Максимальное суммарное количество баллов: 100

Задание 1

Составьте программу, которая принимает на вход в первой строке восьмеричную цифру D -- одну из {0, 1, 2, 3, 4, 5, 6, 7}, во второй строке целое положительное число L и в третьей строке целое неотрицательное число N, записанное в шестнадцатеричной системе. В этой записи используются десятичные цифры и заглавные латинские буквы {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. Число L -- это длина шестнадцатеричной записи числа N, и оно не более чем 49152. Программа находит, количество вхождений цифры D в запись числа N, если его перевести в восьмеричную систему. В начале записи числа N могут стоять незначащие нули, которые не следует учитывать при подсчёте количества вхождений D = 0. Незначащим является любой нуль, стоящий левее первой ненулевой цифры, или, если N = 0, то все нули, кроме самого правого.

Входные данные:

В первой строке содержится символ D — восьмеричная цифра (одна из {0, 1, 2, 3, 4, 5, 6, 7}).

Во второй строке содержится целое положительное число L — длина записи числа N в шестнадцатеричной системе, в которой могут быть незначащие нули (0 < L < 49152).

В третьей строке содержится непустая последовательность символов, являющаяся записью в шестнадцатеричной системе числа N — целого, неотрицательного (в этой записи L символов). В записи числа N используются десятичные цифры и заглавные латинские буквы {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

Выходные данные:

В первой и единственной строке выводится неотрицательное целое число, равное искомому количеству вхождений цифры D в восьмеричную запись числа N (от 0 до 65535).

Баллы: 20

Решение:

Решение В решении нужно в цикле последовательно считывать и обрабатывать введённые цифры шестнадцатеричной записи числа. Для перевода в восьмеричную систему из шестнадцатеричной можно использовать тот факт, что 2 4 = 16 и 2 3 = 8. Значит, тройку подряд идущих шестнадцатеричных цифр можно переводить в четвёрку восьмеричных. Заранее неизвестно будет ли шестнадцатеричная запись числа N (после отбрасывания незначащих нулей) иметь длину, кратную трём. Если длина не кратна трём выходит, что выделяется префикс из первых значащих шестнадцатеричных цифр (одной или двух), который переводится отдельно, а остальные цифры сгруппируются по тройкам, и каждая такая тройка будет переведена в четвёрку восьмеричных цифр. Поэтому считаем сколько незначащих нулей попалось во вводе, вычитаем их количество из L и переводим, зная кратность количества значащих цифр трём.

