Зарегистрировано №_____
«___»_____2019

_____ _____
подпись (расшифровка подписи)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра математического и программного обеспечения информационных систем

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПУЗЫРЬКОВОЙ СОРТИРОВКИ ПО ПЛОЩАДИ ДЛЯ ОБЪЕКТОВ ТИПА ТРЕУГОЛЬНИК

Курсовая работа
по дисциплине «Структуры и алгоритмы компьютерной обработки данных»

студента очного формы обучения
направления подготовки 02.03.02
«Фундаментальная информатика и информационные технологии».
Профиль подготовки: Супервычисления
2 курса группы 12001801
Капустина Виктора Сергеевича

Научный руководитель:
асс. Курлов В.В.

Допущен к защите
«___»_____2019 г.

_____ _____
Подпись (расшифровка подписи)

Оценка_____2019г.
«___»_____
Подпись (расшифровка подписи)

Белгород 2019

ВВЕДЕНИЕ

В наши дни информационные технологии играют одну из важнейших ролей уже в повседневной жизни. Компьютеры применяются в очень многих сферах деятельности человека. Это приводит к тому, что используется огромное количество информации. Чтобы в этих данных не запутаться, требуется их отсортировать. Для этого в информационной сфере существует компьютерная обработка данных. Основными её элементами являются алгоритмы сортировки. Они призваны упростить эту первостепенную процедуру, которая применима во многих областях программирования.

В общем смысле алгоритмы сортировки представляют из себя целенаправленный процесс упорядочивания данных элементов. На данный момент мало какие из проблем вызывали такого творческого подхода и разнообразия решений, как сортировка. Хоть она до сих пор и не имеет некого универсального метода, который бы резко выделялся среди других, но опираясь на цель и входные данные, существует возможность подобрать максимально оптимальный алгоритм для каждого частного случая.

Для написания этой курсовой работы, целью которой является изучение и программная реализация пузырьковой сортировки по площади для объектов типа “Треугольник” можно выделить следующие основные задачи:

1. Изучение теории по предметной области
2. Разработка приложения, для реализации пузырьковой сортировки по площади для объектов типа “Треугольник”
3. Подведение итогов на основе анализа выполненной работы

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Характеризация используемого объекта

Треугольник – геометрическая фигура, образованная тремя отрезками, которые соединяют три точки, не лежащие на одной прямой. Указанные три точки называют вершинами, а отрезки, соединяющие их – сторонами треугольника. Каждая из них образует в вершинах углы, исходя из этого, эту фигуру можно определить еще и как многоугольник с количеством углов равным трем. Изображение данного объекта показано на (Рис. 1.1)

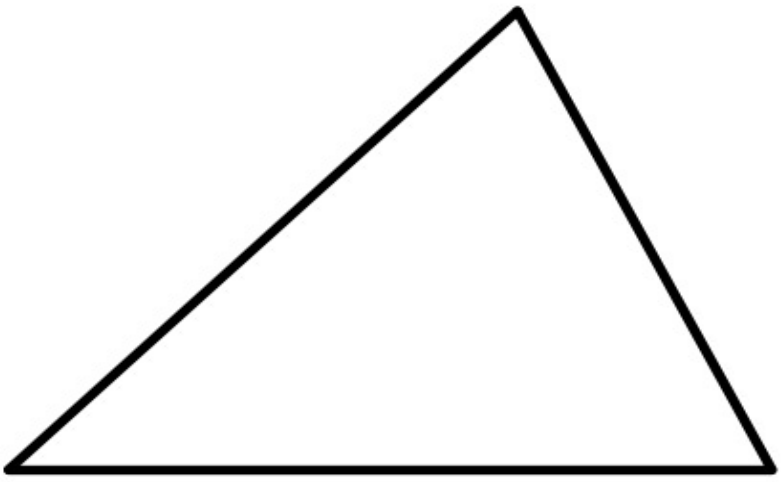

Рис. 1.1 Пример треугольника

Более того, треугольник может различаться по числу его сторон: разносторонний, равносторонний и равнобедренный.

Разносторонним является самый частый треугольник, стороны которого не совпадают.

