```
::::::::::::::
TestBench.make
::::::::::::::
VPATH = ../Class.Core:../Class.Figures:../Class.GPData

INC = -I../Class.Core    \
      -I../Class.Figures \
      -I../Class.GPData  \


.SUFFIXES : .o .cpp .c

.cpp.o :
        g++ -c -g -I${HOME}/include ${INC} $<

.c.o :
        g++ -c -g -I${HOME}/include ${INC} $<


#---------------------------------------------------------------
# Classes
#---------------------------------------------------------------
OBJ  =                                   \


#---------------------------------------------------------------
# Angles.o : Angles.cpp Angles.hpp Core.hpp
#       g++ -c -g -I${HOME}/include ${INC} Angles.cpp

all : ${OBJ} Angles.o

SRC = cordic_tb01.cpp cordic_tb01.hpp \
      cordic_tb02.cpp cordic_tb02.hpp \
      cordic_tb03.cpp cordic_tb03.hpp \


print : TestBench.make ${SRC}
        /bin/more $? > TestBench.print

tar : TestBench.make ${SRC}
        tar cvf TestBench.tar $?


clean :
        \rm -f *.o *~ *#
::::::::::::::
cordic_tb01.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
```

```
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

#include "Core.hpp"
#include "Angles.hpp"
#include "Figures.hpp"
#include "cordic_tb01.hpp"

string GnuTerm;
string ofExt;


//-----------------------------------------------------------------------------
//   Purpose:
//
//     Explore Angles Space using Class Angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.02.13
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


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

  double pi = 3.141592653589793;
  double x, y, z;
  int    nBreak     = 0;    // number of such breaking events
  int    nBreakInit = 0;    // initialize the nBreak counter
  char   path[256]  = "";   // path string in the binary angle tree
  // ----------------------------------------------------------------
  // nIters      : Number of Iteration = Height of binary angle tree
  // nAngles     : Number of Angles    = Number of Leaf Nodes
  // th         : threshold for breaking the cordic algorithm's loop
  // ----------------------------------------------------------------
  int    nIters      = 10;
  int    nAngles     = 1 << nIters;
  double th         = 0.0;
  // ----------------------------------------------------------------
  // GnuTerm    : for gnuplot (wxt: monitor, emf: file)
  // nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
  // plotEn     : enable plotting
  // ----------------------------------------------------------------
         GnuTerm   = "wxt";

  int    nPoints  = 1000;
  int    plotEn    = 1;
  // ----------------------------------------------------------------
  // useTh      : thresholding
  // useThDisp  : display thresholding statistics
  // useATAN    : use atan() instead angles array values
  // ----------------------------------------------------------------
  int    useTh     =0;
  int    useThDisp =0;
  int    useATAN   =0;


//====================================================================
// Setting parameters by class Para constructor
//--------------------------------------------------------------------
// Class Para in Angles_tb.hpp
//--------------------------------------------------------------------

  Para P;
//--------------------------------------------------------------------

  nIters       = P.nIters;
  nAngles      = P.nAngles;
  th           = P.th;

  GnuTerm      = P.GnuTerm; // "post eps color ";  // wxt, x11 or emf
  ofExt        = P.ofExt;   // wxt, x11 or emf


  nPoints     = P.nPoints;
  plotEn       = P.plotEn;

  useTh        = P.useTh;
  useThDisp    = P.useThDisp;
  useATAN      = P.useATAN;


//====================================================================
// Setting parameters by command line arguments
//--------------------------------------------------------------------

  cout << "----------------------------------------------------------\n";
  cout << "Angles_tb    "                  << endl;
  cout << "  [nIters]    : atoi(argv[1]) " << endl;
  cout << "  [th]        : atof(argv[2]) " << endl;
  cout << "  [GnuTerm]   :     (argv[3]) " << endl;
  cout << "  [nPoints]   : atoi(argv[4]) " << endl;
  cout << "  [plotEn]    : atoi(argv[5]) " << endl;
  cout << "  [useTh]     : atoi(argv[6]) " << endl;
  cout << "  [useThDisp] : atoi(argv[7]) " << endl;
  cout << "  [useATAN]   : atoi(argv[8]) " << endl;
  cout << "----------------------------------------------------------\n";

  if (argc > 1 ) nIters    = atoi(argv[1]);
                 nAngles   = 1 << nIters;
  if (argc > 2)  th        = atof(argv[2]);
  if (argc > 3)  GnuTerm   =      argv[3];
  if (argc > 4)  nPoints   = atoi(argv[4]);
  if (argc > 5)  plotEn    = atoi(argv[5]);
  if (argc > 6)  useTh     = atoi(argv[6]);
  if (argc > 7)  useThDisp = atoi(argv[7]);
  if (argc > 8)  useATAN   = atoi(argv[8]);

//--------------------------------------------------------------------
// end of setting parameters
//====================================================================


//====================================================================

  cout << "Angle_tb parameters " << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   nIters      = " << nIters      << endl;
  cout << "   nAngles     = " << nAngles     << endl;
  cout << "   th          = " << th         << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   GnuTerm     = " << GnuTerm    << endl;
  cout << "   nPoints     = " << nPoints   << endl;
  cout << "   plotEn      = " << plotEn     << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   useTh       = " << useTh      << endl;
  cout << "   useThDisp   = " << useThDisp  << endl;
  cout << "   useATAN     = " << useATAN    << endl;
  cout << "----------------------------------------------------------\n";
```

```
//====================================================================
// # include "cordic_check.cpp"
//====================================================================

  //--------------------------------------------------------------------------
  // x = 1.0, y = 0.0, z = [-pi/2, +pi/2], step = pi/(2*nPoints)
  //--------------------------------------------------------------------------
  FILE * fp;
  int i;

  double cosz, sinz;
  double max_err=0.0, max_errn=0.0;
  double xx=0.0, yy=0.0, zz=0.0;
  double sum_xx =0.0, sum_xx2 =0.0;
  double sum_yy =0.0, sum_yy2 =0.0;
  double sum_xx_n =0.0, sum_xx2_n =0.0;
  double sum_yy_n =0.0, sum_yy2_n =0.0;
  int    cnt_xx =0, cnt_yy =0;

  //...........................
