Chapter XII

Sieves: a User's Guide

I would like to give credit and express my thanks to Gérard Marino, a programmer who works with me at CEMAMu. He has adapted my own program which I originally wrote in "Basic" into "C."

The program is divided into two parts:

A. Generation of points on a straight line from the logical formula of the sieve.

B. Generation of the logical formula of the sieve from a series of points on a straight line.

A. GENERATION OF POINTS ON A STRAIGHT LINE FROM THE LOGICAL FORMULA OF THE SIEVE

Example:

DEFINITION OF A SIEVE:

```
L =   [() * () * . . . * ()]
    + [() * () * . . . * ()]
    + . . .
    + [() * () * . . . * ()]
```

In each parenthesis are given in order: modulus, starting point (taken from the set of integers)

[] + [] is a union

() * () is an intersection

Given the formula of a sieve made out of unions and intersections of moduli, the program reduces the number of intersections to one and keeps only the given unions. The abscissa of the final points of the sieve are computed from these unions and displayed.

NUMBER OF UNIONS ? = 2

---

union 1: number of modules ? = 2 modulus 1 ? = 3
start ? = 2
modulus 2 ? = 4
start ? = 7

union 2: number of modules ? = 2 modulus 1 ? = 6
start ? = 9
modulus 2 ? = 15
start ? = 18

FORMULA OF THE SIEVE:
L = [[3, 2) * (4, 7)]
+ [[6, 9) * (15, 18)]

REDUCTION OF THE INTERSECTIONS:
union 1
[(3,2) * (4,7))] = (12,11)

decompression into prime modules ?
(press 'y' for yes, any other key for no): y

(12,11) = 4,3) * (3,2)
union 2
[(6,9) * (15,18))] = (30,3)

decompression into prime modules ?
(press 'y' for yes, any other key for no): y

(30,2) = 2,1) * (3,0) * (5,3)

SIMPLIFIED FORMULA OF THE SIEVE:
L = L(12, 11) + (30, 3)

POINTS OF THE SIEVE CALCULATED WITH THIS FOPRMULA:
rank of first displayed point ? = 0
press <enter> to get a series of 10 points
Rank

