Продвинутые задачи:

Эти задачи нужно оформить в 3-х файлах (1 файл на задачу) в соответствии с правилами оформления http://style.vdi.mipt.ru/CodeStyle.html и прислать мне на почту vladimir.biryukov@phystech.edu.

1. **Задача 1 - Продвинутый helloworld:** Вывести на экран строку `!\@#$^&%` Если возникнут вопросы по этой или по другим задачам, то ответы можно найти на stackoverflow. Просто загуглите, например, "how to print backslash c"или "how to print backslash c stackoverflow".

2. **Задача 2 - Целочисленные переменные:** Различные целочисленные типы языка C представлены в следующей таблице:

тип	размер (байт)	диапазон значений ($2^{\#bits}$)	модификатор
char	1	от -128 до 127	%hhd
short	2	от -32768 до 32767	%hd
int	4	примерно от -2-х миллиардов до 2-х миллиардов	%d
long	4 или 8	такой же как у int или long long в зависимости от системы	%ld
long long	8	примерно от -10^{19} до 10^{19}	%lld
unsigned char	1	от 0 до 255	%hhu
unsigned short	2	от 0 до 65535	%hu
unsigned int	4	примерно от 0 до 4-х миллиардов	%u
unsigned long	4 или 8	такой же как у unsigned int или unsigned long long	%lu
unsigned long long	8	от 0 до $2^{64} \approx 2 * 10^{19}$	%llu
16-ричная система	-	-	%x
указатель	8	$2^{64} \approx 2 * 10^{19}$	%p

   (a) **Произведение чисел:** Напишите функцию, которая вычисляет произведение 2-х положительный чисел $a < 2^{32}$ и $b < 2^{32}$. Проверьте вашу функцию на следующих значениях:

вход	выход
2 2	4
2000000000 2	4000000000
1444444444 777777777	1123456788654320988
4222222222 3777777777	15950617279827160494

   (b) **Факториал:** Напишите функцию, которая вычисляет факториал числа $n \leq 20$. Проверьте вашу функцию на следующих значениях:

вход	выход
0	1
1	1
5	120
10	3628800
20	2432902008176640000

   (c) **Размещения:** В комбинаторике размещением (из n по k) A_n^k называется упорядоченный набор из k различных элементов из некоторого множества различных n элементов. Размещения вычисляются следующим образом: $A_n^k = \frac{n!}{(n-k)!}$. Напишите функцию, которая будет вычислять размещения при условии, что $A_n^k < 2^{64}$. Проверьте вашу функцию на следующих значениях:

вход	выход
5 2	20
20 10	670442572800
30 12	41430393164160000
60 11	13679492361575040000

   (d) **Число Фибоначчи:** Наидите n-е число Фибоначчи для $n \leq 93$. Проверьте на следующих значениях:

вход	выход
5	5
20	6765
50	12586269025
60	1548008755920
75	2111485077978050
93	12200160415121876738

3. **Задача 3 - Передача в функцию по адресу:**
   Краткое введение в указатели (указатели это очень просто):

```
int main() {
    // Предположим у нас есть переменная:
    int x = 42;
    // Положение этой переменной в памяти характеризуется двумя числами - адресом и
    // размером переменной. Узнать их можно, используя операторы & и sizeof:
    printf("Size and address of x = %d and %llu\n",  sizeof(x), &x);
    // Обратите внимание, что для отображения адреса использовался модификатор %llu, так
    // как в 64 битных системах адрес это 64 битное число. Также можно было бы
    // использовать модификатор %p.

    // Для работы с адресами в языке C вводится специальный тип, который называется
    // указатель. Введём переменную для хранения адреса переменной x:
    int* address_of_x = &x;
    // Теперь в переменной address_of_x типа int* будет храниться число - адрес
    // переменной x. Если бы x был бы не int, а float, то для хранения адреса x нужно было
    // бы использовать тип float* .

    // Ну хорошо, у нас есть переменная, которая хранит адрес x. Как её дальше
    // использовать? Очень просто - поставьте звёздочку перед адресом, чтобы получить
    // переменную x.
    // *address_of_x это то же самое, что и  x
    *address_of_x += 10;
    printf("%d\n", x);
    // Запомните: &  -  по переменной получить адрес
    //            *  -  по адресу получить переменную
}
```

   Как использовать указатели для передачи в функции адреса переменной:

```
#include <stdio.h>
// Эта функция не удвоит значение
void doubler_naive(int x) {
        x *= 2;
}

// Эта функция сработает, но таким образом можно изменить только одну переменную за раз.
// К тому же, тут происходит 2 лишних копирования переменной x. Что может быть плохо,
    если переменная x будет не типа int, а, например, структурой большого размера.
int doubler(int x) {
        return 2*x;
}

// Лучший способ - передача по адресу
void doubler_by_address(int* address_of_x) {
        *address_of_x *= 2;
}

int main() {
        int x = 79;
        printf("%d\n", x);
        doubler_naive(x);
        printf("%d\n", x);
```

```
    x = doubler(x);
    printf("%d\n", x);
    doubler_by_address(&x);
    printf("%d\n", x);
}
```

(a) **Меняем переменную по адресу:** Пусть в функции main() определена переменная `float x = 4.53`. Вам нужно ввести переменную типа `float*` и сохранить в ней адрес x. А затем увеличить x в 2 раза, используя только указатель.

(b) **Куб:** Напишите функции `float cube1(float x)` и `void cube2(float* address_of_x)`, которые будут возводить значение переменной в куб двумя разными методами. Вызовите обе функции в функции main().

(c) **Swap:** Напишите функцию `void swap(int* address_of_a, int* address_of_b)`, которая меняет значения 2-х переменных типа int местами. Используйте эту функцию в функции `main()`.

(d) **Меняем указатель:** В приведённой ниже программе программист хотел написать функцию, которая бы меняла указатель address таким образом, чтобы он хранил адрес глобальной переменной. Но, к сожалению, он сделал ошибку.

```
#include <stdio.h>

int global_x = 22;

void change(int* p) {
    p = &global_x;
}

int main() {
    int x = 11;
    int* address = &x;


    printf("Address of x = %p.  Address of global_x = %p\n\n", &x, &global_x);
    printf("before: address = %p, value = %d\n", address, *address);
    // Пытаемся изменить указатель, чтобы он хранил адрес глобальной переменной
    change(address);
    printf("after:  address = %p, value = %d\n", address, *address);
}
```

Исправьте этот код. Нужно чтобы переменная `address` поменялась внутри функции `change` без изменения переменных `x` и `global_x`.