The exhaustive problem solver solves cryptology problems of order 5. A random order 5 problem can be generated and given one possible solution by calling the solve function with random as a parameter. The user also has the option to solve a specific solution by entering solve followed by the parameter numbers with 5 numbers separated by commas and the parameter goal with the target number.

The random call generates 6 random numbers, placing 5 of them into the problem’s numbers, and the final number as the goal. From here, both the random and specific crypto problems are solved in identical fashion.

The order 5 crypto problems take advantage of recursive calls in order to determine which arithmetic will lead to the goal. The first two numbers given in the problem numbers are the first to be sent through the numerous recursive calls. The first recursive call occurs with combos.

Combos will rearrange the combinations of numbers provided throughout the process until a solution is either found or all combinations have occurred. The purpose of using combos is to ensure that all possible arithmetic can occur between the numbers given. The actual arithmetic occurs in the lowest form of the crypto function.

Within the top-level call of crypto with 5 numbers and a goal, crypto is recursively called within itself. The first time it is called within itself, crypto receives a number combination from combos. Crypto is called on the first two numbers, while creating a temporary variable, “SG”, to store the sum, product, difference, or quotient between the two numbers. The second crypto call within itself takes in the remaining three numbers, “SG”, and the goal. By attempting the crypto function on these four problem numbers, more recursion occurs. The process of recursion continues until an eventual solution is reached.

The first solution processed is then stored in the knowledge base. In order to be cleanly displayed, the solution is passed through the function displayResult. This function grabs the expressions from the results and displays them without the preceding “ex.” The expressions frequently contain other expressions within them. When an expression contains an expression, the best way to remove the inside “ex’s” is through recursion once again. When the displayResult function discovers it has another expression within itself, it will call displayResult on that expression. The “struggle” with this recursive printing is when to print the text. The text that must be written prior to the expression need to be written before the displayResult call. This ensures that the when the recursively called displayResult writes its clarified expression, it is not written until the proper time. Once the recursive displayResult is finished, the rest of the text can be written.