ГУАП

КАФЕДРА № 53

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент, кандидат технических наук |  |  |  | Кузнецов В.А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ДОПОЛНИТЕЛЬНОМ ЗАДАНИИ №1 |
| ОПТИМАЛЬНЫЙ ПЕРСОНАЖ/КОМАНДА. |
|  |
| по курсу: Основы программирования |
|  |

РАБОТУ ВЫПОЛНИЛ(А)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 5138 |  |  |  | Воробьев В.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Задание:** создать программу, находящая оптимальную команду или персонажа для исходных данных.

**Исходные данные задачи:**

1. Персонаж обладает несколькими характеристиками:

− Количество наносимого им урона противнику в секунду. Персонаж наносит урон все время, кроме времени, на которое он выведен из строя. Если персонаж наносит урон нескольким соперникам, указанный урон наносится каждому.

− Количество соперников, которым наносится урон. Не более 3.

− Здоровье персонажа. Наносимый урон вычитается из текущего здоровья, при достижении значения 0, персонаж выбывает.

− Способность выводить из строя одного соперника. Задается двумя значениями в секундах: первое - на которое соперник выводится из строя, второе – интервал вывода из строя. Второе как минимум в двое больше первого. Персонаж выводит из строя, как только удается это сделать.

1. Персонажи могут сражаться один против одного, и в командах по 3 персонажа.

Победой команды считается, когда все персонажи противоположной команды выбыли. Каждого персонажа можно выбирать только один раз (не зависимо от того, в какой команде уже выбран этот персонаж).

**Задача:**

**А.** Оптимальный выбор 1х1. Дан набор персонажей с произвольно заданными характеристиками, не менее 10 персонажей. Соперник выбрал одного персонажа. Выбрать среди оставшихся персонажей тех, кто побеждает персонажа соперника с указанием времени сражения и процента оставшегося здоровья. В случае если таких персонажей несколько, то ранжировать их по: времени сражения или проценту (от первоначального) оставшегося здоровья. Вариант ранжирования (сортировки задается) как входной параметр.

**В.** Оптимальная команда. Дан набор персонажей с произвольно заданными характеристиками, не менее 10 персонажей. Соперник выбрал трех персонажей. Собрать команду из трех персонажей среди оставшихся таких, которые побеждают команду соперника с указанием времени сражения. В случае если вариантов таких команд несколько, то ранжировать их по времени сражения. Считать, что персонажи наносят урон и выводят из строя в первую очередь соперника с наибольшим уроном.

**Примечания к исходному заданию:**

По причине неясности некоторых аспектов в задании, я оговорю, как я понял, их здесь:

1. Команды ходят по очереди. (первые – противники).
2. Персонаж выбывает, как только его здоровье опускается ниже 1, не дожидаясь конца хода.
3. Отсчет восстановление способности выбивания персонажа начинается от момента применения этой способности, а не от следующего хода.
4. Персонаж ВСЕГДА пытается вывести из строя вражеского героя с максимальным уроном. Если он уже выведен, то берется максимальное время вывода из строя.

**Описание исходного кода:**

1. Структура **Champion**, хранящая всю информацию о чемпионе. **stunAbility** < 1 , если герой не способен оглушать. Остальные поля структуры тривиальны для понимания.

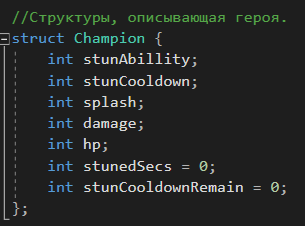
****

Рисунок 1 – код Champion

1. Структура **MyFightResult**, представляющая всю информацию об итоге сражения. Была спроектирована, для удобства сортировки результатов. championIndexes – индексы из исходного массива чемпионов.