Равносторонним будет соответственно треугольник с полностью идентичными друг другу сторонами. Также, этот вид треугольников имеет несколько, применимых только к нему свойств:

- Все его углы равны между собой и имеют значение в 60°
- Медианы, биссектрисы и высоты равностороннего треугольника совпадают и равны по формуле (a√3)/2

Равнобедренным же является треугольник, у которого две из трех сторон будут равны. Они будут называться боковыми, а оставшаяся из трех - основанием.

Каждый из видов треугольников имеет свои пути для нахождения площади треугольника, но так же имеется более универсальный сбособ - *нахождение площади треугольника по трем его вершинам*

$S\ ABC=½*|\ (x2-x1)(y3-y1)-(x3-x1)(y2-y1)|$

На абстрактном уровне представления, объект треугольник будет содержать значения всех его вершин и площади, а также метод, для их получения.

1.2 Анализ алгоритмов сортировок

Одной из наиболее важных целей является рационализация использования алгоритмов сортировки для различных задач. Для этого используется определенная система характеристик, основываясь на следующих свойствах:

1. *Устойчивость* – свойство, при котором устойчивая сортировка не меняет взаимного расположения равных элементов.

2. *Естественность поведения* - эффективность метода при применении с уже полностью или частично упорядоченными данными.
3. *Время сортировки* - Отражения быстродействия метода сортировки
4. *Память* - требование дополнительной памяти для хранения данных.

С их помощью можно оценить общую эффективность того или иного алгоритма сортировки, которых на сегодняшний день существует немало, хотя их подходы и эффективность различны. Основными же являются: *Сортировка выбором, Пузырьковая сортировка, Сортировка вставками, Сортировка расческой, Сортировка Шелла, Сортировка деревом, Шейкерная Сортировка, Пирамидальная сортировка.*

Сортировка выбором: Происходит путем нахождения минимума в массиве после текущего элемента и обмена с ним, при необходимости. Таким образом, после n-ой операции, эти n элементов будут стоять на своих местах.

Сортировка вставками: Самая частая сортировка. Происходит путем создания массива, в котором и получится итог сортировки после завершения алгоритма. Процесс заключается во вставки элементов из исходного массива, тем самым сразу сортируя этот массив.

Сортировка расческой: Очередная модификация для пузырьковой сортировки. Основная идея строится на проблем “черепах” в оригинальном алгоритме. Для ее решения используется перестановка элементов, стоящих на расстоянии, при необходимости. Расстояние обмена элементов местами же с проходами сужается, пока не дойдет до 1, тем самым перейдя на пузырьковую сортировку.

Сортировка Шелла: Фактически является усовершенствованным вариантом сортировки вставками, аналогично как сортировка пузырьком трансформируется в сортировку расческой. Основная идея заключается в том, что подгруппы элементов также сортируются вставками, но только они не пойдут в ряд в этой же подгруппе, а выбираются равномерно с некоторым шагом. После прохода по ряду элементов,мы уменьшаем длину шага в заданное количество раз, пока расстояние между элементами этих подмножеств не достигнет единицы.

Сортировка деревом: По сути является реализацией поиска в глубину, с выстраиванием двоичного дерева поиска перед этим.

Шейкерная сортировка: Так же известна как сортировка перемешиванием или коктейльная. Является улучшением пузырьковой сортировки. После прохода в одну сторону, алгоритм пойдет обратно, только с противоположным условием, тем самым практически удваивая эффективность.

Пирамидальная сортировка: Представляет из себя полное бинарное дерево, для которого выполняется следующей свойство: *Приоритет вершины больше приоритетов ее потомков.* Если элемент больше родителя, то он “поднимается” на уровни выше до тех пор, пока не будет соблюдено приведенное выше свойство.

1.3 Алгоритм пузырьковой сортировки

Сортировка пузырьком: Упорядочивание происходит в результате многократного, последовательного перебора элементов и сравнения их пар между собой. Если сравниваемые элементы не отсортированы друг относительно друга – то происходит их замена друг на друга. С каждым проходом, его путь уменьшается на один элемент, т.к. наибольший(или

наименьший, в зависимости от направления прохода) элемент будет на своем месте.