0	3	11	23	33	35	47	59	63	71	83
10	93	95	107	119	123	131	143	153	155	167
20	179	183	191	203	213	215	227	239	243	251
30	263	273	275	287	299	303	311	323	333	335
40	347	359	363	371	383	393	395	407	419	423
50	431	443	453	455	467	479	483	491	503	513
60	515	527	539	543	551	563	573	575	587	599
70	603	611	623	633	635	647	659	663	671	683
80	693	695	707	719	723	731	743	753	755	767
90	779	783	791	803	813	815	827	839	843	851
100	863	873	875	887	899	903	911	923	933	935
110	947	959	963	971	983	993	995	1007	1019	1023
120	1031	1043	1053	1055	1067	1079	1083	1091	1103	1113
130	1115	1127	1139	1143	1151	1163	1173	1175	1187	1199

```
Line# Source Line
    1 #include <stdiuo.h >
    2 #include <stdlib.h>
    3 #include <coniuo.h>
    4
    5 /* ---------------------------------- types definitions ------------ ------------------ */
    6 typedef struct                  /* period ( congruence class )                    */
    7 {
    8 short mod;                     /* modulus of the period                          */
    9 short ini;                     /* starting point                                 */
   10   } periode;
   11 typedef struct                  /* intersection of several periods                */
   12 {
   13 short   clnb;                  /* number of terms in the intersection            */
   14 periode *cl;                   /* terms in the intersection                      */
   15 periode clr;                    /* resulting period                               */
   16 unsigned long ptval;           /* current point value                            */
   17 } inter;
   18 /* ---------------------------------- function prototypes ---------------------------- */
   19 periode ReducInter(short u); /* computation of the intersections                */
   20 short   Euclide(short m1,short m2);  /* computation of the LCD                  */
   21 short   Meziriac(short c1,short c2); /* computation of "dzeta"                  */
   22 void    Decompos(periode pr);/* decomposition into prime factors                */
   23
   24 /* ---------------------------------------- variables ---------------------------------- */
   25 inter  *fCrib;                  /* sieve formula                                  */
   26 short   unb = 0;                /* number of unions in the formula                */
   27
   28 short   u0, u1, u = 0;          /* current union index                            */
   29 short   i = 0;                  /* current intersection index                     */
   30 unsigned long lastval,n0,ptnb = 0;
   31 periode CL_EMPTY = { 0, 0 };/* empty period                                       */
   32
   33 #define NONEMPTY   1
   34 short   flag      = 0;
   35 short   decomp    = 0;
   36
   37 /*=================================== */
   38 void main(void)
   39 {
   40 printf("SIEVES: user's guide\n\n"
   41         "A. GENERATION OF POINTS ON A STRAIGHT LINE FROM\n"
   42         "  THE LOGICAL FORMULA OF THE SIEVE\n\n"
   43         "Example:\n"
   44         "---------------------------------\n"
   45         "DEFINITION OF A SIEVE:\n"
   46         "   L = [() * () * ... * ()]\n"
   47         "       + [() * () * ... * ()]\n"
   48         "       + ...\n"
   49         "       + [() * () * ... * ()]\n\n"
   50         "In each parenthesis are given in order: modulus, starting point\n"
   51                         "(taken from the set of integers)\n"
```

```
Line#  Source Line
   52          "[] + [] is a union\n"
   53          "() * () is an intersection\n\n");
   54      printf("--------------------------------\n"
   55          "Given the formula of a sieve made out of unions and\n"
   56          "intersections of moduli, the program reduces the number of\n"
   57          "intersections to one and keeps only the given unions.\n"
   58          "Then, the abscissa of the final points of the sieve are\n"
   59          "computed from these unions and displayed.\n\n");
   60      /* ---------------------- get the formula of the sieve ------------------------- */
   61      while (unb == 0)
   62          {
   63          printf("NUMBER OF UNIONS ? = ");
   64          scanf("%d",&unb);
   65          }
   66      fCrib = (inter *)(malloc (sizeof(inter) * unb));
   67      if (fCrib == NULL)
   68          {
   69          printf("not enough memory\n");
   70          exit(1);
   71          }
   72      printf("---------------------------------\n");
   73      for (u = 0; u  unb; u++)
   74          {
   75          printf("union %d: number of modules ? = ", u + 1);
   76          scanf("%d",&fCrib[u].clnb);
   77          printf("\n");
   78          fCrib[u].cl = (periode *)(malloc (sizeof(periode) * fCrib[u].clnb));
   79          if (fCrib[u].cl == NULL)
   80            {
   81            printf("not enough memory\n");
   82            exit(1);
   83            }
   84          for (i = 0; i  fCrib[u].clnb; i++)
   85            {
   86            printf("\n                modulus %d ? = ", i + 1);
   87            scanf("%d",&fCrib[u].cl[i].mod);
   88            printf("                start ?    = ");
   89            scanf("%d",&fCrib[u].cl[i].ini);
   90            }
   91          printf("---------------------------------\n");
   92          }
   93      /* ----------------- reduction of the formula ------------------------- */
   94      printf("FORMULA OF THE SIEVE:\n\n"
   95            "    L = [  ");
   96      for (u = 0; u  unb; u++)
   97          {
   98          if (u != 0)
   99            printf("       + [  ");
  100          for (i = 0; i  fCrib[u].clnb; i++)
  101            {
  102            if (i != 0)
  103              {
```

```
Line#  Source Line
  104            if (i % 4 == 0)
  105             printf("\n          ");
  106             printf("* ");
  107             }
  108           printf("(%5d,%5d) ", fCrib[u].cl[i].mod, fCrib[u].cl[i].ini);
  109           }
  110         printf("]\n");
  111         }
  112       printf("--------------------------------\n");
  113       printf("REDUCTION OF THE INTERSECTIONS:\n\n");
  114       for (u = 0; u  unb; u++)
  115          {
  116          printf("union %d\n      [ ",u + 1);
  117          for (i = 0; i  fCrib[u].clnb; i++)
  118            {