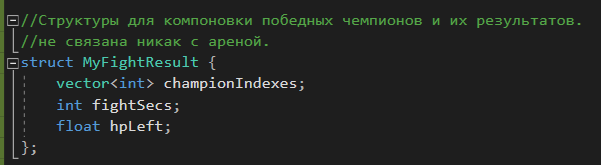


Рисунок 2 – код MyFightResult

1. Класс **Arena**, ответственный за симуляцию сражений двух команд произвольного размера.
   1. приватные методы:
      1. TeamAttack – симулирует атаку одной команды на другую. Проходит в цикле атакующую команду и активирует каждого героя.
      2. ChampionAction – действия чемпиона. Сначала проверяется его состояние и перезарядка способности. Затем выполняются действия атаки. Проверятся способность к выводу из строя, и если она не перезаряжается, выводим из строя самого большого по урону героя противника. Затем атакуем всех вражеских героев, и если их здоровье опустится ниже 0 – удаляем их из команды.
      3. CountHp – подсчет количества здоровья живых героев в команде, указанной через булевый аргумент.
   2. публичные методы
      1. SetTeam – установка команды, переданной через аргументы метода. Сортировка от наибольшего урона к меньшему.
      2. FightTeams – симуляция сражения команд. В процессе выполнения метода изменяются команды (удаляются выбывшие герои). Выполняет симуляцию сражения вызывая действия команд, пока один из векторов не станет пустым. Если выиграла союзная команда – вернет время сражения в секундах, а также процент оставшегося хп, иначе -1 для времени и процента здоровья.



Рисунок 3 – приватные члены Arena

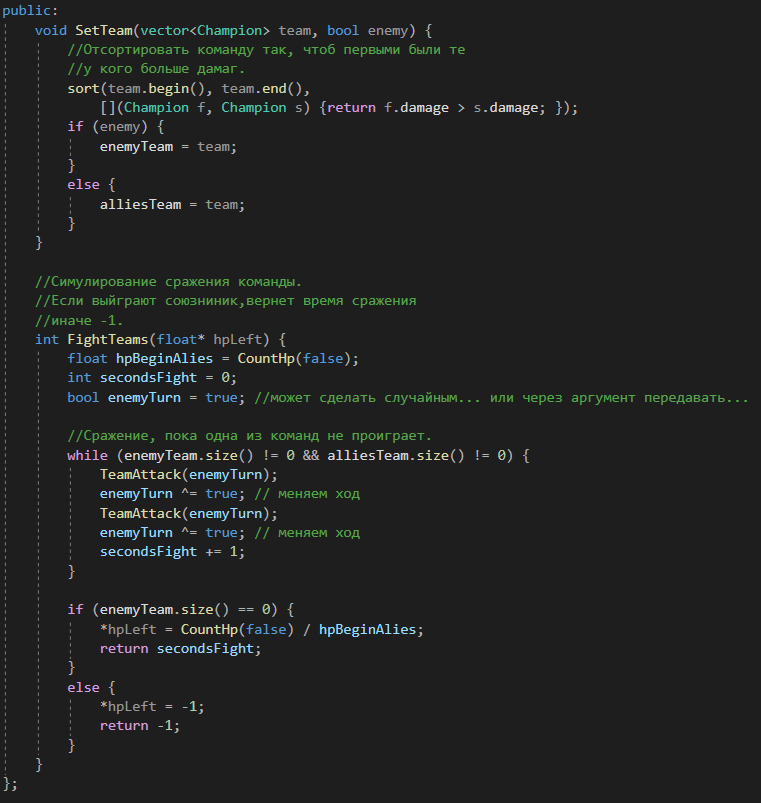


Рисунок 4 – публичные методы Arena

1. Функции для работы с вводом-выводом пользователя.