Код возможного решения

```
program DigitQty1011 (input, output);

var

    C : char;

    D , L, TEMP, ANSWER : word;

function Process3Digit() : word;
```

```
var     I : word;
        C : char;
                        TEMP : word;
(* найти количество вхождений D в переведённые три 16-тиричные цифры без незначащих нулей
*)
begin
        TEMP := 0;
        for I := 1 to 3 do begin
                read(C);
                dec(L);
                if ((C <= '9') and (C >= '0')) then TEMP := TEMP * 16 + ord(C) - ord('0')
                else TEMP := TEMP * 16 + ord(C) - ord('A') + 10;
        end; (* for *)
        I := 0;
        if (TEMP mod 8 = D) then inc(I);
                        if (TEMP div 8 mod 8 = D) then inc(I);
        if (TEMP div 64 mod 8 = D) then inc(I);
        if (TEMP div 512 mod 8 = D) then inc(I);
                        Process3Digit := I
end;


begin
        readln(C);
        D := ord(C) - ord('0');
        ANSWER := 0;
        TEMP := 0;
        readln(L);
        while ((L > 0) and (TEMP = 0)) do begin
                read(C);
                dec(L);
                if ((C <= '9') and (C >= '0')) then TEMP := ord(C) - ord('0')
                else TEMP := ord(C) - ord('A') + 10;
        end; (* while *)
```

```
case (L mod 3) of
0: begin
      if (TEMP div 8 mod 8 <> 0) and (TEMP div 8 mod 8 = D) then inc(ANSWER);
      if (TEMP mod 8 = D) then inc(ANSWER)
  end; (* обработан префикс в 1 цифру *)
1: begin
      read(C);
      dec(L);
      if ((C <= '9') and (C >= '0')) then TEMP := TEMP * 16 + ord(C) - ord('0')
      else TEMP := TEMP * 16 + ord(C) - ord('A') + 10;
      if (TEMP div 64 mod 8 <> 0) and (TEMP div 64 mod 8 = D) then inc(ANSWER);
      if (TEMP div 8 mod 8 = D) then inc(ANSWER);
      if (TEMP mod 8 = D) then inc(ANSWER)
  end; (* обработан префикс в 2 цифры *)
else begin
      read(C);
      dec(L);
      if ((C <= '9') and (C >= '0')) then TEMP := TEMP * 16 + ord(C) - ord('0')
      else TEMP := TEMP * 16 + ord(C) - ord('A') + 10;
      read(C);
      dec(L);
      if ((C <= '9') and (C >= '0')) then TEMP := TEMP * 16 + ord(C) - ord('0')
      else TEMP := TEMP * 16 + ord(C) - ord('A') + 10;
      if (TEMP div 512 mod 8 <> 0) and (TEMP div 512 mod 8 = D) then inc(ANSWER);
      if (TEMP div 64 mod 8 = D) then inc(ANSWER);
      if (TEMP div 8 mod 8 = D) then inc(ANSWER);
      if (TEMP mod 8 = D) then inc(ANSWER)
   end; (* обработаны первые 3 цифры *)
end; (* case *)
while (L > 0) do begin
      ANSWER := ANSWER + Process3Digit();
end (* while *);
```

```
    writeln(ANSWER)
end.
```

Задание 2

Робин Бобин Барабек думает, сколько коров и быка он съест на обед.

Он берет шестнадцатеричное число.

За одно действие он может заменить один из знаков на соседний (0 заменить на 1 или F, A заменить на 9 или B и т.п.). Он ставит себе ограничение сверху на количество действий и старается максимизировать количество подслов BEEF.

Ваша программа в первой строке ввода получает последовательность цифр от 0 до 9 и букв от A до F. Всего не более 100000 знаков. Во второй строке написано ограничение на число ходов. Оно не превышает 1000.

Программа должна напечатать максимальное число подслов BEEF, которое можно получить, не превышая максимального числа ходов.

Баллы: 20

Решение:

```
#include <iostream>
#include <vector>

using namespace std;

string e = "BEEF";
int n;

int idx(char c) {
        if (c >= '0' && c <='9') return c - '0';
        return c - 'A' + 10;
}

int cnt(char t, char f) {
        int i_t = idx(t);
```

```
        int i_f = idx(f);
        if (i_t > i_f) {
                swap(i_t, i_f);
        }
        return min(i_f - i_t, i_t + 16 - i_f);
}

int dist(const char *b) {
        int r = 0;
        for (int i = 0; i < 4; ++i) {
           r += cnt(e[i], b[i]);
        }
        return r;
}

int main() {
   string line;
   cin >> line;
   int k;
   cin >> k;
   n = line.size();
   vector<vector<int>> v(n + 1, vector<int>(k + 1, 0));

   for (int i = 4; i <= n; ++i) {
         for (int j = 0; j <= k; ++j) {
        v[i][j] = v[i-1][j];
            int cnt = dist(line.data() + i - 4);
            if (cnt <= j) {
                    v[i][j] = max(v[i][j], v[i - 4][j - cnt] + 1);
        }
         }
   }
```

```
    int ans = 0;

    for (int j = 0; j <= k; ++j) {
            if (v[n][j] > ans) {
                    ans = v[n][j];
            }
    }
    cout << ans;
}
```

