save;
end;

TimeObj = object
StartDate  : Date;
CurrentDate : Date;
constructorInit(InitDay, InitMonth, InitYear : integer);
destructor Done; virtual;
procedure Increment(IncDay, IncMonth, IncYear : integer);
function Day    : integer;
function Month  : integer;
function Year   : integer;
function Days   : integer;
function Months : integer;
function Years  : integer;
function Season : string;
functionMonthStr : string;
functionEndOfMonth :boolean;
procedureSaveState(FileName : string);
end;

AgeArray = array [0..MaxAgeFemale..Male] oflongint;
SummaryPtr = ^Summary;
Summary = record
Sum    :AgeArray;
SumSq  :AgeArray;
Next   :SummaryPtr;
end;

PossumPtr = ^Possum;

PossumListPtr = ^PossumList;

PossumList = object(List)
  constructorInit;
  destructor Done; virtual;
  procedureAddPossum(NewPossum :PossumPtr);
end;

PopulationPtr = ^Population;
Population = object(PossumList)
AgesHealthy : array [0..MaxAgeFemale..Male] of byte;
AgesClinical : array [0..MaxAgeFemale..Male] of byte;
AgesInfected : array [0..MaxAgeFemale..Male] of byte;
NewClinical  : array [0..2,Female..Male] of byte; {Incidence }
NewInfected  : array [0..2,Female..Male] of byte; {arrays }
AveNilDens : real;
NoInAve    : integer;
constructorInit;
destructor Done; virtual;
procedure Fill;
procedureDeletePossum(N :NodePtr);
procedure Immigration;
procedureUpdateAges;
procedureUpdateAveNilDens(New Value : integer);
procedure Grow;
procedureGoClinical;
procedure Mating;
function Fecundity : real;
procedure TB;
procedure Move;
procedure Death;
procedureDensityDynamics;
function Size    : integer;

```

```

functionNoMales(Status : byte) : integer;
functionNoFemales(Status : byte) : integer;
functionNilDens : integer;
procedurePrintAges;
functionAgeSummary : string;
functionDenSummary : string;
functionNonDenSummary : string;
procedurePrintList;
procedurePrintDenSummary;
procedurePrintNonDens;
procedureSaveState(FileName : string);
end;

DenListPtr = ^DenList;

DenPtr = ^DenObj;
DenObj = object
id      : word;
X       : LocationType;
Y       : LocationType;
Occupants : PossumListPtr;
CloseDens : DenListPtr;
DateLastTB : Date;
constructorInit(Initid : word;InitX, InitY : LocationType);
destructor Done; virtual;
function DistanceRefX, RefY : LocationType) : real;
function Empty : boolean;
procedureCloseDensList;
functionLoadNextDen : SNodePtr;
procedureCalculateGeography(Dens DenListPtr);
procedureWriteGeography;
procedureReadGeography(DensDenListPtr);
procedure DrawOriginX, OriginY : LocationType);
function Clinical : boolean;
end;

DenList = object$List)
MaxX, MinX, MaxY, MinY : LocationType;
constructorInit;
destructor Done; virtual;
procedureSetFromFile(FileName : string);
procedureCalculateGeography;
procedureLoadGeography;
procedure Draw;
procedureAddDen(Den :DenPtr);
functionClinicalDens : integer;
procedureClearInfection;
end;

Possum = object
Birthday,
DateIndependent,
MaturityDate : Date;
Sex :SexType;
Joey :PossumPtr;
Mother :PossumPtr;
Pregnant :boolean;
DenMemory :DenListPtr;
Den :DenPtr;
NoDenList :SListPtr;
Immigrant :boolean; {Once an animal dens in the area it is no longer considered an immigrant}Joey
are considered to be immigrants}

DateInfected : Date;
DateClinical : Date;
constructorInit(InitBirthday : Date;InitSex :SexType;InitDens :DenListPtr);
destructor Done; virtual;
procedure Reproduce;
procedure Independence;
function Live :boolean; {Grow older or die!}
function Mature :boolean;
function Age : integer;
functionAgeCode : byte;
procedureFindDen;
procedure LeaveDen;
procedure Infect;
function Clinical :boolean;

```

```

function Infected boolean;
end;

var
  Time : TimeObj;
  Model : ModelObj;
  Dens : DenList;
  Pop : Population;
  GraphDriver : integer;
  GraphMode : integer;
  ErrorCode : integer;

function FileExists(Filename : string) boolean;
var
  f : file;
begin
  {$I-}
  Assign(f, FileName);
  Reset(f);
  Close(f);
  {$I+}
  FileExists := ((OResult = 0) and (FileName <> ''));
end;

(*****
***** MODEL *****)
(*****)

constructor ModelObj.Init;
var
  i : integer;
begin
  FillChar(Self, SizeOf(Self), char(0));
  if ParamCount <> 7 then
    begin
      repeat
        write('Enter name of file for Start Population: ');
        readln(PopFile);
        if not fileexists(PopFile) then
          writeln('File does not exist, please enter');
        until fileexists(PopFile);
      repeat
        write('Enter name of file for den site locations: ');
        readln(DenMapFile);
        if not fileexists(DenMapFile) then
          writeln('File does not exist, please enter');
        until fileexists(DenMapFile);
      repeat
        write('Enter name of parameter file: ');
        readln(ParamFile);
        if not fileexists(ParamFile) then
          writeln('File does not exist, please enter');
        until fileexists(ParamFile);
      PossumControl := false;
      YorN := 'N';
      write('Do you want to implement control operations? (Y/N): ');
      readln(YorN);
      if UpCase(YorN[1]) = 'Y' then
        begin
          PossumControl := True;
          write('At what intervals do you want to control? (in days): ');
          readln(ControlInterval);
          write('Duration of control operation? (in days): ');
          readln(ControlPeriod);
          repeat
            write('Which parameter file stores effect of control? : ');
            readln(ControlFile);
            if not fileexists(ControlFile) then
              writeln('File does not exist, please enter');
            until fileexists(ControlFile);
          end;
        end;
      else
        begin
          PopFile := ParamStr(1);
          DenMapFile := ParamStr(2);

```

```

    ParamFile :=ParamStr(3);
    write('Start Population File: ');
    writeln(PopFile);
    write('Parameter File: ');
    writeln(ParamFile);
    write('Den Map File: ');
    writeln(DenMapFile);
    YorN :=ParamStr(4);
    write('Simulate Possum Control Operations :');
    writeln(YorN);
    if UpCase(YorN[1]) = 'Y' then
        begin
            PossumControl := True;
            val(ParamStr(5),ControlInterval, code);
            val(ParamStr(6),ControlPeriod, code);
            ControlFile :=ParamStr(7);
            write('Interval between control operations: ');
            write(ControlInterval);
            writeln(' days');
            write('Duration of possum control operation: ');
            write(ControlPeriod);
            writeln(' days');
            write('Parameter file for control operation: ');
            writeln(ControlFile);
        end;
    end;
    SetFromFileParamFile);
    YorN := 'N';
end;

destructorModelObj.Done;
begin
end;

procedureModelObj.SetFromFile(fileName : string);
var
    FileVar : text; : integer;
    UseDenMemoryCh : char;
begin
    Assign(FileVar,FileName);
    Reset(FileVar);
    for i := 1 to 12 do
        read(FileVar,AdultSurvival[i, Male]);
    for i := 1 to 12 do
        read(FileVar,AdultSurvival[i, Female]);
    for i := 1 to 12 do
        read(FileVar,ImmatureSurvival[i, Male]);
    for i := 1 to 12 do
        read(FileVar,ImmatureSurvival[i, Female]);
    for i := 1 to 12 do
        read(FileVar,Immigration[i, Female]);
    for i := 1 to 12 do
        read(FileVar,Immigration[i, Male]);
    for i := 1 to 12 do
        read(FileVar,MatingProb[i]);
    for i := 1 to 12 do
        read(FileVar,ClinicalAdultSurvival[i]);
    for i := 1 to 12 do
        read(FileVar,ClinicalImmatureSurvival[i]);
    for i := 1 to 12 do
        read(FileVar,BackgroundTBProb[i]);
    for i := 1 to 12 do
        read(FileVar,ImmigrantTBProb[i]);
    for i := 1 to 12 do
        read(FileVar,GoClinicalProb[i]);
        readln(FileVar,ResidentDenThresholdResidentDenWindow);
    for i := 1 to 12 do
        read(FileVar,NoDenMortality[i]);
        readln(FileVar,ImmigrantDenThreshold);
        readln(FileVar,MaleBirthProb,InitInfectProb);
    for i := 0 to 8 do
        read(FileVar,JoeySurvival[i]);
        readln(FileVar,DenTBPeriod,DenTBProb);
        readln(FileVar,DenTBBuffer,BufferTBProb);
        readln(FileVar,MaxDenTravel,DenRejectProb);
        readln(FileVar,UseDenMemoryCh);

```

