TIC1001 — Introduction to Computing and Programming
National University of Singapore

Practice on Looping

Perfect Number

A perfect number is a positive integer that is equal to the sum of its proper divisors. A proper divisor is a positive integer other than the number itself that divides the number evenly (i.e. no remainder). For example, 6 is the smallest perfect number, because the sum of its proper divisors 1, 2, and 3 is equal to 6. 8 is not a perfect number because 1 + 2 + 4 is not equal to 8.
Write a function `is_perfect_number` that accepts a positive integer in the range [1, 10000] and returns true/false depending on whether the number is a perfect number or not.

In [ ]:
```
#include <stdbool.h>
bool is_perfect_number(int num)
{
    int sum = 0;
    for (int i = 1 ; i < num ; i++)
    {
        if (num % i == 0)
        {
            sum = sum + i;
        }
    }
    if (sum == num)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Invert Number

Write a function `invert_number` that reads in a positive integer, reverses the order of each of its digit and returns out the inverted value. For example, if input number is 12345, your program output should be 54321.

In [ ]:
```
int invert_number(int num)
{
    int reverse = 0, remainder;
    while (num != 0)
    {
        remainder = num % 10;
        reverse = reverse * 10 + remainder;
        num /= 10;
    }
    return reverse;
}

/*
Loop 1.
Num = 1234
Remainder = 4
Reverse = 0 * 10 + 4 = 4
Num = 123

Loop2.
Num = 123
Remainder = 3
Reverse = 4 * 10 + 3 = 43
Num = 12

Loop3.
Num = 12
Remainder = 2
Reverse = 43*10 + 2 = 432
Num = 1

Loop4.
Remainder = 1
Reverse = 432*10 + 1 = 4321
Num = 0
*/
```

Digit Counting

Write the function `number_of_digits` that will return the number of digits in an integer. You can safely assume that the integers are non-negative and will not begin with the number 0 other than the integer 0 itself.

In [ ]:
```
int number_of_digits(int num)
{
    int count = 0;
    if (num == 0)
    {
        count = 1;
    }
    else
    {
        while (num != 0)
        {
            num /= 10;
            count ++ ;
        }
    }
    return count;
}
```

nth Digit

Implement a function `nth_digit` that takes as inputs a non-negative integer num and a positive integer n. The function should return the nth digit (digit at position n) of num from the left.

In [ ]:
```
int nth_digit(int num, int n)
{
    if(num > 0 && n > 0)
    {
        //Get length of input 'num'
        int length = log10(num) + 1;

        //If digits are out of bounds,
        //e.g. input 5 digit num but asked for 6th digit
        if (num < pow(10,n-1))
        {
            return 0;
        }
        else
        {
        int a, b;

        //Look for nth digit from right
        a = num / int(pow(10,length-n));

        //Look for (n+1)th digit
        b = int(num / int(pow(10,length-n+1))) * 10;

        //Subtract to get nth digit
        int number = a - b;
        return number;
        }
    }
    else
    {
        //When input is negative
        return 0;
    }
}
```

Leibniz formula for π

In mathematics, the **Leibniz formula for π**, named after Gottfried Leibniz , states that

$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots=\frac{\pi}{4}$

Write a function leibniz_pi that takes in a positive integer specifying the number of terms to add in the Leibniz formula, and return the approximation of π.

In [ ]:
```
#include <math.h>
double leibniz_pi(int n)
{
    int i = 0;
    double sum = 0;

    for(i = 0; i <= n - 1 ; i++)
    {
        sum += double(pow(-1,i)/(2 * i + 1)) * 4;
    }

 return sum;
}
```

-END-