Задание 3

Вася очень любит поворачивать монохромные квадратные картинки.
Особенно ему нравятся такие картинки, которые при этом инвертируются.
Ваша программа получает на вход количество пикселей по одному
измерению квадратной картинки. Это число не превосходит 10000.
Программа должна напечатать максимальное количество различных
картинок из белых и черных пикселей указанного размера, для которых
при повороте на 90 градусов получается картинка, совпадающая с
исходной инвертированной (белый пиксель заменен на черный, а черный --
на белый).
Ответ требуется выводить по модулю 1000000007
Баллы: 20
Решение:

```
n = int(input())

s = n - 1
k = 0
while s > 0:
    k += s
    s -= 2
if n % 2 == 1:
```

```
    print(0)
else:
    print(2**k % 1000000007)
```

Задание 4

Андрей изучает социальные сети и пытается определить скрытые атрибуты пользователей по их друзьям. Поскольку Андрей - профессиональный программист, то он хочет протестировать свою программу прежде чем верить ее результатам. Но для этого требуется много разных графов, похожих на социальные сети. Андрей хочет получать графы с разным количеством пользователей (т.е. вершин графа) и разными отношениями дружбы (т.е. ребрами графа). Отношение дружбы ненаправленное.

В графе не должно быть петель и кратных ребер.

Андрей будет задавать желаемое количество вершин и желаемое среднее количество ребер, инцидентных вершине. Его устроит даже граф, если эти его характеристики будут отличаться от заданных, но не более чем на 20%.

Ваша программа получает на вход 2 целых положительных числа -

N - количество вершин и K - среднее количество ребер у вершины (1 <= N <= 200, 0 <= K <= N - 1)

Программа печатает граф описанного вида.

В первой строке печатается количество вершин графа.

Начиная со следующей строки, печатается матрица смежности графа по строкам.

Вершины нумеруются последовательно, начиная с 0. Элемент матрицы смежности равен 1, если соответствующее ребро входит в граф, и 0, иначе.

Элементы разделяются пробельными символами. Элементы главной диагонал матрицы смежности должны равняться 0.

Если графа описанного вида не существует, программа ничего не печатает.

Баллы: 20

Решение:

```
#include "testlib.h"

enum { MAX = 300 };
```

```
int
main(int argc, char *argv[])
{
    setName("check the graph topology");
    registerTestlibCmd(argc, argv);

    int N = inf.readInt();
    int K = inf.readInt();
    static int matrix[MAX][MAX];

    if (ans.readInt() == 0) {
        ouf.readEof();
        quitf(_ok, "ok");
    }

    int Nouf = ouf.readInt();
    quitif(!(N * 0.8 - 1 <= Nouf), _wa, "N = %d, Nouf = %d, > 20%%", N, Nouf);
    quitif(!(Nouf <= N * 1.2 + 1), _wa, "N = %d, Nouf = %d, > 20%%", N, Nouf);

    for (int i = 0; i < Nouf; ++i) {
        for (int j = 0; j < Nouf; ++j) {
            matrix[i][j] = ouf.readInt();
        }
    }

    quitif(!ouf.seekEof(), _wa, "outfile has unnecessary data at the end");

    for (int i = 0; i < Nouf; ++i) {
        quitif(!(matrix[i][i] == 0), _wa,
                "diagonal item [%d][%d] is not zero", i, i);
        for (int j = 0; j < Nouf; ++j) {
```

```
            quitif(matrix[i][j] != 0 && matrix[i][j] != 1, _wa,
                "matrix item [%d][%d] is not 0 or 1 but %d", i, j, matrix[i][j]);
            quitif(matrix[i][j] != matrix[j][i], _wa,
                "matrix is not symmetric, item [%d][%d] (which equals to %d) "
                "is not equal to the item [%d][%d] (which equals to %d)",
                i, j, matrix[i][j], j, i, matrix[j][i]);
        }
    }

    int edg_num = 0;
    for (int i = 0; i < Nouf; ++i) {
        for (int j = 0; j < Nouf; ++j) {
            edg_num += matrix[i][j];
        }
    }
    quitif(!(0.8 * K * Nouf - 1 <= edg_num), _wa, "graph has %d edges, must be more", edg_num);
    quitif(!(edg_num <= 1.2 * K * Nouf + 1), _wa, "graph has %d edges, must be less", edg_num);

    quit(_ok, "ok");
}
```