```

    if UseDenMemoryCh = 'Y' then UseDenMemory := True
    else UseDenMemory := False;
    readln(FileVar, MatingBuffer, MatingTBProb);
    Close(FileVar);
end;

procedure ModelObj.Go;
const
    uiQuit = 0; { Global constants for user interrupt routine }
    uiSave = 1;
    uiNull = 2;
var
    NoDays, Day: longint;
    AntiTheticRun, AT, Run, NoRuns, Year, NoYears, Month, NoMonths, PrevYear: integer;
    Age, Sex, NilDens: integer;
    YorN, FileName, DenMapFile, PopFile: string;
    AgeFile, DenFile, DenSummaryFile: text;
    DayPrint, MonthPrint, YearPrint, PrintAge, PrintDens, NonDens, AntiThetic: boolean;
    SummaryP, SummaryStart, PrevS: SummaryPtr;
    GraphicalDisplay: boolean;
    Finished: boolean;
    Ch: char;
    EnterSeed, DumpDens, DumpDensDaily, DumpCurrentDens: boolean;
    oldx, oldy: byte;

procedure PrintAgeSummary;
begin
    writeln('Run: ', Run, ' Year: ', Time.Year,
            ' Time.MonthStr: ', Time.MonthStr, ' Pop.Size: ', HealthyMales: Pop.NoMalespHealthy), HealthyFemales: Pop.NoFemalespHealthy));
    writeln('          ', ClinicalMales: Pop.NoMalespClinical), ClinicalFemales: Pop.NoFemalespClinical),
            '          ', ClinicalDens: Dens.ClinicalDens);
    writeln('          ', InfectedMales: Pop.NoMalespInfected), InfectedFemales: Pop.NoFemalespInfected),
            '          ', InfectedDens: Dens.ClinicalDens);
    writeln('          ', Fecundity: ', Pop.Fecundity:5:3);
end;

procedure WriteDens(DayorMonth: integer);
var
    DenNode: SNodePtr;
    Den: DenPtr;
begin
    DenNode := Dens.First;
    while DenNode <> nil do
        begin
            Den := DenNode^.Data;
            if (DumpCurrentDens and Den^.Clinical) or ((not DumpCurrentDens) and Den^.DateLastTB <> BadDate) then
                writeln(DenFile, DayorMonth:5,',', Den^.ID:5);
            DenNode := DenNode^.Next;
        end;
end;

function UserInterrupt: byte;
const
    DisplayText = 'Pause...press Q to quit, S to save, other to continue';
begin
    case ReadKey of
        #27, 'Q', 'q': UserInterrupt := uiQuit;
        'S', 's': UserInterrupt := uiSave;
    else
        begin
            writeln(DisplayText);
            while not KeyPressed do
                begin
                    end;
                case ReadKey of
                    #27, 'Q', 'q': UserInterrupt := uiQuit;
                    'S', 's': UserInterrupt := uiSave;
                end;
            end;
        end;
end;

begin
    clrscr;
    writeln('Simulation Model of a Possum Population');
    writeln;
    NoRuns := 1;

```

```

NoDays := 10000;
write(' Enter number of days: ');
readln(NoDays);
writeln;
write(' Print summary each day (Y/NN=default) ');
readln(YorN);
DayPrint := False;
if UpCase(YorN[1]) = 'Y' then DayPrint := True;
write(' Print summary each month (Y/NN=default) ');
readln(YorN);
MonthPrint := False;
if UpCase(YorN[1]) = 'Y' then MonthPrint := True;
if not MonthPrint then
  begin
    write(' Print summary each year (Y/NN=default) ');
    readln(YorN);
    YearPrint := False;
    if UpCase(YorN[1]) = 'Y' then YearPrint := True;
  end;
write(' Print monthly age distribution to file (Y/NN=default) ');
readln(YorN);
PrintAge := False;
if UpCase(YorN[1]) = 'Y' then PrintAge := True;
if PrintAge then
  begin
    write(' Enter filename to use : ');
    readln(FileName);
    Assign(AgeFile,FileName);
    Rewrite(AgeFile);
    writeln(AgeFile, 'Day,Month,Year,NoDays,MM,MF,IM,IF,JM,JF,IMM,NIMM,IMF,NIMF,IIM,NIIM,IIF,
      NIIF,IJM,NIJM,IJF,NIJF,CMM,NCMM,CMF,NCMF,CIM,NCIM,CIF,NCIF,CJM,NCJM,CJF,NCJF');
  end;
write(' Print monthly den distribution to file (Y/NN=default) ');
readln(YorN);
PrintDens := False;
if UpCase(YorN[1]) = 'Y' then PrintDens := True;
YorN := 'N';
if PrintDens then
  begin
    write(' Enter filename to use : ');
    readln(FileName);
    Assign(DenSummaryFile,FileName);
    Rewrite(DenSummaryFile);
    writeln(DenSummaryFile, 'Day,Month,Year,NoDays,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20');
  end;
write(' Print monthly distribution of denning possums to file (Y/NN=default) ');
readln(YorN);
NonDens := False;
if UpCase(YorN[1]) = 'Y' then NonDens := True;
YorN := 'N';
if NonDens then
  begin
    write(' Enter filename to use : ');
    readln(FileName);
    Assign(NonDenSummaryFile,FileName);
    Rewrite(NonDenSummaryFile);
    writeln(NonDenSummaryFile, 'Day,Month,Year,NoDays,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20');
  end;
write(' Dumpcoordinates of clinical dens (Y/NN=default) ');
readln(YorN);
DumpDens := False;
if UpCase(YorN[1]) = 'Y' then DumpDens := True;
if DumpDens then
  begin
    DumpCurrentDens := true;
    write(' Current or All (C/AC=default) ');
    readln(YorN);
    if UpCase(YorN[1]) = 'A' then DumpCurrentDens := false;
    DumpDensDaily := false;
    write(' Daily or Monthly (D/MM=default) ');
    readln(YorN);
    if UpCase(YorN[1]) = 'D' then DumpDensDaily := true;
    write(' Enter filename to use : ');
    readln(FileName);
    Assign(DenFile,FileName);
    Rewrite(DenFile);
  end;

```

```

if DumpDensDaily then write(DenFile, 'Day ')
else
  write(DenFile, 'Month, ');
  writeln(DenFile, DenID, X, Y');
end;
YorN := 'Y'; {default setting}
write(' Do you want to use antithetic variates (Y/N; Y=default): ');
readln(YorN);
AntiThetic := true;
AntiTheticRun := 2;
if UpCase(YorN[1]) = 'N' then
  begin
    AntiThetic := False;
    AntiTheticRun := 1;
  end;
YorN := 'N';
write(' Enter random number seed (N = randomize (default)) : ');
readln(YorN);
EnterSeed := false;
if UpCase(YorN[1]) = 'Y' then EnterSeed := True;
if EnterSeed = False then
  begin
    randomize;
    FirstSeed := RandSeed;
    rinit(RandSeed, 7654321);
    write(' This is the Seed for the Random Number Generators : ');
    writeln(FirstSeed);
    SurvAdultSeed := i31bit;
    SurvImmatureSeed := i31bit;
    MatingSeed := i31bit;
    TBSurvAdultSeed := i31bit;
    TBSurvImmSeed := i31bit;
    TBImmigrSeed := i31bit;
    SubToClinSeed := i31bit;
    PopDensEmigrSeed := i31bit;
    BirthSexSeed := i31bit;
    TBPrevSeed := i31bit;
    JoeySurvSeed := i31bit;
    TBDenSeed := i31bit;
    TBBufferSeed := i31bit;
    DenRejectSeed := i31bit;
    TBMatingSeed := i31bit;
    AgeIndepSeed1 := i31bit;
    AgeIndepSeed2 := m - AgeIndepSeed1;
    AgeMatureSeed1 := i31bit;
    AgeMatureSeed2 := m - AgeMatureSeed1;
    DailyImmigrFemaleSeed := i31bit;
    DailyImmigrMaleSeed := i31bit;
  end;
if EnterSeed = True then
  begin
    write(' Enter Seed for Random Number Generation : ');
    readln(FirstSeed);
    RandSeed := FirstSeed;
    rinit(RandSeed, 7654321);
    write(' This is the Seed for the Random Number Generators : ');
    writeln(FirstSeed);
    SurvAdultSeed := i31bit;
    SurvImmatureSeed := i31bit;
    MatingSeed := i31bit;
    TBSurvAdultSeed := i31bit;
    TBSurvImmSeed := i31bit;
    TBImmigrSeed := i31bit;
    SubToClinSeed := i31bit;
    PopDensEmigrSeed := i31bit;
    BirthSexSeed := i31bit;
    TBPrevSeed := i31bit;
    JoeySurvSeed := i31bit;
    TBDenSeed := i31bit;
    TBBufferSeed := i31bit;
    DenRejectSeed := i31bit;
    TBMatingSeed := i31bit;
    AgeIndepSeed1 := i31bit;
    AgeIndepSeed2 := m - AgeIndepSeed1;
    AgeMatureSeed1 := i31bit;
    AgeMatureSeed2 := m - AgeMatureSeed1;
  end;

```



```

        BackToNormal := Day +ControlPeriod;
        writeln;
        write('Begin Possum Control Operation on Day : ');
        writeln(Day);
    end;
    if Day = NextControl then
    begin
        SetfromFileControlFile);
        NextControl := Day +ControlInterval;
        BackToNormal := Day +ControlPeriod;
        writeln;
        write('Begin Possum Control Operation on Day : ');
        writeln(Day);
    end;
    if Day = BackToNormal then
    begin
        SetfromFileParamFile);
        writeln;
        write('End Possum Control Operation on Day : ');
        writeln(Day);
    end;
    end;
    if MatingProb[Time.Month] > 0 thenPop.Mating;
    Pop.Immigration;
    Pop.Move;
    Pop.TB;
    Pop.DensityDynamics;
    if DumpDens andDumpDensDaily thenWriteDens(Day);
    NilDens :=Pop.NilDens;
    Pop.UpdateAveNilDens$NilDens);
    write('.');
    if Time.Day = 1 then
    begin
        Pop.GoClinical;
    end;
    if Time.Day = 15 then
    begin
        Pop.Grow;
    end;
    { End of Month }
    if Time.EndOfMonth then
    begin
        writeln;
        inc(NoMonths);
        if (notDayPrint) and (notGraphicalDisplay) thenwriteln;
        if MonthPrint then
        begin
            writeln;
            PrintAgeSummary;
        end;
        if PrintAge then
        writeln(AgeFile,DateToDateString('DD,MM,YY',Time.CurrentDate),',',
            Time.CurrentDate -Time.StartDate,',',Pop.AgeSummary);
        if PrintDens thenwriteln(DenSummaryFile.DateToDateString('DD,MM,YY',
            Time.CurrentDate),',',Time.CurrentDate -Time.StartDate,',',Pop.DenSummary);
        if NonDens then
            writeln(NonDenSummaryFile.DateToDateString('DD,MM,YY',
                Time.CurrentDate),',',Time.CurrentDate -Time.StartDate,',',
                Pop.NonDenSummary);
        if DumpDens and (notDumpDensDaily) thenWriteDens(NoMonths);
    end; { of Month }
    if Time.Year <>PrevYear then
    begin
        PrevYear :=Time.Year;
        if (notMonthPrint) and (YearPrint) then
        begin
            Pop.PrintDenSummary;
            PrintAgeSummary;
        end;
    end; { of Year }
    Pop.UpdateAges;
    if KeyPressed then
    begin
        case UserInterupt of
            uiQuit : Day :=NoDays;
            uiSave :SaveState;

```

