

# Контрольная работа. Динная Арифметика. Классы.

## Вариант 2.

- A. На вход программе подаются два числа —  $A$  и  $B$ .  $A$  — любое число из интервала  $0 \leq A \leq 10^{44}$ . имеет следующий вид: в старших разрядах у него  $k$  девяток, за ними следует некоторая цифра;  $0 \leq k \leq 16$  Например,  $B = 3, B = 90, B = 999995$ . Вычислить произведение  $A$  и  $B$ , не используя операцию умножения.

| Input       | Output  |
|-------------|---------|
| 57<br>91    | 5187    |
| 57<br>99994 | 5699658 |

В дальнейших задачах вам необходимо реализовать некоторый класс. Причем описание класса уже есть в системе. Вам необходимо дописать `main()` и реализацию методов класса. Например, в условии дан код:

```
#include<iostream>
#include<vector>

class SuperClass {
private:
    int eniki_, beniki_;
    std::vector<int> eli_vareniki_;

public:
    SuperClass () : eniki_(0), beniki_(0), eli_vareniki_(std::vector<int>(10, 5)) {
    }

    void Print() const;
    int SuperFunction(const std::vector<int>& param, int pam_pam);

    //some more important methods
```

Ваш код, который вы пошлете в систему, будет дописан к этому, получившаяся программа будет запускаться.

Обратите внимание, что **описание класса не закончено**, в конце нет `};`. Это значит, что вы можете дописать еще несколько методов класса.

Таким образом, программа, которую вы будете сдавать в систему, должна выглядеть примерно так:

```
    // some methods, if you need it
}; // DON'T FORGET THIS LINE

int main() {
    // great code, using class SuperClass
    return 0;
}

void SuperClass::Print() const {
    std::cout << "I love to print\n";
}

int SuperClass::SuperFunction(const std::vector<int>& my_vec, int my_int) {
    // your code here
    return 123456789;
}
```

В дальнейших задачах вам надо написать класс для игры **ПьяницаНавсегда**. В игре участвуют два игрока. У каждого есть колода карт (в нашем случае будут просто целые числа от  $-10^9$  до  $10^9$ ). Каждый игрок достает сверху колоды по одной карте, если они равны, достают следующие, и так до тех пор, пока карты не будут различными, либо у одного из игроков не закончатся карты. В итоге, все карты, которые лежат на столе забирает игрок, чья карта оказалась больше (или у которого еще есть карты) и кладет их вниз своей колоды. Далее процедура повторяется. Если у одного из игроков кончились

карты, он проигрывает. Если карты закончились одновременно у обоих игроков, объявляется ничья.

В. Реализуйте класс для игрока в пьяницу. Представьте, что в реализации класса вы храните карты в обычном векторе. Тогда каждый раз, когда мы вытаскиваем карты и забираем карты со стола, придется циклически сдвигать наш вектор. Это неэффективно. Вообще говоря, для подобной задачи разумно использовать специальную структуру данных — очередь (`std::queue`), но мы ее использовать не будем. Поэтому мыведем два массива, в одном будет храниться верхняя часть колоды игрока, во втором — нижняя. Доставать карты мы будем из верхней части, а складывать в нижнюю. И лишь изредка (когда в верхней части будут кончаться карты) мы будем перекладывать все элементы из нижней части в верхнюю. При этом важно, чтобы последний элемент ‘верхнего’ вектора соответствовал самой верхней карте, а последний элемент ‘нижнего’ вектора соответствовал самой нижней карте. Только в этом случае у нас получится эффективно доставать верхнюю карту и складывать карты вниз. Другими словами, нулевые элементы ‘верхнего’ и ‘нижнего’ векторов лежат в колоде рядом.

В этой задаче нужно реализовать методы вытаскивания, добавления карты и печати всех карт игрока. Для того, чтобы упростить вам задачу, функция `Reallocate()`, которая перекладывает все карты из нижней части в верхнюю будет реализована (и автоматически приклеится системой при запуске программы на сервере). Ее нужно вызывать, когда в верхней части не остается элементов.

Печать колоды должна печатать, начиная с верхней карты. Печать не должна вызывать `Reallocate()`.