Задание 5

в игру "Змейка". Змейка ходит по прямоугольному полю
и собирает еду. Змейка занимает незамкнутую цепочку клеток. Две
соседние клетки змейки имеют одну общую сторону. Еда занимает одну
клетку. Змейка ест только своей головой. Съеденная клетка становится
частью змейки. Змейка не может выходить за границы поля. Змейка может
находиться на любой клетке, если
это не запрещено перечисленными выше условиями.

На вход подается прямоугольное поле: количество строк (до 100) и количество
столбцов (до 100), за ними по строкам подаются числа 0, 1, 2, 3. 0 означает, что
в текущей клетке нет еды и нет змейки. 1 означает, что в текущей
клетке находится еда и нет змейки. 2 или 3 означает, что в текущей

клетке находится змейка. 3 означает, что в клетке находится голова
змейки.
Программа должна составить действия змейки, чтобы та
съела всю еду, или определить, что съесть всю еду нельзя.
Условимся, что клетка поля с минимальными координатами
находится сверху слева, а клетка поля с максимальными координатами
находится снизу справа. Тогда действие 0 означает движение головы
на соседнюю клетку сверху, действие 1 означает движение головы
на соседнюю клетку слева, действие 2 означает движение головы
на соседнюю клетку внизу, действие 3 означает движение головы
на соседнюю клетку справа. Голова может переместиться только на
пустую клетку или клетку с едой.
Гарантируется что по исходным данным положение змейки определяется однозначно</p>
Программа печатает последовательность чисел 0, 1, 2, 3, если
можно съесть всю еду, и -1 иначе.
Баллы:
20
Решение:

```
#include "testlib.h"

#include <vector>
#include <utility>
#include <algorithm>

enum { MAX = 100 };

int N, M, field[MAX][MAX];
std::vector<std::pair<int, int>> snake;

bool
in_field(int x, int y)
{
```

```
    return 0 <= x && x < N && 0 <= y && y < M;
}

int
meals_count()
{
    int c = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (field[i][j] == 1) {
                ++c;
            }
        }
    }
    return c;
}

enum Action { UP = 0, DOWN = 2, LEFT = 1, RIGHT = 3 };

void
read_field()
{
    N = inf.readInt();
    M = inf.readInt();

    snake.clear();
    size_t count_of_2 = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            field[i][j] = inf.readInt();
            if (field[i][j] == 3) {
                snake.emplace_back(i, j);
```

```
            field[i][j] = 0;
        } else if (field[i][j] == 2) {
            ++ count_of_2;
            field[i][j] = 22;
        }
    }
  }

  int last_x = snake[0].first;
  int last_y = snake[0].second;

  while (count_of_2-- > 0) {
    int next_x, next_y;

    for (int action = 0; action <= 3; ++action) {
      next_x = (action == DOWN ? last_x + 1 : action == UP ? last_x - 1 : last_x);
      next_y = (action == LEFT ? last_y - 1 : action == RIGHT ? last_y + 1 : last_y);
      if (in_field(next_x, next_y) && field[next_x][next_y] == 22) {
        snake.emplace_back(next_x, next_y);
        break;
      }
    }
    field[next_x][next_y] = 0;
  }
}

void
move_snake(Action action)
{
  int last_x = snake[0].first;
  int last_y = snake[0].second;
  int next_x = (action == DOWN ? last_x + 1 : action == UP ? last_x - 1 : last_x);
```

```
    int next_y = (action == LEFT ? last_y - 1 : action == RIGHT ? last_y + 1 : last_y);
    auto next = std::pair<int, int>(next_x, next_y);

    if (! in_field(next_x, next_y)) {
        quitf(_wa, "snake goes outside of the field to (%d, %d)", next_x, next_y);
    }

    if (std::count(snake.begin(), snake.end(), next) > 0) {
        quitf(_wa, "snake tryes to eat himself to (%d, %d)", next_x, next_y);
    }

    if (field[next_x][next_y] == 1) {
        field[next_x][next_y] = 0;
    } else {
        snake.pop_back();
    }
    snake.emplace(snake.begin(), next_x, next_y);
}

int
main(int argc, char *argv[])
{
    setName("check the snake route");
    registerTestlibCmd(argc, argv);

    read_field();

    if (ans.readInt() == -1) {
        if (ouf.readInt() != -1) {
            quitf(_wa, "expected -1, but gotten another result");
        } else {
            quitf(_ok, "answer is right");
```

```
    }
  }

  while (!ouf.seekEof()) {
    int action = ouf.readInt();
    if (action < 0 || action > 3) {
      quitf(_pe, "unexpected action %d", action);
    }

    move_snake(static_cast<Action>(action));
  }

  int c;
  if ((c = meals_count()) == 0) {
    quitf(_ok, "ok");
  } else{
    quitf(_wa, "not all cells with meal were eaten, %d stay", c);
  }
}
```