```

        end;
        end;
        Time.Increment(1, 0, 0);
    end; { of Day }
    writeln;
    PrintAgeSummary;
    Pop.Done;
    writeln(Pop.Done');
    Time.Done;
    Dens.ClearInfection;
end; { end of Antithetic run }
end; { of Run }
if PrintAge then CloseAgeFile;
if Printdens then CloseDenSummaryFile;
if NonDens then CloseNonDenSummaryFile;
if DumpDens then CloseDenFile;
writeln;
write('End of Simulation, press Return. ');
readln;
end;

{Random Number Generators }

FUNCTION SurvAdultRandom(VARSurvAdultSeed: extended): extended;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * SurvAdultSeed;
    Temp := Temp / m;
    SurvAdultSeed :=Int(m*Frac(Temp) + 0.5);
    SurvAdultRandom :=SurvAdultSeed *rm;
END;

FUNCTION SurvImmatureRandom(VARSurvImmatureSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * SurvImmatureSeed;
    Temp := Temp / m;
    SurvImmatureSeed :=Int(m*Frac(Temp) + 0.5);
    SurvImmatureRandom :=SurvImmatureSeed*rm;
END;

FUNCTION MatingRandom(VARMatingSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * MatingSeed;
    Temp := Temp / m;
    MatingSeed :=Int(m*Frac(Temp) + 0.5);
    MatingRandom :=MatingSeed*rm;
END;

FUNCTION TBSurvAdultRandom(VARTBSurvAdultSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * TBSurvAdultSeed;
    Temp := Temp / m;
    TBSurvAdultSeed :=Int(m*Frac(Temp) + 0.5);

```

```

    TBSurvAdultRandom :=TBSurvAdultSeed*rm;
END;

FUNCTION TBSurvImmRandom(VARTBSurvImmSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * TBSurvImmSeed;
    Temp := Temp / m;
    TBSurvImmSeed :=Int(m*Frac(Temp) + 0.5);
    TBSurvImmRandom :=TBSurvImmSeed*rm;
END;

FUNCTION SubToClinRandom(VARSubToClinSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * SubToClinSeed;
    Temp := Temp / m;
    SubToClinSeed :=Int(m*Frac(Temp) + 0.5);
    SubToClinRandom :=SubToClinSeed*rm;
END;

FUNCTION PopDensEmigrRandom(VARPopDensEmigrSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * PopDensEmigrSeed;
    Temp := Temp / m;
    PopDensEmigrSeed :=Int(m*Frac(Temp) + 0.5);
    PopDensEmigrRandom :=PopDensEmigrSeed*rm;
END;

FUNCTION BirthSexRandom(VARBirthSexSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * BirthSexSeed;
    Temp := Temp / m;
    BirthSexSeed :=Int(m*Frac(Temp) + 0.5);
    BirthSexRandom :=BirthSexSeed*rm;
END;

FUNCTION TBPrevRandom(VARTBPrevSeed: extended): real;
CONST
    m=2147483647;
    a=742938285;
    rm= 0.4656612875246e-9;    (* 1.0/m *)
VAR
    Temp : extended;
BEGIN
    Temp := a * TBPrevSeed;
    Temp := Temp / m;
    TBPrevSeed :=Int(m*Frac(Temp) + 0.5);
    TBPrevRandom :=TBPrevSeed*rm;
END;

FUNCTION JoeySurvRandom(VARJoeySurvSeed: extended): real;
CONST
    m=2147483647;

```

```

a=742938285;
rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * JoeySurvSeed;
  Temp := Temp / m;
  JoeySurvSeed :=Int(m*Frac(Temp) + 0.5);
  JoeySurvRandom :=JoeySurvSeed*rm;
END;

FUNCTION TBDenRandom(VARTBDenSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * TBDenSeed;
  Temp := Temp / m;
  TBDenSeed :=Int(m*Frac(Temp) + 0.5);
  TBDenRandom :=TBDenSeed*rm;
END;

FUNCTION TBBufferRandom(VARTBBufferSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * TBBufferSeed;
  Temp := Temp / m;
  TBBufferSeed :=Int(m*Frac(Temp) + 0.5);
  TBBufferRandom :=TBBufferSeed*rm;
END;

FUNCTION DenRejectRandom(VARDenRejectSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * DenRejectSeed;
  Temp := Temp / m;
  DenRejectSeed :=Int(m*Frac(Temp) + 0.5);
  DenRejectRandom :=DenRejectSeed*rm;
END;

FUNCTION TBMatingRandom(VARTBMatingSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * TBMatingSeed;
  Temp := Temp / m;
  TBMatingSeed :=Int(m*Frac(Temp) + 0.5);
  TBMatingRandom :=TBMatingSeed*rm;
END;

FUNCTION TBImmigrRandom(VARTBImmigrSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * TBImmigrSeed;

```

```

Temp := Temp / m;
TBImmigrSeed :=Int(m*Frac(Temp) + 0.5);
TBImmigrRandom :=TBImmigrSeed*rm;
END

```

```

FUNCTION MyRandom(VAR MySeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * MySeed;
  Temp := Temp / m;
  MySeed :=Int(m*Frac(Temp) + 0.5);
  MyRandom :=MySeed*rm;
END;

```

```

FUNCTION AgeMatureRandom1(var AgeMatureSeed1 : extended): extended;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * AgeMatureSeed1;
  Temp := Temp / m;
  AgeMatureSeed1 :=Int(m*Frac(Temp) + 0.5);
  AgeMatureRandom1 := AgeMatureSeed1*rm;
END;

```

```

FUNCTION AgeMatureRandom2(var AgeMatureSeed2 : extended): extended;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * AgeMatureSeed2;
  Temp := Temp / m;
  AgeMatureSeed2 :=Int(m*Frac(Temp) + 0.5);
  AgeMatureRandom2 := AgeMatureSeed2*rm;
END;

```

```

function AgeMatureNormal(MeanStdDev, Max, Min : real) : real;
var
  Result1,Result2,w,w1,w2,c : real;
begin
  repeat
    repeat
      w1 := 2 * AgeMatureRandom1(AgeMatureSeed1) - 1;
      w2 := 2 * AgeMatureRandom2(AgeMatureSeed2) - 1;
      w := w1 * w1 + w2 * w2;
    until w < 1;
    c := sqrt(-2*ln(w)/w);
    Result1 := w1 * c;
    Result2 := Mean + Result1 *StdDev
  until (Result2 <= Max) and (Result2 >= Min);
  AgeMatureNormal := Result2;
end;

```

```

FUNCTION AgeIndepRandom1(var AgeIndepSeed1 : extended): extended;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * AgeIndepSeed1;
  Temp := Temp / m;
  AgeIndepSeed1 :=Int(m*Frac(Temp) + 0.5);
  AgeIndepRandom1 := AgeIndepSeed1*rm;

```

```

END;

FUNCTION AgeIndepRandom2(var AgeIndepSeed2 : extended): extended;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * AgeIndepSeed2;
  Temp := Temp / m;
  AgeIndepSeed2 :=Int(m*Frac(Temp) + 0.5);
  AgeIndepRandom2 := AgeIndepSeed2*rm;
END;

function AgeIndepNormal(var AgeIndepSeed1, AgeIndepSeed2 : extended; Mean, StdDev, Max, Min : real) : real;
var
  Result1, Result2, w, w1, w2, c : real;
begin
  repeat
    repeat
      w1 := 2 * AgeIndepRandom1(AgeIndepSeed1) - 1;
      w2 := 2 * AgeIndepRandom2(AgeIndepSeed2) - 1;
      w := w1 * w1 + w2 * w2;
    until w < 1;
    c := sqrt(-2*ln(w)/w);
    Result1 := w1 * c;
    Result2 := Mean + Result1 * StdDev ;
  until (Result2 <= Max) and (Result2 >= Min);
  AgeIndepNormal := Result2;
end;

FUNCTION DailyImmigrMaleRandom(VAR DailyImmigrMaleSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * DailyImmigrMaleSeed;
  Temp := Temp / m;
  DailyImmigrMaleSeed :=Int(m*Frac(Temp) + 0.5);
  DailyImmigrMaleRandom :=DailyImmigrMaleSeed*rm;
END;

FUNCTION DailyImmigrFemaleRandom(VAR DailyImmigrFemaleSeed: extended): real;
CONST
  m=2147483647;
  a=742938285;
  rm= 0.4656612875246e-9;  (* 1.0/m *)
VAR
  Temp : extended;
BEGIN
  Temp := a * DailyImmigrFemaleSeed;
  Temp := Temp / m;
  DailyImmigrFemaleSeed :=Int(m*Frac(Temp) + 0.5);
  DailyImmigrFemaleRandom :=DailyImmigrFemaleSeed*rm;
END;

function DailyImmigrMalePoisson(DailyImmigrMale : real; var DailyImmigrMaleSeed : extended) : integer;
var
  Count : integer;
  ProbZero, Product : real;
begin
  Count := 0;
  Product :=DailyImmigrMaleRandom(DailyImmigrMaleSeed);
  ProbZero :=exp(-DailyImmigrMale);
  while Product >ProbZero do
    {Poisson Distribution}
    begin
      Count := Count + 1;
      Product := Product *DailyImmigrMaleRandom(DailyImmigrMaleSeed);
    end;
  DailyImmigrMalePoisson := Count;
end;

```

