**Примерное содержание отчёта по Заданию № 2.**

(без разбиения на разделы)

Обучающимся необходимо выполнить оформление решения в соответствии с выданной структурой отчёта (по пунктам).

Дано задание вычислить результат арифметического выражения вида:

,

где ,

где 

Исходные данные: 

Проведём анализ расчётного выражения. Расчётное выражение для  содержит следующие компоненты , , .

Данное множество не входит в состав множества исходных данных. Стоит заметить, что в нём присутствуют две константы: Пифагорова , основание натурального логарифма (экспонента) .  рассчитывается в зависимости от сформулированного условия, которое уже пересекается со множеством исходных данных элементом   рассчитывается из выражения, все элементы которого входят в состав множества исходных данных.

Таким образом:

1. Вводятся значения для исходных данных 

2. Рассчитывается *m.*

3. Рассчитывается *t*.

4. Рассчитывается .

Алгоритм решения задачи сводится к решению указанных четырёх подзадач. **Декомпозиция задачи выполнена.**

----------------------------------------------------------------------------------------------------

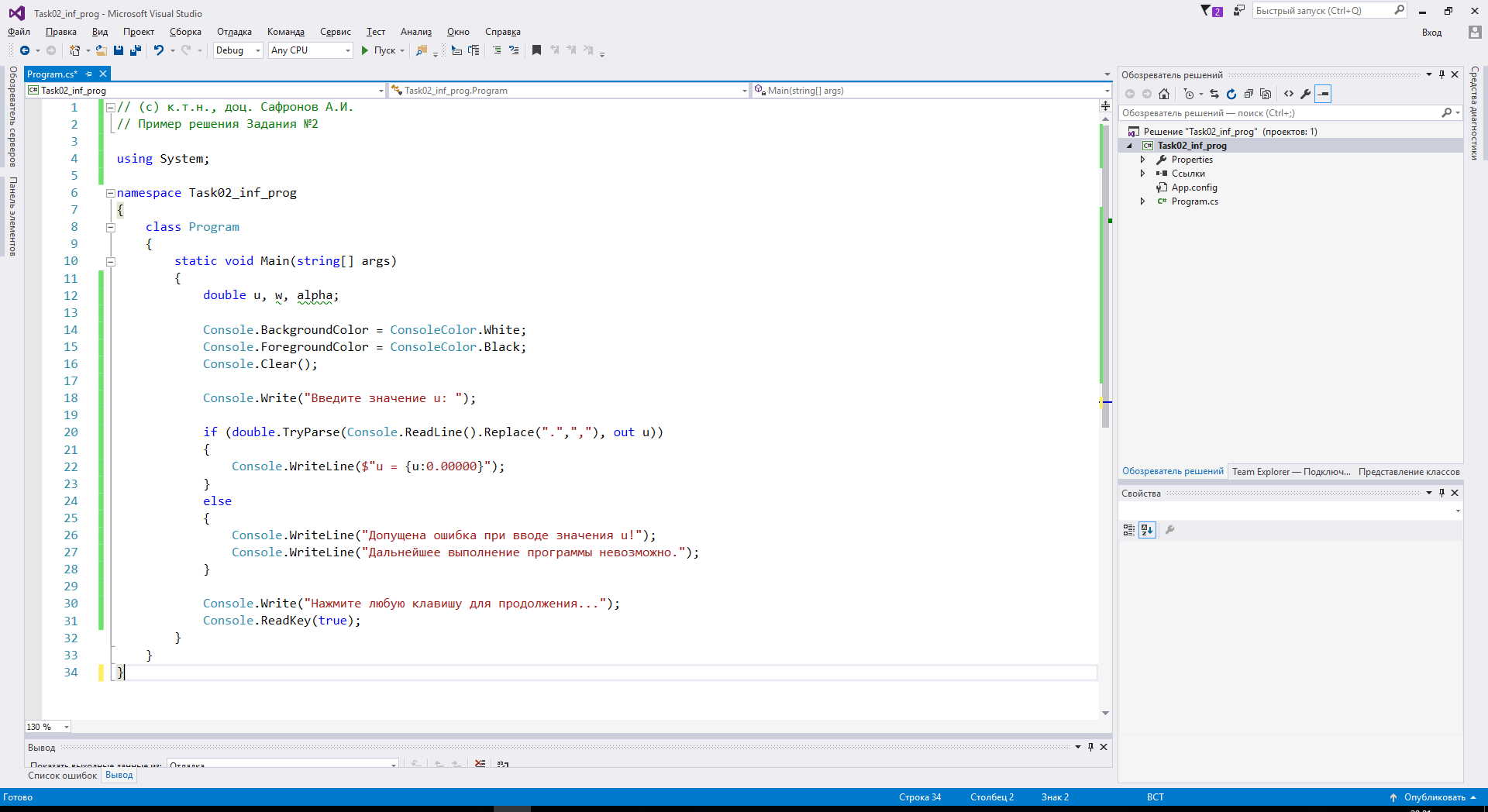
1. Под вводом значений для исходных данных подразумевается их передача в программу из некоторого источника. В качестве источника информации предполагаем пользователя, который через интерфейс (консоль) вводит значения переменных с клавиатуры.

Здесь имеет место человеческий фактор: возможны опечатки, возможны некомпетентные действия пользователя. Человеческий фактор – ограничение которое необходимо пытаться обойти, предусмотреть.

Договоримся, что за диапазон допустимых значений пользователь не выйдет. С математической точки зрения он компетентен и выход за диапазон специально контролировать мы не будем.

При вводе вещественных значений пользователь, действительно, может не знать, какой из десятичных разделителей использовать: «,» или «.». Эту ошибку можно отловить и исправить. Существует метод *Replace(x, y)*, в которой *x* – искомая конструкция, *y* – конструкция, на которую необходимо заменить найденную конструкцию *x*.

Иные ошибки ввода со стороны пользователя предсказать сложно и следует договориться о том, что для всех случаев, когда пользователь ошибается достаточно серьёзно, мы, во-первых, сообщаем ему о совершённой ошибке, во-вторых, отказываемся от выполнения дальнейших расчётов (если хотя бы один из входных параметров не получил своего значения – дальнейший расчёт невозможен). Реализуем это:

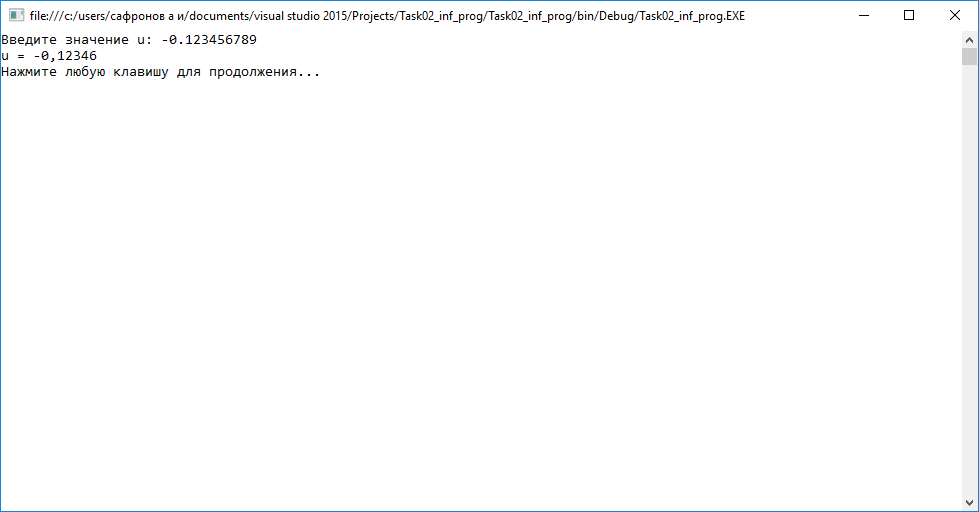


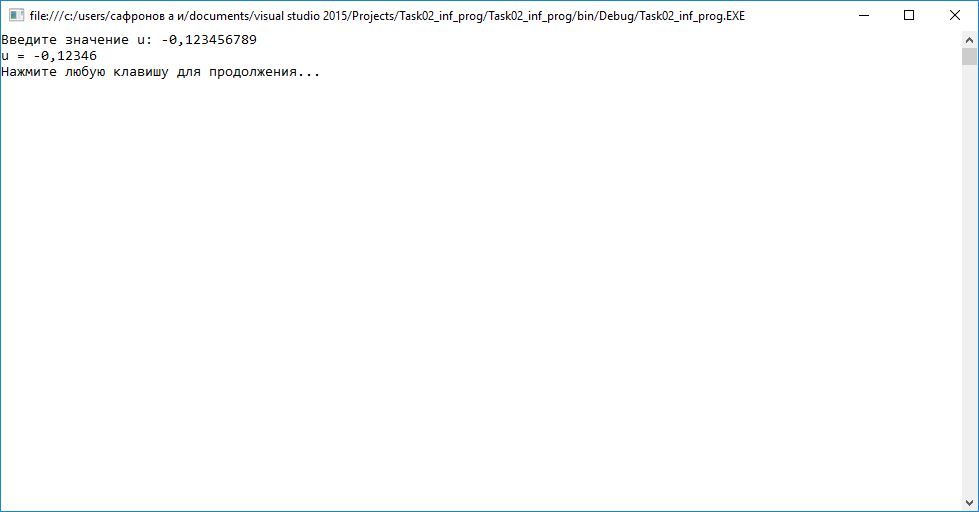
Подберём тестовые примеры:

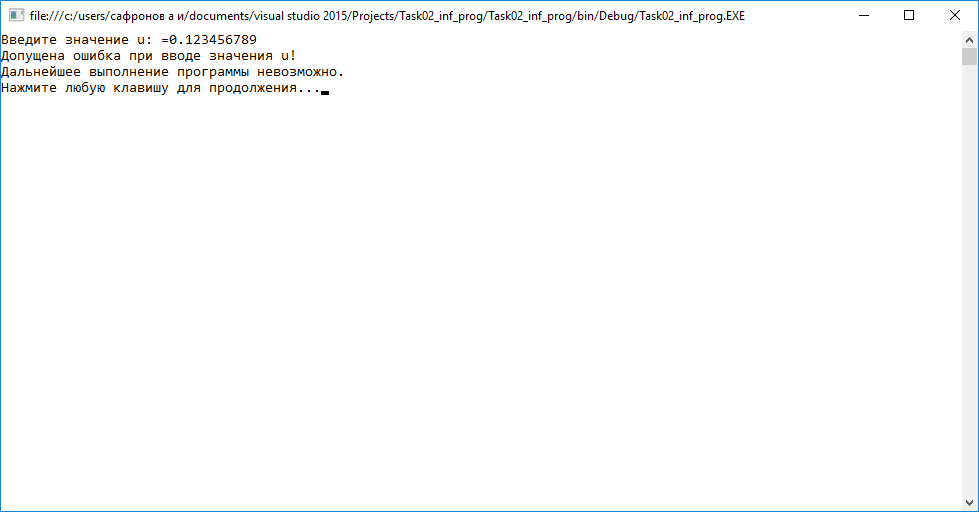
1. -0.123456789 (ввод с разделителем «.»);

2. -0,123456789 (ввод с разделителем «,»);

3. =0.123456789 (опечатка вместо «-» поставлено «=»).





