```
#define MAPM_NUM_DEC_PLC_OPN  240
#define MAPM_NUM_DEC_PLC_SHOW 240

/* ------------------------------------------------------------------
            CDPS2_sqrt2_doac_3rd_order_verif_arb_prec
------------------------------------------------------------------
Function: sqrt(2) difference-of-alternating-coefficients (DOAC), 3rd-order
recurrence relation used to generate denominators, arbitrary precision

Description: allow the user to 'verify' numerically that the sqrt(2) series
represention using the DOAC approach and the 3rd-order recurrence relation
for generating denominators is correct. As more terms in the series are
generated, the approximation of sqrt(2)^2 converges to 2.

Usage: the function allows the user to specify whether the series
representation will converge to sqrt(2) from above or below.

Implementation notes:
• uses Mike's Arbitrary Precision Math (MAPM) C-language library
• I typically first implement a function as 32-bit (not arbitrary precision)
to verify the algorithm and then convert it to arbitrary precision. The
 comments
in the code reflect this approach. The arbitrary precision code is usually
much harder to read / understand than the 32-bit version.
• the comments are from the 'from above' version of the algorithm
------------------------------------------------------------------ */
int CDPS2_sqrt2_doac_3rd_order_verif_arb_prec(
   int      from_above) {  // control: converge from above if nonzero; else
    converge from below.
   int      num_iter;      // number of iterations
   int      i;             // counter
   M_APM    e0_ap = (void *)0;
   M_APM    e1_ap = (void *)0;
   M_APM    e2_ap = (void *)0;
   M_APM    e3_ap = (void *)0;
   M_APM    t_ap  = (void *)0;
   M_APM    t1_ap = (void *)0;
   M_APM    t2_ap = (void *)0;
   M_APM    s_ap  = (void *)0;
   M_APM    C35_ap = (void *)0;
   M_APM    C2_ap  = (void *)0;
   M_APM    C3_ap  = (void *)0;  // new

   char     *d_ap_str   = (void *)0;
   char     *s_ap_str   = (void *)0;

   m_apm_cpp_precision(MAPM_NUM_DEC_PLC_OPN);

                        // -- allocate arbitrary precision variables --
   e0_ap    = m_apm_init();
   e1_ap    = m_apm_init();
```

```
   e2_ap    = m_apm_init();
   e3_ap    = m_apm_init();
   t_ap     = m_apm_init();
   t1_ap    = m_apm_init();
   t2_ap    = m_apm_init();
   s_ap     = m_apm_init();
   C35_ap   = m_apm_init();
   C2_ap    = m_apm_init();
   C3_ap    = m_apm_init();

                        // -- initialize variables --
   m_apm_set_long(C35_ap,35);
   m_apm_set_long(C2_ap,2);
   m_apm_set_long(C3_ap,3);

   if (!from_above) {
      m_apm_set_long(e0_ap,5);
      m_apm_set_long(e1_ap,145);
      m_apm_set_long(e2_ap,4901); }
   else {
      m_apm_set_long(e0_ap,24);
      m_apm_set_long(e1_ap,840);
      m_apm_set_long(e2_ap,28560); }

                        // -- query user --
   printf("Number of iterations? ");
   scanf("%d",&num_iter);

                        // -- generate the first few terms before entering
                         loop --
   printf("\nsqrt(2) =\n");

   if (!from_above)
      printf("\n   1 / 1 +\n");
   else
      printf("\n   3 / 2 -\n");

