Apidologie 39 (2008) 637–649

Online Material

APPENDIX
Paul Williams
Programs written for Dev-C++ version 4.9.9.2
from Bloodshed Software http://www.bloodshed.net

```
//================================================================================
//(1) COLTAIL4.c
//by Paul Williams, the Natural History Museum, London
//version 15.xi.2007
//for Dev-C++ 4.9.9.2
//
//Randomization test for Psithyrus and host data:
//At a coarse scale, is there an association between parasites and hosts
//in their tail colour in the sample of records?

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define       RECORDSEU       24        // number of European parasitism records
#define       HOSTSEU         18        // number of unique European hosts
#define       RECORDSNA       11        // number of N American parasitism records
#define       HOSTSNA         11        // number of unique N American hosts
#define       SIMULATIONS     9999

int main(int argc, char *argv[])
{
int           i,j,k,
              choiceEU[RECORDSEU],choiceNA[RECORDSNA],result[SIMULATIONS+1],
              orderEU[RECORDSEU],orderNA[RECORDSNA],
              randn,randmax,
              mark,gap,tempv,step;
FILE          *fp_out;

//data
char parasite_groups_EU[RECORDSEU][5]= {
    "333D",
    "333D",
    "332x",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "332x",
    "300x",
    "332x",
    "100x",
    "332x",
    "132x",
    "100x",
    "100x",
    "132x",
    "133x",
    "100x",
    "133x",
    "100x",
    "332x",
    "332x",
    "332x"},
    parasite_groups_NA[RECORDSNA][5]= {
    "332x",
    "332x",
    "034x",
```

```
    "334D",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "033x"},
    host_groups_EU[HOSTSEU][5]= { //excluding non-unique hosts as comments
    "333D",
    "324B",
    "333D",
    "324Y",
    "124D",
    "100x",
    "132x",
    "333D",
    "333D",
    "300x",
//    "324B",
    "100x",
    "333D",
//    "333D",
    "100x",
    "100x",
//    "100x",
    "133x",
//    "124D",
    "133x",
    "133x",
//    "333D",
//    "132x",
    "333D"},
    host_groups_NA[HOSTSNA][5]= {
    "033x",
    "033x",
    "034x",
    "033x",
    "343L",
    "332x",
    "113x",
    "033x",
    "233x",
    "332x",
    "033x"};

//open files
if ((fp_out=fopen("simresults.txt","w+")) == NULL)//clear contents of file
       {
    printf("file opening error\n");
       return EXIT_FAILURE;
       }

//simulation
srand((unsigned)time(NULL));                          // seed start of number table
randn= (int)(RAND_MAX/HOSTSEU);
randmax= HOSTSEU*randn;                                // max acceptable score
for (i=1; i<=SIMULATIONS; i++)                        // sim 1->9999
    {
    result[i]= 0;

    //Europe
    //choose offsets into HOSTS codes
    for (j=0; j<RECORDSEU; j++)// for each sim choice with replacement
        {
             do  {
                    randn= rand();
                    } while (randn>randmax);
```

```
        choiceEU[j]= (randn%HOSTSEU);      // offsets 0 -> N-1
        }

    for (j=0; j<RECORDSEU; j++)
        {
        if (strncmp(parasite_groups_EU[j],host_groups_EU[choiceEU[j]],1)==0)
            ++result[i];
        }
       }

//extract results
gap= SIMULATIONS/2;                                  //Shell binary sort
while (gap > 0)
       {
       for (i=1; i<=SIMULATIONS-gap; i++)
             {
             for (j=i; j>=1; j--)
                   {
                   mark= j+gap;
                   if (result[j] > result[mark])    //sort to increasing values
                         {
                         tempv= result[j];
                         result[j]= result[mark];
                         result[mark]= tempv;
                         }
                   else
                         j= 0;
                   }
             }
       gap /= 2;
       }

//write results
fprintf(fp_out,"\n\n Results Europe ####\n\n");
step= SIMULATIONS/1000;
fprintf(fp_out," percentage points:\n");
for (i=1; i<=SIMULATIONS; i++)                       // sim 1->100...
