```
// PointF.cpp
#include "StdAfx.h"
#include ".\pointf.h"
#define  _USE_MATH_DEFINES
#include <math.h>

PointF::PointF(void)
: N(21) // accurate to at least 1.0e-9 for n = 21
, ao(0)
, Ga(0)
, nOrder(10)
{
}

PointF::~PointF(void)
{
}
//////////////////////////////////////////////////////////////////////
double PointF::ab[GSQN] =
{0.234526109519618537, 0.576884629301886424, 1.07244875381781771,
1.72240877644464545, 2.52833670642579507, 3.49221327302199439,
4.61645676974973095, 5.90395850417717807, 7.35812673158943653,
8.98294092506170375, 10.7830186325538887, 12.7636979795992374,
14.9311397434474381, 17.2924543449313719, 19.8558608677867421,
22.6308889326376814, 25.6286360941471892, 28.8621018735170335,
32.3466291442543635, 36.1004948095275791, 40.1457197693999658,
44.5092079943361867, 49.2243949878495185, 54.3337213335559171,
59.8925091630914400, 65.9753772877550561, 72.6876280911816123,
80.1874469777939813, 88.735340417914390, 98.8295428682851138};
double PointF::wb[GSQN] =
{0.210443107938813232, 0.235213229669848016, 0.195903335972881131,
0.129983786286070325, 0.0705786238657170351, 0.0317609125091751521,
0.0119182148348470412, 0.00373881629462726539, 9.80803311479530395e-4,
2.14864918490688578e-4, 3.92034196956540583e-5, 5.93454165850294080e-6,
7.41640471967253239e-7, 7.60456779450040262e-8, 6.35060432863269666e-9,
4.28138116682911178e-10, 2.30589914268529540e-11, 9.79937898678997989e-13,
3.23780202335994162e-14, 8.17182299282348476e-16, 1.54213380965025190e-17,
2.11979233318667315e-19, 2.05442967554444974e-21, 1.34698259129703856e-23,
5.66129410651444598e-26, 1.41856054527834908e-28, 1.91337549154900408e-31,
1.19224876047820526e-34, 2.67151121910151977e-38, 1.33861694210248403e-42};
/*  A Gaussian-Lagrrier quadrature based on 32 nodes. Only the first n
    nodes and corresponding weights are used in the program. User can
    change the number of nodes used in the program by specifying the
    value of computation n. */
double PointF::infintg(double (PointF::*grand)(double))
{
    int i;
    double vk,*pa,*pw;

    pa = ab;
    pw = wb;
    vk = (this->*grand)(0.044489365833267)*0.109218341952385;
    for(i=1;i<=N;++i)
        vk += (this->*grand)(*pa++)**pw++;
    return vk;
}
void PointF::SetNVal(int i)
{
    if(i>30)
        N = 30;
    else
        N = i;
}

double PointF::FunS(double y)
{
    double t,x,S1,S2;
    yo = y;
```

```
    x = 1.0-yo;
    so = gl/2.0+dy; // distance from centerline
    t = so*so+x*x;
    fpt = &PointF::integrandS;
    S1 = infintg(fpt)/so+4.0*so*so/(t*t);

    so = gl/2.0-dy; // distance from centerline
    t = so*so+x*x;
    S2 = infintg(fpt)/so+4.0*so*so/(t*t);
    return S1+S2;
}
// The integrand for stress due to a pair of horizontal point forces on lower edge
double PointF::integrandS(double x)
{
    double sh,chy,shy,xy,xy1,shy1,chy1,t1,t2,t3,t4;
    xy = x*(1.0-yo);
    sh = sinh(x);
    chy = cosh(xy);
    shy = sinh(xy);
    xy1 = x*yo;
    shy1 = sinh(xy1);
    chy1 = cosh(xy1);
// G(a,1-x)
    t1 = (xy1*shy-2*chy1+xy*shy1-2*chy)/(sh+x);
    t2 = (xy1*shy+2*chy1-xy*shy1-2*chy)/(sh-x);
    t1 = (x-1)*(t1+t2);
// H(a,1-x)
    t3 = (xy1*chy-shy1+xy*chy1-shy)/(sh+x);
    t4 = (xy1*chy+shy1-xy*chy1-shy)/(sh-x);
    t3 = x*(t3+t4);
    return (t1+t3)*sin(so*x);
}
////////////////////////////////////////////////////////////////////////
double PointF::add[GSQN] =
{3.8220118431826414e-4, 2.0127000924398829e-3, 4.9416273837414745e-3,
9.1619943579814602e-3, 1.4662870583408546e-2, 2.1429924043507954e-2,
2.9445506659319426e-2, 3.8688709308522369e-2, 4.9135418762999415e-2,
6.0758381392559484e-2, 7.3527274576168277e-2, 8.7408785945670025e-2,
1.0236670035882018e-1, 1.1836199441384390e-1, 1.3535293827526745e-1,
1.5329520455275442e-1, 1.7214198395246456e-1, 1.9184410740103914e-1,
2.1235017432458469e-1, 2.3360668674853737e-1, 2.5555818886887394e-1,