Задание 6

tps://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C обратной польской записи. В выражении должны поддерживаться числа, знаки бинарных операций ‘+’, ‘-’, ‘*’, ‘/’. Польская запись записывается в одну строку, причем элементы польской записи разделяются произвольным количеством пробелов. Пробелы обязательны, когда необходимо отделить два числа друг от друга и допускаются для разделения знаков операций друг от друга или знаков операций и чисел, а также в начале и конце строки. Гарантируется, что польская запись является корректной записью некоторого выражения. Пример польской записи:

2 3 +*

Числа-операнды записываются в пятеричной симметричной системе счисления (см. например, описание "https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F#%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%82%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D

1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F троичной симметричной системы счисления). В качестве отрицательных цифр пятеричной симметричной системы счисления используются цифры B, обозначающая -2, и A, обозначающая -1. Таким образом, число B2 в пятеричной симметричной системе счисления - это число -8 в десятичной системе счисления.

Калькулятор должен обрабатывать 27-разрядные https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D1%81_%D1%84%D0%B8%D0%BA%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B9_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%BE%D0%B9 числа с фиксированной точкой, в которых младшие 11 разрядов отводится под дробную часть, а старшие 16 разрядов - под целую часть. Например, число 120.0102 в виде 27-разрядного числа с фиксированной точкой запишется как 000000000000012001020000000.

На вход подаются числа, у которых целая часть отделяется от дробной символом “точка”, причем гарантируется, что в целой части числа находится не более чем 16 значащих разрядов, а в дробной части - не более чем 11 значащих разрядов. У числа с нулевой дробной частью дробная часть может отсутствовать.

Ваша программа должна вычислить выражение и вывести результат в виде 27-разрядного числа в пятеричной симметричной записи. Если при вычислении возникла ошибка, например, результат вычисления операции после округления не может быть представлен с требуемой разрядностью, должна быть выведена строка ERROR.