    {
    if ((i%(100))==0)
        {
        if (i==500)
            {
            j= 1;
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
             fprintf(fp_out,".....................................\n");
            }
        else if (i==9500)
            {
             fprintf(fp_out,".....................................\n");
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
            }
        else
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
        }
    }
fprintf(fp_out,"\n upper tail threshold:\n");
for (i=1; i<=SIMULATIONS; i++)                       // sim 1->100...
    {
    if (i>9400 && i<=9999)
        {
       fprintf(fp_out,"%5d (%6.4f):   %4d\n",i,1-((double)i/SIMULATIONS),result[i]);
        }
    }

randn= (int)(RAND_MAX/HOSTSNA);
```

```
randmax= HOSTSNA*randn;                                  // max acceptable score
for (i=1; i<=SIMULATIONS; i++)                          // sim 1->9999
    {
    result[i]= 0;

    //N America
    for (j=0; j<RECORDSNA; j++)// for each sim cell
        {
             do  {
                   randn= rand();
                   } while (randn>randmax);
        choiceNA[j]= (randn%HOSTSNA);       // offsets 0 -> N-1
        }

    for (j=0; j<RECORDSNA; j++)
        {
        if (strncmp(parasite_groups_NA[j],host_groups_NA[choiceNA[j]],1)==0)
            ++result[i];
        }
       }

//extract results
gap= SIMULATIONS/2;                                      //Shell binary sort
while (gap > 0)
       {
       for (i=1; i<=SIMULATIONS-gap; i++)
             {
             for (j=i; j>=1; j--)
                   {
                   mark= j+gap;
                   if (result[j] > result[mark])    //sort to increasing values
                         {
                         tempv= result[j];
                         result[j]= result[mark];
                         result[mark]= tempv;
                         }
                   else
                         j= 0;
                   }
             }
       gap /= 2;
       }

//write results
fprintf(fp_out,"\n\n Results N America ####\n\n");
step= SIMULATIONS/1000;
fprintf(fp_out," percentage points:\n");
for (i=1; i<=SIMULATIONS; i++)                          // sim 1->100...
    {
    if ((i%(100))==0)
        {
        if (i==500)
            {
            j= 1;
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
             fprintf(fp_out,".....................................\n");
            }
        else if (i==9500)
            {
             fprintf(fp_out,".....................................\n");
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
            }
        else
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
        }
```

```
    }
fprintf(fp_out,"\n upper tail threshold:\n");
for (i=1; i<=SIMULATIONS; i++)                        // sim 1->100...
    {
    if (i>9400 && i<=9999)
        {
       fprintf(fp_out,"%5d (%6.4f):   %4d\n",i,1-((double)i/SIMULATIONS),result[i]);
        }
    }

fclose(fp_out);
return EXIT_SUCCESS;
}


//=============================================================================
//(2) COLGROUPS4.c
//by Paul Williams, the Natural History Museum, London
//version 15.xi.2007
//for Dev-C++ 4.9.9.2
//
//Randomization test for Psithyrus and host data:
//At an intermediate scale, is there an association between parasites and hosts
//in their colour-pattern groups (from Williams, 2007) in the sample of records?

