```
::::::::::::::
run_cpp_cordic.c
::::::::::::::
#include <stdio.h>


struct Core;

/*-------------------------------------------------------------*/
struct Core * Core_Create();
void Core_Destroy (struct  Core * thisCore);
void Core_cordic(struct Core * thisCore, double *x, double *y, double *z);
/*-------------------------------------------------------------*/




/*-------------------------------------------------------------*/
int main(int argc, char *argv[]) {

  struct Core *p;

  p = (struct Core *) Core_Create();


  double x =.0;
  double y =.0;
  double z =.0;


  // x = 6.072529349476099e-1;
  x = 1.0;
  y = 0.0e0;
  z = 0.0e0;


  printf("---------------------------------\n");
  printf("* before cordic_ghdl \n");
  printf("  x =%f \n", x);
  printf("  y =%f \n", y);
  printf("  z =%f \n", z);
  printf("---------------------------------\n");

  Core_cordic (p, &x, &y, &z);

  printf("---------------------------------\n");
  printf("* after cordic_ghdl \n");
  printf("  x =%f \n", x);
  printf("  y =%f \n", y);
  printf("  z =%f \n", z);
  printf("---------------------------------\n");

  return 0;
}
::::::::::::::
apiCore.cpp
::::::::::::::
#include "Core.hpp"


extern "C" {

 Core * Core_Create() {
   return reinterpret_cast <Core *> (new Core());
 }

  void Core_Destroy (Core * thisCore) {
```

```
    delete reinterpret_cast <Core *> (thisCore);
  }

  void Core_cordic (Core * thisCore, double *x, double *y, double *z) {
    return reinterpret_cast <Core *> (thisCore)->cordic(x, y, z);
  }

}
::::::::::::::
Core.cpp
::::::::::::::
#include "Core.hpp"


using namespace std;


//-----------------------------------------------------------------------
//   Purpose:
//
//     Class Core Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
//
// void   Angles::setnAngles(int nAngles)
//
//-----------------------------------------------------------------------
Core::Core()
{
  setPi();
  setK();
  setAngles();
  setKprod();

  level         = 10;
  nBreak        = 0;
  nBreakInit    = 0;
  threshold     = 0.0001;
  strcpy(path, "");

  useTh         =  1;
  useThDisp     = 1;
  useATAN       = 0;


}

Core::~Core()
{
```

```
}


//------------------------------------------------------------------------
//  Accessor & Changer
//------------------------------------------------------------------------
void    Core::setUseTh     (int flag) { useTh     = flag; }
void    Core::setUseThDisp (int flag) { useThDisp = flag; }
void    Core::setUseATAN   (int flag) { useATAN   = flag; }


int     Core::getUseTh()     { return(useTh);     }
int     Core::getUseThDisp() { return(useThDisp); }
int     Core::getUseATAN()   { return(useATAN);   }

//------------------------------------------------------------------------
void    Core::setLevel      (int l)      { level      = l;      }
void    Core::setNBreak     (int nB)     { nBreak     = nB;     }
void    Core::setNBreakInit (int nBInit) { nBreakInit = nBInit; }
void    Core::setThreshold  (double th)  { threshold  = th;     }
void    Core::setPath       (char *p)    { strcpy(path, p);     }


int     Core::getLevel()       { return(level);       }
int     Core::getNBreak()      { return(nBreak);      }
int     Core::getNBreakInit()  { return(nBreakInit);  }
double  Core::getThreshold()   { return(threshold);   }
void    Core::getPath(char *p) { strcpy(p, path);     }

//------------------------------------------------------------------------
double *Core::getAngles()      { return angles;       }
double *Core::getKprod()       { return kprod;        }



void Core::initAcc ()
{
    max_err =0.0,  max_errn =0.0;

    sum_xx =0.0,   sum_xx2 =0.0;
    sum_yy =0.0,   sum_yy2 =0.0;

    sum_xx_n =0.0, sum_xx2_n =0.0;
    sum_yy_n =0.0, sum_yy2_n =0.0;

    cnt_xx =0.0,   cnt_yy =0.0;
}

//------------------------------------------------------------------------
void Core::cordic ( double *x, double *y, double *z, int& cnt, int& xx, int& yy, int& zz)
//------------------------------------------------------------------------
{

    double cosz, sinz;


    if (cnt == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        cnt++;
        sSCE   =   sSSE =   sSRE = 0.0;
        minSCE = minSSE = minSRE = +1.0e+10;
        maxSCE = maxSSE = maxSRE = -1.0e+10;
    }

    cosz = cos(*z);
```

```
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //.....................................................
