МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет по лабораторной работе №1

по предмету «Тестирование программного обеспечения»

Номер задания 19

Выполнил

студент гр.

Принял:

старший преподаватель Е.В. Старыгина

Ижевск

2023

Содержание

[1. Задание 3](#_Toc129735309)

[2. Блок-схема алгоритма 3](#_Toc129735310)

[3. Тестирование базового пути 4](#_Toc129735311)

[4. Тестирование потоков данных 7](#_Toc129735312)

[5. Области эквивалентности 16](#_Toc129735313)

[6. Контрольный пример 17](#_Toc129735314)

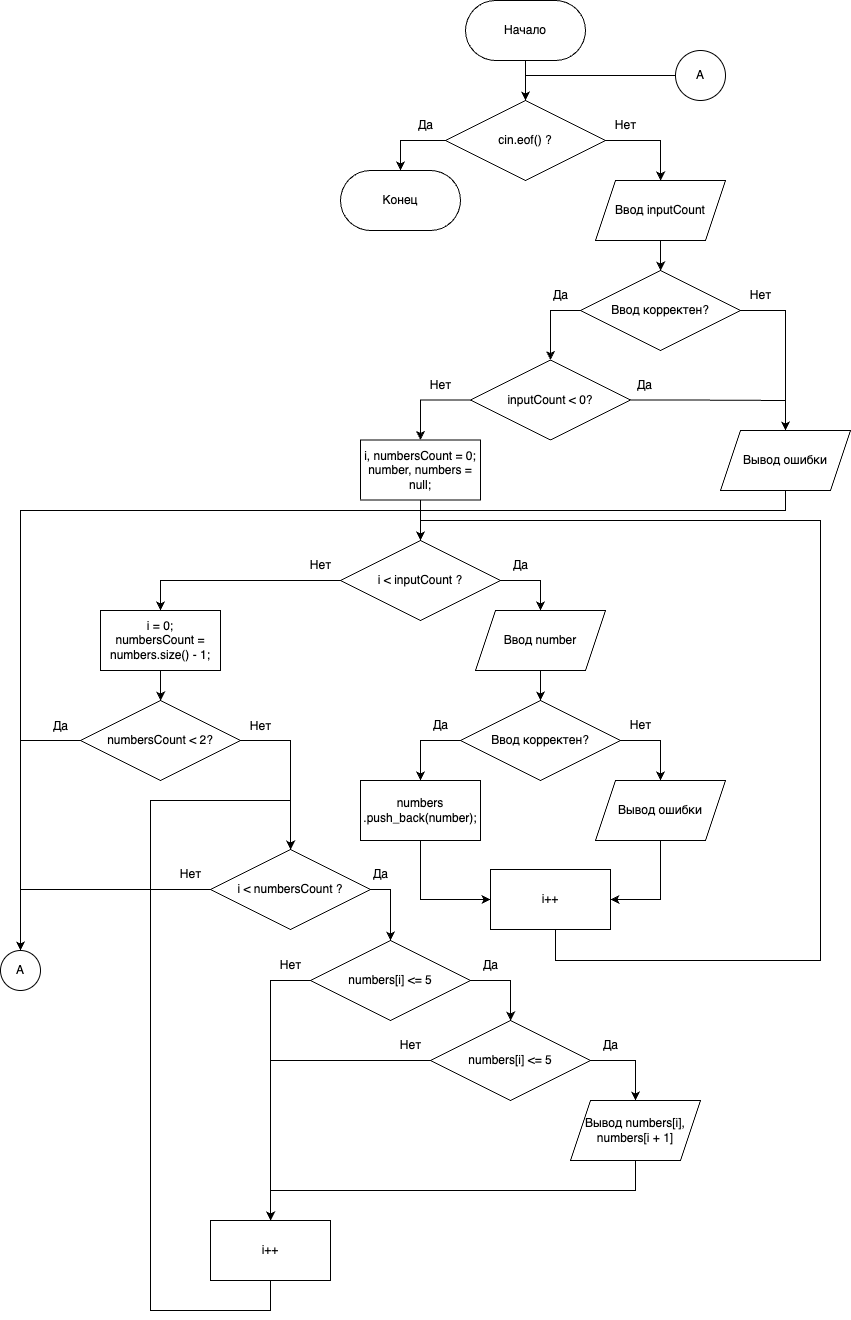
[7. Текст программы 20](#_Toc129735315)

# **Задание**

Дан массив целых чисел а0,...,аn-1. Найти все пары (аi,аi+1), такие что аi<=5 и аi+1>0. Распечатать их значения и номеpа. Если таких паp нет, то выдать сообщение.