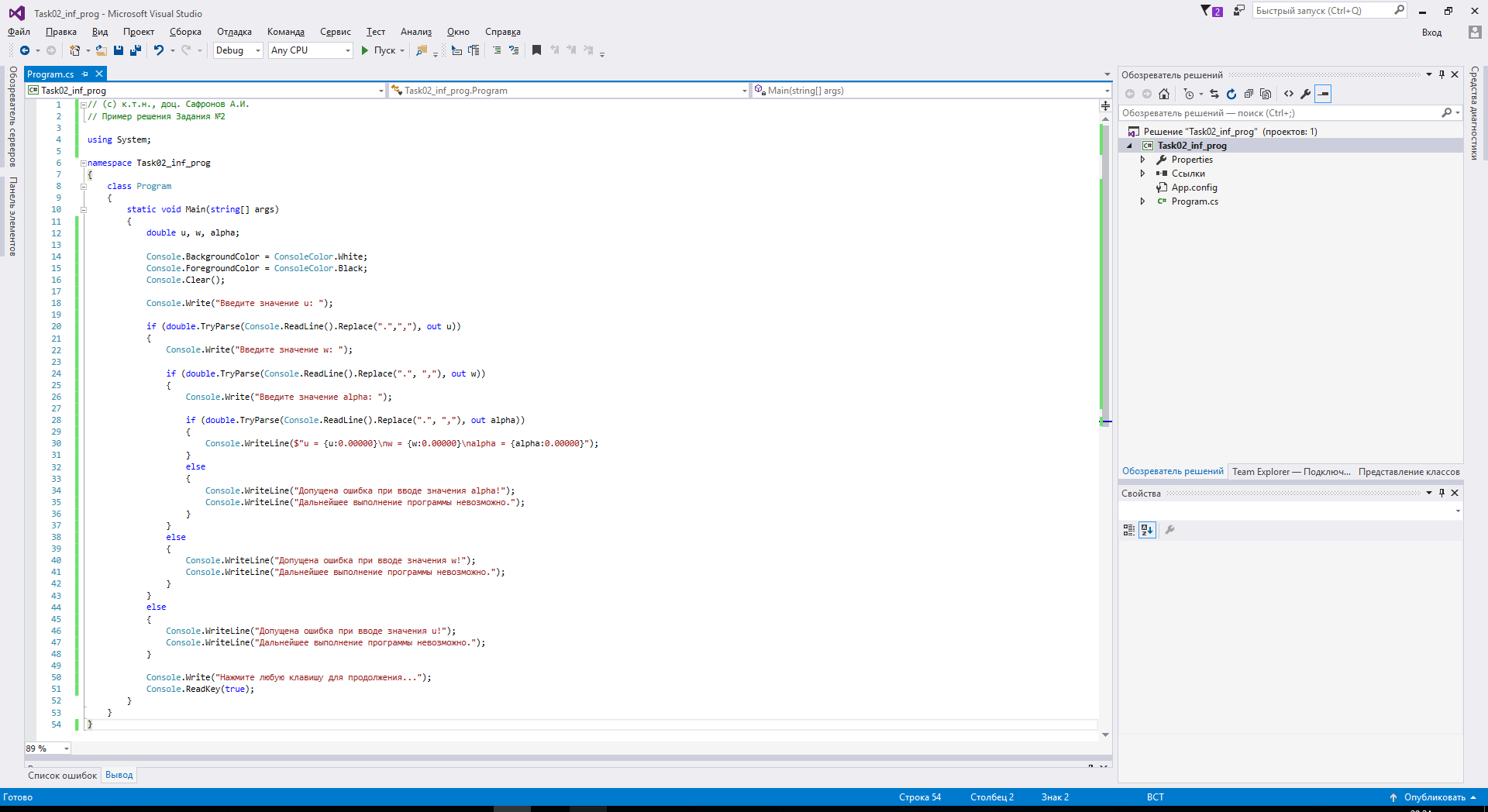


Тестовые примеры 1 и 2 по сути своей – одно и то же. Мы доверились стандартному методу *Replace()* и избежали необходимости дополнительного ветвления алгоритма. Реально рассмотрено два тестовых примера, покрывающих две ветви следующего алгоритма:



Все возможные ситуации отработаны. Программа составлена корректно.

Дополним код вводом оставшихся двух элементов из множества исходных данных с аналогичными проверками.



Далее представлена блок-схема, соответствующая коду этой программы:



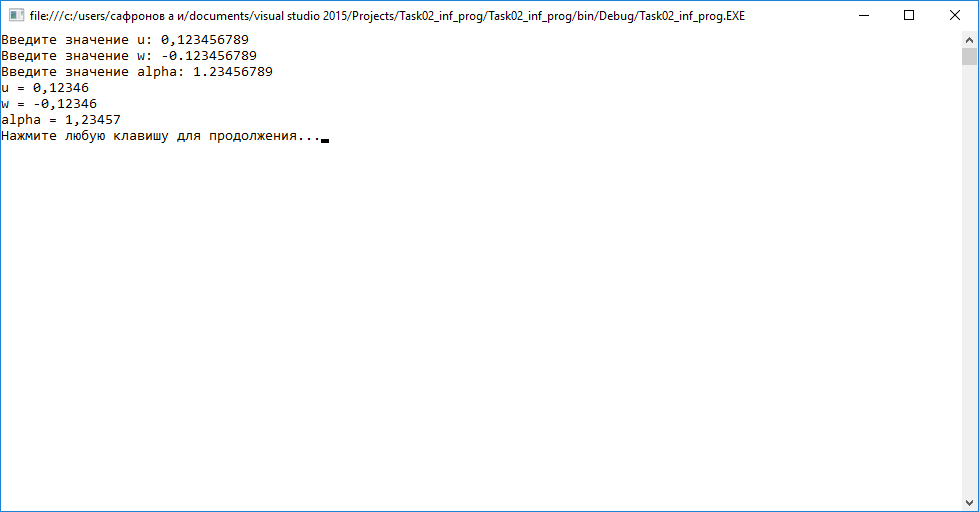
В качестве теста необходимо пройти 4 ветви алгоритма:

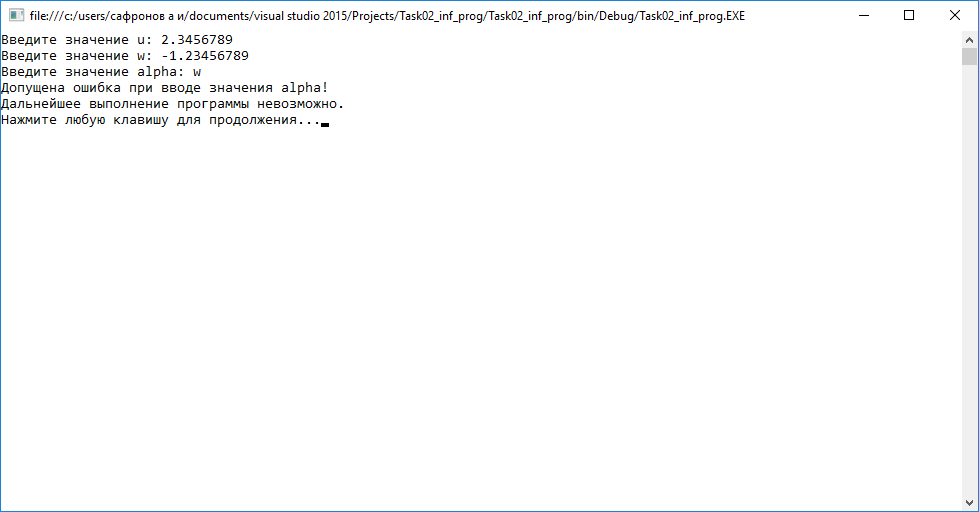
1. Всё введено верно.

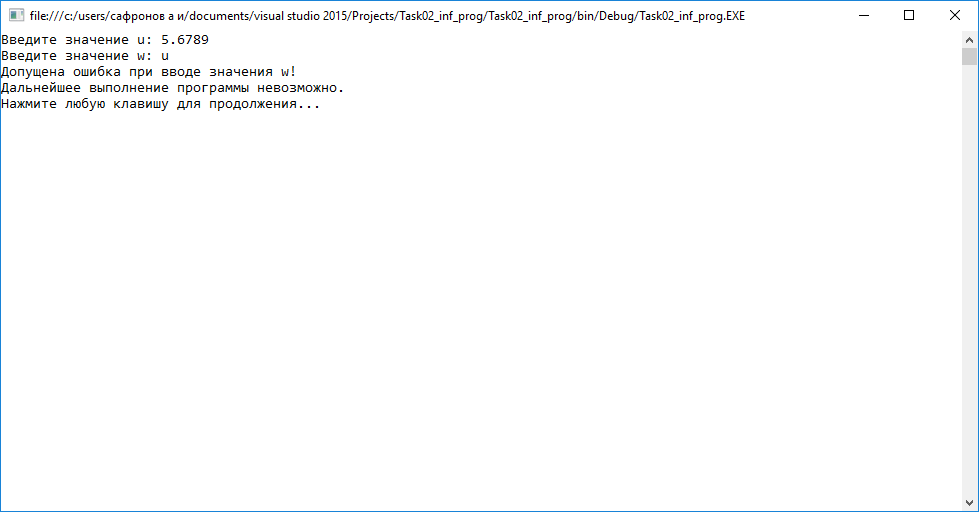
2. Совершена ошибка при вводе *alpha*.