    cordic(x, y, z);
    //.....................................................

    xx = (*x - cosz);
    yy = (*y - sinz);
    zz = (*z);

      SCE  = xx * xx;       SSE  = yy * yy;       SRE  = zz * zz;
     sSCE += SCE;          sSSE += SSE;          sSRE += SRE;
     mSCE  = sSCE/cnt;     mSSE  = sSSE/cnt;     mSRE  = sSRE/cnt;
    rmSCE  = sqrt(mSCE);  rmSSE  = sqrt(mSSE);  rmSRE  = sqrt(mSRE);

    minSCE = (minSCE > SCE) ? SCE : minSCE;
    minSSE = (minSSE > SSE) ? SSE : minSSE;
    minSRE = (minSRE > SRE) ? SRE : minSRE;

    maxSCE = (maxSCE < SCE) ? SCE : maxSCE;
    maxSSE = (maxSSE < SSE) ? SSE : maxSSE;
    maxSRE = (maxSRE < SRE) ? SRE : maxSRE;

}

//-------------------------------------------------------------------------
void Core::cordic ( double *x, double *y, double *z, int& init)
//-------------------------------------------------------------------------
{

    double cosz, sinz;


    if (init == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        init++;
    }

    cosz = cos(*z);
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //.....................................................
    cordic(x, y, z);
    //.....................................................

    xx = (*x - cosz);
    yy = (*y - sinz);

    sum_xx += xx; sum_xx2 += (xx*xx);
    sum_yy += yy; sum_yy2 += (yy*yy);

    if (max_err < fabs(xx)) max_err = fabs(xx);
    if (max_err < fabs(yy)) max_err = fabs(yy);

    if (fabs(cosz) > 1.0e-10) {
        if (max_errn < fabs(xx/cosz))
            max_errn = fabs(xx/cosz);
        sum_xx_n += xx/cosz;
        sum_xx2_n += (xx*xx)/(cosz*cosz);
        cnt_xx++;
    }
    if (fabs(sinz) > 1.0e-10) {
        if (max_errn < fabs(yy/sinz))
```

```
            max_errn = fabs(yy/sinz);
        sum_yy_n += yy/sinz;
        sum_yy2_n += (yy*yy)/(sinz*sinz);
        cnt_yy++;
    }

}


//----------------------------------------------------------------------
void Core::cordic ( double *x, double *y, double *z )
//----------------------------------------------------------------------
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.29
//
//  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)
//
//      level : 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) ).
//
//----------------------------------------------------------------------
{
  double angle;
  double factor;

  double sigma;
  double poweroftwo;
  double theta;

  double xn, yn;

  int j;

  //--------------------------------------------------------------------
```

```
//  Initialize loop variables:
//-----------------------------------------------------------------------
xn = *x;
yn = *y;
theta = *z;

poweroftwo = 1.0;

if (useATAN)
  angle = atan( 1. );
else
  angle = angles[0];



//-----------------------------------------------------------------------
for ( j = 1; j <= level; j++ )
//-----------------------------------------------------------------------
{
    if ( theta < 0.0 ) sigma = -1.0;
    else               sigma = +1.0;

    if ( theta < 0.0 ) path[j-1] = '0';
    else               path[j-1] = '1';
    path[j] = '\0';

    factor = sigma * poweroftwo;

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

    xn = *x;
    yn = *y;

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

    *z = theta;

    //.....................................................................