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define      RECORDSEU      24       // number of European parasitism records
#define      HOSTSEU        18       // number of unique European hosts
#define      RECORDSNA      11       // number of N American parasitism records
#define      HOSTSNA        11       // number of unique N American hosts
#define      SIMULATIONS    9999

int main(int argc, char *argv[])
{
int          i,j,k,
             choiceEU[RECORDSEU],choiceNA[RECORDSNA],result[SIMULATIONS+1],
             orderEU[RECORDSEU],orderNA[RECORDSNA],
             randn,randmax,
             mark,gap,tempv,step;
FILE         *fp_out;

//data
char parasite_groups_EU[RECORDSEU][5]= {
    "333D",
    "333D",
    "332x",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "332x",
    "300x",
    "332x",
    "100x",
    "332x",
    "132x",
    "100x",
    "100x",
    "132x",
    "133x",
    "100x",
    "133x",
```

```
    "100x",
    "332x",
    "332x",
    "332x"},
    parasite_groups_NA[RECORDSNA][5]= {
    "332x",
    "332x",
    "034x",
    "334D",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "333D",
    "033x"},
    host_groups_EU[HOSTSEU][5]= { //excluding non-unique hosts as comments
    "333D",
    "324B",
    "333D",
    "324Y",
    "124D",
    "100x",
    "132x",
    "333D",
    "333D",
    "300x",
//    "324B",
    "100x",
    "333D",
//    "333D",
    "100x",
    "100x",
//    "100x",
    "133x",
//    "124D",
    "133x",
    "133x",
//    "333D",
//    "132x",
    "333D"},
    host_groups_NA[HOSTSNA][5]= {
    "033x",
    "033x",
    "034x",
    "033x",
    "343L",
    "332x",
    "113x",
    "033x",
    "233x",
    "332x",
    "033x"};

//open files
if ((fp_out=fopen("simresults.txt","w+")) == NULL)//clear contents of file
       {
    printf("file opening error\n");
       return EXIT_FAILURE;
       }

//simulation
srand((unsigned)time(NULL));                        // seed start of number table
randn= (int)(RAND_MAX/HOSTSEU);
randmax= HOSTSEU*randn;                              // max acceptable score
for (i=1; i<=SIMULATIONS; i++)                      // sim 1->9999
    {
    result[i]= 0;
```

```
    //Europe
    //choose offsets into HOSTS codes
    for (j=0; j<RECORDSEU; j++)// for each sim choice with replacement
        {
             do  {
                   randn= rand();
                   } while (randn>randmax);
        choiceEU[j]= (randn%HOSTSEU);       // offsets 0 -> N-1
        }

    for (j=0; j<RECORDSEU; j++)
        {
        if (strncmp(parasite_groups_EU[j],host_groups_EU[choiceEU[j]],4)==0)
            ++result[i];
        }
    }

//extract results
gap= SIMULATIONS/2;                                    //Shell binary sort
while (gap > 0)
      {
      for (i=1; i<=SIMULATIONS-gap; i++)
            {
            for (j=i; j>=1; j--)
                  {
                  mark= j+gap;
                  if (result[j] > result[mark])    //sort to increasing values
                        {
                        tempv= result[j];
                        result[j]= result[mark];
                        result[mark]= tempv;
                        }
                  else
                        j= 0;
                  }
            }
      gap /= 2;
      }

//write results
fprintf(fp_out,"\n\n Results Europe ####\n\n");
step= SIMULATIONS/1000;
fprintf(fp_out," percentage points:\n");
for (i=1; i<=SIMULATIONS; i++)                         // sim 1->100...
    {
    if ((i%(100))==0)
        {
        if (i==500)
            {
            j= 1;
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
             fprintf(fp_out,".....................................\n");
            }
        else if (i==9500)
            {
             fprintf(fp_out,".....................................\n");
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
            }
        else
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
        }
    }
fprintf(fp_out,"\n upper tail threshold:\n");
for (i=1; i<=SIMULATIONS; i++)                         // sim 1->100...
```

```
    {
    if (i>9400 && i<=9999)
        {
       fprintf(fp_out,"%5d (%6.4f):   %4d\n",i,1-((double)i/SIMULATIONS),result[i]);
        }
    }

//simulation
randn= (int)(RAND_MAX/HOSTSNA);
randmax= HOSTSNA*randn;                                  // max acceptable score
for (i=1; i<=SIMULATIONS; i++)                          // sim 1->9999
    {
    result[i]= 0;

    //N America
    for (j=0; j<RECORDSNA; j++)// for each sim cell
        {
             do  {
                   randn= rand();
                   } while (randn>randmax);
        choiceNA[j]= (randn%HOSTSNA);       // offsets 0 -> N-1
        }

    for (j=0; j<RECORDSNA; j++)
        {
        if (strncmp(parasite_groups_NA[j],host_groups_NA[choiceNA[j]],4)==0)
            ++result[i];
        }
       }

//extract results
gap= SIMULATIONS/2;                                      //Shell binary sort
while (gap > 0)
       {
       for (i=1; i<=SIMULATIONS-gap; i++)
             {
             for (j=i; j>=1; j--)
                   {
                   mark= j+gap;
                   if (result[j] > result[mark])    //sort to increasing values
                         {
                         tempv= result[j];
                         result[j]= result[mark];
                         result[mark]= tempv;
                         }
                   else
                         j= 0;
                   }
             }
       gap /= 2;
       }

//write results
fprintf(fp_out,"\n\n Results N America ####\n\n");
step= SIMULATIONS/1000;
fprintf(fp_out," percentage points:\n");
for (i=1; i<=SIMULATIONS; i++)                          // sim 1->100...