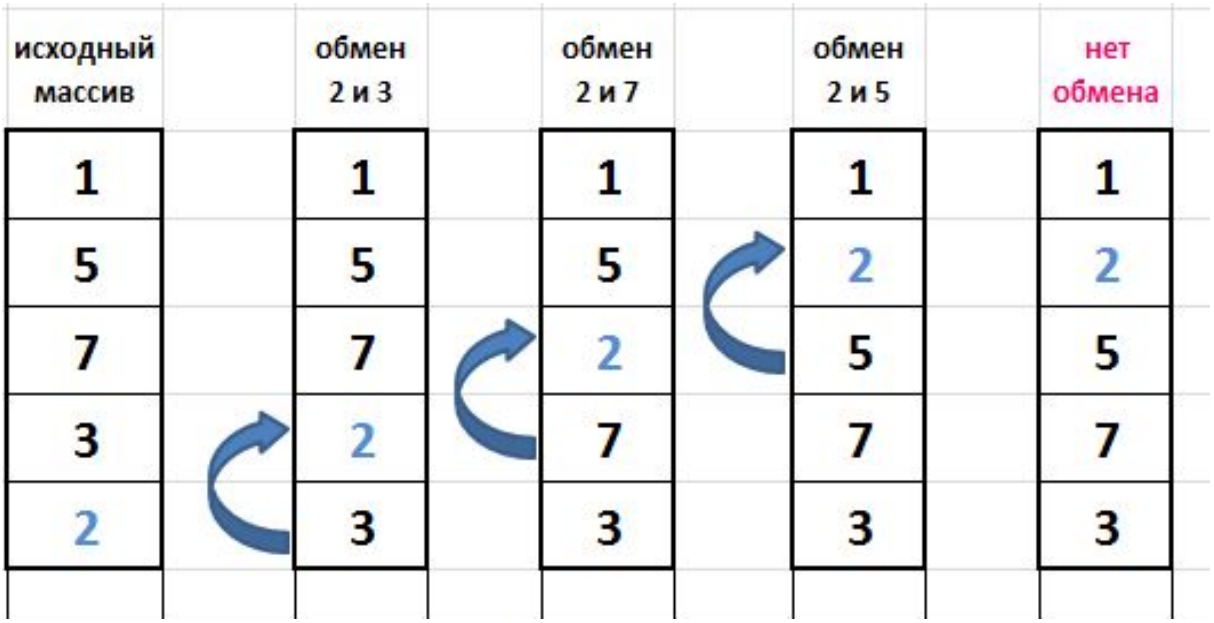

Рис 1.2 Пример пузырьковой сортировки

Является одним из простейшим алгоритмов сортировки, эффективность которого проявляется, только на небольших массивах. Из-за его “сложности” больше считается учебным, между тем все же лежит в основе более совершенных и эффективных алгоритмов (Пирамидальная сортировка, Шейкерная, быстрая сортировка). Без дополнительной модификации число сравнений в лучшем и худшем случае будет равно, различаясь только в числе обменов элементами. Сложностью алгоритма является $O(n^2)$

Эффективность данного алгоритма проявляется в том, что для задач с массивами малой размерности время почти не различается по сравнению с более эффективными алгоритмами. Код при этом на порядок легче и доступнее для понимания. Все это обуславливает общую конкурентоспособную эффективность в данной области задач.

```
ЦИКЛ ДЛЯ J=1 ДО N-1 ШАГ 1
  F=0
  ЦИКЛ ДЛЯ I=1 ДО N-J-1 ШАГ 1
    ЕСЛИ A[I] > A[I+1] ТО ОБМЕН A[I],A[I+1]:F=1
  СЛЕДУЮЩЕЕ I
  ЕСЛИ F=0 ТО ВЫХОД ИЗ ЦИКЛА
СЛЕДУЮЩЕЕ J
```

Рис 1.3 Псевдокод Пузырьковой сортировки

1.4 Выводы из анализа

На основе анализа используемого нами объекта типа “Треугольник”, а также алгоритма его сортировки, можно вывести,что для реализации программы потребуются следующие методы:

1. Создание самого массива под объекты. Так как в вычислениях не обязательно могут быть целочисленные данные, то и содержать придется вещественные числа.
2. Соответственное удаление этого массива для очистки памяти
3. Передача в элемент массива данные ключевого поля объекта типа “треугольник”
4. Сама операция сортировки действующая в созданном массиве.

Будет создан класс удовлетворяющий решению поставленных задач, который и будет содержать вышеуказанные методы.