```

functionDailyImmigrFemalePoisson(DailyImmigrFemale : real, var DailyImmigrFemaleSeed : extended) : integer;
var
  Count : integer;
  ProbZero, Product : real;
begin
  Count := 0;
  Product := DailyImmigrFemaleRandom(DailyImmigrFemaleSeed);
  ProbZero := exp(-DailyImmigrMale);
  while Product > ProbZero do
    {Poisson Distribution}
    begin
      Count := Count + 1;
      Product := Product * DailyImmigrFemaleRandom(DailyImmigrFemaleSeed);
    end;
  DailyImmigrFemalePoisson := Count;
end;

{ End of Random Number Generator Section }

(*****
*****      TIME      *****
*****
*****

constructorTimeObj.Init (InitDay, InitMonth, InitYear : integer);
begin
  StartDate := DMYToDate(InitDay, InitMonth, InitYear);
  CurrentDate := StartDate;
end;

destructorTimeObj.Done;
begin
end;

functionTimeObj.Day : integer;
var
  D, M, Y : integer;
begin
  DateToDMY(CurrentDate, D, M, Y);
  Day := D;
end;

functionTimeObj.Month : integer;
var
  D, M, Y : integer;
begin
  DateToDMY(CurrentDate, D, M, Y);
  Month := M;
end;

functionTimeObj.Year : integer;
var
  D, M, Y : integer;
begin
  DateToDMY(CurrentDate, D, M, Y);
  Year := Y;
end;

functionTimeObj.Days : integer;
var
  D, M, Y : integer;
begin
  DateDiff(StartDate, CurrentDate, D, M, Y);
  Days := D;
end;

functionTimeObj.Months : integer;
var
  D, M, Y : integer;
begin
  DateDiff(StartDate, CurrentDate, D, M, Y);
  Months := M;
end;

functionTimeObj.Years : integer;
var
  D, M, Y : integer;

```

```

begin
  DateDiff(StartDate,CurrentDate, D, M, Y);
  Years := Y;
end;

procedureTimeObj.Increment (IncDay,IncMonth,IncYear : integer);
begin
  CurrentDate :=IncDate(CurrentDate,IncDay,IncMonth,IncYear);
end;

functionTimeObj.Season : string;
begin
  case Month of
    12, 1, 2 : Season := 'Summer';
    3..5   : Season := 'Autumn';
    6..8   : Season := 'Winter';
    9..11  : Season := 'Spring';
  end;
end;

functionTimeObj.MonthStr : string;
begin
  case Month of
    1 : MonthStr := 'January';
    2 : MonthStr := 'February';
    3 : MonthStr := 'March';
    4 : MonthStr := 'April';
    5 : MonthStr := 'May';
    6 : MonthStr := 'June';
    7 : MonthStr := 'July';
    8 : MonthStr := 'August';
    9 : MonthStr := 'September';
    10 : MonthStr := 'October';
    11 : MonthStr := 'November';
    12 : MonthStr := 'December';
  end;
end;

functionTimeObj.EndOfMonth boolean;
var
  D, M, Y : integer;
begin
  DateToDMY(CurrentDate+1, D, M, Y);
  if M <> Month thenEndOfMonth := true elseEndOfMonth := false;
end;

procedureTimeObj.SaveStateFileName : string);
var
  FileVar : text;
  i       : integer;
begin
  Assign(FileVar,FileName);
  Rewrite(FileVar);
  writeln(FileVar,CurrentDate);
  Close(FileVar);
end;

(*****
*****      DEN      *****)
(*****

constructorDenObj.Init(Initid : word;InitX, InitY : LocationType);
begin
  id := Initid;
  X := InitX;
  Y := InitY;
  Occupants := NewPossumListPtr,Init);
  CloseDens := NewDenListPtr,Init);
  DateLastTB := BadDate;
end;

destructorDenObj.Done;
begin
  Occupants^.Done;
  CloseDens^.Done;
end;

```

```

function DenObj.Distance(RefX, RefY : LocationType) : real;
begin
  Distance := sqrt(((RefX - X) * RefX - X)) + ((RefY - Y) * RefY - Y));
end;

function DenObj.Empty : boolean;
begin
  if Occupants^.First = nil then Empty := true else Empty := false;
end;

procedure DenObj.CloseDensList;
var
  tmp, dist : string;
  DenNode : SNodePtr;
  Den : DenPtr;
begin
  tmp := "";
  DenNode := CloseDens^.First;
  while (DenNode <> nil) and (length(tmp) < 250) do
    begin
      Den := DenNode^.Data;
      writeln(Den^.id:4, Distance(Den^.X, Den^.Y):4:0);
      DenNode := DenNode^.Next;
    end;
  readln;
end;

procedure DenObj.CalculateGeography(Dens DenListPtr);
var
  Node, SearchNode : SNodePtr;
  OtherDen, SearchDen, FarDen : DenPtr;
  DistanceToOther : real;
  Finished : boolean;
begin
  CloseDens^.Done;
  CloseDens^.Init;
  Node := Dens^.First;
  while Node <> nil do
    begin
      OtherDen := Node^.Data;
      DistanceToOther := sqrt(((OtherDen^.X - X) * OtherDen^.X - X)) + ((OtherDen^.Y - Y) * OtherDen^.Y - Y));
      if DistanceToOther < Model.MaxDenTravel then
        begin
          if (CloseDens^.Length = 0) then First Den in list
            CloseDens^.AddNode(New SNodePtr, Init(OtherDen))
          else
            begin
              FarDen := CloseDens^.Last^.Data;
              if (DistanceToOther > FarDen^.Distance(X, Y)) then
                begin
                  { Add to end of close den list }
                  CloseDens^.AddNode(New SNodePtr, Init(OtherDen));
                end
              else
                begin
                  if (DistanceToOther < DenPtr(CloseDens^.First^.Data)^.Distance(X, Y)) then
                    begin
                      { Add to beginning }
                      CloseDens^.InsertAfter(New SNodePtr, Init(OtherDen)), nil);
                    end
                  else
                    begin
                      { Insert somewhere in middle }
                      SearchNode := CloseDens^.First;
                      { Check each den in CloseDen list to determine position of best Den }
                      Finished := false;
                      while (not Finished) and SearchNode^.Next <> nil do
                        begin
                          SearchDen := SearchNode^.Next^.Data;
                          if SearchDen^.Distance(X, Y) > DistanceToOther then
                            begin
                              CloseDens^.InsertAfter(New SNodePtr, Init(OtherDen)), SearchNode);
                              Finished := true;
                            end
                        end
                      else
                    end
                end
            end
          end
        end
      end
    end
  end
end;

```



```

Den :DenPtr;
DenNode :SNodePtr;
Found :boolean;
begin
str(Id, strid);
Assign(F, CD'+strid+'.dat');
Reset(F);
DenNode := nil;
Count := 1;
while (notEof(f)) and DenNode = nil) do
begin
readln(F, Denid);
if Count > CloseDens^.Length then
begin
{ Find pointer to den }
DenNode := Dens.First;
Found := false;
while (DenNode <> nil) and (not Found) do
begin
Den := DenNode^.Data;
if (Den^.id = DenId) then Found := true else DenNode := DenNode^.Next;
end;
if DenNode <> nil then
begin
DenNode := NewSNodePtr, Init(Den));
if DenNode = nil then OutOfMemory(LoadNextDen');
CloseDens^.InsertAfterDenNode, CloseDens^.Last);
end
else
begin
writeln(LoadNextDen: Error loading den ',id,', denDenid,' not found in Dens');
halt(1);
end;
end
end
else
inc(Count);
end; { While }
if eof(F) then
begin
writeln(LoadNextDen: Error end of file in den ',id', CloseDens.Length=', CloseDens^.Length);
writeln(Model.DenTBBuffer);
halt(1);
end;
LoadNextDen := DenNode;
Close(F);
end;

function DenObj.Clinical :boolean;
begin
if (DateLastTB = BadDate) or ((Time.CurrentDate -DateLastTB) > Model.DenTBPeriod) then
Clinical := false
else
Clinical := true;
end;

(*****
***** DENLIST *****)
(*****

constructor DenList.Init;
begin
SList.Init;
MaxX := 0;
MaxY := 0;
MinX := 0;
MinY := 0;
end;

procedure DenList.SetFromFile(FileName : string);
var
F : text;
id : word;
X, Y : LocationType;
begin
Assign(F, FileName);
Reset(F);

```

```

readln(F, id, Y, X);
MaxX := X; MinX := X; MaxY := Y; MinY := Y;
repeat
  AddDen(NewDenPtr, Init(id, X, Y));
  if X > MaxX then MaxX := X;
  if X < MinX then MinX := X;
  if Y > MaxY then MaxY := Y;
  if Y < MinY then MinY := Y;
  readln(F, id, Y, X);
until eof(F);
Close(F);
end;

destructor DenList.Done;
begin
  SList.Done;
end;

procedure DenList.CalculateGeography;
var
  DenNode : SNodePtr;
  Den : DenPtr;
  TmpMax : integer;
begin
  DenNode := First;
  TmpMax := Model.MaxDenTravel;
  while DenNode <> nil do
    begin
      Den := DenNode^.Data;
      writeln('Calculating for den 'Den^.Id);
      Den^.CalculateGeographyAddr(Self);
      Den^.WriteGeography;
      Den^.ReadGeographyAddr(Self);
      DenNode := DenNode^.Next;
    end;
  Model.MaxDenTravel := TmpMax;
  LoadGeography;
end;

procedure DenList.LoadGeography;
var
  DenNode : SNodePtr;
  Den : DenPtr;
  Tmp : integer;
begin
  DenNode := First;
  Tmp := Model.MaxDenTravel;
  Model.MaxDenTravel := 0;
  while DenNode <> nil do
    begin
      Den := DenNode^.Data;
      Den^.ReadGeographyAddr(Self);
      DenNode := DenNode^.Next;
    end;
  Model.MaxDenTravel := Tmp;
end;

procedure DenList.AddDen(Den DenPtr);
begin
  AddNode(NewSNodePtr, Init(Den));
end;

function DenList.ClinicalDens : integer;
var
  DenNode : SNodePtr;
  Den : DenPtr;
  Tmp : integer;
begin
  tmp := 0;
  DenNode := First;
  while DenNode <> nil do
    begin
      Den := DenNode^.Data;
      if (Den^.Clinical) then inc(tmp);
      DenNode := DenNode^.Next;
    end;
end;

```

