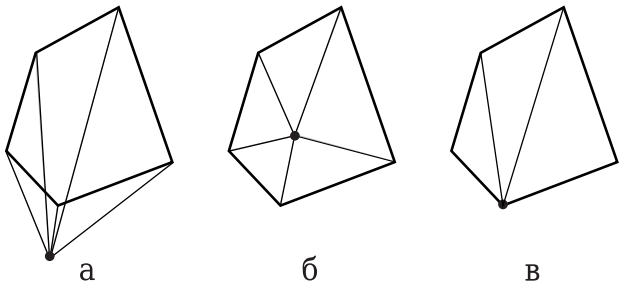
Задание III

Вариант 1

На плоскости заданы выпуклый многоугольник (указаны координаты вершин) и некоторая точка. Программа должна сообщить, находится ли эта точка внутри многоугольника (включая границу) или вне его.

Подсказка. Один из возможных алгоритмов решения этой задачи основан на подсчете суммы площадей треугольников, которые получатся, если соединить указанную точку со всеми вершинами многоугольника (см. рисунок). Если точка лежит вне многоугольника, то суммарная площадь треугольников больше площади многоугольника (вариант «а»); в противном случае площади равны (вариант «б»). Чтобы найти площадь многоугольника, можно разбить его на треугольники, выбрав одну из вершин и соединив ее с остальными, как показано на рисунке «в»



а б в

Для вычисления площади треугольника удобно использовать формулу Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где a, b, c — стороны треугольника, $p = (a+b+c)/2$ — его полупериметр.

Указания по программированию. Программа вначале вводит число вершин многоугольника N и их координаты x_i, y_i $(i = 1, 2, \ldots, N)$. Затем она (многократно) запрашивает координаты точек и каждый раз сообщает, находится ли эта точка внутри или вне многоугольника.

Координаты точки на плоскости удобно хранить в виде массива (вектора) из двух элементов; для хранения вершин многоугольника необходим массив точек (двумерный массив). Вершины размещаются в порядке

обхода контура многоугольника, т. е. в соседних элементах массива находятся вершины, принадлежащие одной стороне.

Напишите три функции:

```
double dist(double a[2], double b[2])
double area(double a[2], double b[2], double c[2])
int isoutside(int n, double p[][2], double z[2])
```

Функция `dist(a,b)` дает расстояние между точками `a` и `b`; функция `area(a,b,c)` вычисляет площадь треугольника с вершинами `a`, `b`, `c`, обращаясь к `dist` для нахождения длин сторон, а логическая функция `isoutside(n,p,z)` вычисляет и сравнивает две площади — площадь многоугольника `p` с `n` вершинами и сумму площадей треугольников с участием точки `z`, возвращая ненулевое значение (т. е. «истина»), если точка `z` находится вне многоугольника, и нуль («ложь»), если точка лежит внутри или на границе. Главная программа `main()` вводит и хранит координаты, обращается к функции `isoutside` и выводит результат.

Вариант 2

Заданы N прямых на плоскости; каждая прямая определена парой чисел — коэффициентами a и b уравнения $y = ax + b$[1]. Программа должна определить количество точек пересечения этих прямых и вывести их координаты. (Если в исходных данных встретилось несколько одинаковых прямых, считать, что это одна и та же прямая).

Исходные данные (количество прямых и их коэффициенты) программа вводит из файла. Образец данных представлен в файле `lines.dat`.

Указания по программированию. Определение прямой хранят в виде массива из двух элементов, содержащих коэффициенты a и b. Точку пересечения также можно представить массивом из двух элементов, содержащих координаты x и y. В программе должен присутствовать массив прямых — двумерный массив `lines[MAXLINES][2]`, а также массив найденных точек пересечения `points[MAXPOINTS][2]`. Константы `MAXLINES` (максимально допустимое количество прямых) и `MAXPOINTS` (максимально возможное количество точек пересечения) можно задать с помощью директивы `#define`. Значение `MAXPOINTS` зависит от `MAXLINES`;

[1] Заметим, что таким способом невозможно задать вертикальную прямую

его определяют, предполагая, что все прямые пересекаются и все точки пересечения различны, т. е. каждая пара прямых дает одну точку.

Алгоритм решения задачи может быть следующим. Вводим прямые по одной и запоминаем их в массиве `lines`. Прочитав очередную прямую, найдем точки ее пересечения с каждой из ранее введенных прямых, хранящихся в массиве `lines`. Если две прямые не параллельны, у них имеется точка пересечения; вычислим ее координаты. Затем сравним эту точку с ранее найденными точками, хранящимися в массиве `points`. Если такая точка уже была, отбросим ее и перейдем к обработке следующей пары прямых. Если же точка встретилась впервые, добавим ее в массив `points` и увеличим на единицу счетчик найденных точек. По окончании ввода и обработки прямых выведем значение счетчика и содержимое массива `points`.

Удобно ввести следующие функции:

```
int crosspoint(double a[2], double b[2], double cp[2])
int isthesame(double p[2], double q[2])
```

Функция `crosspoint` вычисляет координаты точки пересечения двух прямых `a` и `b` и помещает их в массив `cp`, а в качестве своего значения возвращает 1, если прямые непараллельны (т. е. точка пересечения определена), и 0, если прямые параллельны или совпадают (точка пересечения не существует).

Логическая функция `isthesame` сравнивает координаты двух точек `p` и `q` и возвращает значение 1, если точки совпадают, и 0, если они различны. При сравнении необходимо учитывать, что точность вычислений с плавающей точкой ограничена. Поэтому координаты, которые различаются менее чем на единицу в 14-й значащей цифре, следует считать одинаковыми.