При вычислении результата каждой операции в процесс

Баллы: 20

Решение:

```
/* -*- mode: c++; c-basic-offset: 4 -*- */


#include <iostream>

#include <string>

#include <cstdint>

#include <stdexcept>

#include <algorithm>

#include <random>

#include <utility>

#include <iomanip>

#include <cstring>

#include <random>

#include <fstream>


constexpr size_t SCALE = 11;
```

```
constexpr size_t WIDTH = 27;
constexpr int BASE = 5;
constexpr int64_t MAX_VALUE = 3725290298461914062LL;
constexpr int64_t MIN_VALUE = -3725290298461914062LL;

class Mantissa64
{
   int64_t v_{};

public:
   Mantissa64() noexcept = default;
   explicit Mantissa64(int64_t v) noexcept : v_{v} {}
   explicit Mantissa64(const std::string &str);
   int64_t value() const { return v_; }
   std::string to_string() const;
   std::string to_string(uint32_t width) const;

   friend Mantissa64 operator << (Mantissa64 a, uint32_t b);
};

class Mantissa128
{
   __int128 v_{};

public:
   Mantissa128() noexcept = default;
   explicit Mantissa128(__int128 v) noexcept : v_{v} {}
   std::string to_string() const;

   friend Mantissa128 operator << (Mantissa128 a, uint32_t b);
   friend Mantissa128 operator * (Mantissa128 a, Mantissa128 b);
   friend Mantissa128 operator / (Mantissa128 a, int64_t b);
```

```
};

class Fixed64
{
   int64_t v_{};
   uint32_t s_{};

   static int64_t rescale(int64_t value, uint32_t old_scale, uint32_t new_scale);
   static std::string reducetail(const std::string &str, size_t size);
public:
   Fixed64() noexcept = default;
   explicit constexpr Fixed64(int64_t v, uint32_t s = 0) noexcept : v_{v}, s_{s} {}
   explicit Fixed64(Mantissa64 v, uint32_t s = 0) : v_{v.value()}, s_{s}
   {
      if (v_ < MIN_VALUE || v_ > MAX_VALUE) throw std::out_of_range("range error");
   }
   explicit Fixed64(const std::string &str);
   Fixed64(double v, uint32_t s);
   Fixed64(const std::string &str, uint32_t scale);
   Fixed64(const Fixed64 &other) = default;
   Fixed64(const Fixed64 &other, uint32_t new_scale);
   std::string to_string() const;
   double to_double() const;
   Mantissa128 m128() const noexcept { return Mantissa128(__int128(v_)); }
   Mantissa64 m64() const noexcept { return Mantissa64(v_); }
   int64_t value() const noexcept { return v_; }

   static std::string trim(const std::string &str);
   friend Fixed64 operator +(Fixed64 a, Fixed64 b);
   friend Fixed64 operator -(Fixed64 a, Fixed64 b);
   friend Fixed64 operator *(Fixed64 a, Fixed64 b);
   friend Fixed64 operator /(Fixed64 a, Fixed64 b);
```

```
    friend std::ostream & operator << (std::ostream &f, Fixed64 a);
    explicit operator bool() const noexcept { return v_ != 0; }
};

constexpr Fixed64 ONE{int64_t(48828125), SCALE};

Mantissa64::Mantissa64(const std::string &str)
{
    int64_t res = 0;
    for (char c : str) {
        if (__builtin_mul_overflow(res, int64_t(BASE), &res)) {
            throw std::out_of_range("*5 overflow");
        }
        if (c == 'A') {
            if (__builtin_add_overflow(res, int64_t(-1), &res)) {
                throw std::out_of_range("-1 overflow");
            }
        } else if (c == 'B') {
            if (__builtin_add_overflow(res, int64_t(-2), &res)) {
                throw std::out_of_range("-1 overflow");
            }
        } else if (c >= '0' && c <= '2') {
            if (__builtin_add_overflow(res, int64_t(c - '0'), &res)) {
                throw std::out_of_range("+ overflow");
            }
        } else {
            throw std::invalid_argument("invalid symbol");
        }
    }
    v_ = res;
}
```

```
std::string
Mantissa64::to_string() const
{
    int64_t value = v_;
    std::string buf;

    if (!value) {
        return std::string("0");
    } else if (value < 0) {
        while (value) {
            auto rem = -(value % BASE);
            if (!rem) {
                buf += '0';
                value /= BASE;
