1. **(Sides of a Right Triangle)** Write a function that reads three nonzero integers and determines whether they are the sides of a right-angled triangle. The function should take three integer arguments and return 1(true) if the arguments comprise a right-angled triangle, and 0 (false) otherwise. Write a test program that prompts the user to enter a set of integers and display whether the set of integers can form a right triangle.
2. **(Roots of a Quadratic Equation)** A quadratic equation is any equation of the form where *a*, *b*, and *c* are the coefficients of *x*. The roots of a quadratic equation can be calculated by the formula . If the expression, , which is also called the discriminant, is positive then the equation has real roots. If the discriminant is negative, the equation has imaginary (or complex) roots. Write a function that accepts the coefficients of an equation as parameters, check if the roots are real, and calculates the roots of the equation. Write a test program that prompts the user to enter the coefficients and passes them to the function.
3. **(Even or Odd)** Write a program that inputs a series of integers and passes them one at a time to function isEven, which uses the remainder operator to determine whether an integer is even. The function should take an integer argument and return 1 if the integer is even or 0 otherwise. The input ends with zero. The program has to display the results for the input integers.
4. **(Pentagonal numbers)** A pentagonal number is defined as n(3n-1)/2 for n= 1, 2, …, and so on. Therefore, the first few numbers are 1, 5, 12, 22,… Write a function that returns a pentagonal number:

int getPentagonalNumber(int n)

For example, getPentagonalNumber(1) returns 1 and getPentagonalNumber(2) returns 5. Write a test program that uses this method to display the first 100 pentagonal numbers with 10 numbers on each line. Use the %7d format to display each number.

1. **(Display an integer reversed)** Write a function with the following prototype to display an integer in reverse order:

int reverse(int number);

For example, reverse(2345) returns 5432. Write a test program that prompts the user to enter an integer then displays its reversal.

1. Write a program that uses Newton’s method to compute the square root of a positive floating-point number:

Enter a positive number: 3

Square root: 1.73205

Let x be the number entered by the user. Newton’s method requires an initial guess y for the square root of x (we will use y=1). Successive guesses are found by computing the average of y and x/y. The following table shows how the square root of 3 would be found:

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | x/y | Average of y and x/y |
| 3 | 1 | 3 | 2 |
| 3 | 2 | 1.5 | 1.75 |
| 3 | 1.75 | 1.71429 | 1.73214 |
| 3 | 1.73214 | 1.73196 | 1.73205 |
| 3 | 1.73205 | 1.73205 | 1.73205 |

Note that the values of y get progressively closer to the true square root of x. For greater accuracy, your program should use variables of type double rather than float. Have the program terminate when the absolute value of the difference between the old value of y and the new value of y is less than the product of .00001 and y.

Hint: Call the fabs function to find the absolute value of a double.

1. **(*Perfect Numbers*)** An integer number is said to be a perfect number if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because 6=1+2+3. Write a function isPerfect that determines whether parameter number is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect.
2. **(Mersenne prime)** A prime number is called a Mersenne prime if it can be written in the form 2*p*-1 for some positive integer p. Write a program that finds all Mersenne primes with p ≤ 31 and displays the output as follows:

p 2^p - 1

2 3

3 7

5 31

…