```c
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <time.h>

# include "cordic.h"


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

void cordic ( double *x, double *y, double *z, int n )

/******************************************************************************/
/*
  CORDIC returns the sine and cosine using the CORDIC method.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    2012.04.17

  Author:

    Based on MATLAB code in a Wikipedia article.

    Modifications by John Burkardt

    Further modified by Young W. Lim

  Parameters:

    Input:
      *x: x coord of an init vector
      *y: y coord of an init vector
      *z: angle (-90 <= angle <= +90)
       n: number of iteration
         A value of 10 is low.  Good accuracy is achieved
         with 20 or more iterations.

    Output:
      *xo: x coord of a final vector
      *yo: y coord of a final vector
      *zo: angle residue

  Local Parameters:

    Local, real ANGLES(60) = arctan ( (1/2)^(0:59) );

    Local, real KPROD(33), KPROD(j) = product ( 0 <= i <= j ) K(i),
    K(i) = 1 / sqrt ( 1 + (1/2)^(2i) ).
*/
{
# define ANGLES_LENGTH 60
# define KPROD_LENGTH 33

  double angle;
  double angles[ANGLES_LENGTH] = {
    7.8539816339744830962E-01,
    4.6364760900080611621E-01,
    2.4497866312686415417E-01,
    1.2435499454676143503E-01,
    6.2418809995957348474E-02,
    3.1239833430268276254E-02,
    1.5623728620476830803E-02,
    7.8123410601011112965E-03,
    3.9062301319669718276E-03,
    1.9531225164788186851E-03,
```

```
    9.765621895593194040E-04,
    4.882812111948982754E-04,
    2.441406201493617640E-04,
    1.220703118936702042E-04,
    6.103515617420877502E-05,
    3.051757811552609686E-05,
    1.525878906131576210E-05,
    7.629394531101970263E-06,
    3.814697265606496282E-06,
    1.907348632810187035E-06,
    9.536743164059608794E-07,
    4.768371582030888599E-07,
    2.384185791015579824E-07,
    1.192092895507806853E-07,
    5.960464477539055441E-08,
    2.980232238769530367E-08,
    1.490116119384765514E-08,
    7.450580596923827987E-09,
    3.725290298461914045E-09,
    1.862645149230957029E-09,
    9.313225746154785153E-10,
    4.656612873077392577E-10,
    2.328306436538696289E-10,
    1.164153218269348144E-10,
    5.820766091346740722E-11,
    2.910383045673370361E-11,
    1.455191522836685180E-11,
    7.275957614183425903E-12,
    3.637978807091712951E-12,
    1.818989403545856475E-12,
    9.094947017729282379E-13,
    4.547473508864641189E-13,
    2.273736754432320594E-13,
    1.136868377216160297E-13,
    5.684341886080801487E-14,
    2.842170943040400743E-14,
    1.421085471520200371E-14,
    7.105427357601001858E-15,
    3.552713678800500929E-15,
    1.776356839400250464E-15,
    8.881784197001252323E-16,
    4.440892098500626161E-16,
    2.220446049250313080E-16,
    1.110223024625156540E-16,
    5.551115123125782702E-17,
    2.775557561562891351E-17,
    1.387778780781445675E-17,
    6.938893903907228377E-18,
    3.469446951953614188E-18,
    1.734723475976807094E-18 };
  double c2;
  double factor;
  int j;
  double kprod[KPROD_LENGTH] = {
    0.70710678118654752440,
    0.63245553203367586640,
    0.61357199107789634961,
    0.60883391251775242102,
    0.60764825625616820093,
    0.60735177014129595905,
    0.60727764409352599905,
    0.60725911229889273006,
    0.60725447933256232972,
    0.60725332108987516334,
    0.60725303152913433540,
    0.60725295913894481363,
    0.60725294104139716351,
    0.60725293651701023413,
    0.60725293538591350073,
    0.60725293510313931731,
```

```
    0.60725293503244577146,
    0.60725293501477238499,
    0.60725293501035403837,
    0.60725293500924945172,
    0.60725293500897330506,
    0.60725293500890426839,
    0.60725293500888700922,
    0.60725293500888269443,
    0.60725293500888161574,
    0.60725293500888134606,
    0.60725293500888127864,
    0.60725293500888126179,
    0.60725293500888125757,
    0.60725293500888125652,
    0.60725293500888125626,
    0.60725293500888125619,
    0.60725293500888125617 };
  double pi = 3.141592653589793;
  double poweroftwo;
  double s2;
  double sigma;
  double sign_factor;
  double theta;

  double xn, yn;


/*
  Initialize loop variables:
*/
  theta = *z;

  xn = *x;
  yn = *y;

  poweroftwo = 1.0;
  angle = angles[0];

/*
  Iterations
*/
  for ( j = 1; j <= n; j++ )
  {
    if ( theta < 0.0 )
    {
      sigma = -1.0;
    }
    else
    {
      sigma = 1.0;
    }

    factor = sigma * poweroftwo;

    *x =          xn - factor * yn;
    *y = factor * xn +          yn;

    xn = *x;
    yn = *y;

/*
  Update the remaining angle.
*/
    theta = theta - sigma * angle;

    poweroftwo = poweroftwo / 2.0;
/*
  Update the angle from table, or eventually by just dividing by two.
*/
    if ( ANGLES_LENGTH < j + 1 )
```

```
    {
      angle = angle / 2.0;
    }
    else
    {
      angle = angles[j];
    }

    *z = theta;
  }
/*
  Adjust length of output vector to be [cos(beta), sin(beta)]

  KPROD is essentially constant after a certain point, so if N is
  large, just take the last available value.
*/
/*  if ( 0 < n )
  {
    *c = *c * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
    *s = *s * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
  }
*/
/*
  Adjust for possible sign change because angle was originally
  not in quadrant 1 or 4.
*/
/*
  *c = sign_factor * *c;
  *s = sign_factor * *s;
*/


  return;
# undef ANGLES_LENGTH
# undef KPROD_LENGTH
}
```