  119            printf("(%d,%d) ", fCrib[u].cl[i].mod, fCrib[u].cl[i].ini);
  120            if (i != fCrib[u].clnb - 1)
  121              printf("* ");
  122            }
  123          fCrib[u].clr = ReducInter(u);   /* reduction of an intersection */
  124          printf("] = (%d,%d)\n\n", fCrib[u].clr.mod, fCrib[u].clr.ini);
  125          printf("      decomposition into prime modules ?\n"
  126              "      (press 'y' for yes, any other key for no): ");
  127          if (getche() == 'y')
  128            {
  129            printf("\n\n      (%d,%d)", fCrib[u].clr.mod, fCrib[u].clr.ini);
  130            Decompos(fCrib[u].clr);
  131            }
  132          else
  133            printf("\n\n");
  134          }
  135       printf("--------------------------------\n");
  136       /* ---------------- display the simplified formula ---------------------- */
  137       printf("SIMPLIFIED FORMULA OF THE SIEVE:\n\n");
  138       printf("    L = ");
  139       for (u = 0; u  unb; u++)
  140          {
  141          if (u != 0)
  142            {
  143            if (u % 4 == 0)
  144              printf("\n        ");
  145            printf("+ ");
  146            }
  147          printf("(%5d,%5d) ", fCrib[u].clr.mod, fCrib[u].clr.ini);
  148          }
  149       printf("\n--------------------------------\n");
  150       /* ------------------- points of the sieve ---------------------------------- */
  151       printf("POINTS OF THE SIEVE CALCULATED WITH THIS
            FORMULA:\n");
  152       printf("rank of first displayed point ? = ");
  153       scanf("%lu",&n0);
  154       n0 = n0 - n0 % 10;
```

```
Line#  Source Line
  155      printf("\npress  <enter> to get a series of 10 points\n\n"
  156           "Rank    |");
  157      for (u = 0; u  unb; u++)
  158      {
  159         if (fCrib[u].clr.mod != 0 || fCrib[u].clr.ini != 0)
  160           {
  161           fCrib[u].ptval = fCrib[u].clr.ini;
  162           flag = NONEMPTY;
  163           }
  164         else
  165           fCrib[u].ptval = 0xFFFFFFFF;
  166         }
  167      if (flag != NONEMPTY)
  168         return;
  169      u0 = u1 = 0;
  170      lastval = 0xFFFFFFFF;
  171      while (1)
  172         {
  173         for (u = (u0 + 1) % unb; u != u0; u = (u + 1) % unb)
  174           {
  175           if (fCrib[u].ptval fCrib[u1].ptval)
  176             u1 = u;
  177         }
  178         if (fCrib[u1].ptval != lastval)  /* new point */
  179           {
  180           lastval = fCrib[u1].ptval;
  181           if (ptnb = n0)
  182             {
  183             if (ptnb % 10 == 0)
  184                   {
  185                   getch();   /* get a character from the keyboard */
  186                   printf("\n%7lu |", ptnb);
  187                   }
  188             printf("%6lu ", fCrib[u1].ptval);
  189             }
  190           ptnb++;
  191           }
  192         fCrib[u1].ptval += fCrib[u1].clr.mod;
  193         u0 = u1;
  194         }
  195
  196   /* === ===== reduction of an intersection ======== */
  197   periode ReducInter(short u)
  198   {
  199      periode  cl,cl1,cl2,cl3;
  200      short    pgcd,T,n;
  201      long     c1,c2;
  202
  203      cl3 = fCrib[u].cl[0];
  204      for (n = 1; n  fCrib[u].clnb; n++)
  205         {
  206         cl1 = cl3;
```

```
Line#  Source Line
  207         cl2 = fCrib[u].cl[n];
  208         if (cl1.mod  cl2.mod)
  209           {
  210           cl   = cl1;
  211           cl1  = cl2;
  212           cl2  = cl;
  213           }
  214         if (cl1.mod != 0 && cl2.mod != 0)
  215           {
  216           cl1.ini %= cl1.mod;
  217           cl2.ini %= cl2.mod;
  218           }
  219         else
  220           return CL_EMPTY;
  221         /* module resulting from the intersection of 2 modules */
  222         pgcd  = Euclide(cl1.mod, cl2.mod);
  223         c1    = cl1.mod / pgcd;
  224         c2    = cl2.mod / pgcd;
  225         if (pgcd != 1
  226           && ( (cl1.ini - cl2.ini) % pgcd != 0 ))
  227           return CL_EMPTY;
  228         if (pgcd != 1
  229           && ((cl1.ini - cl2.ini) % pgcd == 0)
  230           && (cl1.ini != cl2.ini) && (c1 == c2) )
  231           {
  232           cl3.mod = pgcd;
  233           cl3.ini = cl1.ini;
  234           continue;
  235           }
  236         T = Meziriac((short) c1, (short) c2);
  237         cl3.mod = (short) (c1 * c2 * pgcd);
  238         cl3.ini = (short) (( cl1.ini
  239                   + T * (cl2.ini - cl1.ini) * c1) % cl3.mod);
  240         while (cl3.ini  cl1.ini || cl3.ini  cl2.ini)
  241           cl3.ini += cl3.mod;
  242         }
  243       return cl3;
  244   }
  245   /*=== === decomposition into an intersection ======== */
  246   /*          of prime modules  */
  247   void  Decompos (periode pr)
  248   {
  249       periode  pf;
  250       short    fct;
  251
  252       if (pr.mod == 0)
  253           {
  254           printf(" = (%d,%d)\n", pr.mod, pr.ini);
  255           return;
  256           }
  257       printf(" =");
  258       for ( i = 0, fct = 2; pr.mod != 1; fct++)
```

```
Line#  Source Line
  259        {
  260        pf.mod = 1;
  261        while (pr.mod % fct == 0 && pr.mod != 1)
  262          {
  263          pf.mod  *= fct;
  264          pr.mod  /= fct;
  265          }
  266        if (pf.mod != 1)
  267          {
  268          pf.ini = pr.ini % pf.mod;
  269          pr.ini %= pr.mod;
  270          if (i != 0)
  271            printf(" *");
  272          printf(" (%d,%d)", pf.mod, pf.ini);
  273          i++;
  274          }
  275        }
  276     printf("\n");
  277  }
  278  /* ========== Euclide's algorithm======== */
  279  short Euclide (a1, a2)    /* a1 = a2  0 */
  280  short a1;
  281  short a2;
  282  {
  283     short tmp;
  284
  285     while ((tmp = a1 % a2) != 0)
  286        {
  287        a1 = a2;
  288        a2 = tmp;
  289        }
  290     return a2;
  291  }

  292  /* ======== De Meziriac's theorem ========= */
  293  short Meziriac (c1, c2)   /* c1 = c2  0 */
  294  short c1;
  295  short c2;
  296  {
  297     short T = 0;
  298
  299     if (c2 == 1)
  300        T = 1;
  301     else
  302        while (((++T * c1) % c2) != 1)
  303          ;
  304     return T;
  305  }
```