    {
    if ((i%(100))==0)
        {
        if (i==500)
            {
            j= 1;
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
             fprintf(fp_out,".....................................\n");
            }
        else if (i==9500)
```

```
            {
             fprintf(fp_out,".....................................\n");
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
            }
        else
             fprintf(fp_out,"%5d (%6.2f):     %4d\n",
                i,((double)i/SIMULATIONS),result[i]);
        }
    }
fprintf(fp_out,"\n upper tail threshold:\n");
for (i=1; i<=SIMULATIONS; i++)                        // sim 1->100...
    {
    if (i>9400 && i<=9999)
        {
       fprintf(fp_out,"%5d (%6.4f):   %4d\n",i,1-((double)i/SIMULATIONS),result[i]);
        }
    }

fclose(fp_out);
return EXIT_SUCCESS;
}


//============================================================================
//(3) COLELEM4.c
//by Paul Williams, the Natural History Museum, London
//version 15.xi.2007
//for Dev-C++ 4.9.9.2
//
//Randomization test for Psithyrus and host data:
//At a fine scale, is there an association between parasites and hosts
//in their colour-pattern elements (from Williams, 2007) in the sample of records?

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

//26 elements scored in total, 24 numbered in BJLS Fig. 1, 23 analysed in BJLS
#define      ELEMS          23       // number of colour-pattern elements
#define      RECORDSEU      24       // number of European parasitism records
#define      HOSTSEU        18       // number of unique European hosts
#define      RECORDSNA      11       // number of N American parasitism records
#define      HOSTSNA        11       // number of unique N American hosts
#define      SIMULATIONS    9999

int main(int argc, char *argv[])
{
int          i,j,k,m,
             choiceEU[RECORDSEU+1],choiceNA[RECORDSNA+1],result[SIMULATIONS+1],
             orderEU[RECORDSEU+1],orderNA[RECORDSNA+1],
             randn,randmax,
             observedEU,observedNA,mark,gap,tempv,step,
             parasitesEU[RECORDSEU][ELEMS]= {//parasite-record colour-pattern elements
           //0 X                1                   2
           //1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
             5,  0,0,5,5,0,0,0,0,0,0,0,6,6,6,6,6,6,6,6,6,6,  //barbutellus
             5,  0,0,5,5,0,0,0,0,0,0,0,6,6,6,6,6,6,6,6,6,6,  //barbutellus
             5,  0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,0,6,0,6,  //bohemicus
             5,  0,0,5,5,0,0,0,0,0,0,0,5,0,5,0,5,0,5,0,5,5,  //campestris
             5,  0,0,5,5,0,0,0,0,0,0,0,5,0,5,0,5,0,5,0,5,5,  //campestris
             5,  0,0,5,5,0,0,0,0,0,0,0,5,0,5,0,5,0,5,0,5,5,  //campestris
             5,  0,0,5,5,0,0,0,0,0,0,0,5,0,5,0,5,0,5,0,5,5,  //campestris
             5,  0,0,5,5,0,0,0,0,0,0,0,0,5,5,5,5,0,0,0,0,0,  //flavidus
             5,  0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,0,5,0,5,5,  //maxillosus
             0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,  //maxillosus
```

```
  5,  0,0,0,0,0,0,0,0,0,0,6,0,6,6,6,6,6,0,0,0,0,  //norvegicus
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,5,5,5,5,6,6,  //perezi
  5,  0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,4,4,4,  //quadricolor
  5,  0,0,0,0,0,0,0,0,0,0,0,4,4,4,5,5,5,5,5,4,4,  //quadricolor
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,  //quadricolor d
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,  //rupestris
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,  //rupestris p
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,  //rupestris p
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,  //rupestris
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,  //rupestris p
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,  //rupestris
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,0,0,0,  //sylvestris
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,0,0,0,  //sylvestris
  5,  0,0,0,0,0,0,0,0,0,0,0,5,5,5,6,6,6,6,0,6,0,0}, //vestalis

  parasitesNA[RECORDSNA][ELEMS]= {//parasite-record colour-pattern elements
//0 X              1                   2
//1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
  5,  0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,0,6,0,0,  //ashtoni
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,0,6,0,0,  //ashtoni
  5,  5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  //citrinus lab
  5,  5,5,5,5,0,0,0,0,0,0,5,5,5,5,0,0,0,0,0,0,0,0,  //citrinus
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,5,  //insularis
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,5,  //insularis
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,5,  //insularis
