Семинар #2: Функции. Домашнее задание.

Функции без возвращаемого значения:

- **Задача 1 – Двоичный код наоборот:** Как известно, чтобы напечатать на экран число в шестнадцатиричном или восьмеричном виде, можно использовать спецификаторы `%x` или `%o`. Однако, для двоичного представления числа такого спецификатора нет. Напишите функцию `void print_binary_backwards(int number)`, которая будет печатать число в двоичном виде наоборот (так проще).

вход	выход
2	01
10	0101
97	1000011
1234567	11100001011010110 1001

Функции с возвращаемым значением:

Пример функции, которая принимает на вход 3 числа и возвращает максимальное из них:

```
#include <stdio.h>

int max3(int a, int b)
{
        int result = a;
        if (b > result)
                result = b;
        if (c > result)
                result = c;
        return result;
}
int main()
{
        printf("%d\n", max3(10, 40, -5));  // На место max(10, 40, -5) подставится результат
}
```

- **Задача 2 – Среднее гармоническое:** Написать функцию, `float harmonicmean(float a, float b)`, которая будет вычислять среднее гармоническое. Среднее гармоническое h вычисляется из формулы

$$\frac{1}{h} = \frac{1}{2}\left(\frac{1}{a} + \frac{1}{b}\right)$$

- **Задача 3 – Гамма-функция:** Гамма-функция – это обобщение понятия факториала на вещественные числа. Определяется следующим образом:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

Легко вывести, что $\Gamma(n) = (n-1)!$ для натуральных n. Написать функцию, `double gamma(double x)`, которая будет вычислять значение гамма-функции в точке x, при $x > 1$. Для вычисления интеграла использовать метод трапеций с шагом `step`. Суммирование продолжать до тех пор пока площадь трапеции превышает `eps = 1e-10` (то есть 10^{-10}). `step` и `eps` задать как константы. Понадобятся функции `pow`(возведение в степень) и `exp`(экспонента) из библиотеки `math.h`. Тесты для проверки:

вход	выход
2	1.0
8	5040.0
20	1.216451e+17
1.5	0.8862269255
2.5	1.3293403881
4.14159265	7.1880826955

- **Задача 4 – A500:** Напишите программу, которая будет печатать на экран все числа от 0 до 10^7, которые одновременно в троичном, четвертичном и пятеричном представлении имеют в своей записи лишь нули и единицы. Для решения этой задачи напишите функцию `int check_if_only_01(int num, int basse)`, которая будет проверять, что в записи числа `num` по основанию `base` есть только нули и единицы и будет возвращать `1`, если это так и `0` иначе.

Рекурсия:

- **Задача 5 – Двоичный код:** Напишите функцию `void print_binary(int number)`, которая будет печатать число в двоичном виде, используя рекурсию.

вход	выход
2	10
10	1010
97	1100001
1234567	100101101011010000111

- **Задача 6 – Число Фибоначчи:** Для вычисления чисел Фибоначчи написана следующая программа.

```c
#include <stdio.h>
unsigned long long fib(unsigned int n)
{
    if (n < 2)
        return n;
    else
        return fib(n - 1) + fib(n - 2);
}
int main()
{
    printf("%llu\n", fib(60));
}
```

Однако, в этой программе есть ошибка. При запуске программа зависает и ничего не печатает. Известно, что 60-е число Фибоначчи помещается в тип `unsigned long long` (вообще, в этот тип помещаются все числа Фибоначчи до номера `93` включительно), так что проблема не в переполнении. Найдите в чём причина ошибки. Опишите словесно эту причину и напишите работающую версию функции, вычисляющей числа Фибоначчи. Тесты для проверки:

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

Передача по указателю:

```c
#include <stdio.h>
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 отображения адреса в 16 - ричном виде.

    // 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:

```c
#include <stdio.h>
// Эта функция не удвоит значение, так как функция работает с копией передаваемого параметра
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);
}
```

- **Задача 7 – Меняем переменную по адресу:** Пусть в функции main() определена переменная `float x = 4.53`. Вам нужно ввести создать переменную типа `float*` и сохранить в ней адрес x. А затем увеличить x в 2 раза, используя только указатель.
- **Задача 8 – Куб:** Напишите функции `float cube1(float x)` и `void cube2(float* address_of_x)`, которые будут возводить значение переменной в куб двумя разными методами. Используйте обе функции в функции main() и проверьте их работу.
- **Задача 9 – Перестановка:** 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()` и проверьте её работу.
- **Задача 10 – Квадратное уравнение:** Написать функцию
  `int solve_quadratic(double a, double b, double c, double* px1, double* px2)`,
  которая будет решать квадратное уравнение. Функция должна возвращать целое число - количество корней данного уравнения. Результат решения функция должна записывать по адресам `px1` и `px2`. Если число корней равно нулю, то по этим адресам не нужно ничего записывать. Если число корней равно одному, то этот корень нужно записать по адресу `px1`. Используйте эту функцию в программе. Пользователь вводит 3 числа, программа должна напечатать:
  - `No roots`, если корней нет
  - `One root: 1.4`, если есть 1 корень, равный 1.4 (к примеру)
  - `Two roots: 3.1 and 8.7`, если есть 2 корня, равные 3.1 и 8.7 (к примеру)