CAAM 420 Fall 2012 Homework 6

T. Warburton

November 7, 2012

You must complete the following task by noon on Monday 11/19/12.

Your solutions to the homework must be committed to your Subversion repository in a sub-directory HW06. You will commit your files as follows:

i. Use \LaTeX and Kile to write and format your homework.

ii. Submit the following files HW06/hw06.pdf, HW06/hw06.tex.

iii. Your source code files for Q1 in

   - HW06/Makefile,
   - HW06/main.cpp,
   - HW06/symbolic.hpp.

iv. Submit the documentation directory created by Doxygen as HW06/man

You may work on your own or with one other student. If you work in a group of two you must share code with your partner via Subversion. We will check the logs to make sure that reasonable sharing and an equitable load sharing has taken place. If it is clear that one student is the sole contributor then both students will lose points.

You may consult the internet and programming texts.

\[1\] But do remember to cite all contributing sources.
Q1 Purpose: working with a C++ class, data container & operator overloading.

Your task is to create a symbolic class (and possibly other classes as needed) that makes this code work as expected without editing main.cpp.

Listing 1: Q1/main.cpp

```cpp
#include "symbolic.hpp"

main()
{
    /// demonstrate some automatic differentiation using the symbolic class
    symbolic x = 1.4, y = 2.1, z = 1.5; /// uses copy constructor
    symbolic f = (x+y)+y; /// overloaded operator+
    symbolic g = y*x*x; /// overloaded operator*
    symbolic h = sin(x*y); /// overloaded operator* and sine function
    symbolic i = sin(sin(x*y));
    symbolic j = 1./x; /// uses overloaded power function
    symbolic k = exp(1./(1.+x*y)); /// uses overloaded exponential
    symbolic l = exp(log(x*y)); /// uses overloaded natural log and exponential
    symbolic m = pow(x*y*z, x);
    cout << "dfdx = " << derivative(f, x) << endl;
    cout << "dfdy = " << derivative(f, y) << endl;
    cout << "dgdx = " << derivative(g, x) << " , exact =" << evaluate(2*y*x) << endl;
    cout << "dgdy = " << derivative(g, y) << " , exact =" << evaluate(x*x) << endl;
    cout << "dhdx = " << derivative(h, x) << " , exact=" << evaluate(y*cos(x*y)) << endl;
    cout << "dhdy = " << derivative(h, y) << " , exact=" << evaluate(x*cos(x*y)) << endl;
    cout << "didx = " << derivative(i, x) << " , exact=" << evaluate(y*cos(x*y)*cos(sin(x*y))) << endl;
    cout << "didy = " << derivative(i, y) << " , exact=" << evaluate(x*cos(x*y)*cos(sin(x*y))) << endl;
    cout << "djdx = " << derivative(j, x) << " , exact=" << evaluate(-1./(x*x)) << endl;
    cout << "dkdx = " << derivative(k, x) << " , exact=" << -evaluate(exp(1./(1+x*y))*y/pow(1+x*y, 2)) << endl;
    cout << "dkdy = " << derivative(k, y) << " , exact=" << -evaluate(exp(1./(1+x*y))*x/pow(1+x*y, 2)) << endl;
    cout << "dldx = " << derivative(l, x) << " , exact=" << evaluate(y) << endl;
    cout << "dmdx = " << derivative(m, x) << " , exact=" << evaluate(x*log(x*y*z)) << endl;
}
```
\begin{verbatim}
<< evaluate((log(x*y*z) + 1.0)*exp(x*log(x*y*z))) << endl;
\end{verbatim}

i. [65 points] Five points per correctly evaluated expression displayed in the output.

ii. [30 points] We will review your symbolic class implementation for correctness, readability and style.

iii. [10 points] Run your code with valgrind and include the output in a verbatim environment or use \verbatiminput. Explain any errors that you were not able to resolve.

iv. [10 points] Include the output from running this main code in your report using the \verbatim environment. Explain any deviations from expected output.

v. [10 points] Include your C++ header file(s) in the \LaTeX\ report using the \lstlisting environment provided in the listings package.

vi. [25 point] Make sure your symbolic class is well commented with Doxygen style comments. Use Doxygen to generate a manual for your symbolic class. Add and commit the Doxygen manual to your HW06/Q1 svn directory.

vii. [Extra credit: 20 points] Use an STL container class object inside your symbolic class to represent the list of variable dependencies and the derivatives of the expression with respect to these variables.