

# Семинар #1: Ссылки и перегрузка операторов. Домашнее задание.

## Задача 1: Пространство имён

Создайте пространство имён по имени `myspace`. В этом пространстве имён создайте функцию `void printNTimes(const char* str, int n = 10)`, которая будет печатать строку `str` `n` раз. Для печати используйте `std::cout` из библиотеки `iostream`. Вызовите эту функцию из `main`.

## Задача 2: Куб

Напишите функцию `cubeV`, которая будет принимать одно число типа `int` и возвращать куб этого числа. Вызовите эту функцию из функции `main`, чтобы возвести переменную типа `int` в куб.

## Задача 3: Куб по ссылке

Напишите функцию `cubeR`, которая будет принимать одно число типа `int` по ссылке и возводить это число в куб. Вызовите эту функцию из функции `main`, чтобы возвести переменную типа `int` в куб..

## Задача 4: Подсчёт символов

Напишите функцию `void countLetters(const char* str, int& numLetters, int& numDigits, int& numOther)`, которая будет принимать на вход строку `str` и подсчитывать число букв и цифр в этой строке. Количество букв нужно записать в переменную `numLetters`, количество цифр – в переменную `numDigits`, а количество остальных символов – в переменную `numOther`. Вызвать эту функцию из функции `main`.

## Задача 5: Изменение структуры

Пусть у нас есть следующая структура:

```
struct Book
{
    char title[100];
    int pages;
    float price;
};
```

Напишите функцию `addPrice`, которая будет принимать на вход структуру `Book` по ссылке и некоторое число `x` типа `float`. Эта функция должна увеличивать цену переданной книги на `x`.

## Задача 6: Передача структуры по константной ссылке

Пусть у нас есть следующая структура:

```
struct Book
{
    char title[100];
    int pages;
    float price;
};
```

Напишите функцию `isExpensive`, которая будет принимать на вход структуру `Book` по константной ссылке. Эта функция должна возвращать значение типа `bool`. Если цена книги больше чем 1000, то функция должна вернуть `true`, иначе функция должна вернуть `false`. Протестируйте эту функцию в функции `main`.

## Задача 7: Математический вектор

В файлах `code/complex.h` и `code/complex.cpp` лежит реализация комплексного числа с перегруженными операторами. Используйте её в качестве примера для решения этой задачи.

- Создайте структуру `Vector3f` - вектор в трёхмерном пространстве с полями `x`, `y`, `z` типа `float` в качестве координат. Перегрузите следующие операторы для работы с вектором. Для передачи вектора в функции используйте ссылки и, там где возможно, модификатор `const`.
  - Сложение векторов (+)
  - Вычитание (-)
  - Умножение вектора на число типа `float` (число \* вектор и вектор \* число)
  - Деление вектора на число типа `float` (вектор / число)
  - Скалярное произведение (\*)
  - Унарный -
  - Унарный +
  - Проверка на равенство == (должна возвращать тип `bool`)
  - Проверка на неравенство != (должна возвращать тип `bool`)
  - Операторы += и -= (вектор += вектор)
  - Операторы \*= и /= (вектор \*= число)
  - Оператор вывода `ostream >>` вектор. Выводите вектор в виде (`x`, `y`, `z`).
  - Оператор ввода `istream <<` вектор
  - Функция `float squaredNorm(const Vector3f& a)`, которая вычисляет квадрат нормы вектора.
  - Функция `float norm(const Vector3f& a)`, которая вычисляет норму вектора.
  - Функция `void normalize(Vector3f& a)`, которая нормализует вектор `a`.
- Поместите весь ваш код в отдельный файл `vector3f.h` и подключите к файлу `main.cpp`.
- Протестируйте ваши функции:

```
#include <iostream>
#include "vector3f.h"

using std::cout, std::endl;

int main()
{
    Vector3f a = {1.0, 2.0, -2.0};
    Vector3f b = {4.0, -1.0, 3.0};
    cout << "a = " << a << endl << "b = " << b << endl;
    cout << "a + b = " << a + b << endl;
    cout << "-a = " << -a << endl;
    cout << "Scalar product of a and b = " << a * b << endl;
    a /= 5;
    cout << "a after a /= 5;" << a << endl;
    normalize(b);
    cout << "Normalized b:" << b << endl;
    a += b;
    cout << "a after a+= b;" << a << endl;
}
```