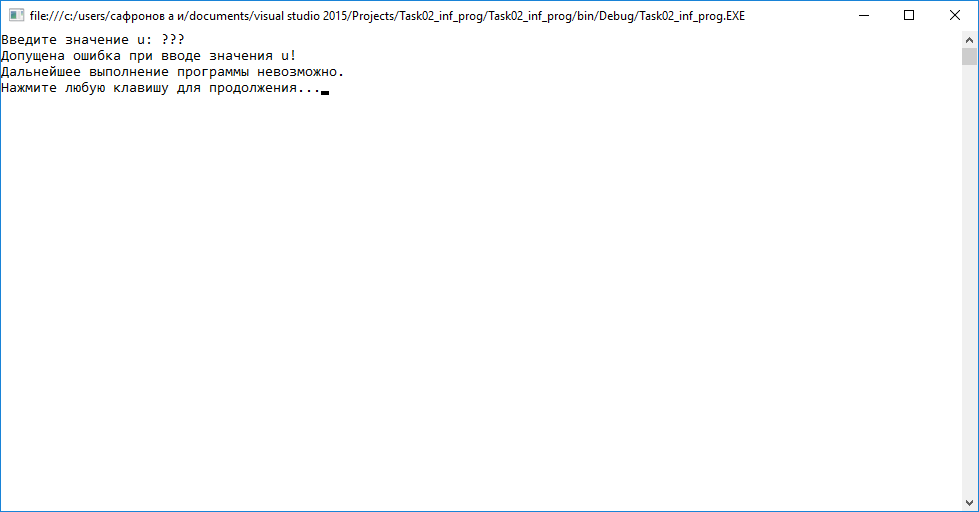
3. Совершена ошибка при вводе *w*.

4. Совершена ошибка при вводе *u*.









Программа ввода / вывода множества исходных данных работает корректно.

Во избежание нагромождений введём в эту блок-схему ссылки. Они будут задействованы при решении второй и последующих подзадач.



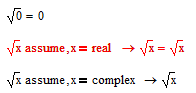
----------------------------------------------------------------------------------------------------

2. Каждая из подзадач содержит ограничения. Потому для разумного подбора значений переменных, входящих во множество исходных данных, необходимо определиться, в первую очередь, с ограничениями.

При расчёте *m* стоит обратить внимание на то, что выражение дробное и при возникновении нуля в знаменателе оно перестаёт существовать в области вещественных значений, то есть нужна проверка на нулевой знаменатель (1).

Важно отметить, что одновременно при нулевом числителе и нулевом знаменателе имеется неопределённость 0 / 0, которая в пределе раскрывается как 0.

Однако, это далеко не всё. Знаменатель содержит операцию извлечения квадратного корня. Корень числа в области вещественных значений существует только в диапазоне от нуля включительно до бесконечности (со знаком «+») – это (2).



Подкоренное выражение содержит функцию тангенса. Тангенс, как известно,  – это дробь, в которой знаменатель обуславливает ситуацию, что при всех  значение функции в области вещественных чисел перестаёт существовать (3).

Таким образом, пока не существует (3), то не существует (2), пока не существует (2), не существует (1), пока не существует (1) – не существует *m*.

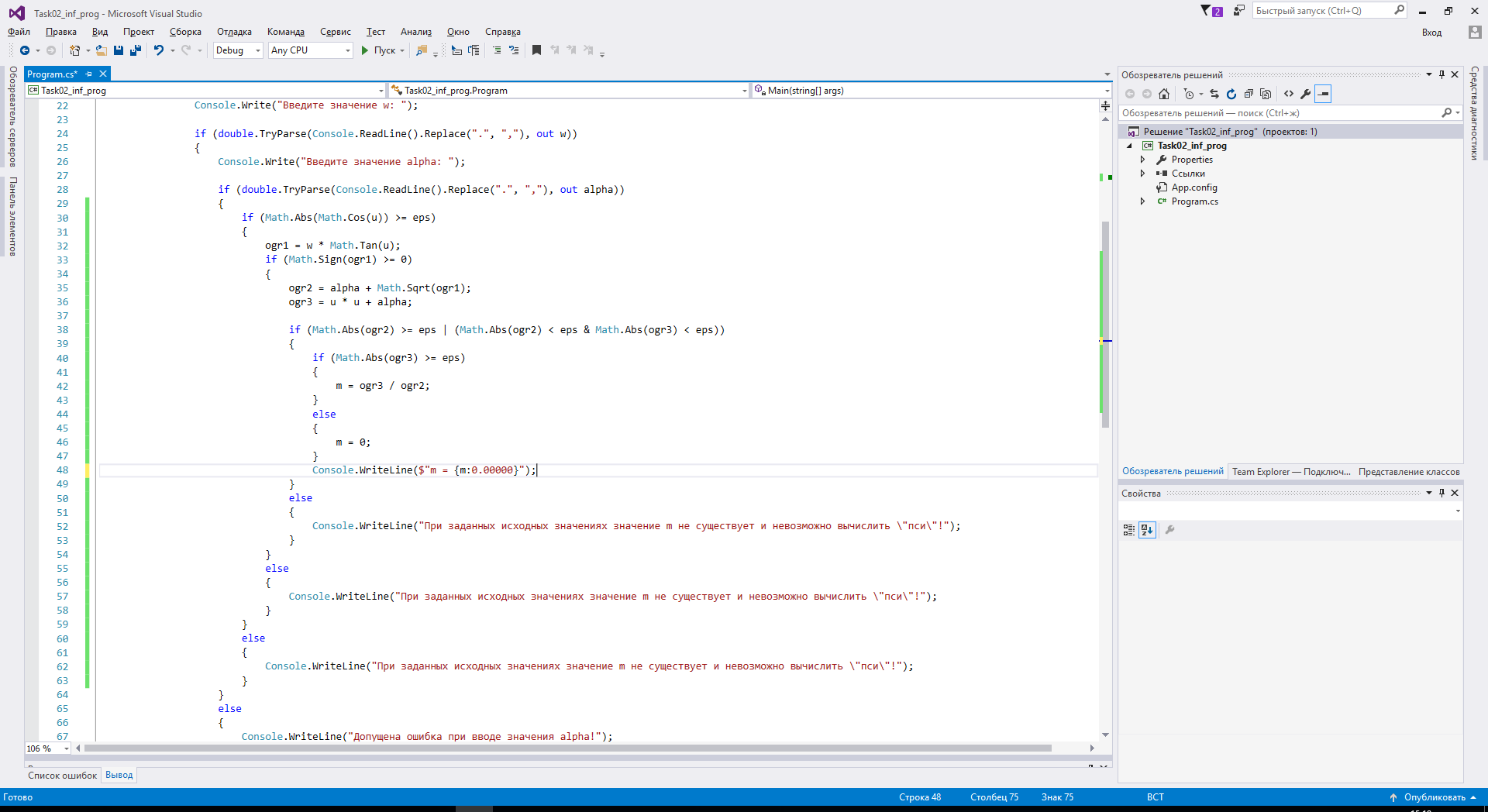
На основании изложенного в первую очередь необходимо проверить существование (3), затем при положительном исходе проверить существование (2), опять же, при положительном исходе проверить существование (1) и если все эти «двери» пройдены, ключ ко всем замкам подобран, то можно вычислять *m*.