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,5,  //insularis
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,5,  //insularis
  5,  0,0,5,5,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,5,0,0,  //suckleyi
  5,  0,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //variabilis

  allhostsEU[RECORDSEU][ELEMS]= {//host-record colour-pattern elements
//0 X              1                   2
//1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
  5,  0,0,5,5,5,5,5,0,0,0,0,0,6,6,6,6,6,6,6,6,6,6,  //hortorum
  4,  4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //hypnorum
  5,  0,0,0,0,0,0,5,5,0,0,0,0,0,0,0,6,6,6,6,6,6,  //lucorum
  4,  4,4,4,4,5,5,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,  //humilis
  4,  4,4,4,4,5,5,4,0,4,4,0,0,0,0,4,0,4,4,4,4,4,  //pascuorum
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3,  //pomorum
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,  //pratorum
  5,  0,0,5,5,5,5,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,  //jonellus
  5,  0,0,5,5,5,5,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,  //argillaceus
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,5,5,5,  //ruderatus cors
  4,  4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //hypnorum
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,  //terrestris xanth
  5,  0,0,0,0,0,5,5,5,0,0,0,0,0,0,0,6,6,6,6,6,6,  //soroeensis
  5,  0,0,0,0,0,5,5,5,0,0,0,0,0,0,0,0,0,6,6,6,6,6,  //soroeensis
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //soroeensis prot
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius
  5,  0,0,5,5,5,5,5,5,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius dec
  4,  4,4,4,4,5,5,4,0,4,4,0,0,0,0,4,0,4,4,4,4,4,4,  //pascuorum
  5,  0,0,5,5,5,5,5,5,0,0,0,0,0,0,0,4,4,4,4,4,4,4,  //sichelii
  5,  0,0,5,5,5,5,5,5,5,5,5,0,0,5,5,4,4,4,4,4,4,4,  //sylvarum
  5,  0,0,5,5,5,5,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,  //jonellus
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,  //pratorum
  5,  0,0,0,0,0,0,5,5,0,0,0,0,0,0,0,6,6,6,6,6,6,6}, //terrestris

  allhostsNA[RECORDSNA][ELEMS]= {//host-record colour-pattern elements
//0 X              1                   2
//1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
  5,  0,5,5,5,5,5,4,4,5,5,0,0,0,0,0,0,0,0,0,0,0,0,  //affinis
  5,  0,0,0,0,0,0,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,  //terricola
  5,  5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  //impatiens
  5,  0,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,  //vagans
  6,  0,0,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,  //appositus
  5,  0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,0,0,0,0,  //fervidus cali
  1,  0,0,1,1,0,5,0,5,5,5,3,3,3,3,3,3,3,3,0,0,0,0,  //flavifrons
```

```
              5,  0,5,5,5,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,  //nevadensis
              5,  0,0,5,5,5,5,3,3,3,3,3,3,3,5,5,5,5,0,0,0,0,  //ternarius
              5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //terricola occ
              5,  0,0,0,0,0,0,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0}, //pensylvanicus

              uniqhostsEU[HOSTSEU][ELEMS]= {//host-record colour-pattern elements
               //excluding non-unique hosts as comments
            //0 X               1                   2
            //1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
              5,  0,0,5,5,5,5,5,0,0,0,0,0,6,6,6,6,6,6,6,6,6,6,  //hortorum
              4,  4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,  //hypnorum
              5,  0,0,0,0,0,0,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //lucorum
              4,  4,4,4,4,5,5,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,  //humilis
              4,  4,4,4,4,5,5,4,0,4,4,0,0,0,0,4,0,4,4,4,4,4,4,  //pascuorum
              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3,3,  //pomorum
              5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,  //pratorum
              5,  0,0,5,5,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,6,  //jonellus
              5,  0,0,5,5,5,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,6,  //argillaceus
              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,5,5,5,  //ruderatus cors
//              4,  4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //hypnorum
              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,  //terrestris xanth
              5,  0,0,0,0,0,5,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //soroeensis
//              5,  0,0,0,0,0,5,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,  //soroeensis
              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //soroeensis prot