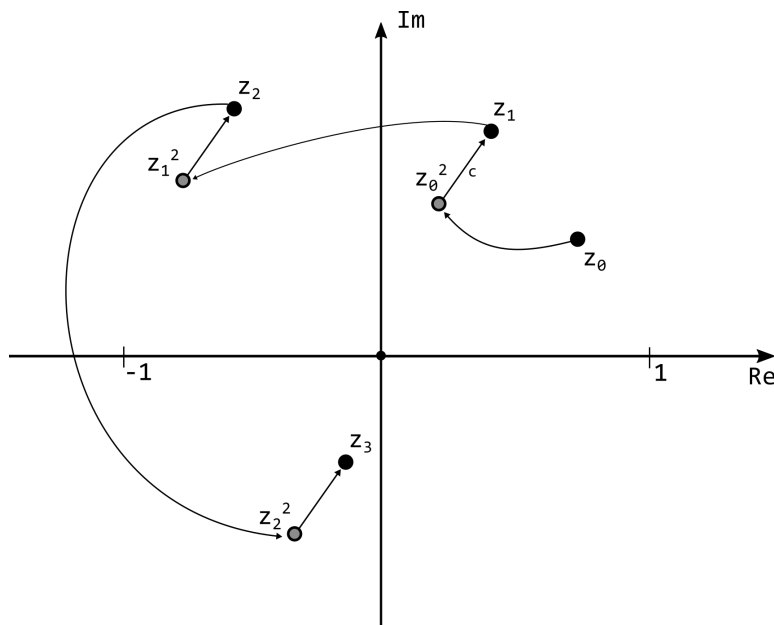
## Задача 8: Убегающая точка

- Предположим, что у нас есть комплексная функция  $f(z) = z^2$ . Выберем некоторое комплексное число  $z_0$  и будем проводить следующие итерации:

$$z_1 = f(z_0) \quad z_2 = f(z_1) \quad \dots \quad z_{k+1} = f(z_k) \quad \dots \quad (1)$$

В зависимости от выбора точки  $z_0$  эта последовательность либо разойдётся, либо останется в некоторой ограниченной области. Будем называть точку  $z_0$  убегающей, если  $z_k \rightarrow \infty$  при  $k \rightarrow \infty$ . Найдите область неубегания для функции  $z^2$ , т.е. множество всех начальных значений  $z_0$ , при которых последовательность (1) остаётся ограниченной (это можно сделать в уме).

- Julia:** Для функции  $f(z) = z^2$  эта область тривиальна, но всё становится сложнее для функции вида  $f(z) = z^2 + c$ , где  $c$  – некоторое комплексное число. Численно найдите область неубегания для функций такого вида. Для этого создайте изображение размера 800x800, покрывающую область  $[-2:2] \times [-2:2]$  на комплексной плоскости. Для каждой точки этой плоскости проведите  $N \approx 20$  итераций и, в зависимости от результата, окрасьте пиксель в соответствующий цвет (цвет можно подобрать самим, он должен быть пропорционален значению  $z_N$  – меняться от яркого если  $z_N$  мало и до черного если  $z_N$  большое). Используйте класс `Complex` и перегруженные операторы. Пример работы с изображениями в формате `ppm` можно посмотреть в файле `complex_image.cpp`. Программа должна создавать файл `julia.ppm`.



- Нарисуйте изображение для  $c = -0.4 + 0.6i$ ;  $c = -0.70 - 0.38i$ ;  $c = -0.80 + 0.16i$  и  $c = 0.280 + 0.011i$ .
- Добавьте параметры командной строки: 2 вещественных числа, соответствующие комплексному числу  $c$ , и целое число итераций  $N$ .
- Mandelbrot:** Зафиксируем теперь  $z_0 = 0$  и будем менять  $c$ . Численно найдите все параметры  $c$ , для которых точка  $z_0$  не является убегающей. Для этого создайте изображение размера 800x800, покрывающую область  $[-2:2] \times [-2:2]$  возможных значений  $c$  на комплексной плоскости. Программа должна создавать файл `mandelbrot.ppm`.
- Анимация:** Программа `complex_movie.cpp` создаёт множество изображений и сохраняет их в папку `animation` (если у вас нет такой папки – создайте её). Эти изображения представляют собой отдельные кадры будущей анимации. Чтобы их объединить в одно видео можно использовать программу `ffmpeg` (Нужно скачать тут: [www.ffmpeg.org](http://www.ffmpeg.org) и изменить переменную среды `PATH` в настройках Windows или Linux). После этого можно будет объединить все изображения в одно видео такой командой:

```
ffmpeg -r 60 -i animation/complex_%03d.ppm complex_movie.mp4
```

Создайте анимацию из изображений множеств Julia при  $c$  линейно меняющемся от  $(-1.5 - 0.5i)$  до  $i$ .