```

    ClinicalDens :=tmp;
end;

procedureDenList.Draw;
var
    DenNode :SNodePtr;
    Den : DenPtr;
begin
    DenNode := First;
    whileDenNode <> nil do
        begin
            Den := DenNode^.Data;
            Den^.Draw(MinX,MinY);
            DenNode :=DenNode^.Next;
        end;
    end;
end;

procedureDenList.ClearInfection;
var
    DenNode :SNodePtr;
    Den : DenPtr;
begin
    DenNode := First;
    whileDenNode <> nil do
        begin
            Den := DenNode^.Data;
            Den^.DateLastTB :=BadDate;
            DenNode :=DenNode^.Next;
        end;
    end;
end;

(*****
*****      POSSUM      *****)
(*****

constructorPossum.Init(InitBirthday : Date;InitSex :SexType;InitDens :DenListPtr);
begin
    Birthday :=InitBirthday;
    Sex :=InitSex;
    if InitDens = nil thenDenMemory := NewDenListPtr,Init)
    else
        DenMemory :=InitDens;
    Den := nil;
    Joey := nil;
    Mother := nil;
    Pregnant := false;
    DateInfected :=BadDate;
    DateClinical :=BadDate;
    { Calculate maturity date}
    MaturityDate := Birthday + round(AgeMatureNormal(547,46,730,365));
    DateIndependent :=BadDate;
    NoDenList := NewSListPtr,Init);
    Immigrant := true;
end;

destructorPossum.Done;
var
    SN : SNodePtr;
    DP : ^Date;
begin
    if Joey <> nil then Dispose(Joey, Done);
    if Den <> nil thenLeaveDen;
    Dispose(DenMemory, Done);
    SN := NoDenList^.First;
    while SN <> nil do
        begin
            DP := SN^.Data;
            Dispose(DP);
            SN^.Data := nil;
            SN := SN^.Next;
        end;
    Dispose(NoDenList, Done);
end;

procedurePossum.Reproduce;

```

```

function DetermineSex :SexType;
const
  MaleProb = 0.5;
begin
  if BirthSexRandom(BirthSexSeed) < Model.MaleBirthProb then
    DetermineSex := Female
  else
    DetermineSex := Male;
  end;
  { DetermineSex }
begin
  if Pregnant then
    begin
      Joey := NewPossumPtr,Init(Time.CurrentDate,DetermineSex, nil);
      Joey^.DateIndependent := Joey^.Birthday + round(AgeIndepNormal(AgeIndepSeed1, AgeIndepSeed2,150,10,180,120));
      Joey^.Mother := Addr(Self);
      Pregnant := false;
    end;
  end; { Possum.Reproduce }

procedure Possum.Independence;
var
  MothersDen :SNodePtr;
begin
  if (DateIndependent <> BadDate) and (DateIndependent <= Time.CurrentDate) then
    begin
      { Give female joey den memory of mother }
      if Sex = Female then Den := Mother^.Den;
      if Model.UseDenMemory then
        begin
          MothersDen := Mother^.DenMemory^.First;
          while MothersDen <> nil do
            begin
              DenMemory^.AddDenPtr(MothersDen^.Data);
              MothersDen := MothersDen^.Next;
            end;
          end;
          { Remove Mother-Joey link }
          Mother^.Joey := nil;
          Mother := nil;
          DateIndependent := BadDate;
          if Live then Pop.AddPossum(Addr(Self));
        end;
      end;
    end;

function Possum.Live :boolean; { Returns false if possum dies }
var
  LiveTmp :boolean;
begin
  LiveTmp := True;
  if DateClinical = BadDate then
    begin
      {for non-clinical possums}
      if Mature then
        begin
          if (SurvAdultRandom(SurvAdultSeed) > Model.AdultSurvival[Time.Month, Sex]) then
            LiveTmp := False;
          end
        else
          begin
            if (SurvImmatureRandom(SurvImmatureSeed) > Model.ImmatureSurvival[Time.Month, Sex]) then
              LiveTmp := False;
            end;
          end
        end
      else
        begin
          if Mature then
            begin
              if (TBSurvAdultRandom(TBSurvAdultSeed) > Model.ClinicalAdultSurvival[Time.Month]) then
                LiveTmp := False;
              end
            else
              begin
                if (TBSurvImmRandom(TBSurvImmSeed) > Model.ClinicalImmatureSurvival[Time.Month]) then
                  LiveTmp := False;
                end
              end
            end
          end
        end
      end
    end;

```

```

        end;
    end;
    if (Joey <> nil) and ((noLiveTmp) or (JoeySurvRandom{JoeySurvSeed} > Model.JoeySurvival{Joey^.Agediv 30})) then
    begin
        Dispose(Joey, Done);
        Joey := nil;
    end;
    Live := LiveTmp;
end;

function Possum.Mature : boolean;
begin
    if MaturityDate <= Time.CurrentDate then
        Mature := true
    else
        Mature := false;
    end;
end;

function Possum.Age : integer;
begin
    Age := Time.CurrentDate - Birthday;
end;

function Possum.AgeCode : byte;
begin
    if Mother <> nil then
        AgeCode := 0
    else if Mature then
        AgeCode := 2
    else AgeCode := 1;
end;
end;

procedure Possum.Infect;
{ Infect a possum }
var
    AgeStatus : byte;
begin
    if (DateInfected = BadDate) then
    begin
        DateInfected := Time.CurrentDate;
        inc(Pop.NewInfected{AgeCode, Sex});
    end;
end;

function Possum.Infected : boolean;
begin
    if (DateInfected <> BadDate) then
        Infected := true
    else
        Infected := false;
    end;
end;

function Possum.Clinical : boolean;
begin
    if (DateClinical = BadDate) or (DateClinical > Time.CurrentDate) then
        Clinical := false
    else
        Clinical := true;
    end;
end;

procedure Possum.FindDen;
var
    Found, Found2 : boolean;
    Count : longint;
    DenNode : SNodePtr;
    OldDen : DenPtr;
    TotalTravel : integer;
    DensTried : DenList;
end;

function NodeInList(First, Node : SNodePtr) : boolean;
var
    Found : boolean;
    TriedNode : SNodePtr;
begin
    TriedNode := First;
    Found := false;

```

```

while (not Found) and (TriedNode <> nil) do
  begin
    if TriedNode^.Data = Node^.Data then Found := true;
    TriedNode := TriedNode^.Next;
  end;
  NodeInList := Found;
end;

procedure FindEmpty (FirstProspect : SNodePtr; var Den : DenPtr; var Found : boolean; var TotalTravel : integer; UseCloseDens : boolean);
var
  ClosestDen,           {Closest den to target}
  Target : DenPtr;     {Last den visited}
  Distance, ShortestDistance : integer;
  ProspectiveNode, TriedNode : SNodePtr;
  Finished : boolean;
  NewDate : ^Date;
begin
  Target := Den;
  if FirstProspect <> nil then Finished := false else Finished := true;
  Found := false;
  while (not Finished) do
    begin
      ClosestDen := nil;
      ShortestDistance := maxint;
      if UseCloseDens then
        ProspectiveNode := Target^.CloseDens^.First
      else
        ProspectiveNode := FirstProspect;
      while (ProspectiveNode <> nil) do
        begin
          Den := ProspectiveNode^.Data;
          { Not already tried, calculate distance}
          if (not NodeInList(DensTried.First, ProspectiveNode)) then
            begin
              Distance := round(sqrt(1.0 * (Target^.X - Den^.X) * (Target^.X - Den^.X) +
                (Target^.Y - Den^.Y) * (Target^.Y - Den^.Y)));
              if UseCloseDens then
                begin
                  ClosestDen := Den;
                  ShortestDistance := Distance;
                  ProspectiveNode := nil; end
                else
                  begin
                    if (Distance < ShortestDistance) then
                      begin
                        ShortestDistance := Distance;
                        ClosestDen := Den;
                      end;
                    ProspectiveNode := ProspectiveNode^.Next;
                  end;
                end;
            end;
          else
            begin
              ProspectiveNode := ProspectiveNode^.Next;
              { If end of close den list load in another one}
              if UseCloseDens and ProspectiveNode = nil then
                begin
                  ProspectiveNode := Target^.LoadNextDen;
                end
            end;
          end; { While }
        if ClosestDen = nil then
          begin
            Found := false;
            Finished := true;
          end
        else
          begin
            TotalTravel := TotalTravel + ShortestDistance;
            { Distance to next den is too far, possum didn't make it, therefore
              make possum use last den visited whether occupied or not }
            if TotalTravel > Model.MaxDenTravel then
              begin
                Found := true;
                Den := Target;
                Finished := true;
              end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