              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius
//              0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius
              5,  0,0,5,5,5,5,5,5,0,0,0,0,0,0,0,3,3,3,3,3,3,3,  //lapidarius dec
//              4,  4,4,4,4,5,5,4,0,4,4,0,0,0,0,4,0,4,4,4,4,4,4,  //pascuorum
              5,  0,0,5,5,5,5,5,5,0,0,0,0,0,0,4,4,4,4,4,4,4,4,  //sichelii
              5,  0,0,5,5,5,5,5,5,5,5,5,0,0,5,5,4,4,4,4,4,4,4,  //sylvarum
//              5,  0,0,5,5,5,5,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,6,  //jonellus
//              5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,  //pratorum
              5,  0,0,0,0,0,0,5,5,0,0,0,0,0,0,6,6,6,6,6,6,6,6}, //terrestris

              uniqhostsNA[HOSTSNA][ELEMS]= {//host-record colour-pattern elements
            //0 X               1                   2
            //1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
              5,  0,5,5,5,5,5,4,4,5,5,0,0,0,0,0,0,0,0,0,0,0,0,  //affinis
              5,  0,0,0,0,0,0,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,  //terricola
              5,  5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  //impatiens
              5,  0,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,  //vagans
              6,  0,0,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,  //appositus
              5,  0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,5,0,0,0,0,  //fervidus cali
              1,  0,0,1,1,0,5,0,5,5,5,5,3,3,3,3,3,3,3,3,3,0,0,0,0,  //flavifrons
              5,  0,5,5,5,5,5,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,  //nevadensis
              5,  0,0,5,5,5,5,3,3,3,3,3,3,3,5,5,5,5,5,0,0,0,0,  //ternarius
              5,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,6,  //terricola occ
              5,  0,0,0,0,0,0,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0}; //pensylvanicus

FILE          *fp_out;

//open file
if ((fp_out= fopen("simresults.txt","w+")) == NULL)//clear contents of file
       {
    printf("file opening error\n");
       return EXIT_FAILURE;
       }

//get test data
observedEU= 0;
observedNA= 0;
for (j=1; j<=RECORDSEU; j++)              // for simulated parasite records
    {
    for (k=1; k<=ELEMS; k++)
        {
        m= parasitesEU[j-1][k-1]-allhostsEU[j-1][k-1];
        if (m>0)
            observedEU += m;
```

```
        else if (m<0)
            observedEU -= m;
        }
    }
for (j=1; j<=RECORDSNA; j++)              // for simulated parasite records
    {
    for (k=1; k<=ELEMS; k++)
        {
        m= parasitesNA[j-1][k-1]-allhostsNA[j-1][k-1];
        if (m>0)
            observedNA += m;
        else if (m<0)
            observedNA -= m;
        }
    }
fprintf(fp_out," Observed mismatch value EU= %6.2f\n Observed mismatch value NA=
%6.2f\n\n",
    ((float)observedEU/(float)RECORDSEU),((float)observedNA/(float)RECORDSNA));

//simulation
srand((unsigned)time(NULL));                          // seed start of number table
randn= (int)(RAND_MAX/HOSTSEU);
randmax= HOSTSEU*randn;                               // max acceptable score

for (i=1; i<=SIMULATIONS; i++)                        // sim 1->9999
    {
    result[i]= 0;

    //Europe
    //choose offsets into RECORDS codes (from within tdata[RECORDS])
    for (j=0; j<RECORDSEU; j++)// for each sim cell
        {
             do  {
                    randn= rand();
                    } while (randn>randmax);
        choiceEU[j]= (randn%HOSTSEU);      // offsets 0 -> N-1
        }
    for (j=0; j<RECORDSEU; j++)              // for simulated parasite records
        {
        for (k=1; k<=ELEMS; k++)
            {
            m= parasitesEU[j][k-1]-uniqhostsEU[choiceEU[j]][k-1];
            if (m>0)
                result[i] += m;
            else if (m<0)
                result[i] -= m;
            }
        }
      }

//extract results
gap= SIMULATIONS/2;                                   //Shell binary sort
while (gap > 0)
       {
       for (i=1; i<=SIMULATIONS-gap; i++)
              {
              for (j=i; j>=1; j--)
                     {
                     mark= j+gap;
                     if (result[j] > result[mark])    //sort to increasing values
                            {
                            tempv= result[j];
                            result[j]= result[mark];
                            result[mark]= tempv;
                            }
                     else
                            j= 0;
                     }
```

```
                }
        gap /= 2;
        }

    //write results
    fprintf(fp_out,"\n\n Simulation results Europe ####\n\n");
    step= SIMULATIONS/1000;
    fprintf(fp_out," percentage points:\n");
    for (i=1; i<=SIMULATIONS; i++)                          // sim 1->100...
