Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра экономической математики, информатики и статистики (ЭМИС)

ПЕРЕГРУЗКА ОПЕРАЦИЙ

Отчет по лабораторной работе по дисциплине «Программирование»

Студентка гр.548-1 К.Е.Анисимова

«\_\_» \_\_\_\_\_\_ 2019 г.

Доцент кафедры ЭМИС

Шельмина Е. А.

«\_\_»\_\_\_\_\_\_\_ 2019г.

Томск 2019

**Лабораторная работа №6**

**Перегрузка операций**

Цель работы: освоить и применить на практике функцию перегрузки операций.

Теоритические сведения

С++ поддерживает перегрузку операторов (operator overloading). За небольшими исключениями большинство операторов С++ могут быть перегружены, в результате чего они получат специаль­ное значение по отношению к определенным классам. Например, класс, определяющий связан­ный список, может использовать оператор + для того, чтобы добавлять объект к списку. Другой класс может использовать оператор + совершенно иным способом. Когда оператор перегружен, ни одно из его исходных значений не теряет смысла. Просто для определенного класса объектов определен новый оператор. Поэтому перегрузка оператора + для того, чтобы обрабатывать свя­занный список, не изменяет его действия по отношению к целым числам.

Операторные функции обычно будут или членами, или друзьями того класса, для которого они используются. Несмотря на большое сходство, имеется определенное различие между спосо­бами, которыми перегружаются операторные функции-члены и операторные функции-друзья. В этом разделе мы рассмотрим перегрузку только функций-членов. Позже в этой главе будет пока­зано, каким образом перегружаются операторные функции-друзья.

Для того, чтобы перегрузить оператор, необходимо определить, что именно означает опера­тор по отношению к тому классу, к которому он применяется. Для этого определяется функция-оператор, задающая действие оператора.

Общая форма записи функции-оператора для случая, когда она является членом класса, имеет вид:

*тип имя\_класса::operator#(список\_аргументов)*

*{*

*// действия, определенные применительно к классу*

*}*

Здесь перегруженный оператор подставляется вместо символа #, а тип задает тип значений, возвращаемых оператором. Для того, чтобы упростить использование перегруженного оператора в сложных выражениях, в качестве возвращаемого значения часто выбирают тот же самый тип, что и класс, для которого перегружается оператор. Характер списка аргументов определяется не­сколькими факторами.

В ранних версиях С++ было невозможно определить, предшествует или следует за операндом перегруженный оператор ++ или --. Например, для объекта О следующие две инструкции были идентичными:

O++;

++O;

Однако более поздние версии С++ позволяют различать префиксную и постфиксную форму опе­раторов инкремента и декремента. Для этого программа должна определить две версии функции operator++(). Одна их них должна быть такой же, как показано в предыдущей программе. Другая объявляется следующим образом: loc operator++(int х);

Если ++ предшествует операнду, то вызывается функция operator++(). Если же ++ следует за операндом, то тогда вызывается функция operator++(int х), где х принимает значение 0.

Действие перегруженного оператора по отношению к тому классу, для которого он опреде­лен, не обязательно должно соответствовать каким-либо образом действию этого оператора для встроенных типов С++. Например, операторы << и >> применительно к cout и cin имеют мало общего с их действием на переменные целого типа. Однако, исходя из стремления сделать код более легко читаемым и хорошо структурированным, желательно, чтобы перегруженные опера­торы соответствовали, там где это возможно, смыслу исходных операторов. Например, оператор + по отношению к классу three\_d концептуально сходен с оператором + для переменных целого типа. Мало пользы, например, можно ожидать от такого оператора +, действие которого на соответствующий класс будет напоминать действие оператора ||. Хотя можно придать перегру­женному оператору любой смысл по своему выбору, но для ясности его применения желательно, чтобы его новое значение соотносилось с исходным значением.

Имеются некоторые ограничения на перегрузку операторов. Во-первых, нельзя изменить при­оритет оператора. Во-вторых, нельзя изменить число операндов оператора. Наконец, за исклю­чением оператора присваивания, перегруженные операторы наследуются любым производным классом. Каждый класс обязан определить явным образом свой собственный перегруженный опе­ратор =, если он требуется для каких-либо целей. Разумеется, производные классы могут пере­грузить любой оператор, включая и тот, который был перегружен базовым классом. Следующие операторы не могут быть перегружены: . :: \* ?

Задача 1.

1. Для строки символов реализовать операции:

a. сравнение строк (операция ==);

b. удаление из строки заданного символа (операция –).

Кроме того, членом класса сделать функцию с именем strset() для удаления из первой строки всех символов, встречающихся во второй строке.

Скриншоты кода программы и его результата представлены на рисунках 1.1-1.4.