Помимо типа данных самого алгоритма, нам потребуется класс треугольника, который будет содержать в себе координаты каждой вершины и площадь треугольника, а также метод для их получения.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Разработка программы

Программная реализация Пузырьковой сортировки по площади для объектов типа треугольник была составлена, опираясь на объектно-ориентированное программирование. Оно подразумевает использование классов, объектов и их методов.

Класс - основной элемент ООП, описывающий абстрактный тип данных, а также его реализацию.

Объект или же экземпляр класса является реализацией класса, некоторой “сущностью” принимающая определенные свойства и методы.

Метод - Та же функция, только уже в рамках класса и его объектов.
При реализации программы использовались следующие библиотеки:

- Библиотека iostream – определяет объекты ввода-вывода на стандартных потоках. Cout – стандартный выходной поток, cin – стандартный входной поток.
- Библиотека math.h – используется для выполнения простых математических операций.

Листинг 1. Подключение библиотек.

```
#include<iostream>
#include <math.h>
```

Конец листинга 1.

Также было подключено пространство имен std, для определения области кода для библиотеки iostream

Листинг 2. Подключение пространства имен.

```
using namespace std;
```

Конец листинга 2.

После этого объявляем класс tri, описание которого будет позже.

Листинг 3

```
Class tri;
```

Конец листинга 3

2.2 Разработка классов

В ходе программной реализации было использовано два класса: Класс str - основной тип данных, содержащий массив, в который и будут заноситься объекты типа треугольник. Он включает в себя 5 методов:

Метод Create - служит для создания массива с указанным количеством элементов, которое вводит пользователь.

Листинг 4. Метод Create

```
void create(float *&arr, int quant)
{
	arr = new float[quant];
}
```

Конец листинга 4.

Метод Delete - Отвечает за очистку массива из памяти, по завершению основного процесса программы. Принимает на вход массив.

Листинг 5. Метод Delete

```
void Delete(float *arr)
{
	delete[]arr;
	arr = NULL;
}
```

Конец листинга 5.

Метод Count - Вставляет в выбранную ячейку массива полученный элемент и переключается на следующую. Принимает на вход массив и значение площади объекта типа треугольник.

Листинг 6. Метод Count.

```
void count(float *arr, float num)
{
arr[Lst_E] = num;
Lst_E++;
}
```

Конец листинга 6.

Метод Show - служит для отображения получившегося массива на экране. Выводит значение с точностью до 1 числа после запятой.

Листинг 7. Метод show.

```
void show(float *arr)
{
for (int i = 0; i < Lst_E; i++) printf(" %.1f\n", arr[i]);
}
```

Конец листинга 7.

Метод bubble - Основной метод, являющийся алгоритмом пузырьковой сортировки. Принимает на вход имеющийся массив, и после сортирует все его элемента по возрастанию. Для этого в начале создается переменная flag, которая копирует значение количества элементов массива. После этого на этом диапазоне происходит прохождение проверки всех соседних элементов слева направо. Если левый элемент больше правого, то они меняются местами, путем ввода дополнительной переменной t, которая способствует этому. После прохождение всего массива, подразумевается,что максимальный элемент встал в самую правую ячейку, поэтому мы сужаем диапазон на

единицу и начинаем проверять массив заново, пока в итоге диапазон не останется равен двум.

Листинг 8. Метод bubble.

```
void bubble(float *arr)
{
	int flag = Lst_E;
	while (flag != 2)
	{
		for (int i = 0; i < flag - 1; i++)
		{
			if (arr[i] > arr[i + 1])
			{
				float t;
				t = arr[i + 1];

				arr[i + 1] = arr[i];
				arr[i] = t;
			}

		}
		flag--;
	}

}
```

Конец листинга 8.

Помимо методов, класс содержит в себе параметр Lst_E, который обозначает индекс последнего элемента массива.

Листинг 9. Объявление переменной Lst_E.

```
int Lst_E = 0;
```

Конец листинга 9.

Заносить в массив мы будем элементы другого класса и чтобы был доступ, класс необходимо объявить дружественным.

Листинг 10. Дружественные классы.

```
friend tri;
```

Конец листинга 10.