//   s = 1.5;

   if (!from_above)
      m_apm_set_double(s_ap,1.0);
   else
      m_apm_divide(s_ap,MAPM_NUM_DEC_PLC_OPN,C3_ap,C2_ap);

//   printf("\n   %d / %d -\n",2,e0);

   d_ap_str = m_apm_to_fixpt_stringexp(0,e0_ap,'.',',',3);
   if (!from_above)
      printf("\n   %d / %s +\n",2,d_ap_str);
   else
      printf("\n   %d / %s -\n",2,d_ap_str);
```

```
   free(d_ap_str);

//   s -= (double)2/e0;

   m_apm_divide(t_ap,MAPM_NUM_DEC_PLC_OPN,C2_ap,e0_ap);
   if (!from_above)
      m_apm_add(t1_ap,s_ap,t_ap);
   else
      m_apm_subtract(t1_ap,s_ap,t_ap);
   m_apm_copy(s_ap,t1_ap);

//   printf("\n   %d / %d -\n",2,e1);

   d_ap_str = m_apm_to_fixpt_stringexp(0,e1_ap,'.',',',3);
   if (!from_above)
      printf("\n   %d / %s +\n",2,d_ap_str);
   else
      printf("\n   %d / %s -\n",2,d_ap_str);
   free(d_ap_str);

//   s -= (double)2/e1;

   m_apm_divide(t_ap,MAPM_NUM_DEC_PLC_OPN,C2_ap,e1_ap);
   if (!from_above)
      m_apm_add(t1_ap,s_ap,t_ap);
   else
      m_apm_subtract(t1_ap,s_ap,t_ap);
   m_apm_copy(s_ap,t1_ap);

//   printf("\n   %d / %d -\n",2,e2);

   d_ap_str = m_apm_to_fixpt_stringexp(0,e2_ap,'.',',',3);
   if (!from_above)
      printf("\n   %d / %s +\n",2,d_ap_str);
   else
      printf("\n   %d / %s -\n",2,d_ap_str);
   free(d_ap_str);

//   s -= (double)2/e2;

   m_apm_divide(t_ap,MAPM_NUM_DEC_PLC_OPN,C2_ap,e2_ap);
   if (!from_above)
      m_apm_add(t1_ap,s_ap,t_ap);
   else
      m_apm_subtract(t1_ap,s_ap,t_ap);
   m_apm_copy(s_ap,t1_ap);

                        // -- loop --
   for (i=0; i<num_iter; i++) {

//     e3 = 1*e0 - 35*e1 + 35*e2;
```

```
      m_apm_multiply(t2_ap,e2_ap,C35_ap);
      m_apm_multiply(t1_ap,e1_ap,C35_ap);
      m_apm_subtract(t_ap,t2_ap,t1_ap);
      m_apm_add(e3_ap,e0_ap,t_ap);

//      printf("\n   %d / %d -\n",2,e3);

      d_ap_str = m_apm_to_fixpt_stringexp(0,e3_ap,'.',',',3);
      if (!from_above)
         printf("\n   %d / %s +\n",2,d_ap_str);
      else
         printf("\n   %d / %s -\n",2,d_ap_str);
      free(d_ap_str);

//      s -= (double)2/e3;

      m_apm_divide(t_ap,MAPM_NUM_DEC_PLC_OPN,C2_ap,e3_ap);
      if (!from_above)
         m_apm_add(t1_ap,s_ap,t_ap);
      else
         m_apm_subtract(t1_ap,s_ap,t_ap);
      m_apm_copy(s_ap,t1_ap);
/*
      e0 = e1;
      e1 = e2;
      e2 = e3;
*/
      m_apm_copy(e0_ap,e1_ap);
      m_apm_copy(e1_ap,e2_ap);
      m_apm_copy(e2_ap,e3_ap); }

   printf("\n   ...\n");

//   printf("\nsqrt(2) = %30.27lf\n",s);

   s_ap_str = m_apm_to_fixpt_stringexp(MAPM_NUM_DEC_PLC_SHOW,s_ap,'.',',',3);
   printf("\nsqrt(2) = %s\n",s_ap_str);
   free(s_ap_str);

//   printf("\nsqrt(2)^2 = %30.27lf\n",s*s);

   m_apm_multiply(t_ap,s_ap,s_ap);
   m_apm_copy(s_ap,t_ap);

   s_ap_str = m_apm_to_fixpt_stringexp(MAPM_NUM_DEC_PLC_SHOW,s_ap,'.',',',3);
   printf("\nsqrt(2)^2 = %s\n",s_ap_str);
   free(s_ap_str);

   printf("\n");
                        // -- free the variables --
```

```
m_apm_free(e0_ap);
m_apm_free(e1_ap);
m_apm_free(e2_ap);
m_apm_free(e3_ap);
m_apm_free(t_ap);
m_apm_free(t1_ap);
m_apm_free(t2_ap);
m_apm_free(s_ap);
m_apm_free(C35_ap);
m_apm_free(C2_ap);
m_apm_free(C3_ap);

return (TRUE); }
```