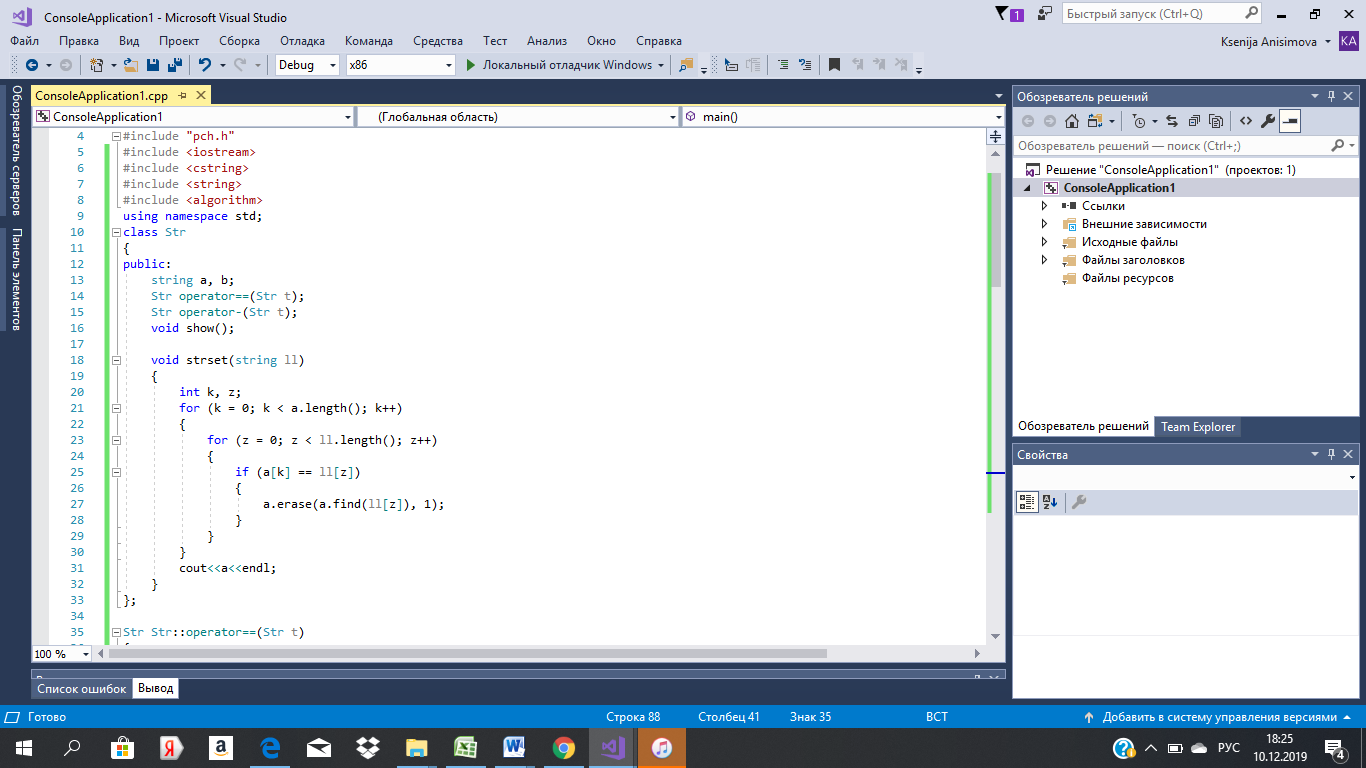


Рисунок 1.1 – скриншот кода программы

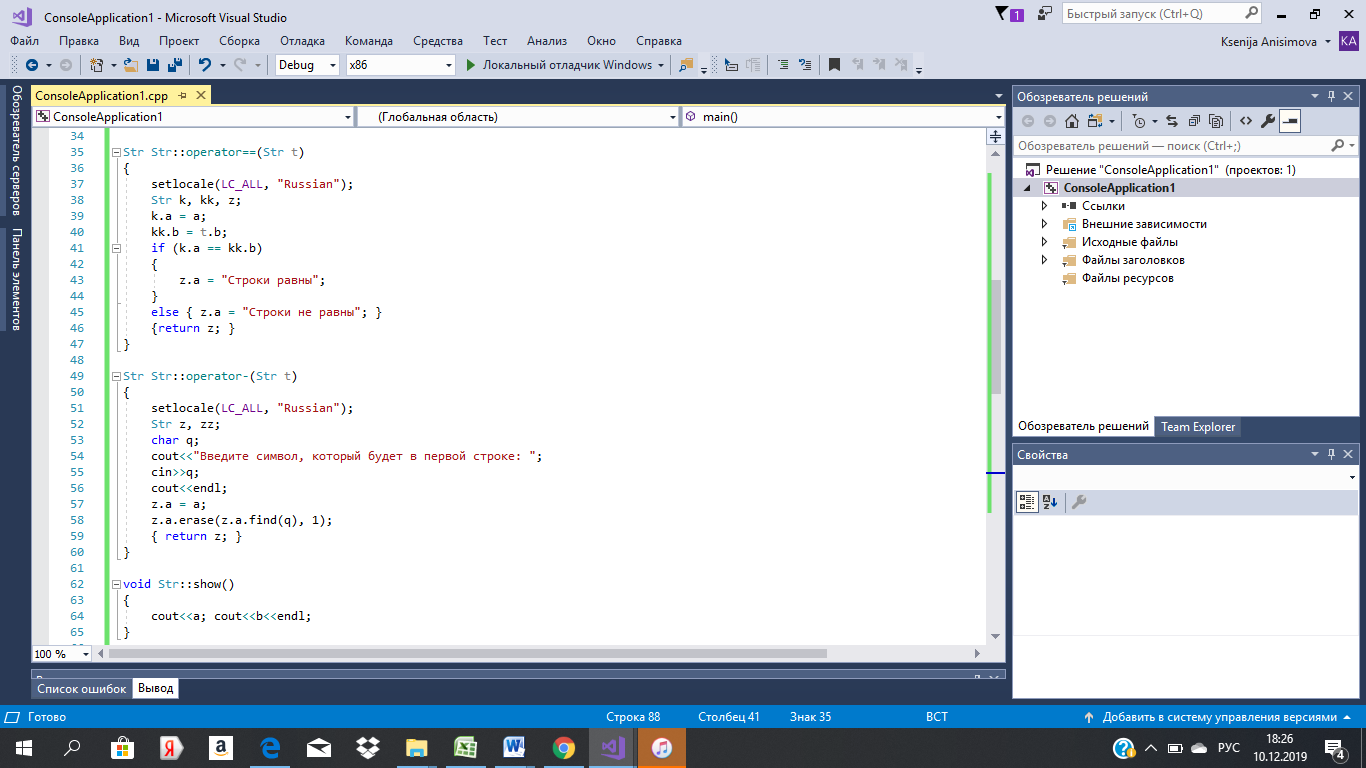


Рисунок 1.2 – скриншот кода программы

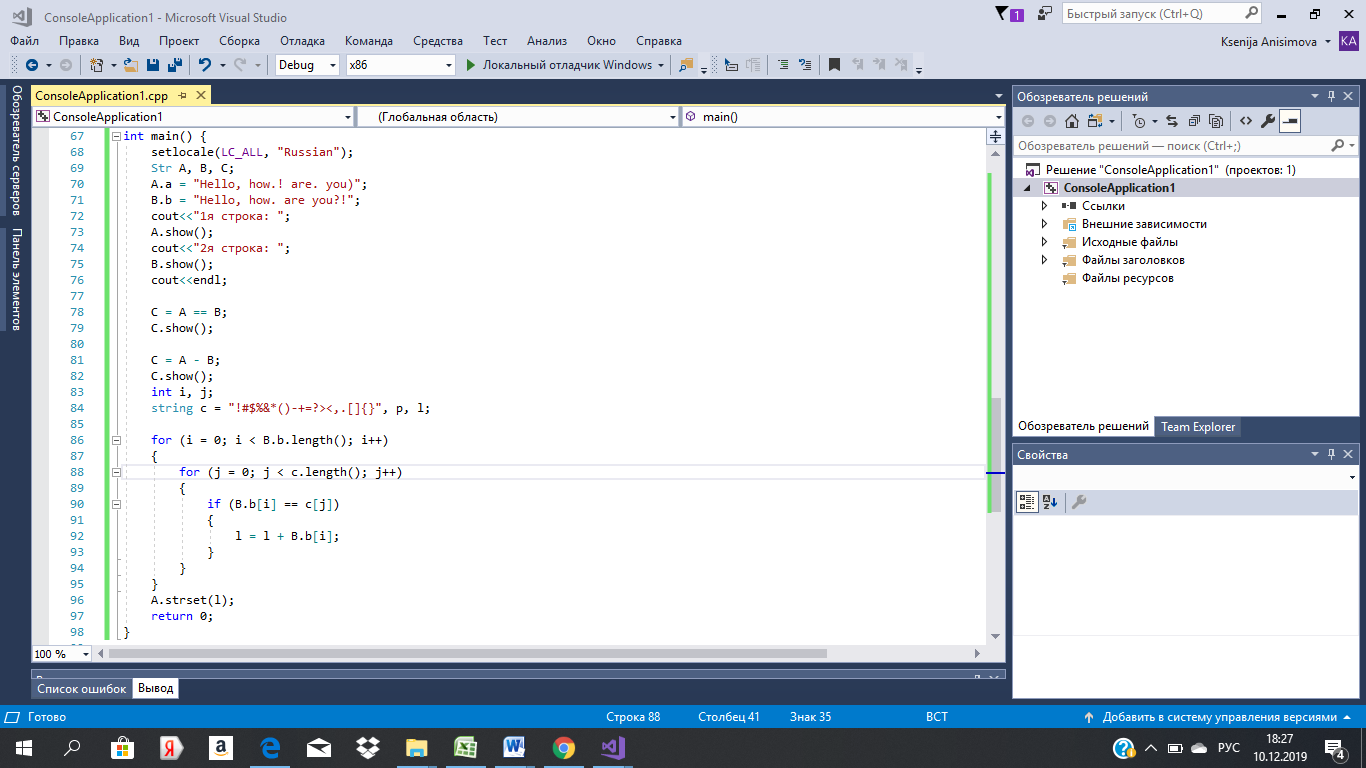


Рисунок 1.3 – скриншот кода программы