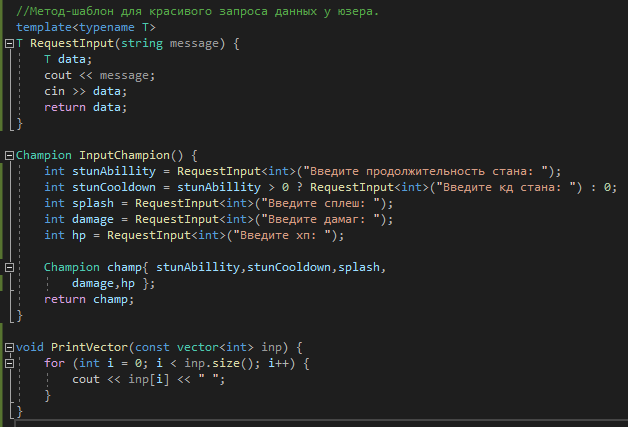


Рисунок 5 – функции для ввода/вывода.

1. Функция для преобразования индексов исходного вектора, в массив элементов на основе этих индексов.

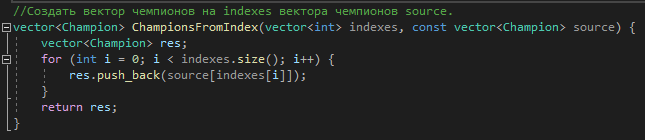


Рисунок 6 – код ChampionsFromIndex

1. Дальше пойдет описание кода в main(). Cначала мы считываем исходные данные (количество чемпионов, их характеристики, количество чемпионов противника и их индексы). Для задачи A – размер количества чемпионов должен быть введен = 1, для задачи В – 3.

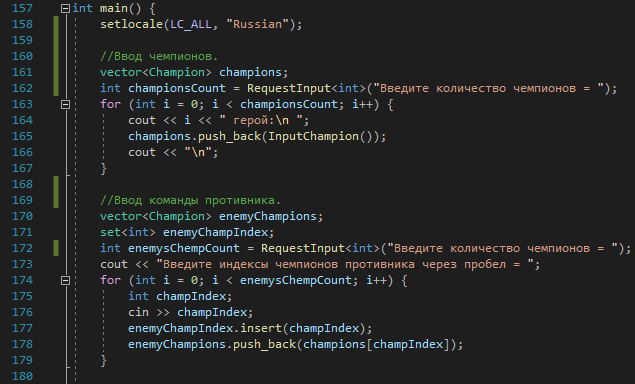


Рисунок 7 – часть кода main (ввод данных)

1. Создания строки bitmask из 0 и 1 перестановки которой, означают индексы из массива героев, которых стоят включить в команду. Выполняем цикл генерации команд, пока есть следующая перестановка. В этом цикле мы смотрим, уникальная ли команда ( противник не брал таких героев ) и если это выполняется -> симулируем схватку в объекте класса Arena и записываем в массив результатов.

