5.26- (***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 perfect 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. Challenge the power of your computer by testing numbers much larger than 1000.

SOLUTION

```
#include<stdio.h>

int perfect(int x);

int main()
{

   int a, b, c;

   printf("\nPlease enter your range of number: ");
   scanf("%d", &a);
   printf("The list of perfect number between 1 and  %d is\n", a);

   for (c = 1; c <= a; c++) {

                b = perfect(c);
                if (b == c) {
                   printf("%d\n", c);
                }

   }

   return 0;
}

int perfect(int x)
{
   int i, sum = 0;

   for (i = 1; i < x; i++) {
                if (x % i == 0)
                   sum += i;
   }

   return sum;
}
```

5.28- (***Reversing Digits***) Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

SOLUTION

```
#include <stdio.h>

void revesesDigit();
int  main()
 {
              int i;
           revesesDigit(i);
            getche();
            return 0;
}

void revesesDigit(){
             int n;
          printf("Enter a four digit number ");
          scanf("%d",&n);
          int temp;
          temp = n;
          int d1 = temp % 10;
          temp = temp / 10;
           int d2 = temp % 10;
          temp = temp / 10;
           int d3 = temp % 10;
          temp = temp / 10;
           int d4 = temp % 10;
          temp = temp / 10;
           int d5 = temp % 10;
          temp = temp / 10;
          printf("%d%d%d%d",d1,d2,d3,d4);
    }
```

5.16- (***Exponentiation***)Write a function integerPower(base, exponent) that returns the value of $base^{Exponent}$.

For example, integerPower(3, 4) = 3 * 3 * 3 * 3. Assume that exponent is a positive, nonzero integer, and base is an integer. Function integerPower should use for to control the calculation. (Do not use any math library functions)

SOLUTION

```
#include <stdio.h>

long int integerPower(int base, unsigned int exponent);

int main(void)
{
   int b;
   unsigned int exp;

   printf("Enter base: ");
   scanf("%d", &b);
   printf("Exponent base: ");
   scanf("%u", &exp);

   printf("\nBase^Exponent = %15li\n", integerPower(b, exp));
   return 0;

}

long int integerPower(int base, unsigned int exponent)
{
   unsigned int x = 1;
   long int result = 1;

   for (x = 1; x <= exponent; x++) {
               result *= base;
   }

   return result;
}
```

5.10- (***Rounding Numbers***) An application of function floor is rounding a value to the nearest integer. The statement y = floor(x + .5); will round the number x to the nearest integer and assign the result to y. Write a program that reads several numbers and uses the preceding statement to round each of these numbers to the nearest integer. For each number processed, print both the original number and the rounded number.

SOLUTION

```
#include <stdio.h>
#include <math.h>

/* to be compiled with "gcc -lm 05_10.c" */

double myround(double number);

int main(void)
{
   double x;

   printf("Enter a number: ");
   while (scanf("%lf", &x) != EOF) {/* %lf --> double */
                printf("myround(%.3f) = %.3f\n", x, myround(x));
                printf("Enter a number: ");
   }

   return 0;

}

double myround(double number)
{
   return floor(number + .5);
}
```