Следующим прописывается класс tri. Он является описанием объекта треугольника и содержит в себе следующие параметры:

X1 - Координата x первой вершины

Y1 - Координата y первой вершины

X2 - Координата x второй вершины

Y2 - Координата y второй вершины

X3 - Координата x третьей вершины

Y3 - Координата y третьей вершины

Value - Площадь треугольника

Листинг 11. Инициализация переменных класса tri.

```
float x1, y1,x2,y2,x3,y3;
float value;
```

Конец листинга 11.

Единственным методом этого класса является взятие всех приведенных переменных и расчет из них площади треугольника.

Листинг 12. Метод GetData.

```
void GetData()
{
	cout << "Input vertices of the triangle" << endl;
```

```
        cout << "x1 ";
        cin >> x1;
        cout << "y1 ";
        cin >>y1;
        cout << "x2 ";
        cin >> x2;
        cout << "y2 ";
        cin >> y2;
        cout << "x3 ";
        cin >> x3;
        cout << "y3 ";
        cin >> y3;
        value = 0.5*abs( ( (x1 - x3)*(y2 - y3) ) - ( (x2 - x3)*(y1 - y3) ) );
}
```

Конец листинга 12.

2.3 Главная функция

Всякая программа всегда содержит main в своем коде. Обычно это глобальная функция и она указывает точку старта программы. В начале нашей программы инициализируется пустой массив с указанием в NULL. После этого создается тип данных для сортировки с названием bubble.

Листинг 13. Начало главной функции программы.

```
int main()
{
        float *ArrStr = NULL;
        str bubble;
```

Конец листинга 13.

Чтобы создать массив, нам потребуется знать количество элементов,чтобы под них создать соответствующее количество ячеек в массиве. Поэтому создаем переменную и запрашиваем пользователя ввести ее значение.

Листинг 14. Получение количества элементов.

```
int quant;

	cout << "Input a value of triangles" << endl;

	cin >> quant;
```

Конец листинга 14.

Получив нужное значение собственно и используем метод для создания массива.

Листинг 15.

```
bubble.create(ArrStr, quant);
```

Конец листинга 15.

После создания массива необходимо инициализировать элементы для занесения в массив. Для этого создаем объект типа треугольник. После этого проходим цикл в зависимости от значения , которое было введено пользователем ранее. В начале вызывается метод, чтобы получить от пользователя данные по треугольнику с последующим расчетом площади по этим же данным,а потом вызывается метод для занесения получившегося значения в ячейку массива.

Листинг 16. Получение и внесение площади треугольника в массив.

```
tri tr1;


for (int i = 0; i < quant; i++)

	{

		tr1.GetData();
```

```
            bubble.count(ArrStr, tr1.value);
        }
```

Конец листинга 16.

После заполнения массива наступило время для основного алгоритма данной работы - применения алгоритма сортировки. После этого программа выведет уже отсортированный массив на экран пользователя и очистит данные из памяти.

Листинг 17. Применение алгоритма сортировки.

```
bubble.bubble(ArrStr);
bubble.show(ArrStr);
bubble.Delete(ArrStr);
```

Конец листинг 17.

После этого действие программы подходит к концу и чтобы консольное окно сразу не закрылось применяется функция паузы, которая будет ждать любого нажатия на клавишу. Также возвращается значение 0, как отсутствие ошибок.

Листинг 18. Завершение программы.

```
system("pause");
        return 0;
}
```

Конец листинг 18.

2.4 Тестирование программы

Неотъемлемой частью любой работы должна быть тестированием работоспособности. В данном случае для теста возьмем значения десяти треугольников и проверим получившиеся значения.

Таблица 1

Тестовые значения

№	(x1,y1)	(x2,y2)	(x3,y3)	Площадь
1	(23,11)	(12, 5)	(1, 0)	5.5
2	(24,12)	(13,6)	(1, 0)	3
3	(25,11)	(10, 5)	(1, 0)	10.5
4	(22, 11)	(10, 5)	(2, 0)	6
5	(25, 12)	(13, 4)	(3, 0)	16
6	(24, 13)	(11, 7)	(8, 0)	36.5
7	(25, 14)	(12, 8)	(8, 0)	40
8	(26, 13)	(11, 7)	(8, 0)	43.5
9	(27, 14)	(12, 5)	(3, 0)	3
10	(30, 20)	(15, 10)	(8, 0)	40

По итогу программа должна вывести уже отсортированные по возрастанию значения.