  th =  compute_threshold(nIters);
  //...........................


  Core C;

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);

  C.setLevel(nIters);
  C.setThreshold(th);

  cout << "cordic core parameters " << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   useTh     = " << C.getUseTh()      << endl;
  cout << "   useThDisp = " << C.getUseThDisp()  << endl;
  cout << "   useATAN          = " << C.getUseATAN()           << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   level            = " << C.getLevel()             << endl;
  cout << "   threshold        = " << C.getThreshold()         << endl;
  cout << "----------------------------------------------------------\n";


  //--------------------------------------------------------------------------
  // I=0: finding max_err & max_errn
  // I=1: writing scaled data into files
```

```
//--------------------------------------------------------------------------
for (int I=0; I<2; ++I) {
//--------------------------------------------------------------------------
C.setNBreak(nBreak=0);
C.setNBreakInit(nBreakInit=0);

if (I==1) fp = fopen("test.dat", "w+");

for (i=-nPoints; i<=nPoints; ++i) {
  x = 1.0;
  y = 0.0;
  z = zz = (pi / (2*nPoints)) * (i);

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

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

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

  if (I==0) {
    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++;
    }
  } else {
    fprintf(fp, "%f", zz);                            // col(1)
    fprintf(fp, " %f %f ", cosz, sinz);               // col(2,3)
    fprintf(fp, " %f %f ", x, y);                     // col(4,5)
    fprintf(fp, " %g %g ", xx/max_err, yy/max_err);   // col(6,7)
    xx /= cosz;
```

```
      yy /= sinz;
      fprintf(fp, " %g %g ", xx/max_errn, yy/max_errn); // col(8,9)
      fprintf(fp, " \n");
    }

  } /* end of i */


  if (I==0) {
    cout << "max_err  = " << max_err  << endl;
    cout << "max_errn = " << max_errn << endl;
    double avg = 0.0, mse = 0.0, rms =0.0;
    cout << "......................................................\n";
    avg = sum_xx / (2*nPoints+1);
    mse = sum_xx2 / (2*nPoints+1);
    rms = sqrt(mse);
    rms = sum_xx2;
    cout << "E[(x-cosz)]               : cos err avg = " <<  avg << endl;
    cout << "E[(x-cosz)^2]             : cos err mse = " <<  mse << endl;
    cout << "sqrt{E[(x-cosz)^2]}       : cos err rms = " <<  rms << endl;
    cout << "......................................................\n";
    avg = sum_yy / (2*nPoints+1);
    mse = sum_yy2 / (2*nPoints+1);
    rms = sqrt(mse);
    cout << "E[(y-sinz)]               : sin err avg = " <<  avg << endl;
    cout << "E[(y-sinz)^2]             : sin err mse = " <<  mse << endl;
    cout << "sqrt{E[(y-sinz)^2]}       : sin err rms = " <<  rms << endl;
    cout << "......................................................\n";
    avg = sum_xx_n / cnt_xx;
    mse = sum_xx2_n / (cnt_xx*cnt_xx);
    rms = sqrt(mse);
    cout << "E[(x-cosz)/cosz]          : cos nerr avg = " << avg << endl;
    cout << "E[(x-cosz)/cosz}^2]       : cos nerr mse = " << mse << endl;
    cout << "sqrt{E[(x-cosz)/cosz}^2]} : cos nerr rms = " << rms << endl;
    cout << "......................................................\n";
    avg = sum_yy_n / cnt_yy;
    mse = sum_yy2_n / (cnt_yy*cnt_yy);
    rms = sqrt(mse);
    cout << "E[(y-sinz)/sinz]          : sin nerr avg = " << avg << endl;
    cout << "E[(y-sinz)/sinz}^2]       : sin nerr mse = " << mse << endl;
    cout << "sqrt{E[(y-sinz)/sinz}^2]} : sin nerr rms = " << rms << endl;
  } else {
    fclose(fp);
  }

cout << "I= " << I << endl;

  //--------------------------------------------------------------------------
  }  /* end of I */
  //--------------------------------------------------------------------------
  if (plotEn ==0) return 0;

  //--------------------------------------------------------------------------
  // ** GnuTerm ** MUST Be set
  //--------------------------------------------------------------------------
  ofstream myout;

  int nemf = (GnuTerm.compare("eps") != 0);

cout << "nemf= " << nemf << endl;

  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;



  myout << "set xlabel \"uniform scaled angles\" " << endl;
  myout << "set ylabel \"error using (x, cosz) or (y, sinz)\" " << endl;
  myout << "set yrange [-1.2:+1.2]" << endl;

  myout << "set output 'tb01.error.cos.emf'" << endl;
  myout << "set title \"cos error plot ";
  myout << "(max_err=" << max_err << ")\"" << endl;

  myout << "plot 'test.dat' using 1:2 w points,  ";
  myout << "     'test.dat' using 1:4 w points,  ";
  myout << "     'test.dat' using 1:6 w points  ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;

  myout << "set output 'tb01.error.sin.emf'" << endl;
  myout << "set title \"sin error plot ";
  myout << "(max_err=" << max_err << ")\"" << endl;

  myout << "plot 'test.dat' using 1:3 w points,  ";
  myout << "     'test.dat' using 1:5 w points,  ";
  myout << "     'test.dat' using 1:7 w points  ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;

  myout << "set output 'tb01.error.all.emf'" << endl;
  myout << "set title \"cos, sin error plot ";
