Кубок Псковской области по программированию среди школьников - 2016
Разбор задач финального тура

A. Текст

Простая задача. Нужно было аккуратно считать входную строку и для каждого символа определить его тип: буква (находится в диапазоне ‘A’-’Z’ или ‘a’-’z’), цифра (в диапазоне ‘0’-’9’), пробел (их может быть несколько подряд), знак препинания.

Тонкости реализации для языка Pascal:

- для считывания количества символов использовать readln, а не read
- использовать вхождение во множества для проверки введенных символов, например: if (someToken in [‘a’..’z’, ‘A’..’Z’]) then inc(answerForAlpha);
- Нужно помнить, что строки в паскале индексируются с 1, а не с 0, а в нулевом индексе у строк хранится их длина.

B. Шифр ROT13

Особенности реализации аналогичны задаче A: нужно считать строку, найти все латинские буквы (прописные и строчные нужно обрабатывать отдельно).

Для Pascal удобно для реализации использовать функции chr (символ по данному ASCII-коду) и ord (ASCII-код для данного символа).

Чтобы упростить реализацию и не обрабатывать отдельно случаи с выходом за границы алфавита, можно было определить строку с буквами латинского алфавита и дублировать ее. Например, для алфавита “abc” дублированная строка выглядит так: “abcabc”. В этом случае можно всегда смещать индекс нужного символа на 13 и не переживать за выход за пределы алфавита.

C. Матричный принтер

Главная сложность в решении - сохранить представления цифр. Символы можно сохранить в трехмерном массиве [цифра (можно даже как char)][координатаX][координатаY], в двумерном массиве строк [цифра][координатаY строки в представлении цифры], в массиве из пяти строк, в котором по смещению находить подстроки, соответствующие цифре.

Если трехмерные массивы вызывают трудности, то можно было обойтись и двумерным массивом, решая задачу отдельно для каждый строки. Сначала обработать ответ для первой строки результата, затем для второй и так до пятой. При этом не забыть, что для pascal в переменную типа строка можно поместить только 255 символов (выводить символы или сразу без меморизации, или сохранять в массив char).

D. Проверка прогрессии

Ключевой момент в решении проблемы с точностью для вещественных чисел (для геометрической прогрессии) заключается в применении свойства геометрической прогрессии: A[i] * A[i] = A[i - 1] * A[i + 1] , так можно избавиться от деления. И решать задачу в длинных целых без потери точности.

Определение арифметической прогрессии особой сложности не представляет, можно воспользоваться определением, данным в условии задачи.

Также нужно не забыть про случаи с нулями: например, 5 0 0 или 0 0 0.

Практически все решения сравнивали вещественные числа в явном виде, как-то if (a = b), где a, b вещественного типа. В программировании это не всегда приемлемо, сравнивать

вещественные числа стоит с какой-то точностью: if (fabs(a - b) < eps). Однако в этой задаче это бы не помогло из-за проблем с точностью, нужна точность большая, чем позволяет тип double (для pascal могло пройти решение с использованием типа extended).

E. Составные числа

Для поиска простых чисел во входной последовательности (и поиска простых делителей) можно воспользоваться алгоритмом Решета Эратосфена (https://ru.wikipedia.org/wiki/Решето_Эратосфена). Искать нужно все простые числа, меньшие $\sqrt{10^9}$ (<31623).

Отдельно нужно рассматривать случаи больших простых чисел *(числа большие этого значения и не делящиеся ни на одно из найденных простых будут также простыми, при этом в дальнейшем проверять деление на них необязательно)*.

Для сортировки можно воспользоваться алгоритмом, не меняющим относительный порядок равных элементов (stable sort), либо использовать для сортировки компаратор, учитывающий количество делителей и позицию во входных данных.

Можно обойтись без сортировки, разделив числа по массивам по количеству делителей (делителей будет не более 12). Записывать числа в этот массив следует по мере считывания их из потока ввода.

Приведем участок кода для определения количества простых делителей и их степени для какого-то числа number. Массив prime[] заполнен простыми числами до $\sqrt{10^9}$

```
for (int i = 0; i < primeCount && prime[i] <= number; ++i) {
        if (number % prime[i] == 0)  {
                deg = 0;
                while (number % prime[i] == 0) {
                        number /= prime[i];
                        deg++;
                }
                divisorCount++;
                // do something
        }
}

if (number > 1) {               /* number is prime */
        deg = 1;
        divisorCount++;
        // do something
}
```

F. Проверка доступа

Символ “*” равен любому слову. можно было аккуратно разделить все маски доступа на слова (разделить по “/”). а дальше проверять для каждой строки доступа, чтобы совпадала эта последовательность слов, учитывая случай со звездочкой (если в маске встретилась *, то проверять равенство слов со строкой доступа не нужно). количество слов при проверке должно совпадать (а также роль пользователя для маски и проверяемой строки).

G. Черепаха

Вычислить максимальное смещение влево и вверх, т.е. минимальные координаты по оси X и оси Y (назовем их minX и minY). Затем следует второй раз пройтись по массиву, который был подан на вход (для этого следует его сохранить), при этом принять начальную координату пути [0; 0] за [minX, minY] точку пути, т.е. при выводе пути смещать все координаты на эту величину, таким образом исключая факт выхода за границы массива. Определив также величины maxX и maxY, не возникнет проблем с определением величины прямоугольного поля, для которого следует вывести ответ.

Можно заранее сместить все точки пути на [+1000; +1000], таким образом, увеличить массив для обработки и исключить вероятность выхода за пределы этого массива. Определить прямоугольную область для вывода ответа аналогично описанному ранее решению [minX; maxX] , [minY, maxY].

Посещенные точки можно хранить в двумерном массиве 2001x2001, либо в виде отсортированного массива точек (или множества), в котором точки можно искать при выводе.

H. Покраска

Для каждой пустой клеточки (идём слева-направо и сверху-вниз) следовало поставить первую из цифр, которая удовлетворяла условию задачи. Цифры можно было перебирать циклом от 0 до 9, для каждой незаполненной ячейки массива. Чтобы отдельно не обрабатывать выход за границы массива, можно было несколько расширить входной массив отступив сверху, снизу, слева и справа дополнительные строки/столбцы и заполнить их пустыми клетками.

Чтобы код был более красивым и привлекательным, не стоит использовать емкие if конструкции с проверкой [i - 1][j] , [i + 1][j], [i - 1][j - 1] ячеек, а можно определить вспомогательные массивы:

```
dx[] = {  1,  1,  1,  0, -1, -1, -1 ,  0};
dy[] = { -1,  0,  1,  1,  1,  0, -1,  -1};
```

а дальше воспользоваться вашим любимым циклом, например

```
for (int k = 0; k < 8; ++k)
        if ( array[ i + dx[k] ][ j + dy[k] ] == someValue )
                // do something or nothing
```

не забывайте, и не игнорируйте тот факт, что красивый, компактный и читаемый код легче отлаживать.