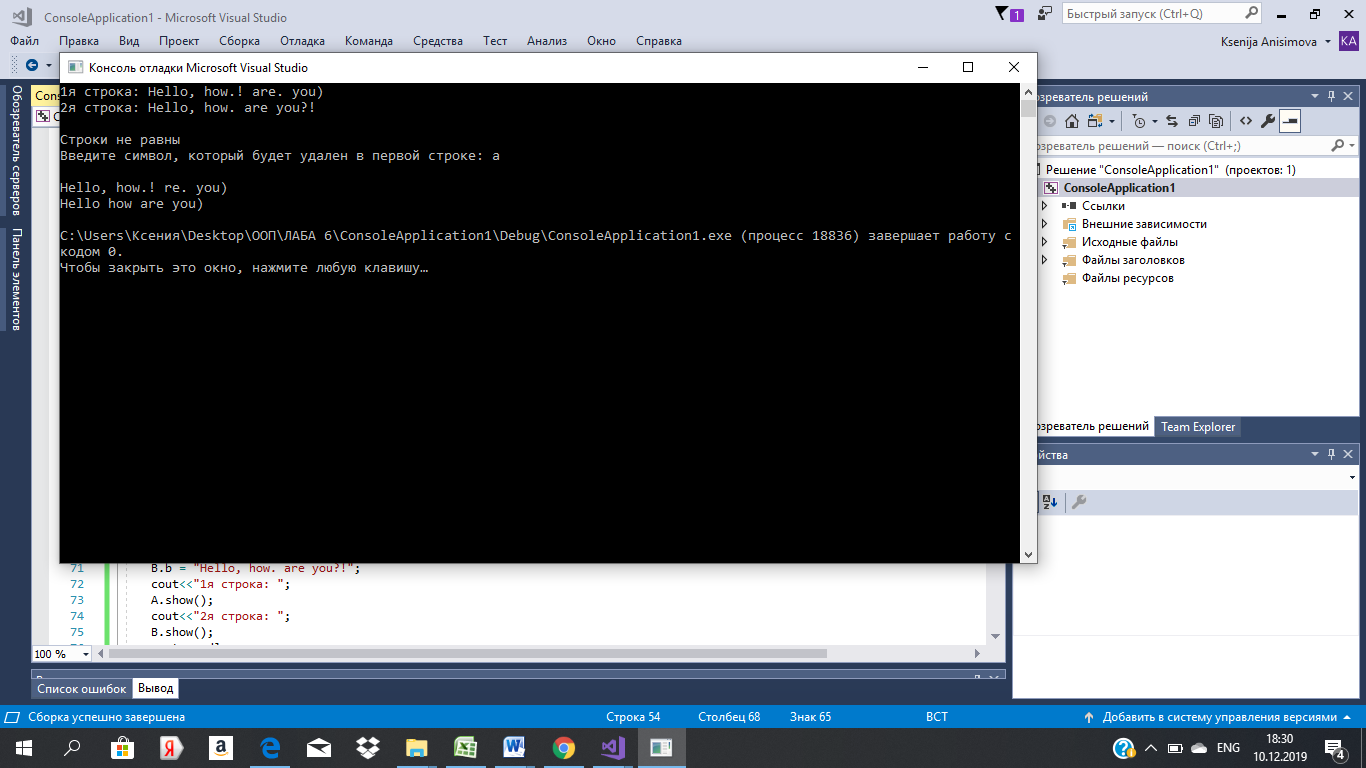


Рисунок 1.4 – скриншот результата программы

Задача 2.

Определить класс-строку. В класс включить два конструктора: для определения класса строки символов и путем копирования другой строки (объекта класса строки).

Определить операции над строками:

* >> перевертывание строки (запись символов в обратном порядке);
* ++ нахождение наименьшего слова в строке.

Скриншоты кода программы и его результата представлены на рисунках 2.1-2.3.