  myout << "(max_err=" << max_err << ")\"" << endl;

  myout << "plot 'test.dat' using 1:2 w points,  ";
  myout << "     'test.dat' using 1:3 w points,  ";
  myout << "     'test.dat' using 1:4 w points,  ";
  myout << "     'test.dat' using 1:5 w points,  ";
  myout << "     'test.dat' using 1:6 w points,  ";
  myout << "     'test.dat' using 1:7 w points   ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;


  myout << "set output 'tb01.errorn.cos.emf'" << endl;
  myout << "set title \"cos normalized error plot ";
  myout << "(max_err=" << max_errn << ")\"" << endl;

  myout << "plot 'test.dat' using 1:2 w points,  ";
  myout << "     'test.dat' using 1:4 w points,  ";
  myout << "     'test.dat' using 1:8 w points  ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;

  myout << "set output 'tb01.errorn.sin.emf'" << endl;
  myout << "set title \"sin normalized error plot ";
  myout << "(max_err=" << max_errn << ")\"" << endl;

  myout << "plot 'test.dat' using 1:3 w points,  ";
  myout << "     'test.dat' using 1:5 w points,  ";
  myout << "     'test.dat' using 1:9 w points  ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;

  myout << "set output 'tb01.errorn.all.emf'" << endl;
  myout << "set title \"cos, sin normalized error plot ";
  myout << "(max_err=" << max_errn << ")\"" << endl;

  myout << "plot 'test.dat' using 1:2 w points,  ";
  myout << "     'test.dat' using 1:3 w points,  ";
  myout << "     'test.dat' using 1:4 w points,  ";
  myout << "     'test.dat' using 1:5 w points,  ";
  myout << "     'test.dat' using 1:8 w points,  ";
  myout << "     'test.dat' using 1:9 w points   ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;




  myout.close();


cout << "* before gnuplot ... " << endl;
```

```
  system("gnuplot command.gp");



  return 0;

}
::::::::::::::
cordic_tb01.hpp
::::::::::::::
using namespace std;


#define useXSampling    10;
#define useXPartition   20;
#define useXSubtree     30;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Para
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.02
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------

// #define FOUT

//-----------------------------------------------------------------------------
// Data structure for gnuplot call
//-----------------------------------------------------------------------------

class Para {
  public:
  Para();
  int           nIters;
  int           nAngles;
  double        th;

  char          GnuTerm[256]; // "post eps color ";  // wxt, x11 or emf
  char          ofExt[256];   // wxt, x11 or emf


  int           nPoints;
  int           plotEn;

  int           useTh;
  int           useThDisp;
  int           useATAN;

};

// ----------------------------------------------------------------
// nIters      : Number of Iteration = 18Height of binary angle tree
// nAngles     : Number of Angles    = Number of Leaf Nodes
// th         : threshold for breaking the cordic algorithm's loop
// ----------------------------------------------------------------
// GnuTerm    : for gnuplot (wxt: monitor, emf: file)
// nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
// plotEn     : enable plotting
// ----------------------------------------------------------------
// UseTh      : flags for thresholding
// UseThDisp  : flags for displaying threshold statistics
// useATAN    : flags for using atan() function
// ---------------------------------------------------------------
Para::Para()  {

  nIters       = 11;
  nAngles      = 1 << nIters;
  th           = 0.001;

#ifdef FOUT
  strcpy(GnuTerm, "eps"); // eps or wxt
  strcpy(ofExt,  ".eps"); // .eps or .wxt
#else
  strcpy(GnuTerm, "wxt"); // eps or wxt
  strcpy(ofExt,  ".wxt"); // .eps or .wxt
#endif

  nPoints     = 1000;
  plotEn       = 1;

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

}
::::::::::::::
cordic_tb02.cpp
::::::::::::::
# include <cstdlib>
# include <cmath>
# include <iostream>
# include <iomanip>
# include <fstream>
# include <string.h>

using namespace std;

#include "Core.hpp"
#include "Angles.hpp"
#include "Figures.hpp"
#include "cordic_tb02.hpp"

string GnuTerm;
string ofExt;


int dispOnlyDiff    = 1;    // show only different paths (optimal vs actual)
int compareAngles   = 0;    // compare angles from two paths
int checkElemAngles = 0;    // show elementary angle information


//-----------------------------------------------------------------------------
//   Purpose:
//
//     Explore Angles Space using Class Angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.02.13
//
//
//  Author:
//
//    Young Won Lim
//
```

```
//  Parameters:
//
//-----------------------------------------------------------------------------
// double conv2angle(char * path)
// double compare_angles(  // Out: return the difference between two angles
//      double *angles,    // In:  angles array point
//      char   *path1,     // In:  optimal path
//      char   *path2)     // In:  actual path
// void compare_paths(
//      Angles *AngPt,     // In:  Angles class pointer
//      double *angles,    // In:  angles array point
//      int     i,         // In:  index to the array A
//      char   *path,      // Out: path computed by cordic Core