B. GENERATION OF THE LOGICAL FORMULA OF THE SIEVE FROM A SERIES OF POINTS ON A STRAIGHT LINE

Example:

Given a series of points, find the starting points with their moduli (periods).
NUMBER OF POINTS ? = 12

---

abscissa of the points

point 1 = 59	point 2 = 93	point 3 = 47	point 4 = 3
point 5 = 63	point 6 = 11	point 7 = 23	point 8 = 33
point 9 = 95	point 10 = 71	point 11 = 35	point 12 = 83

POINTS OF THE SIEVE (ordered by their increasing abscissa):
Rank 0 3 11 23 33 35 47 59 63 71 83
10 93 95

FORMULA OF THE SIEVE:
In each parenthesis are given in order:
(modulus, starting point, number of covered points)
L = (30, 3, 4}+(12, 11, 8)
period of the sieve: P = 60

Line# Source Line

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4  #include <string.h>
 5
 6  /* ---------------------------- types definitions ---------------------------------------- */
 7  typedef struct                  /* period ( congruence class )                          */
 8  {
 9          short mod;              /* modulus of the period
10          short ini;              /* starting point                                       */
11     short couv;                  /* number of covered points                             */
12  } periode;
13  /* ---------------------------- function prototypes -------------------------------------- */
14  unsigned long Euclide(unsigned long m1,
15          unsigned long m2); /* computation of the LCD                                    */
16  /* ---------------------------- variables and constants ---------------------------------- */
17  periode*perCrib;                /* periods of the sieve                                 */
18  short   perTotNb = 0;           /* number of periods in the formula                     */
19  long    *ptCrib;                /* points of the crible                                 */
20  long    *ptReste;               /* points outside the periods                           */
21  short   ptTotNb = 0;            /* number of points in the sieve                        */
22  short   p,ptnb;
23  long    ptval;
24  unsigned long percrib;
25
```

```
Line#  Source Line
  26   periodeper;
  27
  28   #define NON_REDUNDANT 0
  29   #define REDUNDANT     1
  30   #define COVERED      -1L
  31   short  flag;
  32
  33   /* ================================== */
  34   void main(void)
  35   {
  36      printf("B. GENERATION OF THE LOGICAL FORMULA OF THE
       SIEVE FROM\n"
  37           "  A SERIES OF POINTS ON A STRAIGHT LINE\n\n"
  38           "Example:\n"
  39           "--------------------------------\n"
  40           "Given a series of points, find the starting points\n"
  41           "with their moduli (periods).\n\n");
  42         /* --------- entry of the points of the sieve and their sorting -------*/
  43      while (ptTotNb == 0)
  44         {
  45         printf ("NUMBER OF POINTS ? = ");
  46         scanf("%d",&ptTotNb);
  47          }
  48      ptCrib   = (long *)(malloc (ptTotNb * sizeof(long)));
  49      ptReste  = (long *)(malloc (ptTotNb * sizeof(long)));
  50      perCrib  = (periode *)(malloc (ptTotNb * sizeof(periode)));
  51      if (ptCrib == NULL || ptReste == NULL || perCrib == NULL)
  52         {
  53         printf("not enough memory\n");
  54         exit(1);
  55         }
  56      printf("--------------------------------\n"
  57           "abscissa of the points:\n");
  58      for (p = 0; p  ptTotNb; p++)
  59         {
  60         if (p % 4 == 0)
  61           printf("\n   ");
  62         printf("point %2d = ", p + 1);
  63         scanf("%ld", &ptval);
  64         for (ptnb = 0;
  65            ptnb  p && ptval  ptCrib[ptnb];
  66            ptnb++)
  67           ;
  68         if (ptnb  p)
  69           {
  70         if (ptval  ptCrib[ptnb])  /* new point                    */
  71               memmove(&ptCrib[ptnb + 1], &ptCrib[ptnb],
  72                  sizeof(long) * (p - ptnb));
  73          else  /* point already exist   */
  74               {
  75               p--;
```