2.7814741173073420e-1, 3.0131542263712170e-1, 3.2500177889796658e-1,
3.4914468551848464e-1, 3.7368115641547325e-1, 3.9854717874097075e-1,
4.2367787988459235e-1, 4.4900769671886297e-1, 4.7447054664601283e-1};
double PointF::wkb[GSQN] =
{9.8072668083514134e-4, 2.2804620030062086e-3, 3.5761774958745448e-3,
4.8627309151780669e-3, 6.1366317539060523e-3, 7.3945329424689573e-3,
8.6331464938068718e-3, 9.8492388730505907e-3, 1.1039636574159522e-2,
1.2201233593772101e-2, 1.3330999262075445e-2, 1.4425986044091701e-2,
1.5483337184198697e-2, 1.6500294137953705e-2, 1.7474203758266676e-2,
1.8402525211577409e-2, 1.9282836603504086e-2, 2.0112841295499124e-2,
2.0890373895444246e-2, 2.1613405906248049e-2, 2.2280051017541744e-2,
2.2888570026572980e-2, 2.3437375375404533e-2, 2.3925035292547804e-2,
2.4350277528205763e-2, 2.4711992673367795e-2, 2.5009237054089127e-2,
2.5241235193398702e-2, 2.5407381834409172e-2, 2.5507243519348632e-2};
/*  A modified Gaussian quadrature using 41 nodes which is exact for a
    polynomial of up to order 59 */
double PointF::intek(double ak, double bk,double (PointF::*grand)(double))
{
    int i;
    double sk,vk,ad,*pa,*pw;
    sk = bk - ak;
    vk = 0.025540559720393109*(this->*grand)(ak+.5*sk);
    pa = add;
    pw = wkb;
    for(i=1;i<=GSQN;++i)
    {
        ad = sk**pa++;
```

```
        vk += ((this->*grand)(ak+ad)+(this->*grand)(bk-ad))**pw++;
    }
    return vk*sk;
}
// Integrand for the first part of the integral defined in [0,1]
double PointF::Fun1(double v)
{
    double t1,t2;
    t1 = v*(1.0-ao)/(1-ao*v);
    t2 = (1.0-ao*v)*(2.0*ao+Ga-3.0*Ga*v*ao);
    return t2*acos(t1)*FunS(ao*v);
}
// Integrand for the second part of the integral defined  in [0,pi/2]
double PointF::Fun2(double u)
{
    double su,t1,t2,t3;
    su = sin(u);
    t1 = 1.0-ao;
    t2 = t1+ao*su;
    t3 = t1*t1*(t1+(ao-Ga)*su)/(t2*t2*t2);
    return t3*FunS(ao*su/t2);
}
// Compute the integral for a given value of a
double PointF::FunF(double a)
{
    ao = a;          // store the value of normalized crack size
    Ga = 3.0*(1.0-7.0*ao)*pow(1.0-ao,5.0)/28.0;
    double f1,f2;
    gpt = &PointF::Fun1;
    f1 = intek(0.0,1.0,gpt);
    gpt = &PointF::Fun2;
    f2 = intek(0.0,M_PI_2,gpt);
    return (f1+f2)/M_PI_2;
}

double PointF::Legend(double x)
{
    double v0,v1,v2;
    int j,n;

    n = nOrder;
    x = 2.*x-1.;         /* Convert x from [0,1] to [-1,1]   */
    if(n>0)
    {
        v0 = 1.;
        v1 = x;
        for(j=2;j<=n;j++)
        {
            v2 = v1*x;
            v2 = 2.*v2-v0-(v2-v0)/j;
            v0 = v1;
            v1 = v2;
        }
    }
    else
        v1 = 1.0;
    return v1;
}// Integrand for the first part of the integral defined in [0,1]
double PointF::kFun1(double v)
{
    double t1,t2;
    t1 = v*(1.0-ao)/(1-ao*v);
    t2 = (1.0-ao*v)*(2.0*ao+Ga-3.0*Ga*v*ao);
    return t2*acos(t1)*Legend(ao*v);
}
// Integrand for the second part of the integral defined  in [0,pi/2]
double PointF::kFun2(double u)
{
```

```
    double su,t1,t2,t3;
    su = sin(u);
    t1 = 1.0-ao;
    t2 = t1+ao*su;
    t3 = t1*t1*(t1+(ao-Ga)*su)/(t2*t2*t2);
    return t3*Legend(ao*su/t2);
}
// Compute the integral for a given value of a
double PointF::kFunF(double a)
{
    ao = a;          // store the value of normalized crack size
    Ga = 3.0*(1.0-7.0*ao)*pow(1.0-ao,5.0)/28.0;
    double f1,f2;
    gpt = &PointF::kFun1;
    f1 = intek(0.0,1.0,gpt);
    gpt = &PointF::kFun2;
    f2 = intek(0.0,M_PI_2,gpt);
    return (f1+f2)/M_PI_2;
}


double PointF::CFun(double aVal)
{
    double y1,y2;
    double x,y,fm;
    y1 = FunF(aVal);
    y2 = kFunF(aVal);
    x = M_PI_2*aVal;
    y = sin(x);
    y = pow(1.0-y,3);
    fm = (0.752+2.017*aVal+0.3695*y)*sqrt(tan(x)/x)/cos(x);

    return aVal*y1*y2*fm*fm;
}

double PointF::AveStn(double aDepth)
{
    hpt = &PointF::CFun;
    return intek(0.,aDepth,hpt);
}
```