Number Theoretic Calculator

# Why a Number Theoretic Calculator

Much of modern cryptography is based on number theory. The cryptographic calculations not only deal with special number theoretic functions, they also deal with very large numbers. To study and practice cryptography requires a special calculator with both of these features.

# What Is It?

The calculator uses Reverse Polish Notation (RPN). I found RPN easy to program and not too difficult to use.

The calculator uses only integers. This requires the implementation of two parts for division. The first part is what we normally think of as division, but the result is truncated. The second part provides the remainder or residue of the division.

The calculator includes a special Modulus register. This holds the value of the modulus used by the modular arithmetic functions. The Modular Residue and Modular Multiplicative Inverse functions always use the modulus value in their calculations. When the Use Modulus box is checked, several function buttons change name to indicate that they are now modular functions.

The Greatest Common Divisor and division Residue functions never use the modulus.

The calculator has some elements of error detection. If the modulus is < 2, or if a modular function is selected and the modulus is not valid, a modulus error is indicated. A carriage return or space in a number entry window is not a problem, but other invalid digits, such as comma, period, or alphabetic, will cause an unhandled exception error. The user may select Continue, correct the error, and continue working.

# Number Theoretic Algorithms

The number theoretic algorithms used by the calculator may be found in Common.cpp, along with two conversion routines - one converts the input string to an MPIR integer, the other converts an MPIR integer to an output string.

Modular multiplicative inverse and greatest common divisor routines use the well known Extended Euclid algorithm .

The regular exponentiation and modular exponentiation routines use the Fast Modular Exponentiation algorithm.

# How to Make Your Own

The source files and compiled calculator are in NumberTheoreticCalculator.zip. I built the calculator using Microsoft Visual Studio 2010 (VS2010). To accommodate integer values necessary for cryptography, the calculator uses the Multiple Precision Integers and Rational (MPIR) library. Once properly installed in VS2010, the library is very easy to use and program with.

The files needed for the calculator are as follows.

1. Calc\_Form.h
2. Calc\_Form.resx
3. Calc\_Functions.cpp
4. Calc\_Functions.h
5. Common.cp
6. Common.h

Instructions on installing the MPIR library into VS2010 may be found in the document MPIR\_load\_notes\_v1.docx.

Instructions on installing the calculator into VS2010, and compiling may be found in the document Number\_Theoretic\_Calculator\_load\_notes\_v1.docx.

The file NumTheoryCalc.exe is the compiled executable. It was compiled with Visual C++ 2008 and should run on Windows XP, Vista, and 7 without additional installation.

# If You Want To Modify the Calculator

The calculator program operation is very straight forward (assuming you understand the peculiarities of Microsoft Visual Studio programming :-). Each function is assigned an enumeration in Calc\_Functions.h. The do\_Function routine in Calc\_Form.h processes all functions and passes the function number to the CalcFunc routine in Calc\_Functions.cpp. A switch statement handles the activities peculiar to the function specified. If an error is detected within a switch case, the value of errRtn is set to the correct error indicator, otherwise, errNo is returned. Finally, the correct stack scrolling is set in the value of intScroll and the switch case completes.

I had a particularly fun time getting the register scrolling to work correctly. When a number is entered in the accumulator, the previous number should auto-scroll onto the stack. This I programmed using the TextChanged event. The difficulty arose that any new digit to the accumulator triggered the TextChanged event and single digits scrolled up the stack. This was dealt with using the bolAScroll variable. After the first digit is entered, the bolAScroll value is set to false and further scrolling is disabled.

Some of the functions operate on the accumulator only and thus change its value. Yet these functions should not cause auto-scroll of the stack. In a manner similar to bolAScroll, the bolNoScroll variable disables stack scroll for the appropriate functions.

If you are adding new features, you may benefit from a definition in Common.h. There you will find MULTI\_PRES defined. If you un-define this value, the multi-precision integer class variable type mpz\_class will be redefined as 64 bit integer type. This should make variables easier to trace and debugging easier. When the calculator works with 64 bit integers, redefine MULTI\_PRES and it should work with multi-precision integers.

# Suggestions for Future Work

#### Error Handling

As mentioned above, known possible errors are not handled.

#### Number Display

Displaying large numbers can lead to confusion. We humans typically place a separator such as a comma between every three digits. The commas or other characters would need to be removed before the text input number is converted to a multi-precision number. Likewise, a multi-precision result could be displayed with a separator character every three digits.

#### Additional Functions

A nice to have function that comes to mind is binomial expansion using the accumulator (Acc) and stack 1 (S1).

If binomial expansion is implemented, a factorial function would also be nice.

#### Specific Cryptography

Paillier cryptography uses a specially defined function

Any functions special to cryptography, such as this one, or others like it, could be implemented.

#### Miller-Rabin

The Miller-Rabin primality test could be implemented. The result, either composite or possibly prime, could be displayed in the current error window.

#### Random Number Generation

Generating a random number within a range specified by stack 1 and the accumulator could be implemented.

Also, a random prime number within a specified range could be implemented.

# Works Cited

*Euclidean algorithm.* (2011, June 30). Retrieved July 7, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Euclid\_algorithm

Garrett, P. (2004). *The Mathematics of Coding Theory.* Upper Saddle River, New Jersey: Pearson Prentice Hall.

*MPIR home page*. (n.d.). Retrieved July 9, 2011, from MPIR: http://www.mpir.org/

*Reverse Polish notation.* (2011, July 4). Retrieved July 20, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Reverse\_Polish\_notation