Описание класса к первой задаче:

```
#include<iostream>
#include<vector>
#include<string>

class Alcoholic {
private:
    std::vector<int> top_, bottom_;

    void Reallocate();

public:
    Alcoholic(const std::vector<int>& cards): top_(cards), bottom_(std::vector<int>()) {
    }

    int PopCard();           // возвращает верхнюю карту и удаляет ее из колоды.
                           // Если колода пуста, должна вернуть -1000000001

    void PushCard(int card); // добавляет карту вниз колоды
    void Print() const;
```

```
void Alcoholic::Reallocate() {
    for (auto rit = bottom_.rbegin(); rit != bottom_.rend(); ++rit) {
        top_.push_back(*rit);
    }
    bottom_.clear();
}
```

В первой строке входного файла записано количество карт игрока. В следующей строке сами карты. Первой написана верхняя карта. В следующей строчке написано число  $m$  — количество операций с колодой. Далее идет  $m$  строк, в каждой из которых операция, которую нужно выполнить.

- **add  $x$**  — нужно добавить карточку с числом  $x$ .
- **pop** — нужно вытащить верхнюю карту и напечатать ее в ответ.
- **print** — нужно напечатать всю колоду, начиная с верхней карты.

Если в какой-то момент задается запрос, который невозможно выполнить (просят вытащить карту из пустой колоды), надо напечатать `GameOver` и закончить выполнение программы.

| Input | Output   |
|-------|----------|
| 2     | 1 2 3    |
| 1 2   | 1        |
| 7     | 2        |
| add 3 | 3        |
| print | GameOver |
| pop   |          |
| pop   |          |
| pop   |          |
| pop   |          |
| pop   |          |

- C. Реализуйте одну фазу игры пьяница. Первый игрок (*\*this*) выкладывает карту на стол, потом второй игрок выкладывает карту. При равенстве выкладывают еще по карте. Карты первого игрока всегда ниже карт второго игрока. Победитель берет карты со стола, переворачивает и кладет вниз своей колоды. То есть, нижней картой победителя оказывается последняя из выложенных карт второго игрока.

```
#include<iostream>
#include<vector>
#include<string>

class Alcoholic {
private:
    std::vector<int> top_, bottom_;

    void Reallocate();
    int PopCard();
    void PushCard(int card);

public:
    Alcoholic(const std::vector<int>& cards): top_(cards), bottom_(std::vector<int>()) {
    }

    void Print() const;
    void MakeTurn(Alcoholic& enemy);
};
```

Обратите внимание, что `PopCard` и `PushCard` переехали в `private`-часть класса. Видимо, чтобы игроки не жульничали во время игры.

В первых двух строках входного файла количество карт и сами карты первого игрока. В третьей и четвертой строках количество карты и карты второго игрока. В ответ выведите карты первого и второго игрока после одной фазы игры. В случае, если у кого-то закончились карты, выводите пустую строчку.

| Input                    | Output     |
|--------------------------|------------|
| 3<br>1 2 3<br>1<br>4     | 2 3<br>1 4 |
| 3<br>1 1 1<br>2<br>1 1   | 1 1 1 1 1  |
| 3<br>1 1 1<br>3<br>1 1 1 |            |

- D. Реализуйте функцию `Play()` для игры двух игроков в пьяницу. Как известно, игра в пьяницу может затянуться, поэтому играть до победы не всегда позволяет время. Функция должна принимать на вход количество раундов, через которое точно надо остановиться. На вход подаются количество карт и карты первого игрока, количество карт и карты второго игрока, в пятой строчке количество ходов. Программа должна вывести колоду первого и второго игроков через заданное количество ходов. Если игра закончилась раньше, выведите какой игрок победил (**First** или **Second**)

или Draw в случае ничьей.

```
#include<iostream>
#include<vector>
#include<string>

class Alcoholic {
private:
    std::vector<int> top_, bottom_;

    void Reallocate();
    int PopCard();
    void PushCard(int card);

public:
    Alcoholic(const std::vector<int>& cards): top_(cards), bottom_(std::vector<int>()) {
    }

    void Print() const;
    void MakeTurn(Alcoholic& enemy);
    void Play(Alcoholic& enemy, int rounds);
};
```

| Input             | Output  |
|-------------------|---------|
| 3 1 2 3 1 4 3     | 2 1 3 4 |
| 3 1 1 1 2 1 1 5   | First   |
| 3 1 1 1 3 1 1 1 3 | Draw    |