Стоит отметить, что точное попадание в ноль, учитывая специфику вычислителей персональных компьютеров, событие очень редкое (с точки зрения Теории вероятностей – невозможное), потому нужно задаваться некоторой, наперёд заданной точностью (ε – эпсилон).

Как правило, в программах общего вида, пользователя просят ввести значение точности. В рассматриваемом примере примем допущение – выставим точность до 5 знаков после плавающей запятой константой .

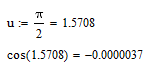
Составим код для расчёта подзадачи поиска *m*.



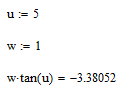


Подберём тестовые примеры:

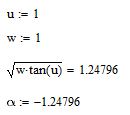
1. Косинус оборачивается в ноль, например, в точке π / 2 с учётом заданной точности:

.

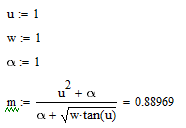
2. Подкоренное выражение отрицательно:

.

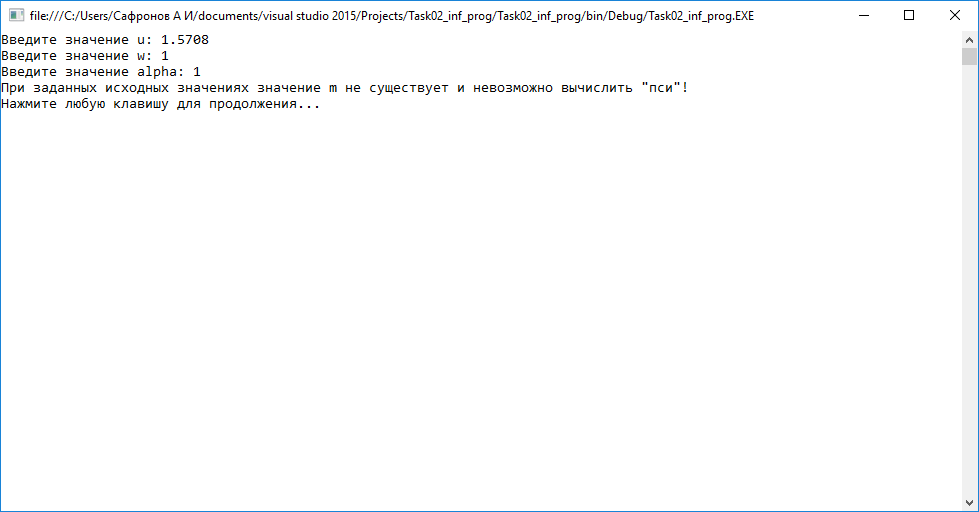
3. Выбрано такое значение , при котором знаменатель обернётся в ноль с учётом заданной точности, например:

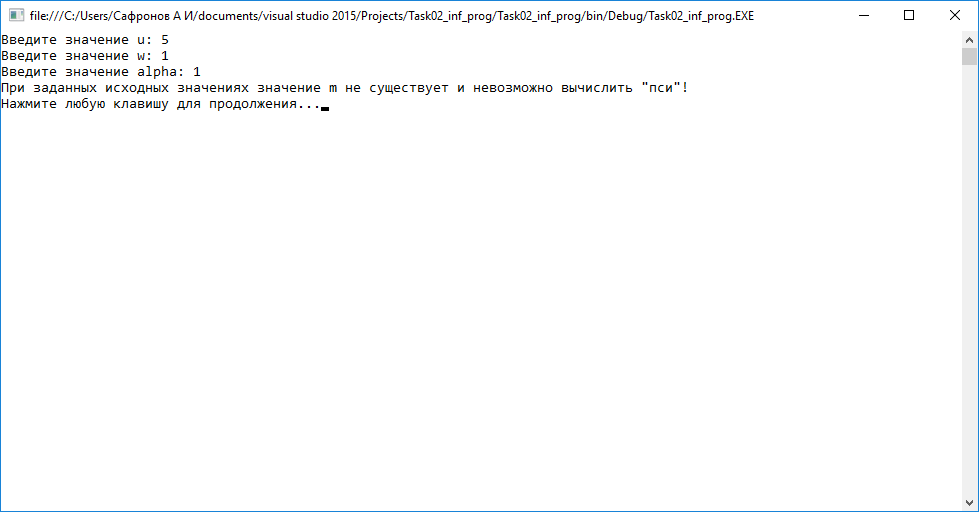


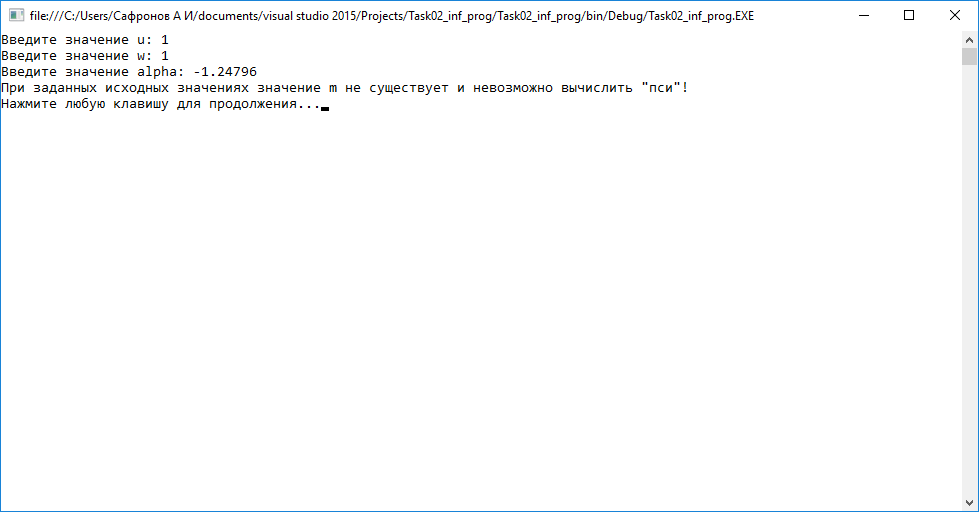
4. Произвольный пример, когда *m* существует и должно быть равно некоторому вещественному результату с учётом заданной точности:

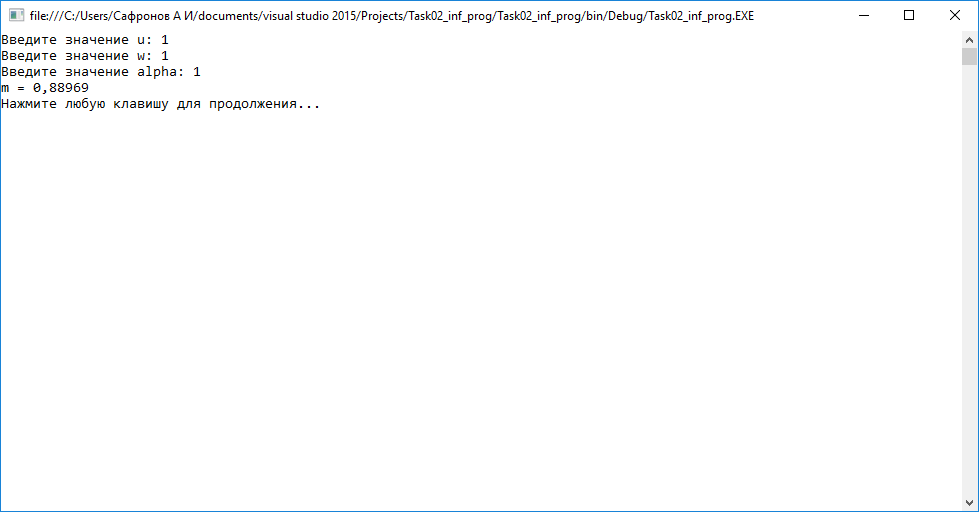


Проверим это:









Результаты тестирования совпали с тем, что ожидалось получить согласно проведённым ранее расчётам.

Представленный выше код соответствует следующей схеме алгоритма:



Далее так же, чтобы не создавать нагромождений введём ссылки 3, 4 и будем решать следующую подзадачу внутри этой пары ссылок:



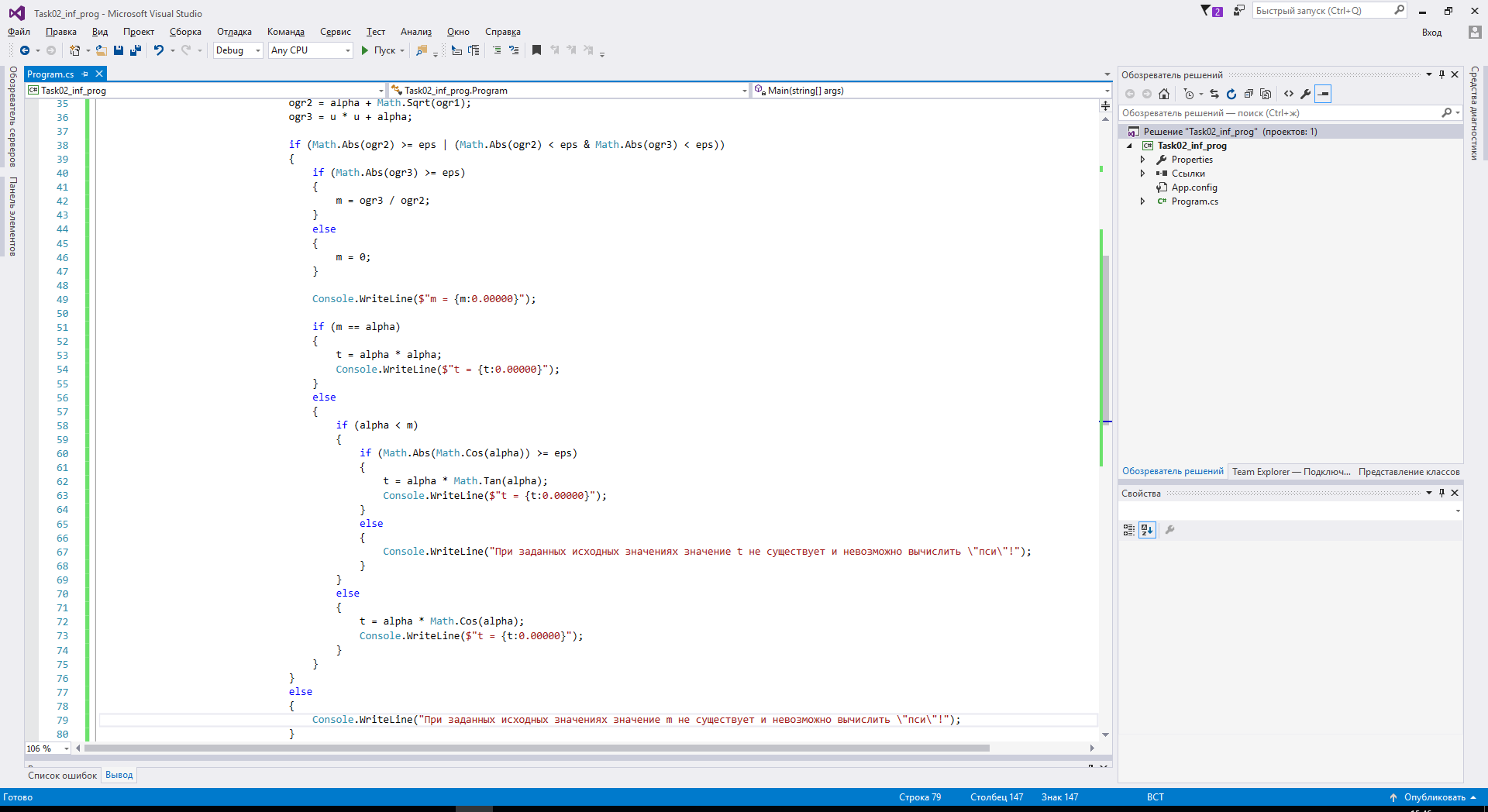
----------------------------------------------------------------------------------------------------

3. На данном этапе решается подзадача расчёта значения *t*.



Анализируем. *t* складывается из собственных трёх условий, а также одного вложенного условия – проверка существования .

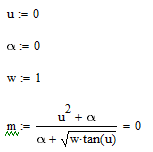
Составим код:



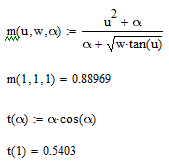
Подберём тестовые примеры:

1. Ситуация :

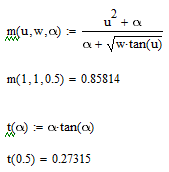
Редкая, но возможная.