        { Add new date to no den list }
        New(NewDate);
        NewDate^ := Time.CurrentDate;
        NoDenList^.AddNode(New$NodePtr, Init(NewDate));
    end
else
begin
    DensTried.AddDen(ClosestDen);
    { Is den occupied }
    if (ClosestDen^.Empty)
    and (DenRejectRandomDenRejectSeed > Model.DenRejectProb) then
        begin
            Found := true;
            Den := ClosestDen;
            Immigrant := false;
            Finished := true;
        end
    else
        begin
            Target := ClosestDen;
            Finished := False;
        end;
    end;
end;
end;
end;
if found then Den^.Occupants^.AddPossum(Addr(Self));
end; { FindEmpty }
begin
    DensTried.Init;
    TotalTravel := 0;
    OldDen := Den;
    { Try last nights den first, or first den in memory if no den }
    if Den = nil then { No den last night }
    if DenMemory^.First = nil then
        begin
            { Find den at random from all dens }
            Den := Dens.RandomNode^.Data; end
    else
        begin
            { Find a den at random in the memory and then use the closest den to it }
            Den := DenMemory^.RandomNode^.Data;
        end;
    if (Den^.Empty) and DenRejectRandomDenRejectSeed > Model.DenRejectProb) then
        begin
            Found := true;
            Immigrant := false;
            Den^.Occupants^.AddPossum(Addr(Self));
            DenNode := New$NodePtr, Init(Den);
            if (DenMemory^.First = nil) or (noNodeInList(DenMemory^.FirstDenNode)) then
                DenMemory^.AddDen(Den);
            Dispose(DenNode, Done);
        end
    else
        begin
            DensTried.AddDen(Den);
            Found := false;
            if Model.UseDenMemory then
                begin
                    { Work through den memory }
                    DenNode := DenMemory^.First;
                    FindEmpty(DenNode, Den, FoundTotalTravel, false);
                end;
            if (not Found) then
                begin
                    { Starting from the last den tried work thru ALL dens using closest den list }
                    DenNode := Dens.First;
                    FindEmpty(DenNode, Den, FoundTotalTravel, true);
                    { Add den to list, if applicable }
                    DenNode := New$NodePtr, Init(Den);
                    if Found and Model.UseDenMemory or (noNodeInList(DenMemory^.FirstDenNode)) then
                        begin
                            DenMemory^.AddDen(Den);
                        end;
                    Dispose(DenNode, Done);
                end;
            end;
        end;
end;
end;
end;

```

```

    if not Found then Den := nil
    else
        if Clinical then
            Den^.DateLastTB :=Time.CurrentDate;
        DensTried.Done;
    end;

procedurePossum.LeaveDen;
var
    N, N2 : NodePtr;
begin
    if Den <> nil then
        begin
            N := Den^.Occupants^.First;
            while (N <> nil) do
                begin
                    if N^.Data = Addr(Self) then
                        begin
                            Den^.Occupants^.DeleteNode(N);
                            N := nil;
                        end
                    else
                        N := N^.Next;
                    end;
                end;
            end;
        end;
end;

(*****
*****      POSSUMLIST      *****
*****)

constructorPossumList.Init;
begin
    List.Init;
end;

destructorPossumList.Done;
begin
    List.Done;
end;

procedurePossumList.AddPossum(NewPossum :PossumPtr);
begin
    AddNode(NewNodePtr,Init(NewPossum));
end;

(*****
*****      POPULATION      *****
*****)

constructorPopulation.Init;
begin
    PossumList.Init;
    FillChar(AgesHealthy,sizeof(AgesHealthy), char(0));
    FillChar(AgesClinical,sizeof(AgesClinical), char(0));
    FillChar(NewInfected,sizeof(NewInfected), char(0));
    FillChar(NewClinical,sizeof(NewClinical), char(0));
    AveNilDens := 0;
    NoInAve := 0;
end;

destructor Population.Done;
begin
    while First <> nil do DeletePossum(First);
    PossumList.Done;
end;

procedure Population.Fill;
var
    F      : text;
    NodeP  : NodePtr;
    AgeYears : integer;
    AgeMonths: integer;
    Bday, MidYear : Date;
    Ch     : char;
    Sex   : SexType;

```

```

    Possum : PossumPtr;
    D      : integer;
begin
    MidYear := DMYToDate(1,6, Time.Year);
    if MidYear > Time.CurrentDate then MidYear := MidYear - 365;
    DateDiff(MidYear, Time.CurrentDate, D, AgeMonths, AgeYears);
    Assign(F, PopFile);
    Reset(F);
    while not eof(F) do
        begin
            readln(F, AgeYears, Ch, Ch);
            Bday := IncDateTrunc(Time.CurrentDate, -1*AgeMonths, -1*AgeYears);
            if Ch = 'F' then Sex := Female else Sex := Male;
            AddPossum(New(PossumPtr, Init(Bday, Sex, nil)));
        end;
    Close(F);
    { Infect possums }
    NodeP := First;
    while NodeP <> nil do
        begin
            Possum := NodeP^.Data;
            if (TBPrevRandom(TBPrevSeed) <= Model.InitInfectProb) then
                begin
                    Possum^.DateClinical := Time.CurrentDate;
                    Possum^.DateInfected := Time.CurrentDate;
                end;
            NodeP := NodeP^.Next;
        end;
    Move;
    UpdateAges;
end;

procedure Population.DeletePossum(N : NodePtr);
var
    DeadPossum : PossumPtr;
begin
    DeadPossum := N^.Data;
    Dispose(DeadPossum, Done);
    DeleteNode(N);
end; { Population.Delete }

procedure Population.Immigration;
{ Calculated daily }
var
    Age, i, Mean : integer;
    Sex : integer;
    Bday : Date;
    Possum : PossumPtr;
    NewPossums : integer;
begin
    for Sex := Female to Male do
        begin
            if Time.Day = 1 then
                begin
                    { Initialise monthly Poisson distribution }
                    Mean := Model.Immigration[Time.Month, Sex];
                    if Sex = 0 then DailyImmigrMale := mean / DaysInMonth(Time.Month, Time.Year);
                    if Sex = 1 then DailyImmigrFemale := mean / DaysInMonth(Time.Month, Time.Year);
                end;
                { Determine number of new possums }
                if Sex = 0 then
                    begin
                        NewPossums := round(DailyImmigrFemalePoisson(DailyImmigrFemale, DailyImmigrFemaleSeed));
                    end;
                if Sex = 1 then
                    begin
                        NewPossums := round(DailyImmigrMalePoisson(DailyImmigrMale, DailyImmigrMaleSeed));
                    end;
                { Set Birthday for new possums }
                Bday := IncDateTrunc(Time.CurrentDate, -1*(Time.Month+8), 0) + 15;
                for i := 1 to NewPossums do
                    begin
                        Possum := New(PossumPtr, Init(Bday, Sex, nil));
                        { Test whether clinical or not }
                        if (TBImmigrRandom(TBImmigrSeed) <= Model.ImmigrantTBProb[Time.Month]) then

```

```

        begin
            Possum^.DateClinical := Time.CurrentDate;
            Possum^.DateInfected := Time.CurrentDate;
        end;
        AddPossum(Possum);
    end;
end;

procedure Population.GoClinical;
{ Calculate the date which infected animals become clinical based on monthly probabilities, and distributed randomly through the month }
var
    Possum : PossumPtr;
    Node : NodePtr;
begin
    Node := First;
    while Node <> nil do
        begin
            Possum := Node^.Data;
            if (Possum^.Infected) and not(Possum^.Clinical) and
                (SubToClinRandom(SubToClinSeed) <= Model.GoClinicalProb[Time.Month]) then
                begin
                    Possum^.DateClinical := Time.CurrentDate + trunc(random * DaysInMonth(Time.Month, Time.Year));
                    inc(Pop.NewClinical[Possum^.AgeCode, Possum^.Sex]);
                end;
            Node := Node^.Next;
        end;
    end;
end;

procedure Population.Mating;
{ Mating is based on a buffer zone and a probability of mating. All males in the buffer zone around a female are tested to see if they mate successfully }
var
    FemaleNode,
    MaleNode : NodePtr;
    DNode : SNodePtr;
    NextDen : DenPtr;
    FemalePossum,
    MalePossum : PossumPtr;
begin
    FemaleNode := first;
    while FemaleNode <> nil do
        begin
            FemalePossum := FemaleNode^.Data;
            { Select only female possums that are not pregnant or have joey }
            if (FemalePossum^.Sex = Female) and (FemalePossum^.Joey = nil)
                and FemalePossum^.Mature and (not FemalePossum^.Pregnant) then
                begin
                    { Check each den closest to the female for a male possum }
                    DNode := FemalePossum^.Den^.CloseDens^.First;
                    NextDen := DNode^.Data;
                    while (FemalePossum^.Den^.Distance(NextDen^.X, NextDen^.Y) <= Model.MatingBuffer)
                        and (not FemalePossum^.Pregnant) do
                            begin
                                { Search through occupants for a male }
                                MaleNode := NextDen^.Occupants^.First;
                                while (MaleNode <> nil) do
                                    begin
                                        MalePossum := MaleNode^.Data;
                                        if (MalePossum^.Sex = Male) and
                                            (MatingRandom(MatingSeed) <= Model.MatingProb[Time.Month]) then
                                            begin
                                                FemalePossum^.Pregnant := true;
                                                if (FemalePossum^.Clinical) and
                                                    (TBMatingRandom(TBMatingSeed) <= Model.MatingTBProb) then
                                                    begin
                                                        MalePossum^.Infect;
                                                    end;
                                                if (MalePossum^.Clinical) and
                                                    (TBMatingRandom(TBMatingSeed) <= Model.MatingTBProb) then
                                                        FemalePossum^.Infect;
                                                    end;
                                                MaleNode := MaleNode^.Next;
                                            end;
                                    end;
                                DNode := DNode^.Next;
                                if DNode = nil then DNode := FemalePossum^.Den^.LoadNextDen;
                            end;
                end;
        end;
    end;
end;

```

