Контрольная работа. Динная Арифметика. Классы.
Вариант 1.

A. На вход программе подаются два числа A и B. A — любое число из интервала $0 \leqslant A \leqslant 10^{44}$. имеет следующий вид: в старшем разряде у него 1, за ней следуют k нулей, за ними само число k; $0 \leqslant k \leqslant 16$ Например, $B = 10, B = 1000005$. Вычислить произведение A и B, не используя операцию умножения.

Input	Output
57 10	570
57 100004	5700228

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 `Герои++`. В игре сражаются две армии. Каждая армия состоит из нескольких одинаковых существ. У каждого существа есть урон и количество жизни. Когда встречаются две армии, они начинают по очереди атаковать друг друга, пока одна из них не погибнет полностью.
Подробнее, когда одна армия атакует вторую, она наносит суммарный урон, равный `урон существа` $\times$ `численность армии`. После этого часть второй армии погибает (возможно,

конечно, никто не погибнет, если урона недостаточно), а один из воинов, возможно, потеряет часть здоровья. Другими словами, существа не умеют распределять урон. То есть, в каждой армии может быть только один воин с неполным здоровьем. Например, если войско состояло из 5–ти воинов, по 3 здоровья у каждого, и по ним нанесли урон 10, то трое воинов погибнут, а один останется жить с 2 жизнями. Выжить всем с 1 здоровья не получится. Если эту армию будут атаковать дальше, то в первую очередь урон будет получать уже раненый воин.

B. Реализуйте класс для армии. Поля класса должны быть следующими: название воинов, количество, здоровье и урон каждого воина. Также понадобится здоровье последнего воина (которое может оличаться от максимального здоровья). Количество, урон и здоровье это некоторые положительные числа не больше 10^9. В этой задаче необходимо реализовать две функции — одна из них будет возвращать урон армии, вторая 'получать урон'. Конечно, вам могут понадобиться другие функции, например, для ввода/вывода.
Итак, на первой строке вводятся название воина, количество, начальное здоровье и урон. На второй — количество ударов, которое будет нанесено по армии. Далее несколько чисел — сила каждого удара. После каждого удара надо написать, сколько воинов погибло и какой урон оставшиеся могут нанести (см. пример). Если армия мертва, а ее продолжают бить, то погибае 0 воинов, и урон она может нанести тоже нулевой.
Описание класса к первой задаче:

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


class Army {
 private:
   int size_;
   int hp_, dmg_;
   int last_hp_;
   std::string name_;

 public:
   Army(const std::string& name, int n, int hp, int dmg):
       name_(name), size_(std::max(n, 0)), hp_(hp), last_hp_(hp), dmg_(dmg)
   {}

   int64_t TotalDamage() const; // обратите ВНИМАНИЕ, что урон может не поместиться в int
   int BeHit(int64_t dmg);
```

Input	Output
Dragon 3 50 15 4 45 55 100 100	0 45 2 15 1 0 0 0

C. Реализуйте битву двух армий с описанием происходящего. Для этого понадобится функция для удара одной армии по другой (в ней же разумно реализовать печать). В тестах будут описания двух армий, в ответе должно быть описание битвы. Формат описания будет понятен из примера.

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


class Army {
 private:
   int size_;
   int hp_, dmg_;
   int last_hp_;
```

```
    std::string name_;

 public:
    Army(const std::string& name, int n, int hp, int dmg):
        name_(name), size_(std::max(n, 0)), hp_(hp), last_hp_(hp), dmg_(dmg)
    {}

    bool AllDead() const; // check if army is dead

    int64_t TotalDamage() const; // обратите ВНИМАНИЕ, что урон может не поместиться в int
    int BeHit(int64_t dmg);

    int HitOneTime(Army& enemy) const;
    bool Battle(Army& enemy); // returns true if *this wins the battle, false otherwise
```

Input	Output
ZmeiGorynych 2 50 15 Ivan 10 10 10	2 ZmeiGorynych vs 10 Ivan 2 ZmeiGorynych attack 3 Ivan die 7 Ivan attack 1 ZmeiGorynych die 1 ZmeiGorynych attack 1 Ivan die 6 Ivan attack 1 ZmeiGorynych die Ivan win

D. В этой задаче у армий появляются дополнительные свойства (каждая армия может обладать любым количеством свойств или не обладать ими вовсе). В нашем случае их будет всего три: регенерация, броня и благословление.

- Регенерация — после того, как армия получает урон, раненный воин излечивается на x единиц здоровья. Естественно, у него не может стать больше максимального значения единиц здоровья.
- Броня — армия блокирует 10% урона. Блокируемый урон вычисляется в целых числах, округляется вниз. То есть, если по бронированой армии наносят 9 урона, она заблокирует 0. А если 22 урона, то заблокирует 2.
- Благословление — каждый воин наносит на 5 урона больше.

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

struct Buffs {
    int regen_;
    bool armored_;
    bool blessed_;

    Buffs(): regen_(0), armored_(false), blessed_(false) {
    }

    Buffs(int regen, bool armored, bool blessed) :
        regen_(regen), armored_(armored), blessed_(blessed) {
    }
};

class Army {
 private:
    int size_;
    int hp_, dmg_;
    int last_hp_;
    std::string name_;
    Buffs buffs_;

 public:
```

```
    Army(const std::string& name, int n, int hp, int dmg, Buffs buffs = Buffs()):
        name_(name), size_(std::max(n, 0)), hp_(hp), last_hp_(hp), dmg_(dmg), buffs_(buffs)
    {}

    bool AllDead() const;

    void Bless();             // благословить армию
    void SetRegen(int regen); // установить регенерацию
    void Armor();             // наложить на армию броню

    int64_t TotalDamage() const;
    int BeHit(int64_t dmg);

    int HitOneTime(Army& enemy) const;
    bool Battle(Army& enemy);
```

Обратите внимание, что теперь членом класса `Army` является структура `Buffs`. То есть, все свойства разумно объединить в одну переменную. Со структурами можно обращаться также, как с классами. Единственное отличие — в структурах все переменные считаются `public`.

В каждом тесте 4 строки. В первой и третьей вводятся информая об армиях в том же формате, что и в предыдущей задаче. Во второй и четвертой строчках вводится информация о свойствах армий: регенерация, броня, благословление. Броня и благословление вводится 1, если есть, 0 в противном случае.

Вывод должен быть в том же формате, что и в предыдущей задаче.

Input	Output
ZmeiGorynych 2 50 15 50 1 1 Ivan 10 10 10 0 0 0	2 ZmeiGorynych vs 10 Ivan 2 ZmeiGorynych attack 4 Ivan die 6 Ivan attack 1 ZmeiGorynych die 1 ZmeiGorynych attack 2 Ivan die 4 Ivan attack 0 ZmeiGorynych die 1 ZmeiGorynych attack 2 Ivan die 2 Ivan attack 0 ZmeiGorynych die 1 ZmeiGorynych attack 2 Ivan die ZmeiGorynych win