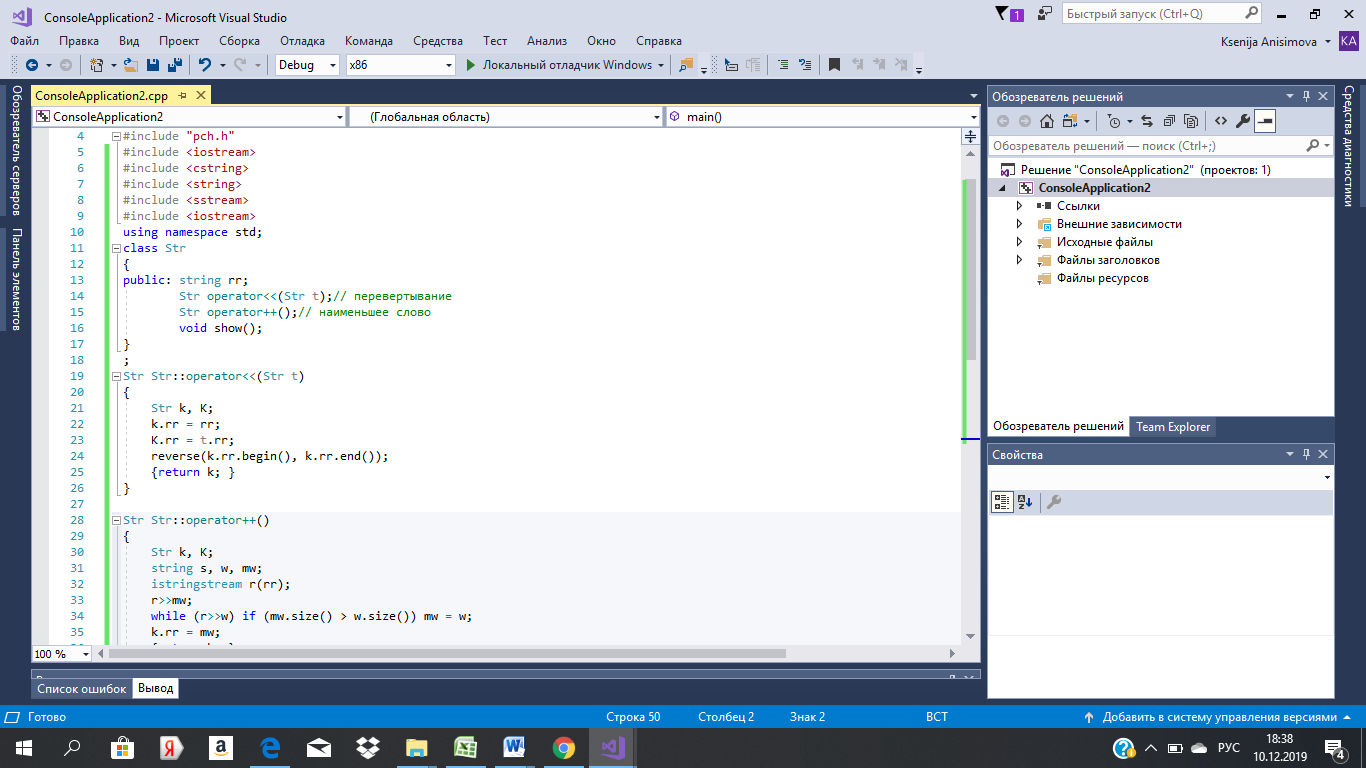


Рисунок 2.1 – скриншот кода программы

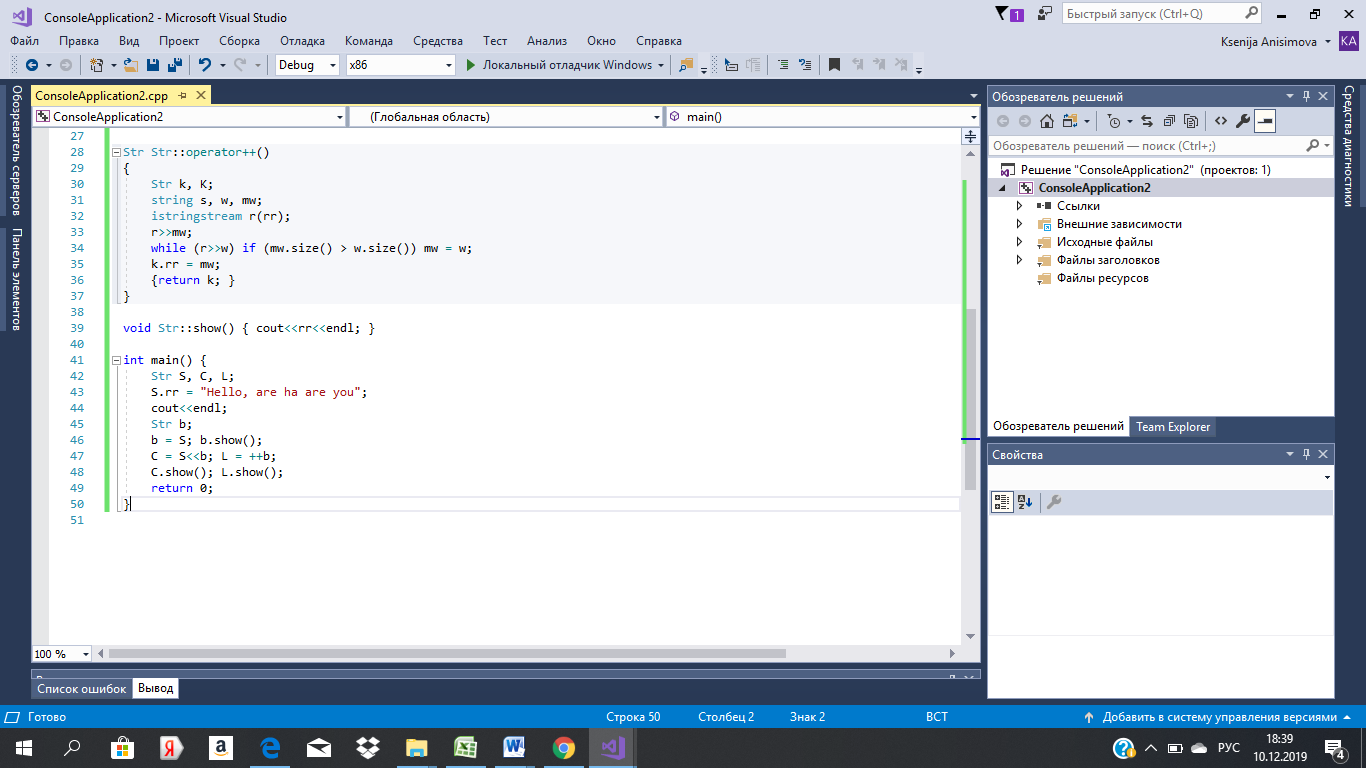


Рисунок 2.2 – скриншот кода программы

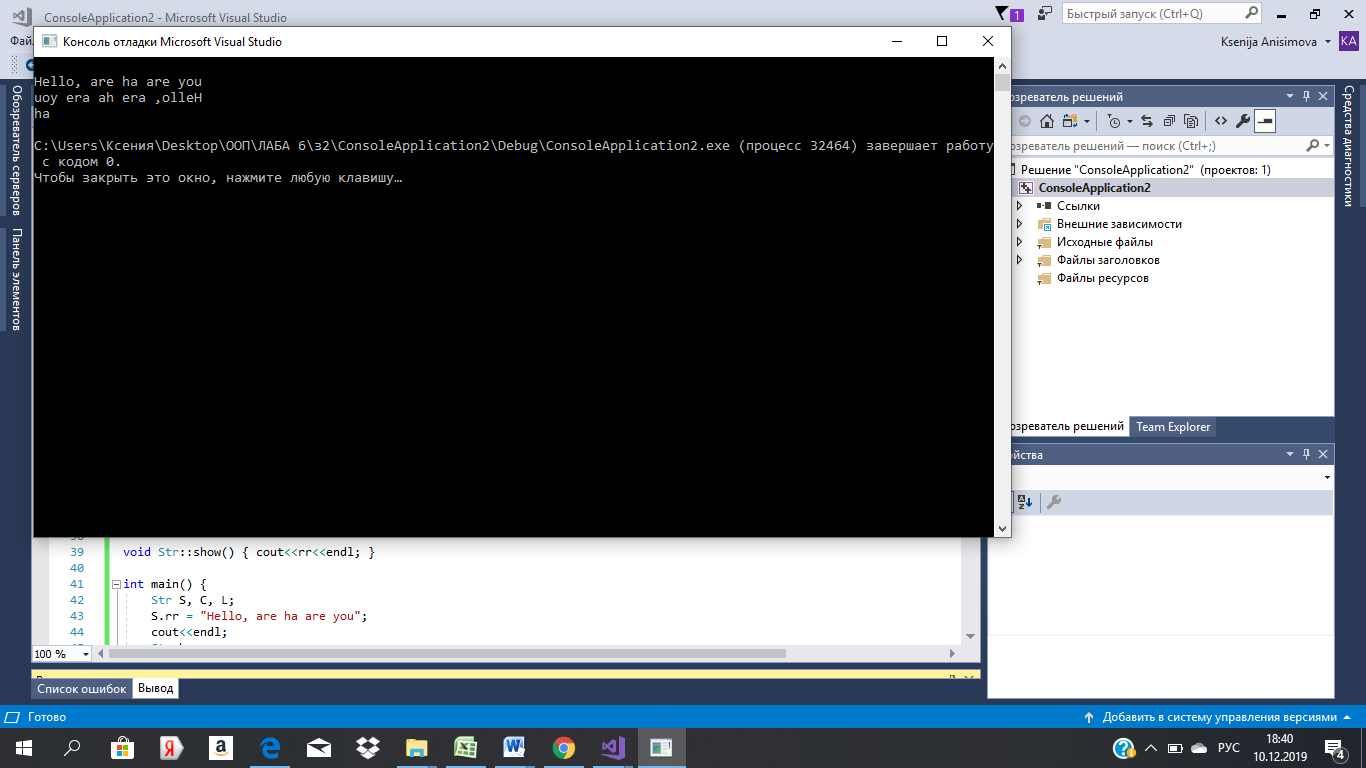


Рисунок 2.3 – скриншот результата программы

Задача 3.

При решении задач необходимо описать класс, который используется для представления элементов динамической структуры данных. Затем разрабатывается класс для работы с используемой динамической структурой данных, которая при тестировании класса может быть построена путем ввода данных: а) с клавиатуры; б) из файла.

Возможны два варианта решения:

1. динамическая структура данных постоянно хранится в памяти;
2. динамическая структура данных хранится в файле.

1. Создать класс для работы со стеком. Элемент стека – действительное число. Применить класс для вывода возрастающих серий последовательности действительных чисел: а) в обратном порядке; б) в том же порядке (серия – упорядоченная последовательность максимальной длины).

Скриншоты кода программы и его результата представлены на рисунках 3.1-3.4.

