Рекурсия задачи и решения

```
// Рекурсивна функция изчислява n!
int FactorialRecursive(int n) {
 if (n <= 1) return 1;
 return n * FactorialRecursive(n - 1);
}

// Итеративна функция изчислява n!
int FactorialIterative(int n) {
    int sum = 1;
    if (n <= 1) return sum;
    while (n > 1) {
     sum *= n;
     n--;
    }
 return sum;
}

//--------------- Фибоначи naive рекурсивна версия ---------------
int FibonacciRecursive(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
 return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);
}

//--------------- Фибоначи итеративна версия --------------------

int FibonacciIterative(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;

    int prevPrev = 0;
    int prev = 1;
    int result = 0;

    for (int i = 2; i <= n; i++) {
      result = prev + prevPrev;
      prevPrev = prev;
      prev = result;
    }
 return result;
}
```

Оптимизация - Опашкова рекурсия

Опашкова рекурсия - частен случай на рекурсия при която всяко рекурсивно извикване се явява последна операция преди връщането от функцията. Подобен вид рекурсия се отличава с това, че може да бъде лесно заменена с итерация по формален начин и е гарантирано коректно преработване на програмния код на функцията от компилатора. Оптимизацията на опашкова рекурсия се състои в нейното преобразуване в плоска итерация. Тази опитмизация е реализирана в много от съществуващите компилатори.

Примери:

```
// Фибоначи с опашкова рекурсивна функция
int fib(int term, int val = 1, int prev = 0) {
    if(term == 0) return prev;
    if(term == 1) return val;
 return fib(term - 1, val + prev, val);
}
// факториел с опашкова рекурсивна функция
unsigned int fact_r(unsigned int n, unsigned int a) {
    if (n == 0)  return a;
    return fact_r(n - 1, n * a);
}

// предефиниране на факториел функцията в стандартен вид с един аргумент
unsigned int fact(unsigned int n)
{
   return fact_r(n, 1);
}

//Рекурсивна функция за сума
int recsum(int x) {
    if (x == 1) {
        return x;
    } else {
        return x + recsum(x - 1);
      }
}

// сума с опашкова рекурсивна функция
int tailrecsum(int x, int running_total = 0) {
    if (x == 0) {
        return running_total;
    } else {
        return tailrecsum(x - 1, running_total + x);
    }
}
```

Задачи за рекурсия

```
//Задача 1
//Да се отпечата правоъгълен триъгълник от символи, като потребителя въвежда броя
//на символите в основата и символа, чрез който да се печата триъгълника
void print_triangle(size_t, char);
a
aa
aaa
aaaa
aaaaa
aaaaaa
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaa
void print_triangle(size_t n, char ch) {
        if(n > 0) print_triangle(n-1, ch);
        for(size_t i = 0; i < n; i++)
                cout << ch;
        cout << endl;
}

//Задача 2
//Да се напише рекурсивна функция, която печата равнобедрен триъгълник от символи.
//Символът, чрез който се печата се въвежда от потребителя. Броят редове, на които
//да се разположи триъгълника, отново се въвежда от потребителя
//Пример: 4 *
//Изход:
//    *
//   ***
//  *****
// *******
void print_isosceles_triangle(size_t, size_t, char);
```

```
void print_isosceles_triangle(size_t c, size_t n, char ch) {
//функцията трябва да печата на всеки ред определен брой интервали и знаци, за да
//се получи желаният триъгълник ако броя редове за печатане е n, то на всеки ред i
//имаме да печатаме n-i интервала и i на брой символа и отново n-i интервала
//на последния ред имаме да печатаме 2*n-i символа, а на всеки ред символите са
//нечетен брой

     if (n > 0) print_isosceles_triangle(c, n-1, ch);
         for(size_t i = 0; I < 2*c - 1; i++) {
             if(i < c-n-1 || i > c-n-1+2*n) cout << " ";
             else cout << ch;
         }
        cout << endl;
}

// а. сумата на елементите
int sum(int * arr, size_t sz) {
 return (sz > 0 ) ? arr[sz-1] + sum(arr, sz-1) : 0;
}

//б. броят на ненулевите елементи
int non_zero(int * arr, size_t sz){
    if (sz > 0) return (arr[sz-1]) ? 1 + non_zero(arr, sz-1) : non_zero(arr, sz-1);
 return 0;
}

//в. най-малкият елемент
int minimum(int * arr, size_t sz, int min) {
    if (sz > 1) {
       if (arr[sz-1] < min) return minimum(arr, sz-1, arr[sz-1]);
          return minimum(arr, sz-1, min);
      }
 return min;
}

//г. извеждат елементите на масива на екрана
void print(int * arr, size_t sz) {
     if(sz > 1) print(arr, sz-1);
        cout << arr[sz-1] << endl;
}
```

```
//д. въвеждат елементите на масива от клавиатурата

void fill(int * arr, size_t) {
        if (sz > 1) fill(arr, sz-1);
        cout << "arr[" << sz - 1 << "] = ";
        cin >> arr[sz-1];
}

//е. извеждат елементите на масива в обратен ред

void print_reverse(int * arr, size_t sz) {
     if (sz >= 1) {
        cout << arr[sz-1] << endl;
        print_reverse(arr, sz-1);
     }
}

// изчислява сумата за дадено n в следния вид: 1 + 1 / 2 + 1 / 3 + . . . + 1 / n
double sum_1_over_n(int n) {
 return n == 0 ? 0 : 1.0 / n + sum_1_over_n(n-1);
}
```