```

        NextDen := DNode^.Data;
    end;
    end;
    FemaleNode := FemaleNode^.Next;
end; { Population.Mating }

function Population.Fecundity : real;
var
    PNode : NodePtr;
    Joeys,
    Total : integer;
begin
    Joeys := 0;
    Total := 0;
    PNode := First;
    while PNode <> nil do
    begin
        if PossumPtr(PNode^.Data)^.Sex = Female then
        begin
            if PossumPtr(PNode^.Data)^.Mature then inc(Total);
            if PossumPtr(PNode^.Data)^.Joey <> nil then inc(Joeys);
        end;
        PNode := PNode^.Next;
    end;
    if Total = 0 then Fecundity := 0 else Fecundity := Joeys/Total;
end;

procedure Population.TB;
{ Four mechanisms of TB infection are included here: a) Denning in a clinical den (clinical den), b) Denning in a buffer zone around an
clinical possum, c) Mother (clinical) to joey}
var
    N, PN : NodePtr;
    Possum, P : PossumPtr;
    Count, Offset : integer;
    DenNode : SNodePtr;
    Den : DenPtr;
begin
    N := First;
    while N <> nil do
    begin
        Possum := N^.Data;
        { Denning in infected den }
        if (not Possum^.Clinical) then
        begin
            if (Possum^.Den <> nil) and (Possum^.Den^.Clinical) and
                (TBDenRandom(TBDenSeed) <= Model.DenTBProb) then
                Possum^.Infect;
        end
        else
        begin
            { Mother to joey transfer }
            if (Possum^.Joey <> nil) then Possum^.Joey^.Infect;
            { Denning in buffer zone }
            if Possum^.Den <> nil then
                DenNode := Possum^.Den^.CloseDens^.First
            else
                DenNode := nil;
            while (DenNode <> nil) do
            begin
                Den := DenNode^.Data;
                { Is Den in buffer zone }
                if (Den^.Distance(Possum^.Den^.X, Possum^.Den^.Y) < Model.DenTBBuffer) then
                begin
                    { Infect all possums in this Den, even if current den }
                    PN := Den^.Occupants^.First;
                    while PN <> nil do
                    begin
                        P := PN^.Data;
                        if (TBBufferRandom(TBBufferSeed) <= Model.BufferTBProb) then P^.Infect;
                        PN := PN^.Next;
                    end; { while }
                    if (DenNode^.Next = nil) then
                    begin
                        DenNode := Possum^.Den^.LoadNextDen;
                    end
                end;
            end;
        end;
    end;
end;

```

```

        else
            DenNode := DenNode^.Next;
        end
    else
        DenNode := nil;
    end; {While}
end; {Clinical Possum}
N := N^.Next;
end;
end; {Population.TB}

procedure Population.UpdateAges;
var
    N : NodePtr;
    Age : integer;
    Sex : integer;
    Possum : PossumPtr;
begin
    for Age := 0 to MaxAge do
        for Sex := Female to Male do
            begin
                AgesHealthy[Age, Sex] := 0;
                AgesInfected[Age, Sex] := 0;
                AgesClinical[Age, Sex] := 0;
            end;
            N := First;
            while N <> nil do
                begin
                    Possum := N^.Data;
                    Age := Possum^.Age div 365;
                    if Age > MaxAge then Age := MaxAge;
                    if Possum^.Clinical then
                        inc(AgesClinical[Age, Possum^.Sex])
                    else if Possum^.Infected then
                        inc(AgesInfected[Age, Possum^.Sex])
                    else
                        inc(AgesHealthy[Age, Possum^.Sex]);
                    if Possum^.Joey <> nil then
                        begin
                            Age := 0;
                            Sex := Possum^.Joey^.Sex;
                            if Possum^.Joey^.Clinical then
                                inc(AgesClinical[Age, Sex])
                            else if Possum^.Joey^.Infected then
                                inc(AgesInfected[Age, Sex])
                            else
                                inc(AgesHealthy[Age, Sex]);
                        end;
                    N := N^.Next;
                end;
            end;
        end; {Population.UpdateAges}

procedure Population.UpdateAveNilDens(NewValue : integer);
begin
    inc(NoInAve);
    if NoInAve = 1 then AveNilDens := NewValue
    else AveNilDens := AveNilDens + (NewValue - AveNilDens)/NoInAve;
end;

procedure Population.Grow;
{ Perform life functions on each possum. Note that reproduction occurs before death and joeys are subject to death, ie each loop starts a new breeding season}
var
    N, N2 : NodePtr;
    Joey, Adult : PossumPtr;
    JoeyPop : Population; {Temporary object to hold newly independent joeys}
begin
    JoeyPop.Init;
    N := First;
    while N <> nil do
        begin
            Adult := N^.Data;
            if Adult^.Joey <> nil then
                Adult^.Joey^.Independence;
            Adult^.Reproduce;
            N2 := N^.Next;    {Copy N in case possum dies, but only after any new possums are added}
        end;
    end;
end;

```

```

        if not Adult^.Live then DeletePossum(N);
        N := N2;
    end;
    { Insert independent joeys into the population }
    while JoeyPop.First <> nil do
        begin
            Adult := JoeyPop.First^.Data;
            if Adult^.Live then AddPossum(Adult);
            JoeyPop.DeleteNode(JoeyPop.First);
        end;
    JoeyPop.Done;
    UpdateAges;
end; { Population.Grow }

procedure Population.Move;

function DensEmpty : boolean;
var
    DenNode : SNodePtr;
    found : boolean;
begin
    found := false;
    DenNode := Dens.First;
    while DenNode <> nil do
        begin
            if DenPtr(DenNode^.Data)^.Occupants^.Length <> 0 then
                found := true;
                DenNode := DenNode^.Next;
            end;
        if found then DensEmpty := false else DensEmpty := true;
    end; { Dens Empty }
var
    N : NodePtr;
    Possum : PossumPtr;
    Count, Offset : integer;
begin
    N := First;
    while N <> nil do
        begin
            Possum := N^.Data;
            Possum^.LeaveDen;
            N := N^.Next;
        end;
    { Reallocate dens, start with random possum }
    Offset := Random(Length);
    N := First;
    while Offset > 0 do
        begin
            N := N^.Next;
            Offset := Offset - 1;
        end;
    Count := 0;
    repeat
        if N = nil then N := First;
        Possum := N^.Data;
        Possum^.FindDen;
        N := N^.Next;
        inc(Count);
    until Count = Length;
end;

procedure Population.DensityDynamics;
{ This procedure simulates the emmigration and/or death of possums due to competition for den site }
var
    Possum : PossumPtr;
    PNode,
    PNodeNext : NodePtr;
    FirstDate : ^Date;
    PossumGoes : boolean;
    Count : integer;
begin
    PNode := First;
    Count := 0;
    while PNode <> nil do
        begin

```

```

PossumGoes := false;
Possum := PNode^.Data;
PNodeNext := PNode^.Next;
if Possum^.NoDenList^.First <> nil then
  begin
    { Test dying or naffing off due to lack of dens}
    if (Possum^.NoDenList^.Length > Model.ResidentDenThreshold) and
      (PopDensEmigrRandom(PopDensEmigrSeed) <=
      Model.NoDenMortality[Time.Month]/DaysInMonth(Time.Month,Time.Year)) then
      begin
        PossumGoes := true;
      end
    else
      begin
        FirstDate := Possum^.NoDenList^.First^.Data;
        if (Time.CurrentDate - FirstDate^) = Model.ResidentDenWindow then
          begin
            Dispose(FirstDate);
            Possum^.NoDenList^.DeleteNode(Possum^.NoDenList^.First);
          end;
        end;
      end;
    end;
  if Possum^.Immigrant then
    begin
      if Possum^.NoDenList^.Length > Model.ImmigrantDenThreshold then
        begin
          PossumGoes := true;
        end;
      end;
    if PossumGoes then
      begin
        DeletePossum(PNode);
      end;
    PNode := PNodeNext;
  end; { While }
end; {Population.DensityDynamics}

function Population.Size : integer;
var
  Age, Sex, i, tmp : integer;
begin
  tmp := 0;
  for Age := 0 to MaxAge do
    for Sex := Female to Male do
      tmp := tmp + AgesHealthy[Age, Sex] + AgesInfected[Age,Sex] + AgesClinical[Age, Sex];
    Size := tmp;
  end ;

function Population.NoMales(Status : byte) : integer;
var
  Age, Sex, tmp : integer;
begin
  tmp := 0;
  for Age := 0 to MaxAge do
    case Status of
      pHealthy : tmp := tmp + AgesHealthy[Age, Male];
      pInfected : tmp := tmp + AgesInfected[Age, Male];
      pClinical : tmp := tmp + AgesClinical[Age, Male];
    end;
  NoMales := tmp;
end;

function Population.NoFemales(Status : byte) : integer;
var
  Age, Sex, tmp : integer;
begin
  tmp := 0;
  for Age := 0 to MaxAge do
    case Status of
      pHealthy : tmp := tmp + AgesHealthy[Age, Female];
      pInfected : tmp := tmp + AgesInfected[Age, Female];
      pClinical : tmp := tmp + AgesClinical[Age, Female];
    end;
  NoFemales := tmp;
end;

```

```

function Population.NilDens : integer;
var
  tmp : integer;
  Node : NodePtr;
  Possum : PossumPtr;
begin
  Tmp := 0;
  Node := First;
  while Node <> nil do
    begin
      Possum := Node^.Data;
      if Possum^.Den = nil then inc(Tmp);
      Node := Node^.Next;
    end;
  NilDens := Tmp;
end;

procedure Population.PrintAges;
var
  Age, Sex : integer;
begin
  writeln('      ', '0':5, '1':5, '2':5, '3':5, '4':5, '5':5);
  writeln;
  for Sex := Female to Male do
    begin
      if Sex = Female then
        write('Female':10)
      else
        write('Male':10);
    end;
  for Age := 0 to 5 do
    write(AgesHealthy[Age, Sex]:5);
  writeln;
end;
readln;
end;

function Population.AgeSummary : string;
const
  asJoey      = 0;
  asImmature  = 1;
  asMature    = 2;
  asClear     = 0;
  asInfected  = 1;
  asClinical  = 2;
  asNewInfected = 3;
  asNewClinical = 4;
var
  Data : array [Female..Male, asJoey..asMature, asClear..asNewClinical] of integer;
  Node : NodePtr;
  Possum : PossumPtr;
  Age, Status, Sex : integer;
  Result : string;
begin
  Node := First;
  FillChar(Data, sizeof(Data), char(0));
  while Node <> nil do
    begin
      Possum := Node^.Data;
      if Possum^.Mature then Age := asMature else Age := asImmature;
      if Possum^.Clinical then Status := asClinical
      else if Possum^.Infected then Status := asInfected
      else Status := asClear;
      inc(Data[Possum^.Sex, Age, Status]);
      if Possum^.Joey <> nil then
        begin
          if Possum^.Joey^.Clinical then Status := asClinical
          else if Possum^.Joey^.Infected then Status := asInfected
          else Status := asClear;
          inc(Data[Possum^.Joey^.Sex, asJoey, Status]);
        end;
      Node := Node^.Next;
    end;
  Result := "";
  for Status := asClear to asClinical do
    for Age := asMature downto asJoey do
      for Sex := Mde downto Female do

```

