

Семинар #2: Классы и перегрузка операторов. Домашнее задание.

Задача 1. Класс Circle

Допустим, что мы хотим создать программу, которая будет работать с окружностями (это может быть игра или, например, графический редактор). Для того, чтобы сделать код более понятным и удобным в использовании, мы решили создать класс окружности. Кроме того, мы решили использовать уже ранее написанный класс точки в 2D пространстве (файл `point.hpp`). Создайте класс окружности, который будет включать следующие поля:

- Поле `center` типа `Point` – центр окружности.
- Поле `radius` типа `float` – радиус окружности.

И следующие методы:

- Конструктор `Circle(const Point& center, float radius)`, который будет задавать поля `center` и `radius` соответствующими значениями.
- Конструктор по умолчанию `Circle()` – задаются значения, соответствующие единичной окружности с центром в начале координат.
- Конструктор копирования `Circle(const Circle& circle)`
- Сеттеры и геттеры, для полей `center` и `radius`. Поле `radius` нельзя задать отрицательным числом. При попытке задания его отрицательным числом оно должно устанавливаться в значение 0.
- Метод `float area() const`, который будет возвращать площадь поверхности круга.
- Метод `float distance(const Point& p) const`, который будет возвращать расстояние от точки `p`, до ближайшей точки окружности.
- Метод `bool isColliding(const Circle& c) const`, который будет возвращать `true`, если круг пересекается с кругом `c`.
- Метод `void move(const Point& p)`, который будет перемещать кружок на вектор `p`.

Весь начальный код содержится в папке `code/circle`.

Задача 2. Математический вектор

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

- Создайте класс `Vector2f` – вектор в двумерном пространстве с полями `x` и `y` 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)
 - Скалярное произведение (*)
 - Унарный -
 - Унарный +
 - Проверка на равенство == (должна возвращать тип `bool`)
 - Проверка на неравенство != (должна возвращать тип `bool`)
 - Операторы += и -= (вектор += вектор)
 - Операторы *= (вектор *= число)

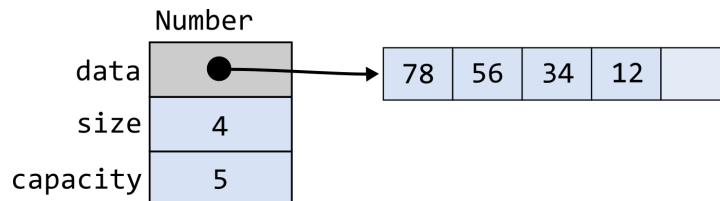
- Оператор вывода `ostream >>` вектор. Выводите вектор в виде `(x, y)`.
- Оператор ввода `istream <<` вектор
- Протестируйте ваши функции:

```
#include <iostream>
#include "vector2f.h"
using std::cout, std::endl;

int main()
{
    Vector2f a = {1.0, 2.0};
    Vector2f b = {4.0, -1.0};
    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;
    a += b;
    cout << "a after a+= b;" << a << endl;
}
```

Задача 3. Большие числа

Стандартные целочисленные типы данных, такие как `int` имеют фиксированный небольшой размер. Соответственно значения, которые можно хранить в переменных этих типов ограничены. Типичное максимальное значение `char` равно $2^7 - 1 = 127$, тип `int` обычно ограничен $2^{31} - 1 = 2147483647$ и даже тип `unsigned long long` имеет ограничение в $2^{64} - 1 \approx 1.8 * 10^{19}$. Хранить действительно большие числа в этих типах невозможно. В этом задании нужно создать класс, с помощью которого можно будет удобно складывать и умножать большие целые положительные числа. Начальный код этого класса содержится в папке `code/number`.



Представление числа 12345678 в памяти с помощью нашего класса `Number`

Подзадачи:

1. **Конструктор по умолчанию:** Напишите конструктор по умолчанию `Number()`, который будет создавать число равное нулю.
2. **Конструктор копирования:** Напишите конструктор копирования `Number(const Number& n)`.
3. **Конструктор из строки:** Напишите конструктор `Number(const std::string& str)`, который будет создавать большое число на основе строки. Предполагаем, что на вход конструктору всегда идёт корректная строка. Например, число из примера можно будет создать так:

```
Number a("12345678");
```

или так:

```
Number a = "12345678";
```

4. **Присваивание:** Напишите оператор присваивания `Number& operator=(const Number& n)`.
5. **Сложение:** Напишите и протестируйте операторы сложения `operator+` и оператор присваивания сложения `operator+=`. Реализовывать оба этих оператора с нуля необязательно. Ведь, если написан один из этих операторов, то очень просто написать другой.
6. **Числа Фибоначчи:** Числа Фибоначчи задаются следующим образом:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

Используйте класс `Number`, чтобы вычислить F_{1000} . Правильный ответ:

```
F(1000) = 43466557686937456435688527675040625802564660517371780402481729089536555417949051890
40387984007925516929592259308032263477520968962323987332247116164299644090653318793829896964992
8516003704476137795166849228875
```