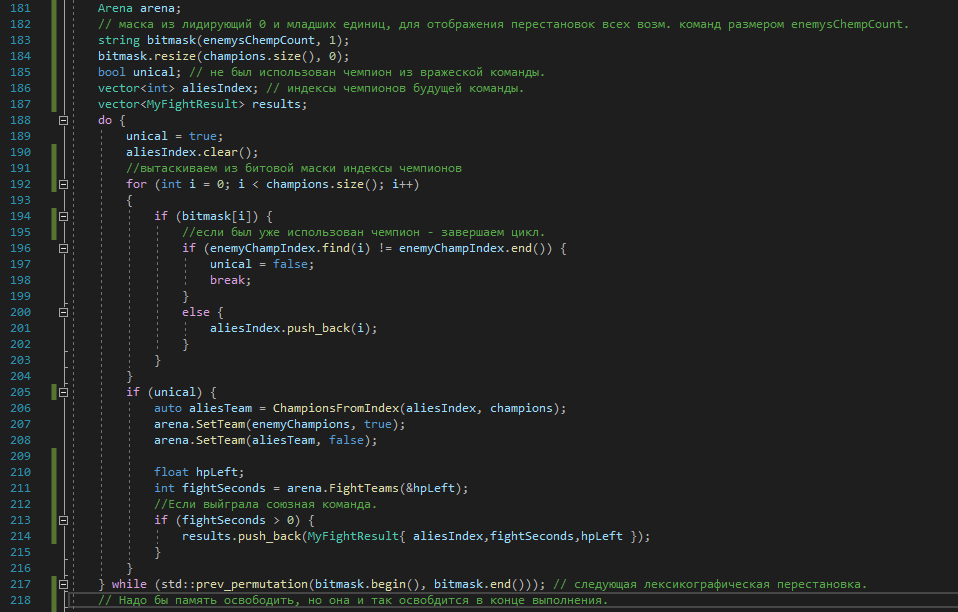


Рисунок 8 - часть кода main (симуляция сражений)

1. Выбор способа ранжирования результатов. Вывод победивших команд, с указанием индексов из ввода.

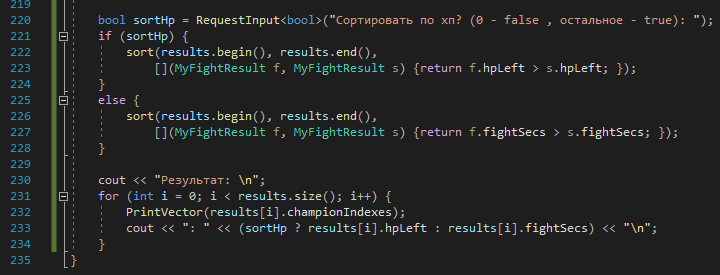


Рисунок 9 - часть кода main (вывод результата)

**Листинг программы:**

Исходный код можно также посмотреть на GitHub. URL - <https://github.com/vladcto/SUAI_homework/main/op_sem2_add1/OP/2%D1%81%D0%B5%D0%BC%D0%B5%D1%81%D1%82%D1%80_%D0%B4%D0%BE%D0%BF_%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B51/source_code.cpp>

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <set>

using namespace std;

//Структуры, описывающая героя.

struct Champion {

int stunAbillity;

int stunCooldown;

int splash;

int damage;

int hp;

int stunedSecs = 0;

int stunCooldownRemain = 0;

};

//Структуры для компоновки победных чемпионов и их результатов.

//не связана никак с ареной.

struct MyFightResult {

vector<int> championIndexes;

int fightSecs;

float hpLeft;

};

//Класс в котором симулируется сражение.

class Arena {

private:

vector<Champion> enemyTeam, alliesTeam;

//Чемпионы команды, чей сейчас ход, выполняют свои действия.

void TeamAttack(bool enemyTurn) {

vector<Champion>& nowTeam = enemyTurn ? enemyTeam : alliesTeam;

for (int i = 0; i < nowTeam.size(); i++) {

ChampionAction(i, enemyTurn);

}

}

//Проверка состояния чемпиона, и его действия.

void ChampionAction(int champIndex, bool enemyTurn) {

Champion& myChampion = enemyTurn ? enemyTeam[champIndex] : alliesTeam[champIndex];

//Перезаряка его способности.

myChampion.stunCooldownRemain = max(myChampion.stunCooldownRemain - 1, 1);

//Если герой в стане, уменьшаем его длительность и пропускаем все действия.

if (myChampion.stunedSecs > 0) {

myChampion.stunedSecs -= 1;

return;

}

//Действия героя.

vector<Champion>& opositTeam = enemyTurn ? alliesTeam : enemyTeam;

//Станим героем.

if (myChampion.stunAbillity > 0 && myChampion.stunCooldownRemain <= 0) {

opositTeam[0].stunedSecs = max(opositTeam[0].stunedSecs, myChampion.stunAbillity);

myChampion.stunCooldownRemain = myChampion.stunCooldown;

}

//Атакуем.

for (int i = 0; i < myChampion.splash && i < opositTeam.size(); i++) {

opositTeam[i].hp -= myChampion.damage;

