CalcLib Example of Calc::Intg

```
Calc::Intg Example
Objective: define a test function with an exact integral. Send test function incremental
   points to 3 versions of integration: Trap (linear), Simp (parabola) & Romb (>parabola)
   Approximate a definite integral with each method. Compare for errors.
References:
   - Press, William H, et al, Numerical Recipes, 2nd Ed, Cambridge Press, 1992.
   - Spiegel, MR, et al, Schaum's Mathematical Handbook, 5th Ed, McGraw-Hill, 2018.
Author: Everett George Copyright (C) 2023
   https://www.jefgeorge.com
   07/06/2023
```

Output:

```
calc::Intg Class Example Application

exact evaluation of fn(x)       fn(x) = 1.5509+i0.32240  (exact)
trapezoid estimate of fn(x)  iTrap(x) = 1.5488+i0.33236  (2 iterations)
Simpson estimate of fn(x)    iSimp(x) = 1.5508+i0.32239  (2 iterations)
Romberg estimate of fn(x)    iRomb(x) = 1.5509+i0.32240  (4 iterations)
```

C++ Source:

```
#include "Cpx.h"                                        // complex variable definitions
#include "Intg.h"                                       // function integral objects

using namespace std;                                    // standard C++ library namespace
using namespace calc;                                   // CalcLib namespace

typedef Cpx<float> CPXf;                                // alias for <yType>

// test function data
const CPXf C1(-1.0f,-1.0f);                             // test function constant
const CPXf C2(-1.0f, 1.0f);                             // test function constant
CPXf  fn(float x) {return sin(C2*x)*C1;}                // test function
CPXf dfn(float x) {return cos(C2*x)*C1*C2;}             // test function derivative

// results print function
void results(ostream &ostr, const char *cDat, CPXf cpf, int nCnt)
{
   ostr<<"   "<<cDat<<" = ";                            // print out title
   cpf.stream(ostr)<<"  ("<<nCnt<<" iterations)";      // print data values
   ostr<<endl;                                          // end the text line
}

int main(void)
{
   // set parameters
   const float xLo   =0.0f;                             // integration lower bound
   const float xHi   =0.25f*Base::PI;                   // integration upper bound
   const float yError=0.05f;                            // integration error
   CPXf y,yTrap,ySimp,yRomb;                            // evaluation variables

   Trap<float,CPXf> iTrap(dfn,yError);                  // define trapezoid integration object
   Simp<float,CPXf> iSimp=iTrap;                        // define Simpson integration object
   Romb<float,CPXf> iRomb=iSimp;                        // define Romberg integration object

   // evaluate equivalent function at same point
   try {
      y=fn(xHi)-fn(xLo);                                // test function
      yTrap=iTrap.eval(xLo,xHi);                        // interpolated function
      ySimp=iSimp.eval(xLo,xHi);                        // differentiated function
      yRomb=iRomb.eval(xLo,xHi);                        // integrated function
   }
   catch(IntgErr& intgErr) {cout<<intgErr<<endl; return 1;}
   catch(...) {cout<<"Unknown execution error..."<<endl; return 1;}

   // print out evaluations comparing the functions
   cout.precision(5);
   cout.setf(ios::showpoint,ios::showpoint);
   cout<<endl<<"   calc::Intg Class Example Application"<<endl<<endl;
   cout<<"   exact evaluation of fn(x)       fn(x) = "; y.stream(cout)<<"  (exact)"<<endl;
   results(cout,"trapezoid estimate of fn(x)
iTrap(x)",yTrap,iTrap.getCount(Intg<float,CPXf>::LOOP_CURRENT));
   results(cout,"Simpson estimate of fn(x)
iSimp(x)",ySimp,iSimp.getCount(Intg<float,CPXf>::LOOP_CURRENT));
   results(cout,"Romberg estimate of fn(x)
iRomb(x)",yRomb,iRomb.getCount(Intg<float,CPXf>::LOOP_CURRENT));
   cout<<endl<<flush;
```

```
    return 0;
}
```