7. **Четность:** Напишите метод `bool isEven() const`, который будет проверять является ли наше число чётным и, если это верно, возвращает `true`, в ином случае возвращает `false`.
8. **Произведение:** Напишите метод `Number operator*(const Number& n) const` - оператор умножения одного числа `Number` на другое. Протестируйте вашу функцию на различных примерах (умножение большого числа на большое, умножение большого числа на небольшое, умножение двух небольших чисел и т. д.).
9. **Факториал:** Используйте написанный оператор для вычисления факториала от 1000.
Правильный ответ:

```
1000! = 40238726007709377354370243392300398571937486421071463254379991042993851239862902059
2044208486969404800479988610197196058631666872994808558901323829669944590997424504087073759
9188236277271887325197795059509952761208749754624970436014182780946464962910563938874378864
8733711918104582578364784997701247663288983595573543251318532395846307555740911426241747434
9347553428646576611667797396668820291207379143853719588249808126867838374559731746136085379
5345242215865932019280908782973084313928444032812315586110369768013573042161687476096758713
4831202547858932076716913244842623613141250878020800026168315102734182797770478463586817016
4365024153691398281264810213092761244896359928705114964975419909342221566832572080821333186
1168115536158365469840467089756029009505376164758477284218896796462449451607653534081989013
8544248798495995331910172335555660213945039973628075013783761530712776192684903435262520001
5888535147331611702103968175921510907788019393178114194545257223865541461062892187960223838
9714760885062768629671466746975629112340824392081601537808898939645182632436716167621791689
0977991190375403127462228998800519544441428201218736174599264295658174662830295557029902432
4153181617210465832036786906117260158783520751516284225540265170483304226143974286933061690
8979684825901254583271682264580665267699586526822728070757813918581788896522081643483448259
9326604336766017699961283186078838615027946595513115655203609398818061213855860030143569452
7224206344631797460594682573103790084024432438465657245014402821885252470935190620929023136
4932734975655139587205596542287497740114133469627154228458623773875382304838656889764619273
838149001407673104466402598994902222176590433990188601856652648506179970235619389701786004
0811889729918311021171229845901641921068884387121855646124960798722908519296819372388642614
8396573822911231250241866493531439701374285319266498753372189406942814341185201580141233448
2801505139969429015348307764456909907315243327828826986460278986432113908350621709500259738
9863554277196742822248757586765752344220207573630569498825087968928162753848863396909959826
2809561214509948717012445164612603790293091208890869420285106401821543994571568059418727489
9809425474217358240106367740459574178516082923013535808184009699637252423056085590370062427
124341690900415369010593398383577793941097002775347200000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

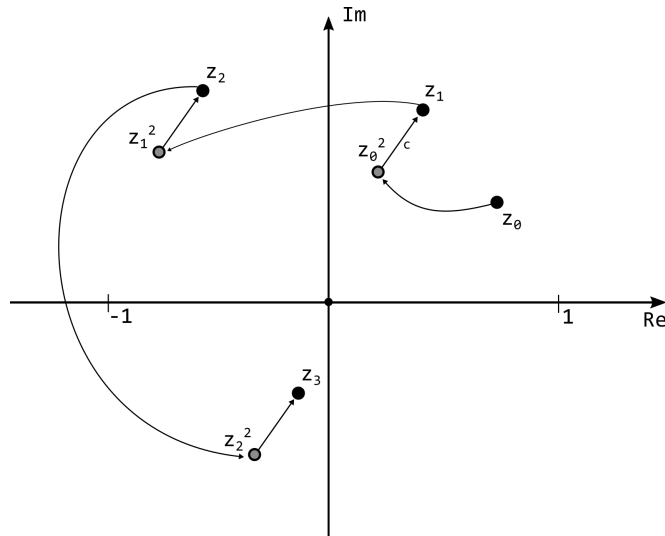
10. **Большие числа с использованием вектора:** Напишите новый класс `Integer`, который будет делать всё то же самое, что и `Number`, но не будет самостоятельно выделять память в куче. Вместо этого для хранения разрядов числа класс `Integer` будет использовать внутри себя класс `std::vector`. Этот класс должен находиться в новом файле `integer.hpp`.

Задача 4. Убегающая точка

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

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

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 $k \rightarrow \infty$. Область неубегания для функции z^2 , т.е. множество всех начальных значений z_0 , при которых последовательность (??) остаётся ограниченной, тривиальна – это единичный круг. Но всё становится сложнее для функции вида $f(z) = z^2 + c$, где c – некоторое комплексное число. Для такой функции найти область неубегания можно только численно.



1. Множество Жюлиа

Численно найдите область неубегания для функций вида $f(z) = z^2 + c$, где c – некоторое комплексное число. 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). Нарисуйте изображения для $c = -0.4 + 0.6i$; $c = -0.70 - 0.38i$; $c = -0.80 + 0.16i$ и $c = 0.280 + 0.011i$.

Программа должна создавать изображения в формате `.ppm`. Для работы с изображениями в формате `.ppm` используйте класс `Image` из файла `code/fractal/image.hpp`. Для просмотра изображений в формате `.ppm` можно использовать программу `IrfanView`. Для комплексных чисел используйте класс `Complex` из файла `code/fractal/complex.hpp`. Начальный код для задачи лежит в файле `code/fractal/complex_image.cpp`.

- Добавьте параметры командной строки: 2 вещественных числа, соответствующие комплексному числу c , и целое число итераций N .

3. Множество Мандельброта

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`.

4. 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`. (`ffmpeg` можно установить с помощью `MSYS2`. Или можно скачать тут: www.ffmpeg.org и изменить переменную среды `PATH` в настройках системы.) 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 .