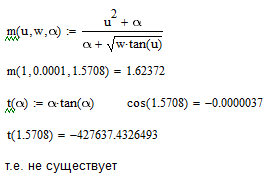
2. Ситуация :

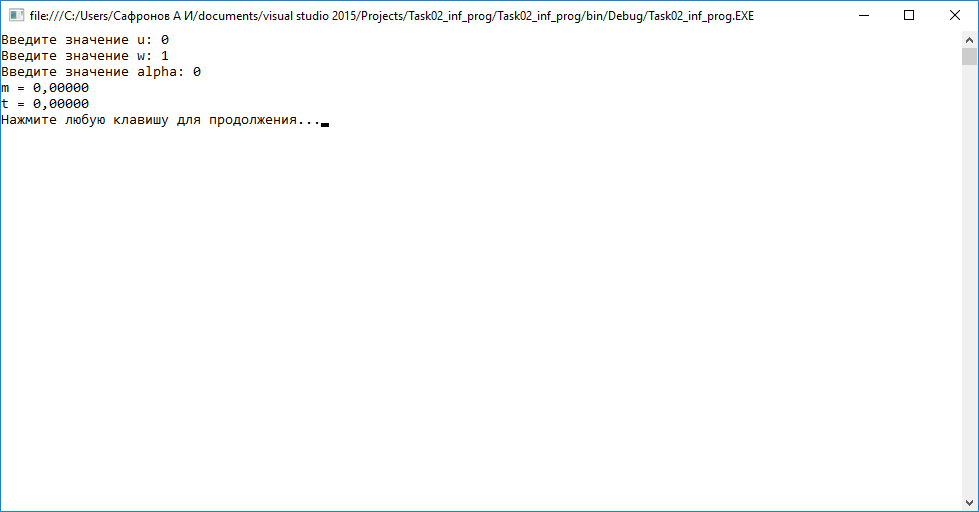


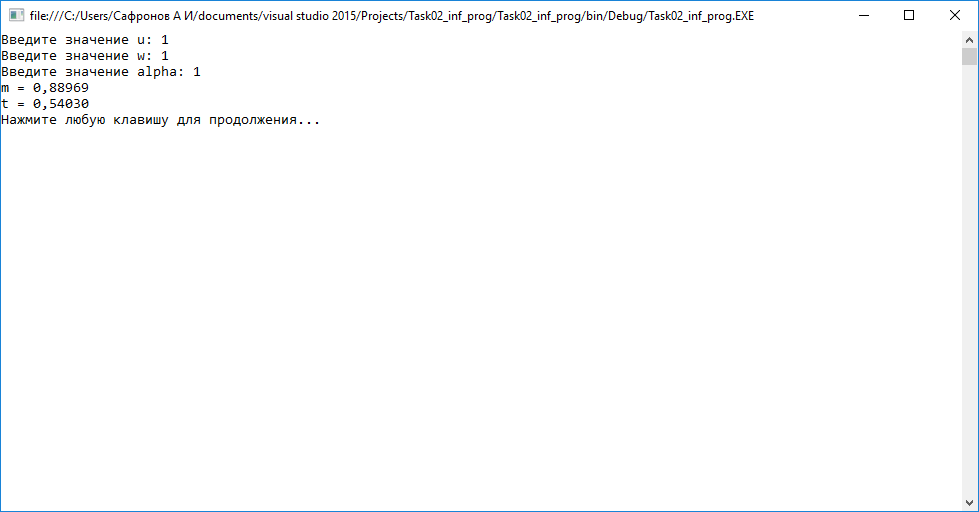
3. Ситуация  и  существует:

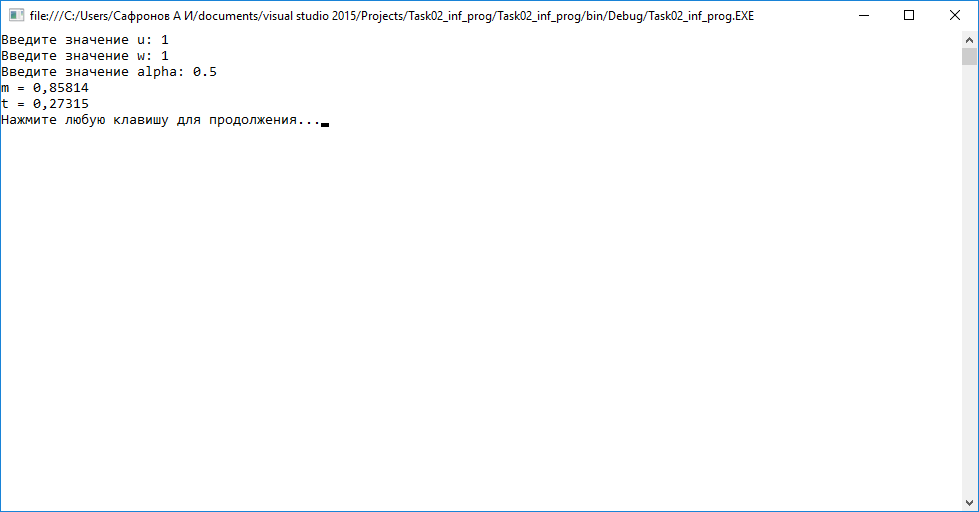


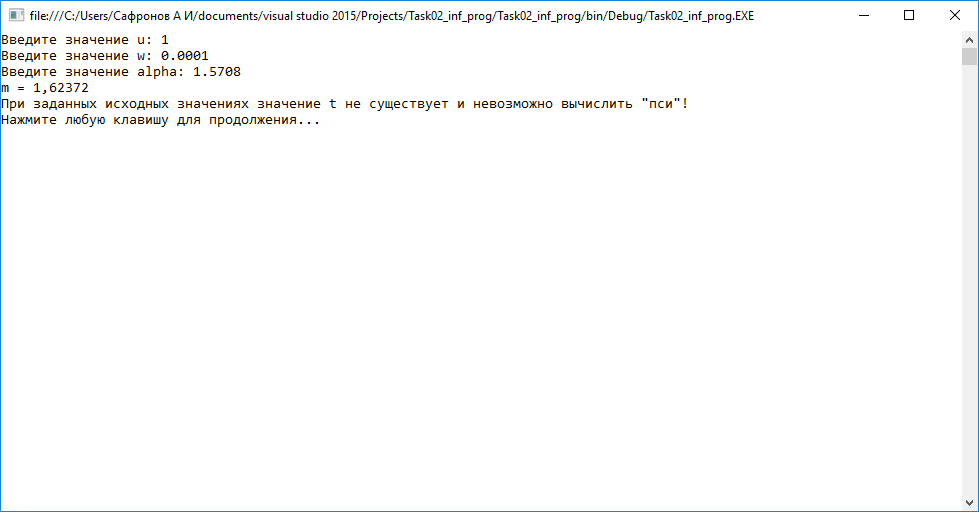
4. Ситуация  и  не существует:











Результаты тестирования совпали с тем, что ожидалось получить согласно проведённым ранее расчётам.

Этот код соответствует следующей схеме алгоритма:



Поскольку *t* входит во множество промежуточных данных, то моменты вывода информации в консоль можно опустить из рассмотрения. Взамен необходимо ввести логический признак существования *t*. По умолчанию его выставить (*true*), а в ветви, где *t* не существует – сбросить (*false*):



----------------------------------------------------------------------------------------------------

4. На данном этапе решается подзадача расчёта значения .