            } else {
                rem = BASE - rem;
                if (rem < 3) {
                    buf += char('0' + rem);
                    value = value / BASE - 1;
                } else if (rem == 3) {
                    buf += 'B';
                    value = value / BASE;
                } else if (rem == 4) {
                    buf += 'A';
                    value = value / BASE;
                }
            }
        }
    } else {
        while (value) {
            auto rem = value % BASE;
            if (rem < 3) {
```

```
            buf += char('0' + rem);
            value /= BASE;
        } else if (rem == 3) {
            buf += 'B';
            value = value / BASE + 1;
        } else if (rem == 4) {
            buf += 'A';
            value = value / BASE + 1;
        }
    }
  }
  std::reverse(buf.begin(), buf.end());
  return buf;
}

Fixed64::Fixed64(const std::string &str)
{
  int64_t res = 0;
  uint32_t scale = 0;
  bool dot = false;
  for (char c : str) {
    if (c == '.') {
      if (dot) throw std::invalid_argument(". again");
      dot = true;
    } else {
      if (dot) ++scale;
      if (__builtin_mul_overflow(res, int64_t(BASE), &res)) {
        throw std::out_of_range("*5 overflow");
      }
      if (c == 'A') {
        if (__builtin_add_overflow(res, int64_t(-1), &res)) {
          throw std::out_of_range("-1 overflow");
```

```
            }
        } else if (c == 'B') {
            if (__builtin_add_overflow(res, int64_t(-2), &res)) {
                throw std::out_of_range("-1 overflow");
            }
        } else if (c >= '0' && c <= '2') {
            if (__builtin_add_overflow(res, int64_t(c - '0'), &res)) {
                throw std::out_of_range("+ overflow");
            }
        } else {
            throw std::invalid_argument("invalid symbol");
        }
      }
    }
    v_ = res;
    if (v_ < MIN_VALUE || v_ > MAX_VALUE) throw std::out_of_range("range error");
    s_ = scale;
}

Fixed64::Fixed64(const std::string &str, uint32_t scale)
    : Fixed64(str)
{
    v_ = rescale(v_, s_, scale);
    if (v_ < MIN_VALUE || v_ > MAX_VALUE) throw std::out_of_range("range error");
    s_ = scale;
}

Fixed64::Fixed64(const Fixed64 &other, uint32_t scale)
    : Fixed64(other)
{
    v_ = rescale(v_, s_, scale);
    if (v_ < MIN_VALUE || v_ > MAX_VALUE) throw std::out_of_range("range error");
```

```
    s_ = scale;
}

// remove leading 0 in the integral part and traling 0 in the fractional part
std::string
Fixed64::trim(const std::string &str)
{
    if (!str.size()) throw std::invalid_argument("empty string");
    if (str[0] == '.') throw std::invalid_argument("starting with .");
    size_t i = 0;
    while (i < str.size() && str[i] == '0') { ++i; }
    if (i == str.size()) return std::string("0");
    std::string res;
    while (i < str.size() && str[i] != '.') {
        if (str[i] == '0' || str[i] == '1' || str[i] == '2' || str[i] == 'A' || str[i] == 'B') {
            res += str[i];
            ++i;
        } else {
            throw std::invalid_argument("invalid symbol");
        }
    }
    if (i == str.size()) return res;
    res += '.';
    ++i;
    if (i == str.size()) throw std::invalid_argument("ending with .");
    while (i < str.size()) {
        if (str[i] == '0' || str[i] == '1' || str[i] == '2' || str[i] == 'A' || str[i] == 'B') {
            res += str[i];
            ++i;
        } else {
            throw std::invalid_argument("invalid symbol");
        }
```

```
    }
    i = res.size();
    while (i > 0 && res[i - 1] == '0') {
        res.resize(i - 1); --i;
    }
    if (res[i - 1] == '.') res.resize(i - 1);
    return res;
}

std::string
Mantissa128::to_string() const
{
    __int128 value = v_;
    std::string buf;

    if (!value) {
        return std::string("0");
    } else if (value < 0) {
        while (value) {
            auto rem = -(value % BASE);