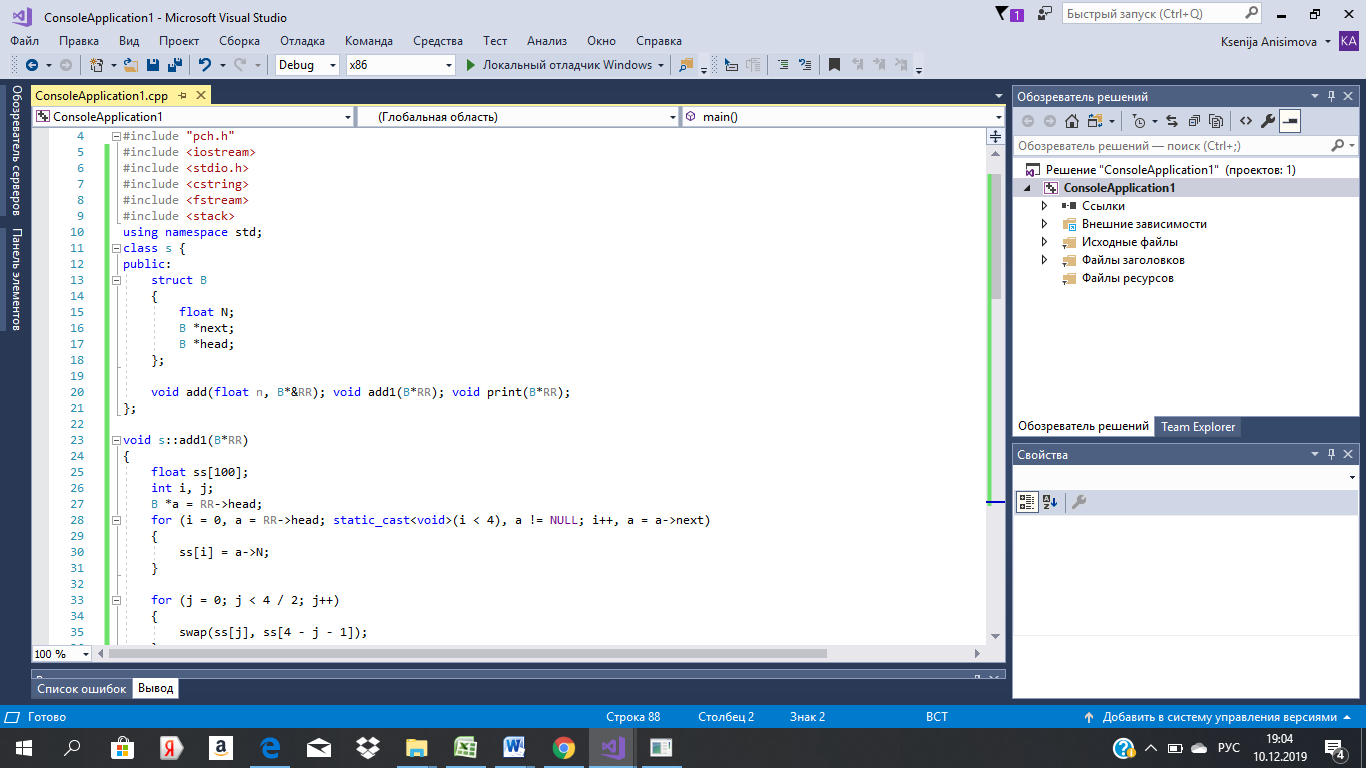


Рисунок 3.1 – скриншот кода программы

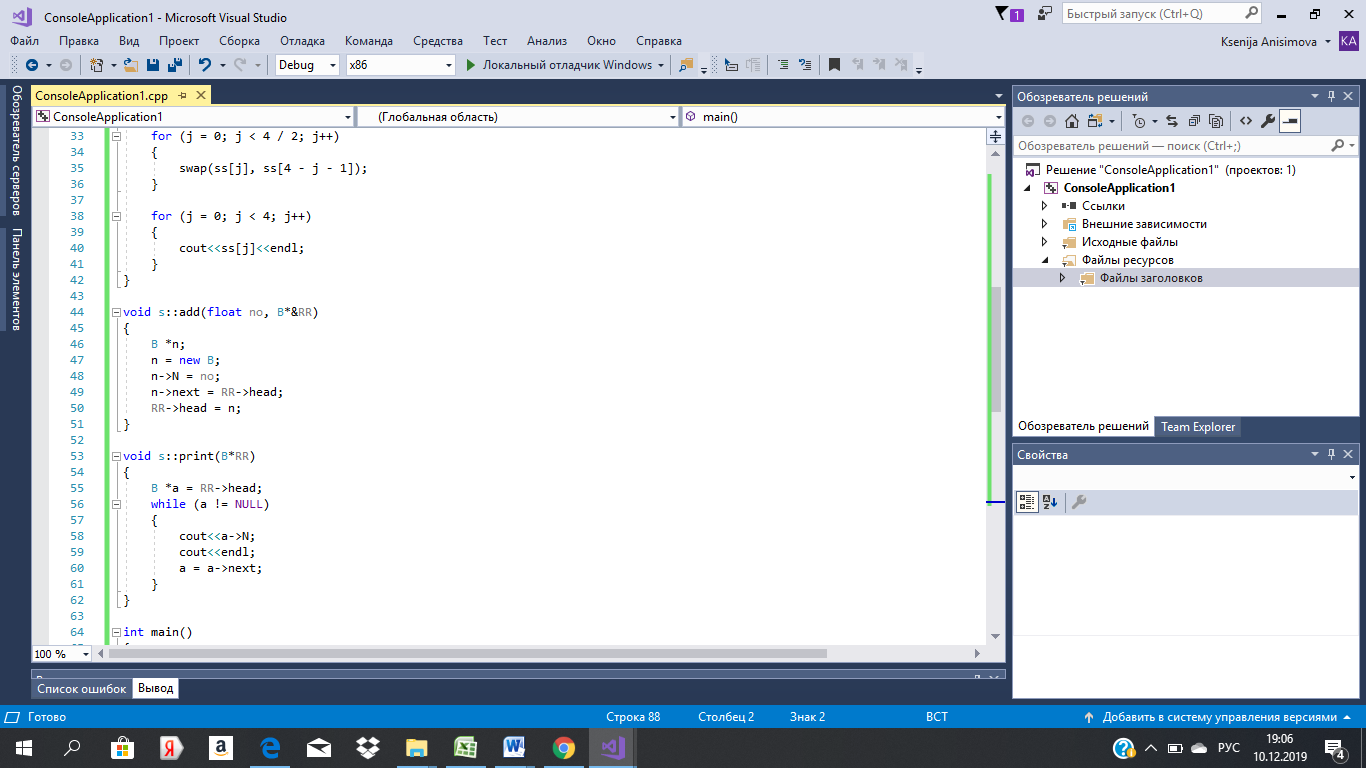


Рисунок 3.2 – скриншот кода программы

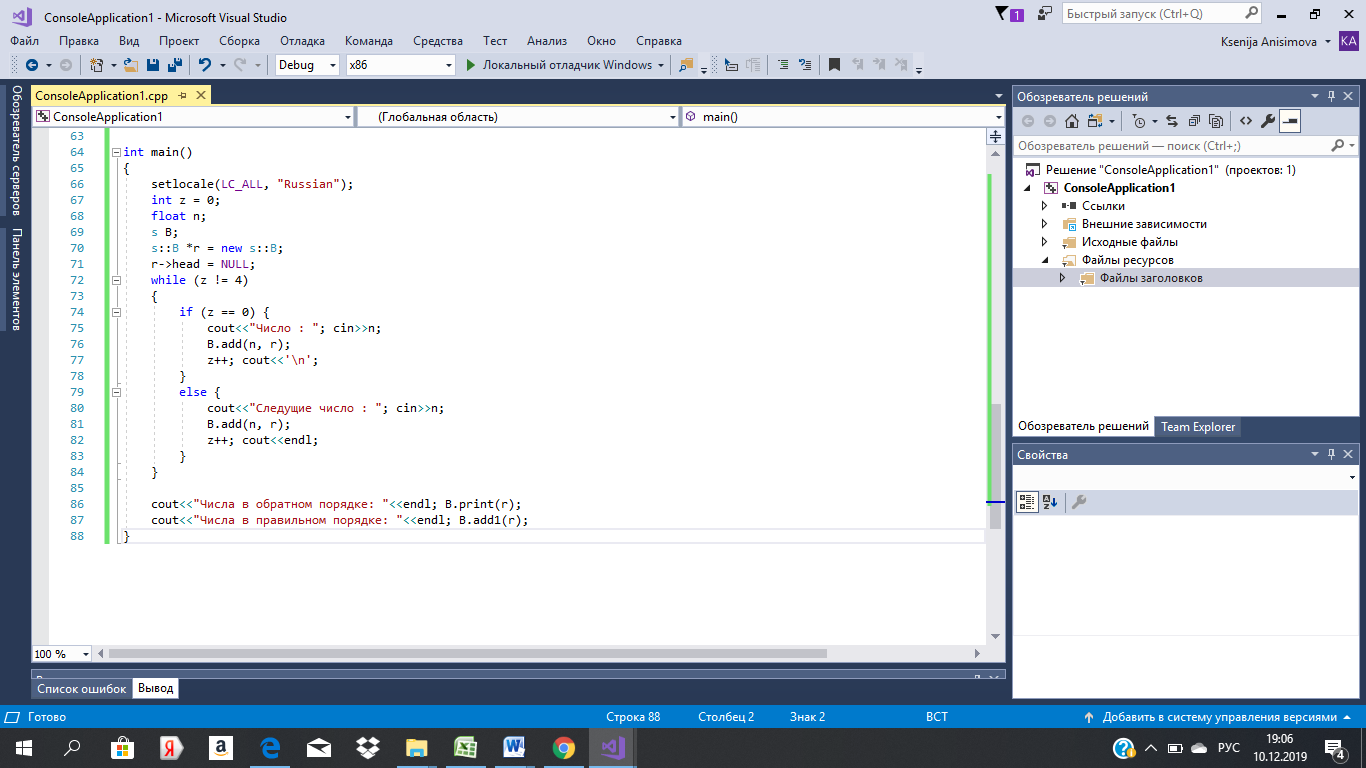