//      int    *diff_cnt,  // Out: the no of different paths (act vs. opt)
//      double *diff_sum ) // Out: accumulation of difference in angles
// void check_Angles_Object(Angles * AngPt, Core * CPt)
//-----------------------------------------------------------------------------
// check_Angles_Object()
// +  compre_paths()
//    +  compare_angles()
//    +  conv2angle()
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Convert the given path into angle value
//-----------------------------------------------------------------------------
double conv2angle(char * path)
{
  int i, j;
  double angle = 0.0;

  for (i=0; i<strlen(path); i++) {
    j = 1 << i;
    if (path[i] == '1') {
      angle += atan( 1. / j );
    } else {
      angle -= atan( 1. / j );
    }
  }

  return angle;

}

//-----------------------------------------------------------------------------
// Check elementary angles based on the given optimal and actual paths
//-----------------------------------------------------------------------------
double compare_angles(  // Out: return the difference between two angles
        double *angles,    // In:  angles array point
        char   *path1,     // In:  optimal path
        char   *path2)     // In:  actual path
//------------------------------------------------------------------------------
{
  double angle =0.0;
  double a1 =0.0, a2=0.0;
  double t1 =0.0, t2=0.0;
  int len1 = strlen(path1);
  int len2 = strlen(path2);
  int len = max(len1, len2);


  for (int i=0; i<len; ++i) {
    angle = angles[i];

    if (i < len1) {
      if (path1[i] == '1') a1 += angle;
      else                 a1 -= angle;
    }

    if (i < len2) {
      if (path2[i] == '1') a2 += angle;
      else                 a2 -= angle;
    }

    //=======================================
    // show elementary angle information
    //=======================================
    if (checkElemAngles) {
      cout << left << setw(10) << i;
      if (i < len1) {
        if (path1[i] == '1')
          cout << left << setw(14) << angle;
        else
          cout << left << setw(14) << -angle;
      } else {
          cout << left << setw(14) << "--";
      }

      cout << left << setw(10) << " ";
      if (i < len2) {
        if (path2[i] == '1')
          cout << left << setw(14) << angle;
        else
          cout << left << setw(14) << -angle;
      } else {
          cout << left << setw(14) << "--";
      }
      cout << endl;
    } /* end of if (checkElemAngles) { */
  }


  //=======================================
  // show elementary angle information
  //=======================================
  if (checkElemAngles) {
    cout << left << setw(10) << " ";
    cout << left << setw(14) << "-------------";
    cout << left << setw(10) << " ";
    cout << left << setw(14) << "-------------";
    cout << endl;

    cout << left << setw(10) << " ";
    cout << left << setw(14) << a1;
    cout << left << setw(10) << " ";
    cout << left << setw(14) << a2;
    cout << endl;
  } /* end of if (checkElemAngles) { */


  cout << left << setw(10) << "";
  cout << left << setw(7) << "diff=";
  cout << left << setw(12) << a1-a2;
  cout << left << setw(10) << "norm err=";
  cout << left << setw(12) << (a1-a2)/a1;

  cout << endl;


  return (a1 - a2);
}


//-----------------------------------------------------------------------------
// Compare actual and optimal paths and corresponding angles
//-----------------------------------------------------------------------------
void compare_paths(
        Angles *AngPt,     // In:  Angles class pointer
        double *angles,    // In:  angles array point
        int     i,         // In:  index to the array A
        char   *path,      // Out: path computed by cordic Core
        int    *diff_cnt,  // Out: the no of different paths (act vs. opt)
        double *diff_sum ) // Out: accumulation of difference in angles
//-----------------------------------------------------------------------------
{
    int n = AngPt->getnIters() + 2;

    double opt_angle = AngPt->A[i];
    double act_angle = conv2angle(path);
    char * opt_path  = AngPt->Ap[i];
    char * act_path  = path;
    int    opt_len   = strlen(opt_path);
    int    act_len   = strlen(act_path);

    if (i==0) {
      *diff_cnt=0;
      *diff_sum=0.0;
    }

    //===========================================
    // difference flag --> print only differences
    //===========================================
    if (dispOnlyDiff) {
      if (!strcmp(opt_path, act_path)) return;
    }

    (*diff_cnt)++;
    (*diff_sum) += abs(opt_angle - act_angle);

    cout << "i="      << left << setw(6)  << i;
    cout << " angle=" << left << setw(14) << opt_angle;
    cout << " opt="   << left << setw(n)  << opt_path;
    cout << " comp="  << left << setw(n)  << act_path ;
    cout << " used="  << left << setw(10) << act_angle;
    if (opt_len != act_len)
    cout << " (* " << opt_len << ", " << act_len << ")";
    cout << endl;


    //===========================================
    // compare angles from two paths
    //===========================================
    if (compareAngles) {
      double diff_angle;
      //...........................................................