        {
        if ((i%(100))==0)
            {
            if (i==500)
                {
                j= 1;
                 fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                    i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSEU));
                 fprintf(fp_out,".....................................\n");
                }
            else if (i==9500)
                {
                 fprintf(fp_out,".....................................\n");
                 fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                    i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSEU));
                }
            else
                 fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                    i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSEU));
            }
        }
    fprintf(fp_out,"\n lower tail threshold:\n");
    for (i=1; i<=SIMULATIONS; i++)                          // sim 1->100...
        {
        if (i>=1 && i<=600)
            {
           fprintf(fp_out,"%5d (%6.4f):   %6.2f\n",
                i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSEU));
            }
        }

    ////////////////////////////////////////////////////////////////////////////
    //simulation
    randn= (int)(RAND_MAX/HOSTSNA);
    randmax= HOSTSNA*randn;                                  // max acceptable score
    for (i=1; i<=SIMULATIONS; i++)                          // sim 1->9999
        {
        result[i]= 0;

        //N America
        for (j=0; j<RECORDSNA; j++)// for each sim cell
            {
                 do  {
                        randn= rand();
                        } while (randn>randmax);
            choiceNA[j]= (randn%HOSTSNA);       // offsets 0 -> N-1
            }
        for (j=0; j<RECORDSNA; j++)               // for simulated parasite records
            {
            for (k=1; k<=ELEMS; k++)
                {
                m= parasitesNA[j][k-1]-uniqhostsNA[choiceNA[j]][k-1];
                if (m>0)
                    result[i] += m;
                else if (m<0)
                    result[i] -= m;
                }
            }
           }
```

```
//extract results
gap= SIMULATIONS/2;                                    //Shell binary sort
while (gap > 0)
       {
       for (i=1; i<=SIMULATIONS-gap; i++)
              {
              for (j=i; j>=1; j--)
                     {
                     mark= j+gap;
                     if (result[j] > result[mark])     //sort to increasing values
                            {
                            tempv= result[j];
                            result[j]= result[mark];
                            result[mark]= tempv;
                            }
                     else
                            j= 0;
                     }
              }
       gap /= 2;
       }

//write results
fprintf(fp_out,"\n\n Simulation results N America ####\n\n");
step= SIMULATIONS/1000;
fprintf(fp_out," percentage points:\n");
for (i=1; i<=SIMULATIONS; i++)                         // sim 1->100...
    {
    if ((i%(100))==0)
        {
        if (i==500)
            {
            j= 1;
             fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSNA));
             fprintf(fp_out,".....................................\n");
            }
        else if (i==9500)
            {
             fprintf(fp_out,".....................................\n");
             fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSNA));
            }
        else
             fprintf(fp_out,"%5d (%6.2f):     %6.2f\n",
                i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSNA));
        }
    }
fprintf(fp_out,"\n lower tail threshold:\n");
for (i=1; i<=SIMULATIONS; i++)                         // sim 1->100...
    {
    if (i>=1 && i<=600)
        {
       fprintf(fp_out,"%5d (%6.4f):   %6.2f\n",
            i,((double)i/SIMULATIONS),(float)result[i]/(float)(RECORDSNA));
        }
    }

//close
fclose(fp_out);
return EXIT_SUCCESS;
}
```