//Если герой сдох, то убираем его из команды.

if (opositTeam[i].hp <= 0) {

opositTeam.erase(opositTeam.begin() + i);

i--;

}

}

}

//Подсчитать количество ХП команды.

int CountHp(bool countEnemy) {

int res = 0;

vector<Champion> myTeam = countEnemy ? enemyTeam : alliesTeam;

for (int i = 0; i < myTeam.size(); i++) {

res += myTeam[i].hp;

}

return res;

}

public:

void SetTeam(vector<Champion> team, bool enemy) {

//Отсортировать команду так, чтоб первыми были те

//у кого больше дамаг.

sort(team.begin(), team.end(),

[](Champion f, Champion s) {return f.damage > s.damage; });

if (enemy) {

enemyTeam = team;

}

else {

alliesTeam = team;

}

}

//Симулирование сражения команды.

//Если выйграют союзниник,вернет время сражения

//иначе -1.

int FightTeams(float\* hpLeft) {

float hpBeginAlies = CountHp(false);

int secondsFight = 0;

bool enemyTurn = true; //может сделать случайным... или через аргумент передавать...

//Сражение, пока одна из команд не проиграет.

while (enemyTeam.size() != 0 && alliesTeam.size() != 0) {

TeamAttack(enemyTurn);

enemyTurn ^= true; // меняем ход

TeamAttack(enemyTurn);

enemyTurn ^= true; // меняем ход

secondsFight += 1;

}

if (enemyTeam.size() == 0) {

\*hpLeft = CountHp(false) / hpBeginAlies;

return secondsFight;

}

else {

\*hpLeft = -1;

return -1;

}

}

};

//Метод-шаблон для красивого запроса данных у юзера.

template<typename T>

T RequestInput(string message) {

T data;

cout << message;

cin >> data;

return data;

}

Champion InputChampion() {

int stunAbillity = RequestInput<int>("Введите продолжительность стана: ");

int stunCooldown = stunAbillity > 0 ? RequestInput<int>("Введите кд стана: ") : 0;

int splash = RequestInput<int>("Введите сплеш: ");

int damage = RequestInput<int>("Введите дамаг: ");

int hp = RequestInput<int>("Введите хп: ");

Champion champ{ stunAbillity,stunCooldown,splash,

damage,hp };

return champ;

}

void PrintVector(const vector<int> inp) {

for (int i = 0; i < inp.size(); i++) {

cout << inp[i] << " ";

}

}

//Создать вектор чемпионов на indexes вектора чемпионов source.

vector<Champion> ChampionsFromIndex(vector<int> indexes, const vector<Champion> source) {

vector<Champion> res;

for (int i = 0; i < indexes.size(); i++) {

res.push\_back(source[indexes[i]]);

}

return res;

}

int main() {

setlocale(LC\_ALL, "Russian");

//Ввод чемпионов.

vector<Champion> champions;

int championsCount = RequestInput<int>("Введите количество чемпионов = ");

for (int i = 0; i < championsCount; i++) {

cout << i << " герой:\n ";

champions.push\_back(InputChampion());

cout << "\n";

}

//Ввод команды противника.

vector<Champion> enemyChampions;

set<int> enemyChampIndex;

int enemysChempCount = RequestInput<int>("Введите количество чемпионов = ");

cout << "Введите индексы чемпионов противника через пробел = ";

for (int i = 0; i < enemysChempCount; i++) {

int champIndex;

cin >> champIndex;

enemyChampIndex.insert(champIndex);

enemyChampions.push\_back(champions[champIndex]);

}

Arena arena;

// маска из лидирующий 0 и младших единиц, для отображения перестановок всех возм. команд размером enemysChempCount.

string bitmask(enemysChempCount, 1);

bitmask.resize(champions.size(), 0);

bool unical; // не был использован чемпион из вражеской команды.