# **Блок-схема алгоритма**

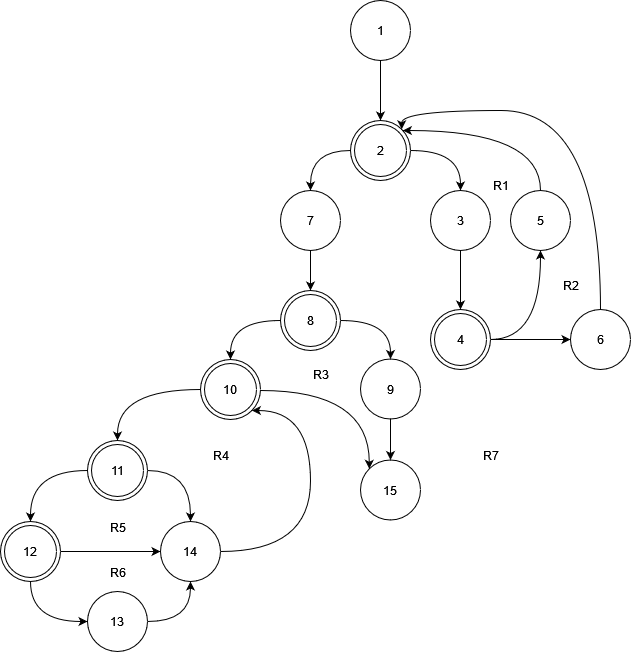
Блок-схема алгоритма представлена на рис. 1.



# **Тестирование базового пути**

* 1. Потоковый граф

Потоковый граф представлен на рис. 2.



* 1. Цикломатическая сложность

Вычислим цикломатическую сложность потокового графа каждым из трёх способов:

1. потоковый граф имеет 7 регионов;
2. V(G) = 20 дуг – 15 узлов + 2 = 7;
3. V(G) = 6 предикатных узлов + 1 = 7.
   1. Базовое множество независимых путей

Перечислим все пути в данном графе:

1. 1 – 2 – 7 – 8 – 9 – 15. Если не введено ни одного значения; если ввод пустой
2. 1 – 2 – 3 – 4 – 5 – 2 – 7 – 8 – 9 – 15. Если единственное введенное число оказалось некорректным;
3. 1 – 2 – 3 – 4 – 6 – 2 – 7 – 8 – 9 – 15. Если единственное введенное число оказалось корректным;
4. 1 – 2 – 3 – 4 – 6 - … - 2 – 7 – 8 – 9 – 15. Если не введено ни одного корректного числа;
5. 1 – 2 – 3 – 4 – 6 - … - 2 – 7 – 8 – 10 – 15. Путь недостижим.
6. 1 – 2 – 3 – 4 – 6 - … - 2 – 7 – 8 – 10 – 11 – 14 – … – 10 – 15. Если введено не менее 2 корректных чисел, каждое из которых > 5;
7. 1 – 2 – 3 – 4 – 6 - … - 2 – 7 – 8 – 10 – 11 – 12 – 14 – … – 10 – 15. Если введено не менее 2 корректных чисел, где в паре первый элемент <= 5, а второй <= 0;
8. 1 – 2 – 3 – 4 – 6 - … - 2 – 7 – 8 – 10 – 11 – 12 – 13 – 14 – … – 10 – 15. Если введено не менее 2 корректных чисел, где в хотя бы одной паре первый элемент <= 5, а второй > 0.
   1. Тестовые варианты

Тестовый вариант для пути 1):

Исходные данные: пустая строка.

Ожидаемые результаты: сообщение об отсутствии возможности найти пару чисел.

Тестовый вариант для пути 2):

Исходные данные: одно некорректное значение number(i).

Ожидаемые результаты: сообщение об ошибке ввода, сообщение об отсутствии возможности найти пару чисел.

Тестовый вариант для пути 3):

Исходные данные: одно корректное значение number(i).

Ожидаемые результаты: сообщение об отсутствии возможности найти пару чисел.

Тестовый вариант для пути 4):

Исходные данные: одно или более некорректных значений number(i).

Ожидаемые результаты: сообщение(-я) об ошибке(-ах) ввода числа(-ел).