```
Line#  Source Line
  76                    ptTotNb--;
  77                    }
  78              }
  79            ptCrib[ptnb] = ptval;
  80        }
  81        printf("\n---------------------------------\n");
  82        /* ---------------------- points of the sieve ------------------------------- */
  83        printf("POINTS OF THE SIEVE (ordered by their increasing
            abscissa):\n\n"
  84              "Rank    |");
  85        for (p = 0; p  ptTotNb; p++)
  86            {
  87            if (p % 10 == 0)
  88              printf("\n%7d |", p);
  89            printf("%6ld ", ptCrib[p]);
  90            }
  91        printf("\n\n---------------------------------\n");
  92        /* ---------------- compute the periods of the sieve ------------------- */
  93        memcpy(ptReste, ptCrib, ptTotNb * sizeof(long));
  94
  95        for (p = 0; p  ptTotNb; p++)
  96            {
  97            if ( ptReste[p] == COVERED )
  98              continue;
  99            /* -------- compute a period starting at current point --------- */
 100            per.mod  = 0;
 101            do
 102              {
 103              per.mod++;
 104              per.ini  = (short) (ptCrib[p] % (long)per.mod);
 105              per.couv = 0;
 106              for (ptnb = 0, ptval = per.ini;
 107                      ptnb  ptTotNb && ptval = ptCrib[ptnb];
 108                      ptnb++)
 109                     {
 110                     if (ptval == ptCrib[ptnb])
 111                       {
 112                       per.couv++;
 113                     ptval += per.mod;
 114                     }
 115                 }
 116              }
 117            while (ptnb  ptTotNb);
 118            /* ---------- check the redundancy of the period ----------------- */
 119            for (ptnb = 0, ptval = per.ini, flag = REDUNDANT;
 120              ptnb  ptTotNb;
 121              ptnb++)
 122              {
 123              if (ptval == ptCrib[ptnb])
 124                {
 125                if (ptval == ptReste[ptnb])
 126                      {
```

```
Line#  Source Line
  127                 ptReste[ptnb] = COVERED;
  128                 flag = NON_REDUNDANT;
  129                 }
  130               ptval += per.mod;
  131               }
  132             }
  133           if (flag == NON_REDUNDANT)
  134             perCrib[perTotNb++] = per;
  135           }
  136       /* ---------- compute the period of the sieve ---------- */
  137       percrib = perCrib[0].mod;
  138       for (p = 1; p  perTotNb; p++)
  139           {
  140           if ((long) perCrib[p].mod = percrib)
  141             percrib *= (long) perCrib[p].mod / Euclide((long)perCrib[p].mod,
                percrib);
  142           else
  143           percrib *= (long) perCrib[p].mod / Euclide(percrib,
              (long)perCrib[p].mod);
  144           }
  145       /* ---------- display the formula of the sieve ---------- */
  146       printf("FORMULA OF THE SIEVE:\n"-
  147             "In each parenthesis are given in order:\n"
  148             "(modulus, starting point, number of covered points)\n\n");
  149        printf("    L = ");
  150       for (p = 0; p  perTotNb; p++)
  151           {
  152           if (p != 0)
  153             {
  154             if (p % 3 == 0)
  155               printf("\n       ");
  156             printf("+ ;
  157             }
  158           printf("(%5d,%5d,%5d) ", perCrib[p].mod,perCrib[p].ini,
              perCrib[p].couv);
  159           }
  160       printf("\n\n    period of the sieve: P = %lu\n", percrib);
  161   }
  162   /* == ========= Euclide's algorithm ============ */
  163   unsigned long Euclide (a1, a2)   /* a1 = a2  0                  */
  164   unsigned long a1;
  165   unsigned long a2;
  166   {
  167       unsigned long tmp;
  168
  169       while ((tmp = a1 % a2) != 0)
  170           {
  171       a1 = a2;
  172           a2 = tmp;
  173           }
  174       return a2;
  175   }
```