      diff_angle = compare_angles(angles, opt_path, act_path);
      //...........................................................
    }

    return;
}


//-----------------------------------------------------------------------------
// For Leaf / All nodes, make statistics report
//-----------------------------------------------------------------------------
void check_Angles_Object(Angles * AngPt, Core * CPt)
{
  int i;
  double x, y, z;

  char path[256];
  int nBreak = 0;

  int diff_cnt = 0;
  double diff_avg = 0.0;

  for (i=0; i<AngPt->getnAngles(); ++i) {
    x = 1.0;
    y = 0.0;
    z = AngPt->A[i];

    CPt->setLevel(AngPt->getnIters());
    CPt->setNBreak(nBreak);
    CPt->setNBreakInit(i);

    //...........................................................
    CPt->cordic(&x, &y, &z);
    //...........................................................

    CPt->getPath(path);                // cordic computed path
    double *angles = CPt->getAngles(); // point to angles array

    //...........................................................
    compare_paths(AngPt, angles,  i, path, &diff_cnt, &diff_avg);
    //...........................................................

  }

  cout << "total diff angles = " << diff_cnt;
  cout << " (" << diff_cnt *100 / AngPt->getnAngles() << "%)";
  cout << "  average diff angle = " << diff_avg / diff_cnt;
  cout << endl;
}


//-----------------------------------------------------------------------------
int main (int argc, char * argv[])
{
  double pi = 3.1415926535897932;
  double x, y, z;
  int    nBreak     = 0;    // number of such breaking events
  int    nBreakInit = 0;    // initialize the nBreak counter
  char   path[256]  = "";   // path string in the binary angle tree


  // ----------------------------------------------------------------
  // nIters      : Number of Iteration = Height of binary angle tree
  // nAngles     : Number of Angles    = Number of Leaf Nodes
  // th         : threshold for breaking the cordic algorithm's loop
  // ---------------------------------------------------------------
  int    nIters      = 10;
  int    nAngles     = 1 << nIters;
  double th         = 0.0;
  // ---------------------------------------------------------------
  // GnuTerm    : for gnuplot (wxt: monitor, emf: file)
  // nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
  // plotEn     : enable plotting
  // ---------------------------------------------------------------
         GnuTerm   = "wxt";

  int    nPoints  = 1000;
  int    plotEn    = 1;
  // ---------------------------------------------------------------
  // useTh      : thresholding
  // useThDisp  : display thresholding statistics
  // useATAN    : use atan() instead angles array values
  // ---------------------------------------------------------------
  int    useTh     =0;
  int    useThDisp =0;
  int    useATAN   =0;


//====================================================================
// Setting parameters by class Para constructor
//--------------------------------------------------------------------
// Class Para in Angles_tb.hpp
//--------------------------------------------------------------------

  Para P;
//--------------------------------------------------------------------

  nIters       = P.nIters;
  nAngles      = P.nAngles;
  th           = P.th;

  GnuTerm      = P.GnuTerm; // "post eps color ";  // wxt, x11 or emf
  ofExt        = P.ofExt;   // wxt, x11 or emf


  nPoints     = P.nPoints;
  plotEn       = P.plotEn;

  useTh        = P.useTh;
  useThDisp    = P.useThDisp;
  useATAN      = P.useATAN;

//====================================================================
// Setting parameters by command line arguments
//--------------------------------------------------------------------

  cout << "----------------------------------------------------------\n";
  cout << "Angles_tb    "                   << endl;
  cout << "  [nIters]    : atoi(argv[1]) " << endl;
  cout << "  [th]        : atof(argv[2]) " << endl;
  cout << "  [GnuTerm]   :     (argv[3]) " << endl;
  cout << "  [nPoints]   : atoi(argv[4]) " << endl;
  cout << "  [plotEn]    : atoi(argv[5]) " << endl;
  cout << "  [useTh]     : atoi(argv[6]) " << endl;
  cout << "  [useThDisp] : atoi(argv[7]) " << endl;
  cout << "  [useATAN]   : atoi(argv[8]) " << endl;
  cout << "----------------------------------------------------------\n";

  if (argc > 1 ) nIters    = atoi(argv[1]);
                 nAngles   = 1 << nIters;
  if (argc > 2)  th        = atof(argv[2]);
  if (argc > 3)  GnuTerm   =      argv[3];
  if (argc > 4)  nPoints   = atoi(argv[4]);
  if (argc > 5)  plotEn    = atoi(argv[5]);
  if (argc > 6)  useTh     = atoi(argv[6]);
  if (argc > 7)  useThDisp = atoi(argv[7]);
  if (argc > 8)  useATAN   = atoi(argv[8]);

//--------------------------------------------------------------------
// end of setting parameters
//====================================================================



//====================================================================

  cout << "Angle_tb parameters " << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   nIters      = " << nIters      << endl;
  cout << "   nAngles     = " << nAngles     << endl;