vector<int> aliesIndex; // индексы чемпионов будущей команды.

vector<MyFightResult> results;

do {

unical = true;

aliesIndex.clear();

//вытаскиваем из битовой маски индексы чемпионов

for (int i = 0; i < champions.size(); i++)

{

if (bitmask[i]) {

//если был уже использован чемпион - завершаем цикл.

if (enemyChampIndex.find(i) != enemyChampIndex.end()) {

unical = false;

break;

}

else {

aliesIndex.push\_back(i);

}

}

}

if (unical) {

auto aliesTeam = ChampionsFromIndex(aliesIndex, champions);

arena.SetTeam(enemyChampions, true);

arena.SetTeam(aliesTeam, false);

float hpLeft;

int fightSeconds = arena.FightTeams(&hpLeft);

//Если выйграла союзная команда.

if (fightSeconds > 0) {

results.push\_back(MyFightResult{ aliesIndex,fightSeconds,hpLeft });

}

}

} while (std::prev\_permutation(bitmask.begin(), bitmask.end())); // следующая лексикографическая перестановка.

// Надо бы память освободить, но она и так освобдится в конце выполнения.

bool sortHp = RequestInput<bool>("Сортировать по хп? (0 - false , остальное - true): ");

if (sortHp) {

sort(results.begin(), results.end(),

[](MyFightResult f, MyFightResult s) {return f.hpLeft > s.hpLeft; });

}

else {

sort(results.begin(), results.end(),

[](MyFightResult f, MyFightResult s) {return f.fightSecs > s.fightSecs; });

}

cout << "Результат: \n";

for (int i = 0; i < results.size(); i++) {

PrintVector(results[i].championIndexes);

cout << ": " << (sortHp ? results[i].hpLeft : results[i].fightSecs) << "\n";

}

}

**Тесты работы программы:**

1. Входные данные:

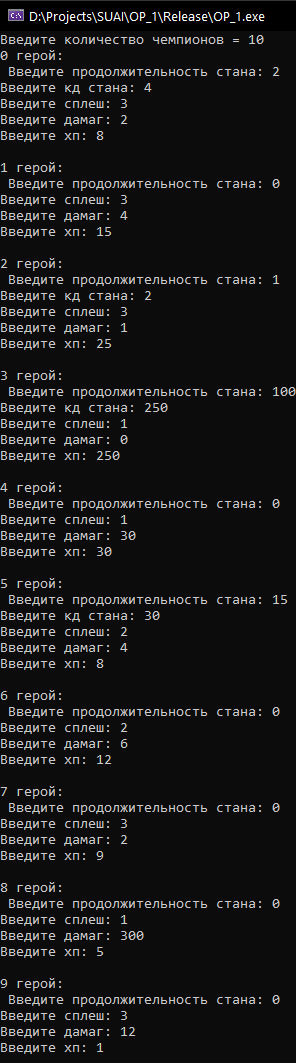


Рисунок 10 – тестовые данные

Результат (задача А):

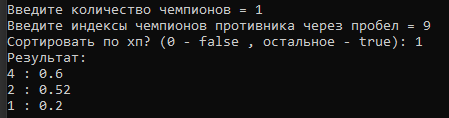


Рисунок 11 – результат тестовых данных

1. Входные данные:

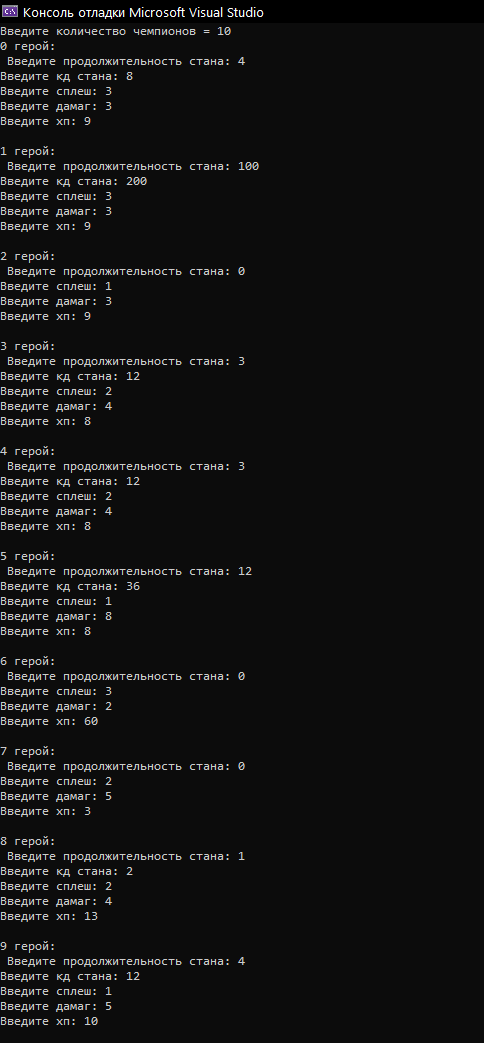


Рисунок 12 – тестовые данные

Результат(задача A):

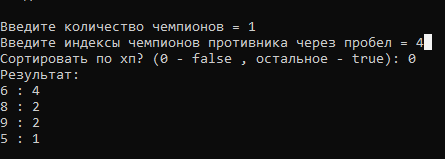


Рисунок 13 - результат тестовых данных

1. Входные данные:

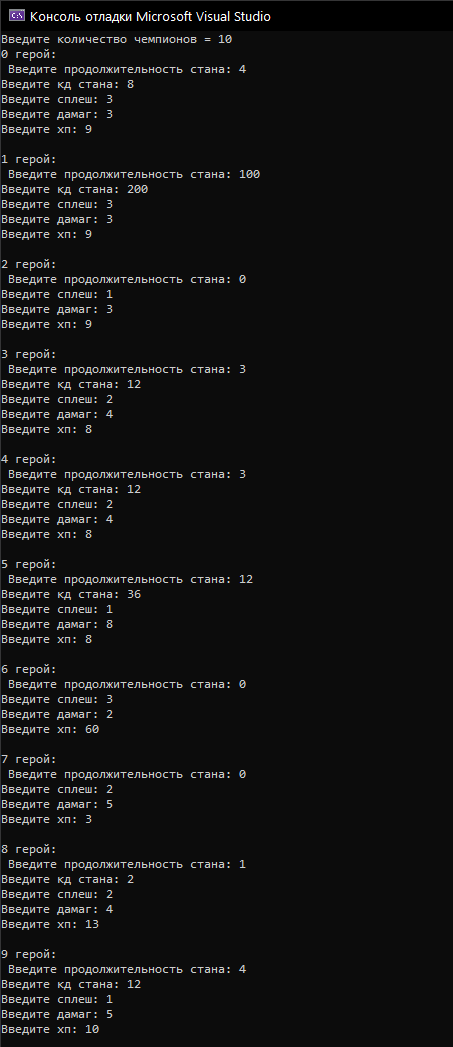


Рисунок 14 – тестовые данные

Результат (задача В):

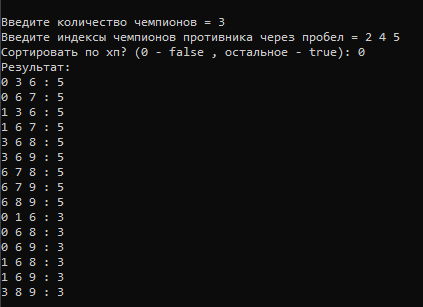


Рисунок 15 - результат тестовых данных

**Вывод**: на основе поставленного задания реализовали алгоритм нахождения оптимальной команды/персонажа. Протестировали алгоритм на тестовых значениях и убедились в правильности написанного алгоритма.