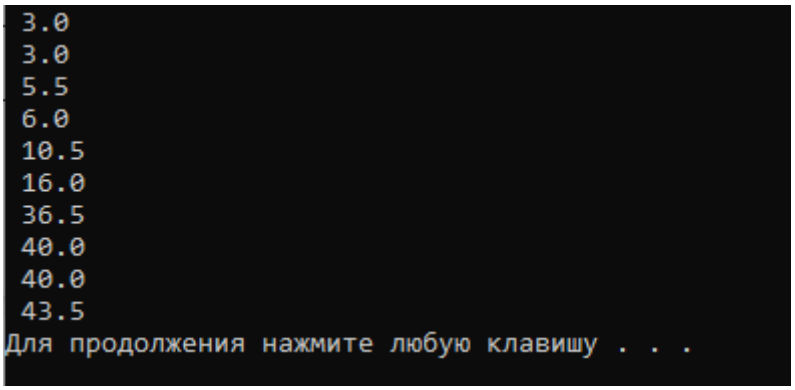

Рис 2.1 Выведенные значения

3. Заключение

В процессе написания курсовой на тему “Программная реализация пузырьковой сортировки по площади для объектов типа треугольник” поставленная цель была выполнена.

Для ее достижения все следующие задачи были решены:

1. Изучение теории по предметной области
2. Разработка приложения, для реализации пузырьковой сортировки по площади для объектов типа “Треугольник”
3. Подведение итогов на основе анализа выполненной работы

После изучения работы алгоритма, можно прийти к выводу, что сложность сортировки может зависеть не только от количества элементов, но и от их упорядоченности. Также, несмотря на просто и не всегда эффективность данного метода сортировки, он является базовым и лежит в основе других более совершенных алгоритмов, поэтому умение пользоваться им просто необходимо.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Уоррен. Генри, С. Алгоритмические трюки для программистов, испр. изд. : Пер. с англ. — М.: Издательский дом "Вильяме", 2004. — 288с. : ил. — Парал. тит. англ.
2. Гудман С. Введение в разработку и анализ алгоритмов / Гудман С., Хидетниеми С. – Москва: Мир, 1981. – 368с
3. https://is.gd/5cs1v6

ПРИЛОЖЕНИЕ

Листинг программы.

```
#include<iostream>
#include <math.h>
using namespace std;
class tri;
class str
{
public:
	int Lst_E = 0;
	friend tri;
	void create(float *&arr, int quant)
	{
		arr = new float[quant];
	}
	void Delete(float *arr)
	{
		delete[]arr;
		arr = NULL;
	}
	void count(float *arr, float num)
	{
		arr[Lst_E] = num;
		Lst_E++;
	}
	void show(float *arr)
	{
		for (int i = 0; i < Lst_E; i++) printf(" %.1f\n", arr[i]);
```

```
        }
        void bubble(float *arr)
        {
            int flag = Lst_E;
            while (flag != 2)
        {
            for (int i = 0; i < flag - 1; i++)
            {
                if (arr[i] > arr[i + 1])
                {
                    float t;
                    t = arr[i + 1];
                    arr[i + 1] = arr[i];
                    arr[i] = t;
                }
            }
                flag--;
        }

        }
};
    class tri
{
public:
    float x1, y1,x2,y2,x3,y3;
    float value;
        void GetData()
        {
            cout << "Input vertices of the triangle" << endl;
```

```
            cout << "x1 ";
            cin >> x1;
            cout << "y1 ";
            cin >>y1;
            cout << "x2 ";
            cin >> x2;
            cout << "y2 ";
            cin >> y2;
            cout << "x3 ";
            cin >> x3;
            cout << "y3 ";
            cin >> y3;
            value = 0.5*abs( ( (x1 - x3)*(y2 - y3) ) - ( (x2 - x3)*(y1 - y3)) );
        }
};

int main()
{
        float *ArrStr = NULL;
        str bubble;
        int quant;
        cout << "Input a value of triangles" << endl;
        cin >> quant;
        bubble.create(ArrStr, quant);
        tri tr1;
        for (int i = 0; i < quant; i++)
        {
            tr1.GetData();
            bubble.count(ArrStr, tr1.value);
```

```
        }
        bubble.bubble(ArrStr);
        bubble.show(ArrStr);
        bubble.Delete(ArrStr);
        system("pause");
        return 0;
    }
```

Конец листинга