  cout << "   th          = " << th         << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   GnuTerm     = " << GnuTerm    << endl;
  cout << "   nPoints     = " << nPoints   << endl;
  cout << "   plotEn      = " << plotEn     << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   useTh       = " << useTh      << endl;
  cout << "   useThDisp   = " << useThDisp  << endl;
  cout << "   useATAN     = " << useATAN    << endl;
  cout << "----------------------------------------------------------\n";
```

```
//====================================================================
// # include "cordic_check.cpp"
//====================================================================

  //....................................
  th = compute_threshold(nIters);
  //....................................

  Core C;

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);

  C.setLevel(nIters);
  C.setThreshold(th);


  // -------------------------------------------------------------
  // LeafAngles : Angles Class for leaf nodes only
  // AllAngles  : Angles Class for all nodes (internal nodes included)
  // -------------------------------------------------------------
  Angles LeafAngles(nIters, nAngles);
  Angles AllAngles(nIters, 2*nAngles-1);


  //--------------------------------------------------------------------------
  // x = 1.0, y = 0.0, z = [0, pi/2], step = pi/200
  //--------------------------------------------------------------------------
  // check_Angles_Object(&LeafAngles, &C);
  check_Angles_Object(&AllAngles, &C);




  return 0;

}
::::::::::::::
cordic_tb02.hpp
::::::::::::::
using namespace std;


//-----------------------------------------------------------------------------
#undef DISP_ONLY_DIFF    // show only different paths (optimal vs actual)
#undef COMPARE_ANGLES    // compare angles from two paths
#undef CHECK_ELEM_ANGLES // show elementary angle information
//-----------------------------------------------------------------------------
```

```
#define DISP_ONLY_DIFF
#define COMPARE_ANGLES
#define CHECK_ELEM_ANGLES


//-----------------------------------------------------------------------------


#define useXSampling    10;
#define useXPartition   20;
#define useXSubtree     30;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Para
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.02
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------

// #define FOUT

//-----------------------------------------------------------------------------
// Data structure for gnuplot call
//-----------------------------------------------------------------------------

class Para {
  public:
  Para();

  int           nIters;
  int           nAngles;
  double        th;
  char          GnuTerm[256]; // "post eps color ";  // wxt, x11 or emf
  char          ofExt[256];   // wxt, x11 or emf


  int           nPoints;
  int           plotEn;

  int           useTh;
  int           useThDisp;
  int           useATAN;

};

// ----------------------------------------------------------------
// nIters      : Number of Iteration = 18Height of binary angle tree
// nAngles     : Number of Angles    = Number of Leaf Nodes
// th         : threshold for breaking the cordic algorithm's loop
// ----------------------------------------------------------------
// GnuTerm    : for gnuplot (wxt: monitor, emf: file)
// nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
// plotEn     : enable plotting
// ----------------------------------------------------------------
// UseTh      : flags for thresholding
// UseThDisp  : flags for displaying threshold statistics
// useATAN    : flags for using atan() function
// ---------------------------------------------------------------
Para::Para()  {

  nIters       = 11;
  nAngles      = 1 << nIters;
  th           = 0.001;

#ifdef FOUT
  strcpy(GnuTerm, "eps"); // eps or wxt
  strcpy(ofExt,  ".eps"); // .eps or .wxt
#else
  strcpy(GnuTerm, "wxt"); // eps or wxt
  strcpy(ofExt,  ".wxt"); // .eps or .wxt
#endif

  nPoints     = 1000;
  plotEn       = 1;

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

}
::::::::::::::
cordic_tb03.cpp
```

```
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

#include "Core.hpp"
#include "Angles.hpp"
#include "Figures.hpp"
#include "cordic_tb03.hpp"

string GnuTerm;
string ofExt;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Explore Angles Space using Class Angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.02
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------



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


  //=================================================================
  // The following parameter values are overrided by
  // first, class Para constructor,
  // then, command line arguments
  //=================================================================

  // ----------------------------------------------------------------
  // nIters      : Number of Iteration = Height of binary angle tree
  // nAngles     : Number of Angles    = Number of Leaf Nodes
  // th         : threshold for breaking the cordic algorithm's loop
  // ----------------------------------------------------------------