Тестовый вариант для пути 6):

Исходные данные: не менее 2 корректных чисел, каждое из которых > 5.

Ожидаемые результаты: возможные сообщения об ошибках ввода, не найдено ни одной пары.

Тестовый вариант для пути 7):

Исходные данные: не менее 2 корректных чисел, где в паре первый элемент <= 5, а второй <= 0.

Ожидаемые результаты: возможные сообщения об ошибках ввода, не найдено ни одной пары.

Тестовый вариант для пути 8):

Исходные данные: не менее 2 корректных чисел, где в хотя бы одной паре первый элемент <= 5, а второй > 0.

Ожидаемые результаты: возможные сообщения об ошибках ввода, найденная(-ые) пара(-ы) чисел.

# **Тестирование потоков данных**

* 1. Информационный граф

Информационный граф представлен на рис. 3.

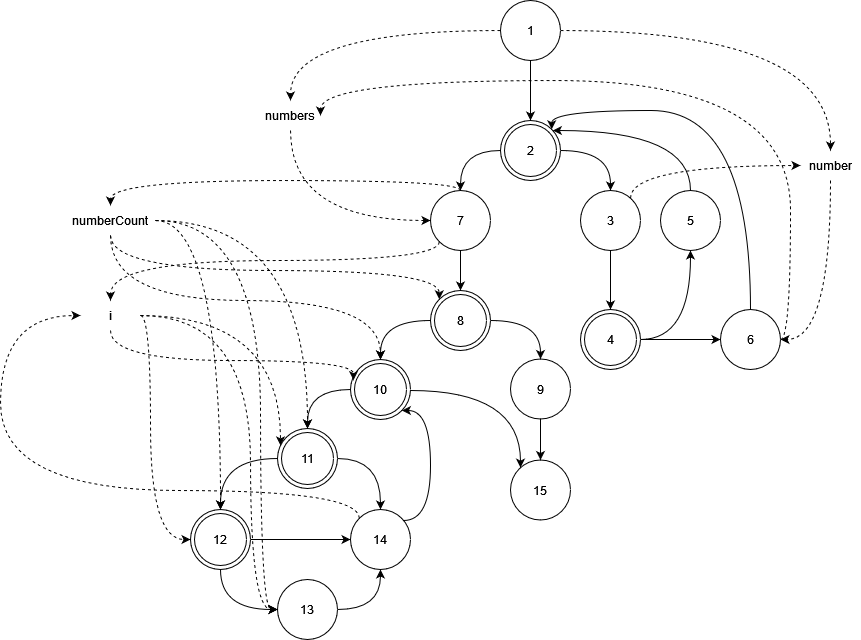


Рисунок 3. Информационный граф

* 1. Формирование полного набора du-цепочек

Сформируем список определений переменных:

1. DEF(1) = {number, 1}
2. DEF(1) = {numbers, 1}
3. DEF(3) = {number, 3}
4. DEF(7) = {numberCount, 7}
5. DEF(7) = {i, 7}
6. DEF(14) = {i, 14}

Сформируем список использования переменных:

1. USE(6) = {number, 6}
2. USE(7) = {numbers, 7}
3. USE(8) = {numberCount, 8}
4. USE(10) = {numberCount, 10}
5. USE(10) = {i, 10}
6. USE(11) = {numberCount, 11}
7. USE(11) = {i, 11}
8. USE(12) = {numberCount, 12}
9. USE(12) = {i, 12}
10. USE(13) = {numberCount, 13}
11. USE(13) = {i, 13}

Сформируем список DU-цепочек:

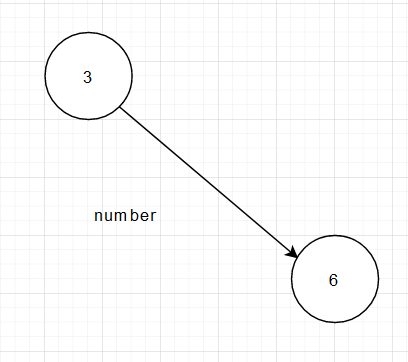
1. DU(number) = {number, 1, 6}
2. DU(number) = {number, 3, 6}
3. DU(numbers) = {numbers, 1, 7}
4. DU(numberCount) = {numberCount, 7, 8}
5. DU(numberCount) = {numberCount, 7, 10}
6. DU(numberCount) = {numberCount, 7, 11}
7. DU(numberCount) = {numberCount, 7, 12}
8. DU(numberCount) = {numberCount, 7, 13}
9. DU(i) = {i, 7, 10}
10. DU(i) = {i, 7, 11}
11. DU(i) = {i, 7, 12}
12. DU(i) = {i, 7, 13}
13. DU(i) = {i, 14, 10}
14. DU(i) = {i, 14, 11}
15. DU(i) = {i, 14, 12}
16. DU(i) = {i, 14, 13}
    1. Построение маршрутов
17. 1 – number – 6: подходят маршруты 3), 5), 6), 7), 8);
18. 3 – number – 6: подходят маршруты 3), 5), 6), 7), 8);
19. 1 – numbers – 7: подходят маршруты 1), 2), 3), 5), 6), 7), 8);
20. 7 – numberCount – 8: подходят маршруты 1), 2), 3), 5), 6), 7), 8);
21. 7 – numberCount – 10: подходят маршруты 5), 6), 7), 8);
22. 7 – numberCount – 11: подходят маршруты 6), 7), 8);
23. 7 – numberCount – 12: подходят маршруты 7), 8);
24. 7 – numberCount – 13: подходит маршрут 8);
25. 7 – i – 10: подходят маршруты 5), 6), 7), 8);
26. 7 – i – 11: подходят маршруты 6), 7), 8);
27. 7 – i – 12: подходят маршруты 7), 8);
28. 7 – i – 13: подходят маршруты 8);
29. 14 – i – 10: подходят маршруты 6), 7), 8);
30. 14 – i – 11: подходят маршруты 6), 7), 8);
31. 14 – i – 12: подходят маршруты 6), 7), 8);
32. 14 – i – 13: подходят маршруты 6), 7), 8);

Убрав дубликаты, получим следующие маршруты:

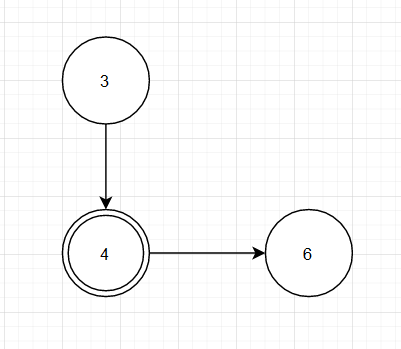
* 1 – 2 – 7 – 8 – 9 – 15. Соответствует маршруту 1)
* 1 – 2 – 3 – 4 – 5 – 2 – 7 – 8 – 9 – 15. Соответствует маршруту 2)
* 1 – 2 – 3 – 4 – 6 – 2 – 7 – 8 – 9 – 15. Соответствует маршруту 3)
* 1 – 2 – 3 – 4 – 6 – 2 – 7 – 8 – 10 – 11 – 12 – 13 – 14 – 15. Соответствует маршруту 8)

Отобразим DU-цепочки для переменных на потоковом графе:

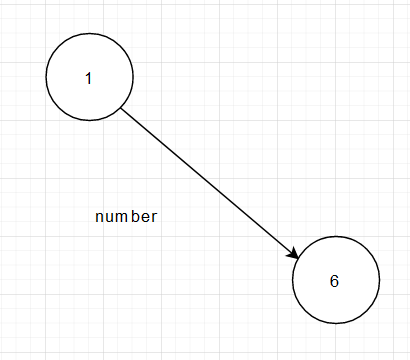
1. DU(number) = {number, 3, 6}. Цепочка:



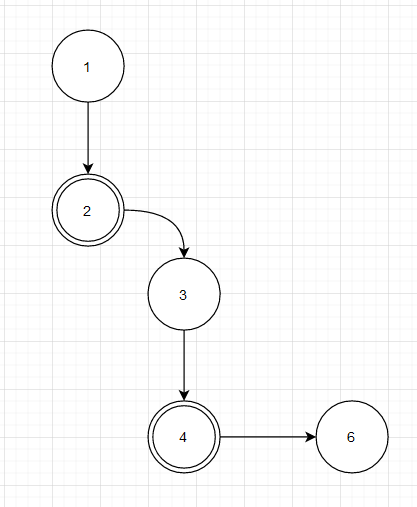
Участок в потоковом графе:



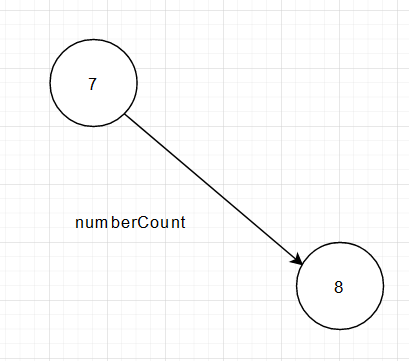
1. DU(number) = {number, 1, 6}. Цепочка:



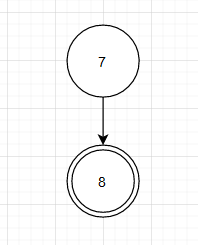
Участок потокового графа:



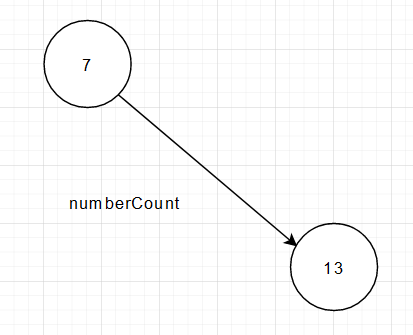
1. DU(numberCount) = {numberCount, 7, 8}. Цепочка:



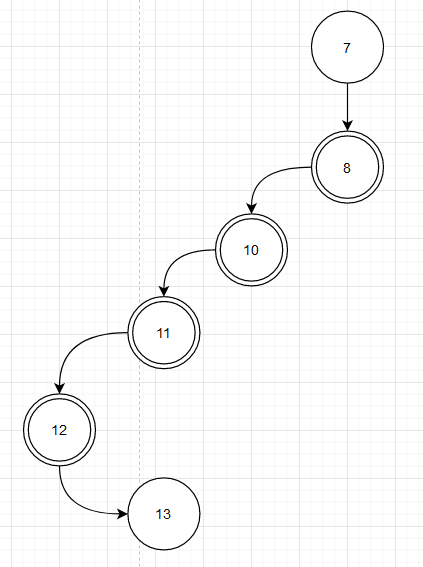
Участок потокового графа:



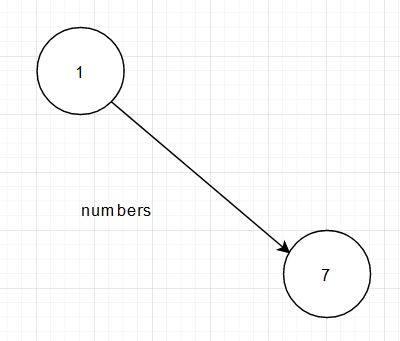
1. DU(numberCount) = {numberCount, 7, 13}. Цепочка:



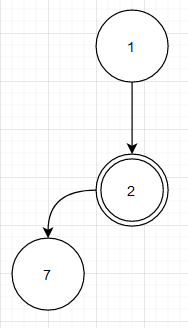
Участок потокового графа:



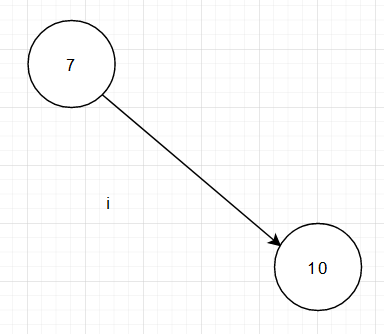
1. DU(numbers) = {numbers, 1, 7}. Цепочка:



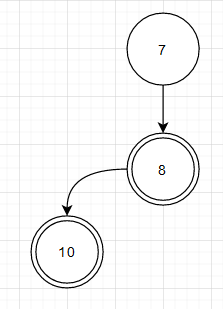
Участок потокового графа:



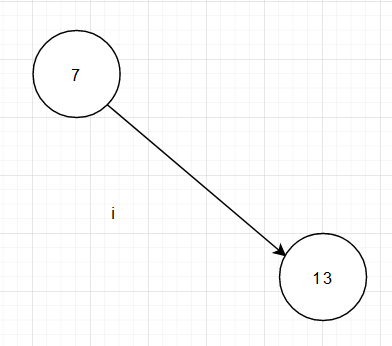
1. DU(i) = {i, 7, 10}. Цепочка:



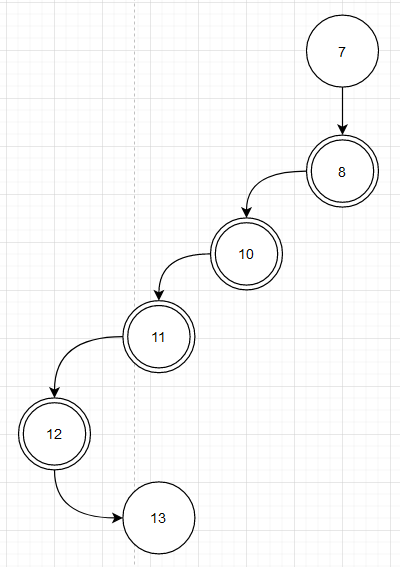
Участок потокового графа:



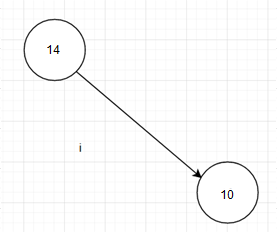
1. DU(i) = {i, 7, 13}. Цепочка:



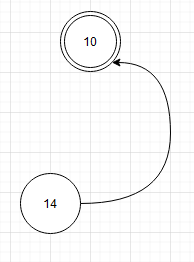
Участок потокового графа:



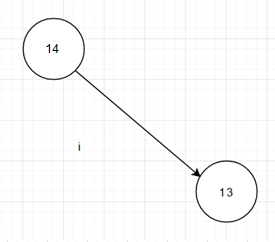
1. DU(i) = {i, 14, 10}. Цепочка:



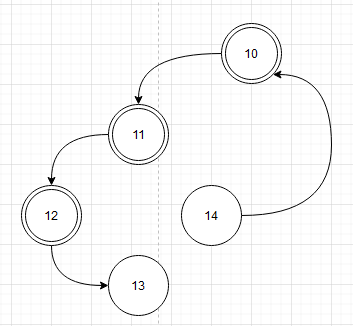
Участок потокового графа:



1. DU(i) = {i, 14, 13}. Цепочка:



Участок потокового графа:



# **Области эквивалентности**

Для number можно выделить области эквивалентности на основании типа вводимых данных, их значении и величины: см Таблица 1.

Таблица 1. Области эквивалентности переменной number

|  |  |
| --- | --- |
| Тип данных | Размер |
| Строка | - |
| Число | В пределах int32 |
| Число | Максимальное значение int32: 2^31 – 1 = 2147483647 |
| Число | Минимальное значение int32: -2^31= -2147483648 |
| Число | Больше, чем максимальное значение int32: 2^31 = 2147483648 |
| Число | Меньше, чем минимальное значение int32: -2^31-1 = -2147483649 |

# **Контрольный пример**

Входные данные для программы сохранены в файл. Структура файла представляет собой:

n срока: исходные данные, номер варианта n, номер набора 1

n + 1 строка: ожидаемые результаты, номер варианта n, номер набора 1  
…

n + k строка: исходные данные, номер варианта n, номер набора k

n + k + 1 строка: исходные данные, номер варианта n, номер набора k

Результаты работы программы сохраняются в файл. Структура файла:

1. Test #<номер варианта>

2. Set #<номер набора>

3. Path: <путь в потоковом графе>

4. Initial data: <начальные данные>

5. Expected output: <ожидаемые результаты>

6. Test output: <реальные результаты>

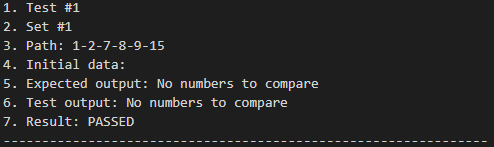
7. Result: <результат теста: пройден / не пройден>

Результаты тестирования:

Тест 1:



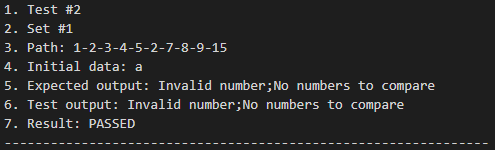
Результаты теста 1:



Тест 2:



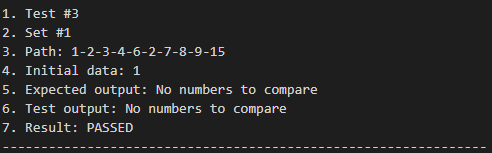
Результаты теста 2:



Тест 3:



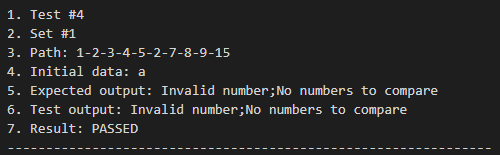
Результаты теста 3:



Тест 4:



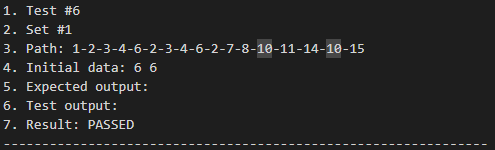
Результаты теста 4:



Тест 6:



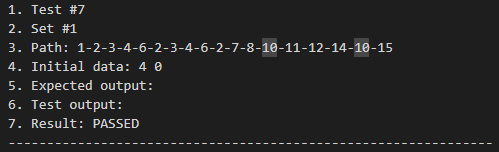
Результаты теста 6:



Тест 7:



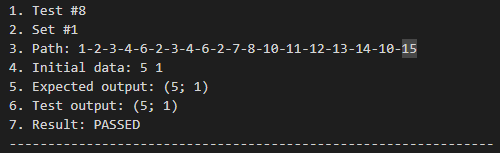
Результаты теста 7:



Тест 8:



Результаты теста 8:



# **Текст программы**

#include <iostream>

#include <cstdint>

#include <vector>

#include <sstream>

#include <string>

#include <limits>

using namespace std;

vector<string> split(string base, char delimeter)

{

    vector<string> result;

    stringstream ss(base);

    string item;

    while (getline(ss, item, delimeter))

        result.push\_back(item);

    return result;

}

struct TestStats

{

    int test, set;

    string path, initialData, expectedOutput, output, result;

    TestStats(string in, string out)

    {

        auto input = split(in, ',');

        auto output = split(out, ',');

        path = "";

        result = "";

        test = stoi(input[1]), set = stoi(input[2]);

        expectedOutput = output[0];

        initialData = input[0];

    }

    bool addGraphNode(int node)

    {

        return addGraphNode(to\_string(node));

    }

    bool addGraphNode(string node)

    {

        if (path == "")

        {

            path += node;

        }

        else

        {

            path += "-" + node;

        }

        return true;

    }

    void addResult(string s)

    {

        if (!result.empty())

        {

            result += ';';

        }

        result += s;

    }

};

void solve(TestStats &test)

{

    stringstream ss(test.initialData);

    test.addGraphNode(1);

    int number;

    vector<int> numbers;

    while (test.addGraphNode(2))

    {

        if (ss.rdbuf()->in\_avail() == 0 || ss.eof())

        { // checks wether string is empty

            break;

        }

        test.addGraphNode(3);

        ss >> number;

        test.addGraphNode(4);

        if (ss.fail())

        {

            test.addGraphNode(5);

            ss.clear();

            ss.ignore(1, ' ');

            test.addResult("Invalid number");

        }

        else

        {

            test.addGraphNode(6);

            numbers.push\_back(number);

        }

    }

    test.addGraphNode(7);

    int numbersCount = numbers.size(), i = 0;

    test.addGraphNode(8);

    if (numbersCount < 2)

    {

        test.addGraphNode(9);

        test.addResult("No numbers to compare");

        test.addGraphNode(15);

        return;

    }

    while (test.addGraphNode(10) && i < numbersCount - 1)

    {

        if (test.addGraphNode(11) && numbers[i] <= 5)

        {

            if (test.addGraphNode(12) && numbers[i + 1] > 0)

            {

                test.addGraphNode(13);

                auto res = "(" + to\_string(numbers[i]) + "; " + to\_string(numbers[i + 1]) + ")";

                test.addResult(res);

            }

        }

        test.addGraphNode(14);

        i++;

    }

    test.addGraphNode(15);

    return;

}

void printResult(TestStats test)

{

    cout << "1. Test #" << test.test << endl;

    cout << "2. Set #" << test.set << endl;

    cout << "3. Path: " << test.path << endl;

    cout << "4. Initial data: " << test.initialData << endl;

    cout << "5. Expected output: " << test.expectedOutput << endl;

    cout << "6. Test output: " << test.result << endl;

    cout << "7. Result: " << (test.expectedOutput == test.result ? "PASSED" : "NOT PASSED") << endl;

    cout << "---------------------------------------------------------------" << endl;

}

int main()

{

    freopen("lab1-tests.txt", "r", stdin);

    freopen("lab1-out.txt", "w", stdout);

    string t, s;

    while (getline(cin, t))

    {

        if (t.empty())

            break;

        getline(cin, s);

        auto test = TestStats(t, s);

        solve(test);

        printResult(test);

    }

    return 0;

}