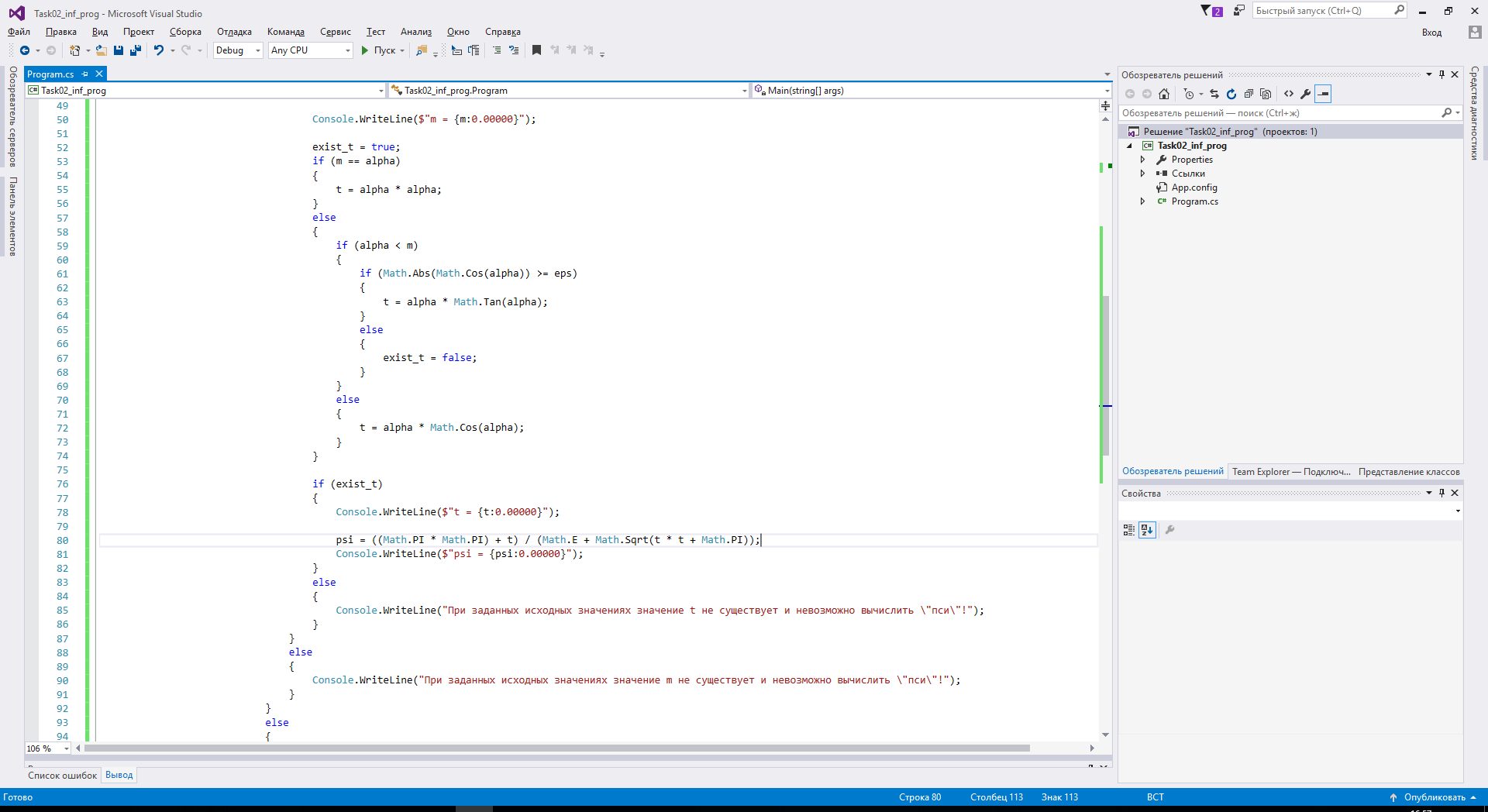


Для данного случая можно доказать, что отрицательное подкоренное выражение недостижимо.  – положительная константа; она **добавляется** к квадрату *t*. Все отрицательные и положительные значения *t* будут возведены в квадрат и результат будет не отрицателен (положительный или ноль). Отрицательным значения возникнуть неоткуда.

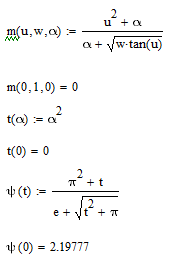
Аналогичным образом можно доказать, что знаменатель никогда не будет обращён в ноль.  – положительная константа, все результаты извлечения корня будут положительными и располагаться правее экспоненты на числовой оси. Проверки избыточны.

Таким образом, для всех существующих *m* и *t* будет существовать . Для тестирования подойдёт ситуация, в которой *m* и *t* содержат чистые нулевые значения.

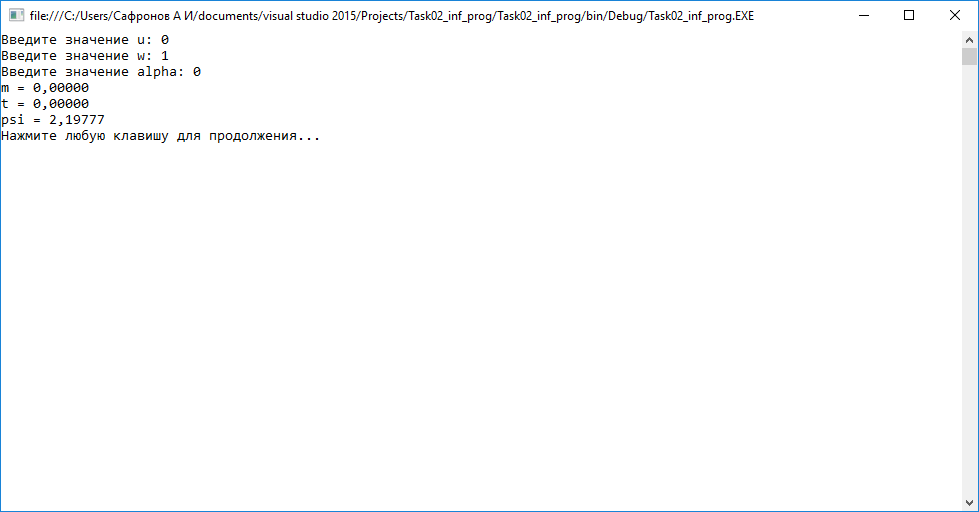
Допишем код в соответствии с вышеизложенным:



Подготовим тестовый пример:



Выполним проверку тестового примера на персональном компьютере с использованием составленной программы:



Полученный результат совпадает с ожиданием.

Дополненному фрагменту кода соответствует блок-схема:



Задача решена.