Семинар #2: Инкапсуляция. Домашнее задание.

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 (файлы `point.h` и `point.cpp`). Создайте класс окружности, который будет включать следующие методы:

- Конструктор `Circle(const Point& acenter, float aradius)`, 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`.
- Метод `float get_area() const`, который будет возвращать площадь поверхности круга.
- Метод `float get_distance(const Point& p) const`, который будет возвращать расстояние от точки `p`, до ближайшей точки окружности.
- Метод `bool is_colliding(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 `0circle`. При компиляции нужно указывать все `.cpp` файлы, которые вы хотите скомпилировать:

```
g++ main.cpp point.cpp
```

- Создайте файлы `circle.h` и `circle.cpp` и перенесите реализацию класса окружности из файла `main.cpp` в эти файлы.

Класс Number (большое число)

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

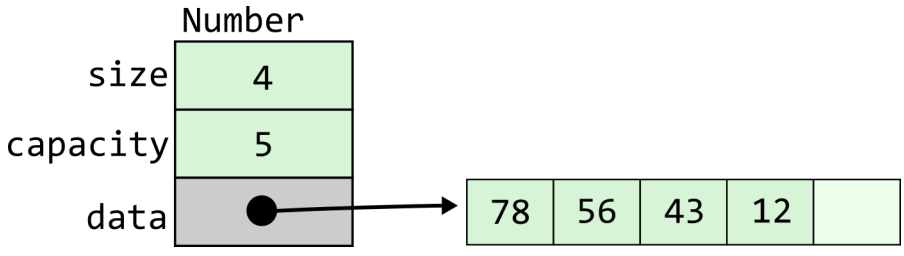

Рис. 1: Представление числа 12345678 в памяти с помощью нашего класса Number

Задания:

- **Числа Фибоначчи:** Числа Фибоначчи задаются следующим образом:

$$fib_1 = 1$$
$$fib_2 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

  Используйте уже написанный класс `Number`, чтобы вычислить fib_{1000}. Правильный ответ:

```
fib(1000) = 43466557686937456435688527675040625802564660517371780402481729089536555417949051890
40387984007925516929592259308032263477520968962323987332247116164299644090653318793829896964992
85160037044761377951668492288 75
```

- **Деструктор:** Напишите деструктор для класса `Number`, чтобы устранить утечки памяти.

- **Конструктор по умолчанию:** Напишите конструктор по умолчанию `Number()`, который будет создавать число равное нулю. Проверьте его работу.

- **Четность:** Напишите метод `bool is_even() const`, который будет проверять является ли наше число чётным и, если это верно, возвращает `true`, в ином случае возвращает `false`.

- **Произведение:** Напишите метод `Number operator*(const Number& right) const` - оператор умножения одного числа `Number` на другое. Протестируйте вашу функцию на различных примерах (умножение большого числа на большое, умножение большого числа на небольшое (< 100) или на ноль, умножение двух небольших чисел и т. д.).

- **Факториал:** Используйте написанный оператор для вычисления факториала от 1000.
  Правильный ответ:

```
1000! = 40238726007709377354370243392300398571937486421071463254379991042993851239862902059
20442084869694048004799886101971960586316668729948085589013238296699445909974245040870737591
91882362772718873251977950595099527612087497546249704360141827809464649629105639388743788647
87337119181045825783647849977012476632889835955735432513185323958463075557409114262417474349
93475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795
53452422158659320192809087829730843139284440328123155861103697680135730421616874760967587134
48312025478589320767169132448426236131412508780208000261683151027341827977704784636586817016
43650241536913982812648102130927612448963599287051149649754199093422215668325720808213331861
16811553615836546984046708975602900950537616475847728421889679646244945160765353408198901 3
85442487984959953319101723355556602139450399736280750137837615307127761926849034352625200015
88885351473316117021039681759215109077880193931781141945452572238655414610628921879602238389
97147608850627686296714667469756291123408243920816015378088989396451826324367161676217916890
97799119037540312746222899880051954444142820121873617459926429565817466283029555702990243 2
41531816172104658320367869061172601587835207515162842255402651704833042261439742869330616 90
89796848259012545832716822645806652676995865268227280707578139185817888965220816434834482 59
93266043367660176999612831860788386150279465955131156552036093988180612138558600301435694 52
72242063446317974605946825731037900840244324384656572450144028218852524709351906209290231 36
49327349756551395872055965422874977401141334696271542284586237738753823048386568897646192 73
83814900140767310446640259899490222221765904339901886018566526485061799702356193897017860 04
08118897299183110211712298459016419210688843871218556461249607987229085192968193723886426 14
83965738229112312502418664935314397013742853192664987533721894069428143411852015801412334 48
28015051399694290153483077644569099073152433278288269864602789864321139083506217095002597 38
98635542771967428222487575867657523442202075736305694988250879689281627538488633969099598 26
28095612145099487170124451646126037902930912088908694202851064018215439945715680594187274 89
98094254742173582401063677404595741785160829230135358081840096996372524230560855903700624 27
12434169090041536901059339838357779394109700277534720000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000
```

- ***Числа-градины:** Возьмём некоторое число n и будем последовательно применять к нему следующую функцию:

$$f(n) = \begin{cases} n/2, & \text{если n - четное} \\ 3n+1, & \text{если n - нечетное} \end{cases}$$

В результате получится некоторая последовательность. Например, если мы взяли число $n = 7$, то получится последовательность:

```
7 -> 22 -> 11 -> 34 -> 17 -> 52 -> 26 -> 13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

Последовательность доходит до 1. Вам нужно написать функцию, которая будет по начальному числу находить длину такой последовательности (`steps`) и максимальное число в этой последовательности(`max`). Например, для числа 7, максимальное число в последовательности будет равно 52, а длина последовательности – 16.
Напишите программу, которая будет по начальному числу находить длину последовательности и максимальный элемент в ней.

Тесты для проверки:

```
n = 7                    steps = 16;      max = 52
n = 256                  steps = 8;       max = 256
n = 1117065              steps = 527;     max = 2974984576
n = 4761963248413673697  steps = 2337;    max = 9926927712374950744648

n = 905607926569729475824397856089724657896289745872640562846587217 71
steps = 1630;
max = 773658021643749360792171137214151494851244403993540980838080564520
```

Для решения этой задачи нужно написать необходимые конструкторы, методы и перегруженные операторы (оператор сравнения, деление на 2).