    // If residual angle is less than a given threshold, then break
    //.....................................................................

    //  cout << right << setw(20) << " j= " << right << setw(4) << j  ;
    //  cout << "  z= " << right << setw(15) << *z;
    //  cout << " < " << right << setw(7) << threshold;
    //  cout << endl;


    if (useTh) {
      static int cntBreak = 0;
      if (nBreakInit == 0) cntBreak = 0;
      if (fabs(*z) < threshold) {
        nBreak = ++cntBreak;

        if (useThDisp) {
          cout << "cntBreak= " << cntBreak;
          cout << "  z= " << right << setw(15) << *z;
          cout << " < " << right << setw(7) << threshold;
          cout << " j= " << right << setw(4) << j << endl;
        }
        break;
      }
    }
```

```
      //.....................................................................
      //  Update the angle from table, or eventually by just dividing by two.
      //.....................................................................
      poweroftwo = poweroftwo / 2.0;

      if (useATAN)
        if ( ANGLES_LENGTH < j+1 )  angle = angle / 2.0;
        else                        angle = angles[j];
      else
        angle = atan( 1. / (1 << j));


  //-----------------------------------------------------------------------
  }  /* end of j */
  //-----------------------------------------------------------------------



  //-----------------------------------------------------------------------
  //  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 ( j > KPROD_LENGTH ) {
    *x = *x * kprod [ KPROD_LENGTH - 1 ];
    *y = *y * kprod [ KPROD_LENGTH - 1 ];
  }
  else {
    *x = *x * kprod [ j - 1 ];
    *y = *y * kprod [ j - 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;

}


//---------------------------------------------------------------------------
//  Initialize the constants: pi, K
//---------------------------------------------------------------------------
void Core::setPi()
{
  pi = 3.141592653589793;
}

void Core::setK()
{
  K = 1.646760258121;
}

//---------------------------------------------------------------------------
//  Initialize the array Angles[ANGLES_LENGTH]
//---------------------------------------------------------------------------
void Core::setAngles()
{
  double angles_in[ANGLES_LENGTH] = {
    7.8539816339744830962E-01,
    4.6364760900080611621E-01,
```

```
    2.44978663126864154 17E-01,
    1.24354994546761435 03E-01,
    6.24188099959573484 74E-02,
    3.12398334302682762 54E-02,
    1.56237286204768308 03E-02,
    7.81234106010111129 65E-03,
    3.90623013196697182 76E-03,
    1.95312251647881868 51E-03,
    9.76562189559319430 40E-04,
    4.88281211194898275 47E-04,
    2.44140620149361764 02E-04,
    1.22070311893670204 24E-04,
    6.10351561742087750 22E-05,
    3.05175781155260968 62E-05,
    1.52587890613157621 07E-05,
    7.62939453110197026 34E-06,
    3.81469726560649628 29E-06,
    1.90734863281018703 54E-06,
    9.53674316405960879 42E-07,
    4.76837158203088859 93E-07,
    2.38418579101557982 49E-07,
    1.19209289550780685 31E-07,
    5.96046447753905544 14E-08,
    2.98023223876953036 77E-08,
    1.49011611938476551 47E-08,
    7.45058059692382798 71E-09,
    3.72529029846191404 53E-09,
    1.86264514923095702 91E-09,
    9.31322574615478515 36E-10,
    4.65661287307739257 78E-10,
    2.32830643653869628 90E-10,
    1.16415321826934814 45E-10,
    5.82076609134674072 26E-11,
    2.91038304567337036 13E-11,
    1.45519152283668518 07E-11,
    7.27595761418342590 33E-12,
    3.63797880709171295 17E-12,
    1.81898940354585647 58E-12,
    9.09494701772928237 92E-13,
    4.54747350886464118 96E-13,
    2.27373675443232059 48E-13,