  int    nIters      = 10;
  int    nAngles     = 1 << nIters;
  double th         = 0.0;
  // ----------------------------------------------------------------
  // GnuTerm    : for gnuplot (wxt: monitor, emf: file)
  // nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
  // plotEn     : enable plotting
  // ----------------------------------------------------------------
         GnuTerm   = "wxt";

  int    nPoints  = 1000;
  int    plotEn    = 1;
  // ----------------------------------------------------------------
  // useTh      : thresholding
  // useThDisp  : display thresholding statistics
  // useATAN    : use atan() instead angles array values
  // ----------------------------------------------------------------
  int    useTh     =0;
  int    useThDisp =0;
  int    useATAN   =0;


//====================================================================
// Setting parameters by class Para constructor
//--------------------------------------------------------------------
// Class Para in Angles_tb.hpp
//--------------------------------------------------------------------

  Para P;
//--------------------------------------------------------------------

  nIters       = P.nIters;
  nAngles      = P.nAngles;
  th           = P.th;

  GnuTerm      = P.GnuTerm; // "post eps color ";  // wxt, x11 or emf
  ofExt        = P.ofExt;   // wxt, x11 or emf


  nPoints     = P.nPoints;
  plotEn       = P.plotEn;
  useTh        = P.useTh;
  useThDisp    = P.useThDisp;
  useATAN      = P.useATAN;


//====================================================================
// Setting parameters by command line arguments
//--------------------------------------------------------------------

  cout << "----------------------------------------------------------\n";
  cout << "Angles_tb    "                  << endl;
  cout << "  [nIters]    : atoi(argv[1]) " << endl;
  cout << "  [th]        : atof(argv[2]) " << endl;
  cout << "  [GnuTerm]   :     (argv[3]) " << endl;
  cout << "  [nPoints]   : atoi(argv[4]) " << endl;
  cout << "  [plotEn]    : atoi(argv[5]) " << endl;
  cout << "  [useTh]     : atoi(argv[6]) " << endl;
  cout << "  [useThDisp] : atoi(argv[7]) " << endl;
  cout << "  [useATAN]   : atoi(argv[8]) " << endl;
  cout << "----------------------------------------------------------\n";

  if (argc > 1 ) nIters    = atoi(argv[1]);
                 nAngles   = 1 << nIters;
  if (argc > 2)  th        = atof(argv[2]);
  if (argc > 3)  GnuTerm   =      argv[3];
  if (argc > 4)  nPoints   = atoi(argv[4]);
  if (argc > 5)  plotEn    = atoi(argv[5]);
  if (argc > 6)  useTh     = atoi(argv[6]);
  if (argc > 7)  useThDisp = atoi(argv[7]);
  if (argc > 8)  useATAN   = atoi(argv[8]);

//--------------------------------------------------------------------
// end of setting parameters
//====================================================================



//====================================================================

  cout << "cordic_tb01 parameters " << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   nIters      = " << nIters      << endl;
  cout << "   nAngles     = " << nAngles     << endl;
  cout << "   th          = " << th         << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   GnuTerm     = " << GnuTerm    << endl;
  cout << "   nPoints     = " << nPoints   << endl;
  cout << "   plotEn      = " << plotEn     << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   useTh       = " << useTh      << endl;
  cout << "   useThDisp   = " << useThDisp  << endl;
  cout << "   useATAN     = " << useATAN    << endl;
  cout << "----------------------------------------------------------\n";



//====================================================================
// # include "cordic_check.cpp"
//====================================================================
  int rnd = 1;

  int flag = 4;
  int flag_basic        =  flag & 1;
  int flag_tscale_stat  =  flag & 2;
  int flag_uscale_stat  =  flag & 4;


  Angles * LA, * AA;
  Figures * F;

  for (int i=0; i< 5; ++i) {
     nAngles = (1 << nIters);
     nPoints = nAngles;

     LA = new Angles(nIters, nAngles);
     AA = new Angles(nIters, 2*nAngles-1);
     F = new Figures();


  //--------------------------------------------------------------------------
  if (flag_basic) {
  //--------------------------------------------------------------------------
  // b. plot_angle_tree       : plot binary angle trees
  // 1. plot_circle_angle     : plot angle vectors on a unit circle
  // 2. plot_line_angle       : plot angle vectors on a linear scal
  // 9. plot_quantization     : plot non-uniform quantization of CORDIC
  //--------------------------------------------------------------------------

if (1) LA->plot_angle_tree(5, 9);
if (1) LA->plot_circle_angle();
if (1) LA->plot_line_angle();
if (1) LA->plot_quantization();

  if (strcmp(GnuTerm.c_str(), "wxt") != 0)
    F->make_figures(flag_basic, LA->epsList, AA->epsList);


  char cmd[256];
  sprintf(cmd, "cp fig_basic.pdf fig_basic%d.pdf", i);
  system(cmd);
  sprintf(cmd, "pdftk fig_basic?.pdf cat output fig_basic_all.pdf", i);
  system(cmd);


}

  //.......................................
