Jan 12, 16 9:07 **flows.cpp** Page 1/1

```cpp
//  file: flows.cpp
//
//  This program determines overflow and underflow limits
//
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      02-Jan-2004  original version, for 780.20 Computational Physics
//      01-Jan-2007  added file output so usable on Windows
//
//  Notes:
//   * compile with:  "g++ -o flows.x flows.cpp"
//   * adapted from: "Projects in Computational Physics" by Landau and Paez
//             copyrighted by John Wiley and Sons, New York
//             code copyrighted by RH Landau
//   * comment: very crude program which produces lots of screen output
//       * now generates file flows.out but could also from command
//          line use:  "flows.x > flows2.out"
//   * How can we figure out the limits more accurately?
//
//*****************************************************************//

// include files
#include <iostream>                // note that .h is omitted
#include <fstream>       // for file output
using namespace std;

//*****************************************************************//
const int max_iterations = 200;         // might not be big enough to cause
                                        // over and underflow
int
main ()
{
  float under = 1.;    // starting values
  float over = 1.;

  ofstream flow_out ("flows.out");   // open a "stream" to flows.out
  for (int i = 0; i < max_iterations; i++)
  {
    under /= 2.;          // divide by two
    over *= 2.;           // multiply by two

    // output both to the screen (cout) and to a file (flow_out)
    cout << i + 1 << ". under: " << under
      << " over: " << over << endl;
    flow_out << i + 1 << ". under: " << under
      << " over: " << over << endl;
  }

  flow_out.close();               // close the output stream
  return 0;                       // successful completion
}
```

Jan 03, 09 17:43 **flows0.py** Page 1/1

```python
#  file: flows0.py
#
#  This program determines overflow and underflow limits
#
#  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
#
#  Revision history:
#      26-Dec-2008  original Python version from flows2.cpp
#
#  Notes:
#   * run with:  "python flows0.py"
#   * adapted from: "Projects in Computational Physics" by Landau and Paez
#             copyrighted by John Wiley and Sons, New York
#             code copyrighted by RH Landau
#   * comment: very crude program which produces lots of screen output
#   * to output to flows_py.out:  "python flows0.py > flows_py.out"
#
#*****************************************************************#

max_iterations = 2000

under = 1.;    # starting values
over = 1.;

for i in range(0, max_iterations, 1):   # or range(max_iterations)
    under /= 2.0;                # divide by two
    over *= 2.0;                 # multiply by two
    print '%d under: %.5e over: %.5e' % (i+1, under, over)
```

```
//  file: precision.cpp
//
//  This program determines machine precision e
//   (the smallest e for which 1 + e does not equal 1)
//  Be careful when interpreting the output!
//
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      02-Jan-2004  original version, for 780.20 Computational Physics
//      02-Jan-2005  changed name from limit to precision for clarity,
//                    added test and made one a const
//      01-Jan-2007  added file output so usable on Windows
//
//  Notes:
//   * compile with:  "g++ -o precision.x precision.cpp"
//   * adapted from: "Projects in Computational Physics" by Landau and Paez
//             copyrighted by John Wiley and Sons, New York
//             code copyrighted by RH Landau
//   * comment: very crude program which produces lots of output
//
//***************************************************************************

// include files
#include <iostream>             // basic input/output functions
#include <iomanip>              // manipulators like setprecision
using namespace std;

//***************************************************************************
const int max_iterations = 600;

int
main ()
{
  float eps = 1.0;              // starting value
  const float one = 1.0;

  ofstream my_out ("precision.out");   // open a "stream" to precision.out
  for (int i = 0; i < max_iterations; i++)
  {
    eps /= 1.1;          // divide by a fixed factor
    float test = one + eps;     // is "test" different from 1.0?

    // output 12 digits in "test" to file
    my_out << setprecision (12) << test << " " << eps  << endl;

  }

  my_out.close();               // close the output stream
  return 0;                     // successful completion
}
```



```
//  file: J0_test.cpp
//
//  C++ Program to demonstrate use of the J0 Bessel function from
//   the gsl numerical library.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      12/26/03  original C++ version, modified from C version
//      01/03/05  switched to <cmath>
//      01/12/16  upgraded notes
//
//  Notes:
//   * Example taken from the GNU Scientific Library Reference Manual
//      for GSL Version 2.1.  URL: http://www.gnu.org/software/gsl/manual/
//   * Compile and link with the GSL libraries using:
//       g++ -Wall -o J0_test J0_test.cpp -lgsl -lgslcblas
//   * GSL routines have built-in
//       extern "C" {
//          <header stuff>
//       }
//      so they can be called from C++ programs without modification
//   * The answer should be J0(5) = -1.775967713143382920e-01
//
//***************************************************************************

// include files
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>

#include <gsl/gsl_sf_bessel.h>  // gsl Bessel special function header file

int
main ()
{
  double x = 5.0;   // just a random test value

  double y = gsl_sf_bessel_J0 (x);   // see the GSL manual for details

  std::cout << "J0(" << x << ") = "
            << std::setprecision(18) << std::setw(20) << y << std::endl;

  return 0;
}
```