    1.13686837721616029 74E-13,
    5.68434188608080148 70E-14,
    2.84217094304040074 35E-14,
    1.42108547152020037 17E-14,
    7.10542735760100185 87E-15,
    3.55271367880050092 94E-15,
    1.77635683940025046 47E-15,
    8.88178419700125232 34E-16,
    4.44089209850062616 17E-16,
    2.22044604925031308 08E-16,
    1.11022302462515654 04E-16,
    5.55111512312578270 21E-17,
    2.77555756156289135 11E-17,
    1.38777878078144567 55E-17,
    6.93889390390722837 76E-18,
    3.46944695195361418 88E-18,
    1.73472347597680709 44E-18 };

    for (int i=0; i<ANGLES_LENGTH; ++i) {
      angles[i] = angles_in[i];
    }

}


//---------------------------------------------------------------------------
```

```
//  Initialize the array kprod[ANGLES_LENGTH]
//------------------------------------------------------------------------
void Core::setKprod()
{

  double kprod_in[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 };

    for (int i=0; i<KPROD_LENGTH; ++i) {
      kprod[i] = kprod_in[i];
    }

}
```

```
::::::::::::::
Core.hpp
::::::::::::::
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <ctime>
#include <string.h>
```

```
using namespace std;


const int ANGLES_LENGTH =60;
const int KPROD_LENGTH =33;

//------------------------------------------------------------------------
//   Purpose:
//
//     Class Core Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------


// ---------------------------------------------------------------
// level      : Number of Iteration = Height of binary angle tree
// path       : path string in the binary angle tree
// threshold  : threshold for breaking the cordic algorithm's loop
// nBreak     : number of such breaking events
// nBreakInit : initialize the nBreak counter
// ---------------------------------------------------------------

class Core
{

public:

  Core();
  ~Core();

  void    setUseTh(int flag);
  void    setUseThDisp(int flag);
  void    setUseATAN(int flag);

  int     getUseTh();
  int     getUseThDisp();
  int     getUseATAN();

  //-------------------------------------------
  void    setLevel(int l);
  void    setPath(char *p);
  void    setThreshold(double th);
  void    setNBreak(int nB);
  void    setNBreakInit(int nBInit);

  int     getLevel();
  void    getPath(char *p);
  double  getThreshold();
  int     getNBreak();
  int     getNBreakInit();
```

```
  //-------------------------------------------
  void    setPi();
  void    setK();
  void    setAngles();
  void    setKprod();

  //-------------------------------------------
  double *getAngles();
  double *getKprod();

  void    initAcc () ;

  void    cordic(double*, double*, double*, int&, int&, int&, int&);
  void    cordic(double *x, double *y, double *z, int& init);
  void    cordic(double *x, double *y, double *z);



public:
    double zz;

    // xx = (*x - cosz); sum_xx += xx; sum_xx2 += (xx*xx);
    // yy = (*y - sinz); sum_yy += yy; sum_yy2 += (yy*yy);

    double xx, sum_xx, sum_xx2;
    double yy, sum_yy, sum_yy2;

    double sum_xx_n, sum_xx2_n;
    double sum_yy_n, sum_yy2_n;

    double max_err, max_errn;
    int    cnt_xx, cnt_yy;


    double    SCE,        SSE,        SRE;
    double   sSCE,       sSSE,       sSRE;
    double   mSCE,       mSSE,       mSRE;
    double  rmSCE,      rmSSE,      rmSRE;
    double minSCE,     minSSE,     minSRE;
    double maxSCE,     maxSSE,     maxSRE;




private:
  int           useTh;
  int           useThDisp;
  int           useATAN;

  int           level;
  char          path[256];

  double        threshold;
  int           nBreak;
  int           nBreakInit;

  double        pi;
  double        K;
  double        angles[ANGLES_LENGTH];
  double        kprod[KPROD_LENGTH];

};
```