  LA->setUseTh(useTh);
  LA->setUseThDisp(useThDisp);
  LA->setUseATAN(useATAN);
  LA->setThreshold(th);
  //.......................................


  //--------------------------------------------------------------------------
  if (flag_tscale_stat) {
  //--------------------------------------------------------------------------
  // angle tree statistics
  //--------------------------------------------------------------------------
  // 3. calc_tscale_statistics              : find Angles Statistics  --> member data
  // 4. plot_tscale_statistics              : plot delta distribution and angle-delta
  // 5. plot_tscale_residual_angles         : plot residuals-angle and residuals-index
  //--------------------------------------------------------------------------
  int binNum =100;

if (1) LA->calc_tscale_statistics();
if (1) LA->plot_tscale_statistics(binNum);
if (1) LA->plot_tscale_residual_angles();              // cordic()


  if (strcmp(GnuTerm.c_str(), "wxt") != 0)
    F->make_figures(flag_tscale_stat, LA->epsList, AA->epsList);


  char cmd[256];
  sprintf(cmd, "cp fig_tscale.pdf fig_tscale%d.pdf", i);
  system(cmd);
  sprintf(cmd, "pdftk fig_tscale?.pdf cat output fig_tscale_all.pdf", i);
  system(cmd);



  }


  //--------------------------------------------------------------------------
  if (flag_uscale_stat) {
  //--------------------------------------------------------------------------
  // uniform scale statistics
  //--------------------------------------------------------------------------
  // 6. calc_uscale_statistics
  // 7. plot_uscale_statistics
  // 8. plot_uscale_residual_angles
  //--------------------------------------------------------------------------
#if 0
  int nPtLeaf = LA->getnAngles()*4;
#else
  int nPtLeaf = nPoints/2;
#endif

if (1) LA->calc_uscale_statistics(nPtLeaf);    // cordic()
if (1) LA->plot_uscale_statistics(nPtLeaf);
if (1) LA->plot_uscale_residual_angles(rnd);              // cordic()
if (1) LA->plot_uscale_histogram(nPtLeaf);   // cordic()

  if (strcmp(GnuTerm.c_str(), "wxt") != 0)
    F->make_figures(flag_uscale_stat, LA->epsList, AA->epsList);


  char cmd[256];
  sprintf(cmd, "cp fig_uscale.pdf fig_uscale%d.pdf", i);
  system(cmd);
  sprintf(cmd, "pdftk fig_uscale?.pdf cat output fig_uscale_all.pdf", i);
  system(cmd);

  }


     //------------------------------------------------------------------------
     delete LA;
     cout << "<<< end of delete LA " << i << endl;
     delete F;
     cout << "<<< end of delete F " << i << endl;
     //------------------------------------------------------------------------

     nIters++;

  }





  return 0;

}


::::::::::::::
```

```
cordic_tb03.hpp
::::::::::::::
using namespace std;


#define useXSampling    10;
#define useXPartition   20;
#define useXSubtree     30;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Para
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.08.02
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Data structure for gnuplot call
//-----------------------------------------------------------------------------

class Para {
  public:
  Para();

  int           nIters;
  int           nAngles;
  double        th;

  char          GnuTerm[256]; // "post eps color ";  // wxt, x11 or emf
  char          ofExt[256];   // wxt, x11 or emf


  int           nPoints;
  int           plotEn;

  int           useTh;
  int           useThDisp;
  int           useATAN;

};

// ----------------------------------------------------------------
// nIters      : Number of Iteration = 18Height of binary angle tree
// nAngles     : Number of Angles    = Number of Leaf Nodes
// th         : threshold for breaking the cordic algorithm's loop
// ----------------------------------------------------------------
// GnuTerm    : for gnuplot (wxt: monitor, emf: file)
// nPoints   : determines the number of uniform samples over [-pi/2, +pi/2]
// plotEn     : enable plotting
// ----------------------------------------------------------------
// UseTh      : flags for thresholding
// UseThDisp  : flags for displaying threshold statistics
// useATAN    : flags for using atan() function
// ---------------------------------------------------------------
Para::Para()  {

  nIters       = 13;
  nAngles      = 1 << nIters;
  th           = 0.001;

  // strcpy(GnuTerm, "wxt");
  // strcpy(ofExt,  ".wxt");
  strcpy(GnuTerm, "eps");
  strcpy(ofExt,  ".eps");


  nPoints     = 1000;
  plotEn       = 1;

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

}
```