Вариант 3

Даны книжная полка длины L и N книг разной толщины (книги пронумерованы, и для каждой книги известна ее толщина d_i). Возможно, размеры полки не позволят разместить на ней все книги. Требуется поставить на полку максимальное число книг (программа должна сообщить номера помещенных на полку книг и их общее количество).

Исходные данные (L, N, d_i $(i = 1, 2, \ldots, N)$) следует прочитать из файла. Образец данных представлен в файле `books.dat`.

Вариант 4

Геометрическая конфигурация N-атомной молекулы задана совокупностью декартовых координат ядер атомов (в ангстремах); указаны также атомные номера. На основании этих данных программа должна сообщить, между какими атомами существуют химические связи (руководствуясь приведенной ниже таблицей типичных длин связей).

C–H	1.09 Å	C–O	1.43 Å	C–S	1.82 Å	O–H	0.96 Å
C–C	1.54 Å	C=O	1.23 Å	N–H	1.02 Å	O–O	1.48 Å
C⋯C	1.40 Å	C–F	1.33 Å	N–N	1.45 Å	O–F	1.42 Å
C=C	1.34 Å	C–Cl	1.77 Å	N=O	1.20 Å	S–H	1.34 Å
C≡C	1.20 Å	C–Br	1.93 Å	N–F	1.36 Å		
C–N	1.47 Å	C–I	2.14 Å	N–Cl	1.75 Å		

Следует учитывать, что в разных молекулах длины связей могут отличаться от указанных выше, причем отличие в бóльшую сторону не превышает 0.12 Å, а отличие в меньшую сторону (каким бы оно ни было) заведомо указывает на наличие связи.

Исходные данные программа должна прочитать из файла. Образцы данных представлены в файлах `molecule1.dat` (ацетальдегид) и `molecule2.dat` (1-фторпропан).

Вариант 5

Написать программу, которая анализирует химическую формулу вещества и вычисляет молекулярную массу и процентное содержание каждого элемента. Формула состоит из символов химических элементов, за которыми могут следовать целые числа, например:

```
Na2SO4
```

По такой формуле программа должна вывести следующий результат:

```
Molecular mass: 142.037
Na: 32.37%
 S: 22.57%
 O: 45.06%
```

Степень сложности этой задачи зависит от того, какие правила вы примете для записи формул.

а) Простейший вариант: брутто-формула. Каждый из элементов указан единственный раз с суммарным стехиометрическим коэффициентом, как в приведенной выше формуле сульфата натрия или в следующих формулах:

`N2H4O3` — нитрат аммония NH_4NO_3
`C2H4O2` — уксусная кислота CH_3COOH

б) Чуть более сложный вариант, когда символ элемента может входить в формулу несколько раз, например:

`NH4NO3` — нитрат аммония
`CH3COOH` — уксусная кислота

в) Наиболее сложный вариант синтаксиса — формула со скобками:

`Al2(SO4)3` — сульфат алюминия $Al_2(SO_4)_3$
`(CH3)2CO` — ацетон $(CH_3)_2CO$

Указания по программированию. Очевидно, в программе должна присутствовать таблица химических элементов, содержащая символы и атомные массы. Информацию об одном элементе можно хранить в структуре

```
struct element_data {
    char   sym[3];
    double mass;
};
```

а таблицу элементов — в массиве таких структур, например:

```
struct element_data table[120];
```

Содержимое таблицы можно загружать перед началом работы из файла либо задать непосредственно в программе с помощью инициализации массива:

```
static struct element_data table[] = {
   { "H" , 1.0079  },
   { "He", 4.0026  },
   { "Li", 6.941   },
```

```
   { "Be", 9.01218 },
   { "B" , 10.81   },
   { "C" , 12.011  },
   { "N" , 14.0067 },
   . . . . . .
   { "Xe", 131.30  }
};

static int tablesize = sizeof(table) / sizeof(table[0]);
```

Так как размер массива `table` здесь явно не указан, то компилятор определит его самостоятельно по количеству заданных инициализаторов. Фактический размер таблицы хранится в переменной `tablesize`, причем инициализирующее значение определено как частное от деления размера всего массива на размер одного элемента.

Для проверки программы не обязательно задавать полную таблицу химических элементов. Вполне можно ограничиться несколькими первыми периодами.

Вариант 6

Структурная формула [органического] соединения задана в виде матрицы смежности. Программа должна выяснить, содержит ли данная структура циклы, и если да, то указать атомы, входящие в цикл(ы).

Подсказка. Все атомы можно разделить на два типа в зависимости от их положения в структуре: находящиеся внутри цепи (цепьевые) и расположенные на конце цепи (концевые). Концевой атом связан лишь с одним другим атомом, а цепьевые атомы образуют две и более связей. Удалим из структуры все концевые атомы. Некоторые из прежних цепьевых атомов в новой структуре станут концевыми. Будем повторять процедуру удаления концевых атомов до тех пор, пока это возможно. Если в конце концов удастся удалить все атомы молекулы, то исходная структура не содержала циклов. Если же на некотором этапе окажется, что в оставшейся части структуры больше нет концевых атомов, то эта часть состоит из циклов (возможно, циклы связаны между собой линейными цепочками атомов).

Вариант 7

Структурная формула органического соединения записана в сокращенной форме — в виде углеродного скелета без отображения атомов водорода. Эта формула задана в виде модифицированной матрицы смежности, недиагональные элементы которой содержат кратности связей, а на диагонали указаны атомные номера. Написать программу, которая строит полную структурную формулу, добавляя отсутствующие атомы водорода.