Рисунок 3.3 – скриншот кода программы

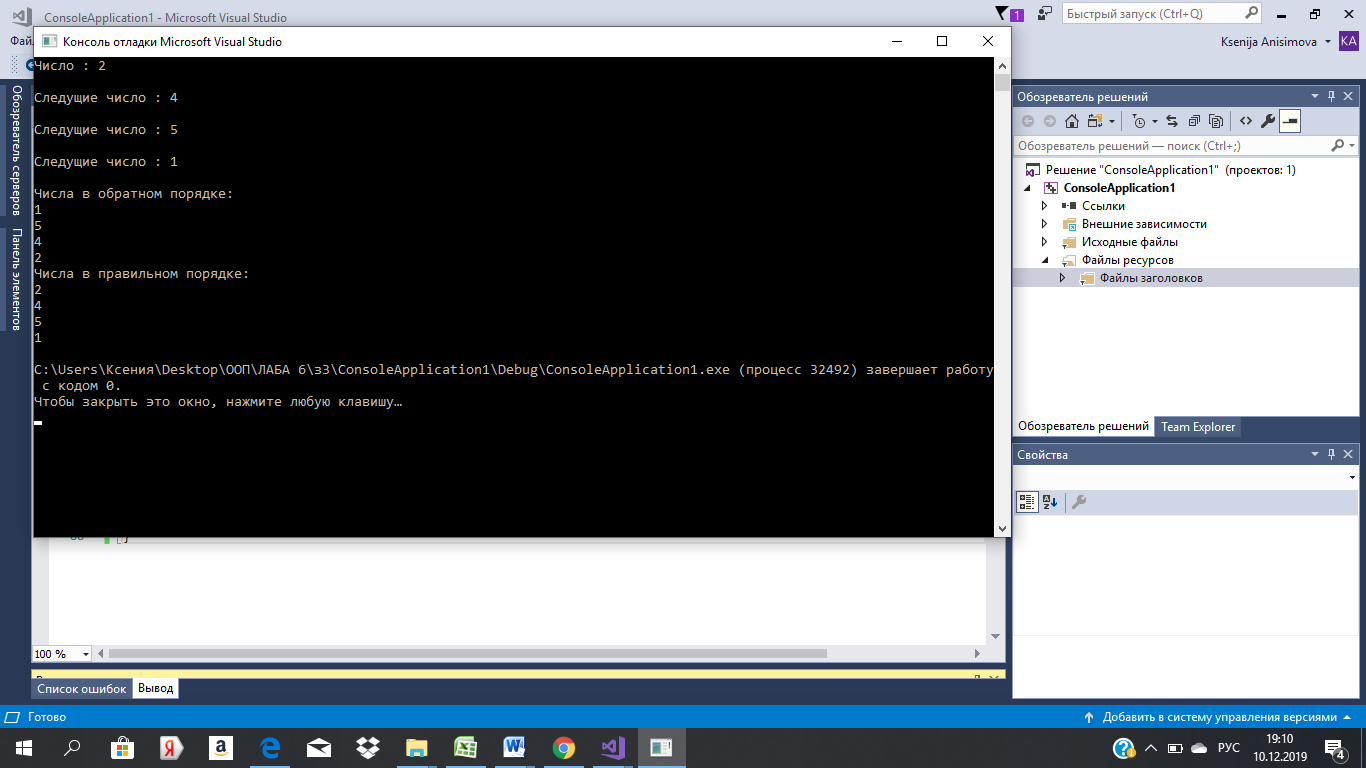


Рисунок 3.4 – скриншот результата программы

Вывод: освоены и применены на практике функции перегрузки операций.