```

begin
    Result := Result + Long2Str(Data[Sex, Age, Status]) + ',';
    if Status = asInfected then
        Result := Result + Long2Str(NewInfected[Age, Sex]) + ','
    else
        if Status = asClinical then
            Result := Result + Long2Str(NewClinical[Age, Sex]) + ',';
        end;
    Result[0] := chr(ord(Result[0]) - 1);
    AgeSummary := Result;
    FillChar(NewInfected, sizeof(NewInfected), char(0));
    FillChar(NewClinical, sizeof(NewClinical), char(0));
end;

procedure Population.PrintList; { Print population to screen }
var
    N : NodePtr;
    Possum : PossumPtr;
begin
    N := First;
    while N <> nil do
        begin
            Possum := N^.Data;
            write(Possum^.Age:2, Possum^.Sex);
            if Possum^.Joey <> nil then
                write(Possum^.Joey^.Age:2, Possum^.Joey^.Sex, ' ')
            else
                write(' ');
            N := N^.Next;
        end;
        writeln;
    end;

procedure Population.PrintDenSummary;
{ Summary of the number of number of possum with den memories of a certain size }
const
    DenLimit = 14;
var
    DenPrintSummary : array [0..DenLimit] of integer;
    N : NodePtr;
    P : PossumPtr;
    i : integer;
    Sum : longint;
begin
    Sum := 0;
    for i := 0 to DenLimit do DenPrintSummary[i] := 0;
    N := First;
    while N <> nil do
        begin
            P := N^.Data;
            i := P^.DenMemory^.Length;
            Sum := Sum + i;
            if i > DenLimit then i := DenLimit;
            inc(DenPrintSummary[i]);
            N := N^.Next;
        end;
    for i := 0 to DenLimit do
        write(DenPrintSummary[i]:4);
    writeln(Sum/Length:6:2);
end;

function Population.DenSummary : string;
{ Summary of the number of number of possum with den memories of a certain size written to file }
const
    DenLimit = 14;
var
    DenFileSummary : array [0..DenLimit] of integer;
    N : NodePtr;
    P : PossumPtr;
    i : integer;
    Sum : longint;
    DenSummaryString : string;
begin
    DenSummaryString := '';
    Sum := 0;
    for i := 0 to DenLimit do DenFileSummary[i] := 0;

```

```

N := First;
while N <> nil do
  begin
    P := N^.Data;
    i := P^.DenMemory^.Length;
    Sum := Sum + i;
    if i > DenLimit then i := DenLimit;
    inc(DenFileSummary[i]);
    N := N^.Next;
  end;
for i := 0 to DenLimit do
  begin
    DenSummaryString := DenSummaryString + Long2Str(DenFileSummary[i]) + ',';
  end;
DenSummary := DenSummaryString;
end;

procedure Population.PrintNonDens;
{ Summary of the frequency of non denning possums }
const
  Limit = 14;
var
  Summary : array [0..Limit] of integer;
  N : NodePtr;
  P : PossumPtr;
  i : integer;
  Sum : longint;
begin
  Sum := 0;
  for i := 0 to Limit do Summary[i] := 0;
  N := First;
  while N <> nil do
    begin
      P := N^.Data;
      i := P^.NoDenList^.Length;
      Sum := Sum + i;
      if i > Limit then i := Limit;
      inc(Summary[i]);
      N := N^.Next;
    end;
  for i := 0 to Limit do
    write(Summary[i]:4);
    writeln(Sum/Self.Length:6:2);
  end;

function Population.NonDenSummary : string;
{ Summary of the frequency of non denning possums during a month }
const
  Limit = 14;
var
  NoDSummary : array [0..Limit] of integer;
  N : NodePtr;
  P : PossumPtr;
  i : integer;
  Sum : longint;
  NonDenSummaryString : string;
begin
  NonDenSummaryString := '';
  Sum := 0;
  for i := 0 to Limit do NoDSummary[i] := 0;
  N := First;
  while N <> nil do
    begin
      P := N^.Data;
      i := P^.NoDenList^.Length;
      Sum := Sum + i;
      if i > Limit then i := Limit;
      inc(NoDSummary[i]);
      N := N^.Next;
    end;
  for i := 0 to Limit do
    begin
      NonDenSummaryString := NonDenSummaryString + Long2Str(NoDSummary[i]) + ',';
    end;
  NonDenSummary := NonDenSummaryString;
end;

```

```

procedure UpdateSummary (S : SummaryPtr; P : Population);
var
  Age, Sex : integer;
  X : longint;
begin
  for Age := 0 to MaxAge do
    for Sex := Female to Male do
      begin
        X := P.AgesHealthy[Age,Sex];
        S^.Sum[Age,Sex] := S^.Sum[Age,Sex] + X;
        S^.Sumsq[Age,Sex] := S^.Sumsq[Age,Sex] + X*X;
      end;
    end;
  end;

procedure TMyApplication.InitMainWindow;
begin
  MainWindow := New(PMyWindow, Init(nil, 'Welcome to PossPop'));
end;

procedure TMyWindow.About(var Msg: TMessage);
var
  AboutWnd: TDialog;
begin
  AboutWnd.Init(@Self, 'About');
  AboutWnd.Execute;
  AboutWnd.Done;
end;

procedure TMyWindow.RunModel(var Msg: TMessage);{procedure to run model in window}
var
  choice : integer;
  RunQuit : string;
begin
  StrCopy(WindowTitle,'PossPop');
  InitWinCrt;
  writeln(' Welcome to the Computer Simulation Model');
  writeln('      "PossPop"');
  writeln;
  writeln('developed by D.U.Pfeiffer, M.Stern and R.S.Morris');
  writeln('      programmed by M.Stern');
  writeln(' Dept. Vet. Clin. Sci., Massey University');
  writeln('      Palmerston North, New Zealand');
  writeln;
  writeln;
  Model.Init;
  writeln(MemAvail);
  Dens.Init;
  Dens.SetFromFile(DenMapFile);
  writeln(MemAvail);
  writeln(Dens.Length,' dens loaded. '); {press return to continue};readln;
  Dens.LoadGeography;
  repeat
    write('Enter R to run model, D to calculate den distances or Q to exit (R=default): ');
    readln(RunQuit);
    RunQuit := UpCase(RunQuit[1]);
    Choice := 0;
    if RunQuit = 'R' then Choice := 0;
    if RunQuit = 'D' then Choice := 7;
    if RunQuit = 'Q' then Choice := 2;
    case Choice of
      0 : Model.Go;
      7 : Dens.CalculateGeography;
      2 : DoneWinCrt;
    end; {Case}
  until Choice = 2;
end;

procedure TMyWindow.MakeFont;
var
  MyLogFont : TLogFont;
begin
  with MyLogFont do
    begin
      IfHeight := 30;
      IfWidth := 0;
    end;
  end;
end;

```

```

        IfEscapement := 0;
        IfOrientation := 0;
        IfWeight := fw_Bold;
        IfItalic := 0;
        IfUnderline := 0;
        IfStrikeOut := 0;
        IfCharSet := ANSI_CharSet;
        IfOutPrecision := Out_Default_Precis;
        IfClipPrecision := Clip_Default_Precis;
        IfQuality := Default_Quality;
        IfPitchAndFamily := Variable_Pitch or ff_Swiss;
        StrCopy(@IfFaceName, 'Helv');
    end;
    MainFontRec := MyLogFont;
end;

procedure TMyWindow.Paint(PaintDC : HDC; var PaintInfo: TPaintStruct);
var
    DC, MemDC: HDC;
    OldBitMap, BitMap: HBitmap;
    BM: TBitmap;
    TextString1, TextString2, TextString3 : array[0..20] of Char;
    MyBitMap : HBitmap;
begin
    BitMap := LoadBitmap(HInstance, 'Possum2');
    MemDC := CreateCompatibleDC(PaintDC);
    OldBitMap := SelectObject(MemDC, BitMap);
    GetObject(BitMap, SizeOf(BM), @BM);
    with SplashRect do
        begin
            Left := 50;
            Top := 70;
            Right := Left + BM.bmWidth;
            Bottom := Top + BM.bmHeight;
            BitBlt(PaintDC, Left, Top, BM.bmWidth, BM.bmHeight, MemDC, 0, 0, SRCCopy);
        end;
    DeleteObject(SelectObject(MemDC, OldBitMap));
    DeleteDC(MemDC);
    DeleteDC(PaintDC);
    StrCopy(TextString1, 'PosPop');
    TextOut(PaintDC, 200, 10, TextString1, StrLen(TextString1));
    StrCopy(TextString2, 'a Computer Simulation Model');
    TextOut(PaintDC, 125, 30, TextString2, StrLen(TextString2));
    StrCopy(TextString3, 'of Bovine Tuberculosis Infection in Feral Possum Populations');
    TextOut(PaintDC, 25, 50, TextString3, StrLen(TextString3));
end;

constructor TMyWindow.Init(AParent: PWindowsObject; ATitle: PChar);
begin
    TWindow.Init(AParent, ATitle);
    with attr do
        begin
            w:=550;{ Force window size }
            h:=650;
        end;
    end;
end;

procedure TMYWindow.GetWindowClass(var WndClass: TWndClass);
begin
    TWindow.GetWindowClass(WndClass);
    WndClass.hIcon := LoadIcon(HInstance, 'PosIco');
    WndClass.Style := CS_DBLCLKS; { Respond to double click }
    WndClass.lpszMenuName := 'MainMenu';
end;

(*****
*****      MAIN      *****
*****)

var
    MyApp: TMyApplication;
begin
    MyApp.Init('Posspo10');
    MyApp.Run;
    MyApp.Done;
    InvalidateRect(0, @SplashRect, True);

```

end.