            if (!rem) {
                buf += '0';
                value /= BASE;
            } else {
                rem = BASE - rem;
                if (rem < 3) {
                    buf += char('0' + rem);
                    value = value / BASE - 1;
                } else if (rem == 3) {
                    buf += 'B';
                    value = value / BASE;
                } else if (rem == 4) {
```

```
                buf += 'A';
                value = value / BASE;
            }
        }
    }
  } else {
    while (value) {
        auto rem = value % BASE;
        if (rem < 3) {
            buf += char('0' + rem);
            value /= BASE;
        } else if (rem == 3) {
            buf += 'B';
            value = value / BASE + 1;
        } else if (rem == 4) {
            buf += 'A';
            value = value / BASE + 1;
        }
    }
  }
  std::reverse(buf.begin(), buf.end());
  return buf;
}

std::string
Fixed64::to_string() const
{
  int64_t value = v_;
  uint32_t scale = s_;
  std::string buf;

  if (!value) {
```

```
    return std::string("0");
  } else if (value < 0) {
    while (value) {
      auto rem = -(value % BASE);
      if (!rem) {
        buf += '0';
        value /= BASE;
      } else {
        rem = BASE - rem;
        if (rem < 3) {
          buf += char('0' + rem);
          value = value / BASE - 1;
        } else if (rem == 3) {
          buf += 'B';
          value = value / BASE;
        } else if (rem == 4) {
          buf += 'A';
          value = value / BASE;
        }
      }
    }
  } else {
    while (value) {
      auto rem = value % BASE;
      if (rem < 3) {
        buf += char('0' + rem);
        value /= BASE;
      } else if (rem == 3) {
        buf += 'B';
        value = value / BASE + 1;
      } else if (rem == 4) {
        buf += 'A';
```

```
            value = value / BASE + 1;
          }
        }
    }
    std::reverse(buf.begin(), buf.end());
    if (buf.size() == scale) {
        buf.insert(0, "0.");
    } else if (buf.size() < scale) {
        buf.insert(0, scale - buf.size(), '0');
        buf.insert(0, "0.");
    } else {
        buf.insert(buf.size() - scale, ".");
    }
    while (buf.size() > 0 && buf[buf.size() - 1] == '0') {
        buf.resize(buf.size() - 1);
    }
    if (buf.size() > 0 && buf[buf.size() - 1] == '.') {
        buf.resize(buf.size() - 1);
    }
    return buf;
}

std::string
Fixed64::reducetail(const std::string &str, size_t size)
{
    if (str.size() <= size) return "0";
    return str.substr(0, str.size() - size);
}

double
Fixed64::to_double() const
{
```

```
    int64_t value = v_;
    uint32_t scale = s_;
    double res = value;
    while (scale > 0) {
        res /= double(BASE);
        --scale;
    }
    return res;
}

std::string
Mantissa64::to_string(uint32_t width) const
{
    std::string res = to_string();
    if (res.size() >= width) return res;
    return std::string(width - res.size(), '0') + res;
}

int64_t
Fixed64::rescale(int64_t value, uint32_t old_scale, uint32_t new_scale)
{
    if (old_scale > new_scale) {
        // divide
        while (new_scale < old_scale) {
            if (value % BASE != 0) {
                throw std::out_of_range("rescale rounding not allowed");
            }
            value /= BASE;
            ++new_scale;
        }
    } else if (old_scale < new_scale) {
        // multiply
```

```
      while (old_scale < new_scale) {
        if (__builtin_mul_overflow(value, BASE, &value)) {
          throw std::out_of_range("*5 overflow");
        }
        ++old_scale;
      }
    }
    return value;
}

Fixed64::Fixed64(double val, uint32_t scale)
{
    for (size_t i = 0; i < scale; ++i) {
      val *= BASE;
    }
    if (val < -9223372036854775808.0 || val >= 9223372036854774784.0) {
      throw std::overflow_error("dtof64");
    }
    v_ = int64_t(val + 0.5);
    if (v_ < MIN_VALUE || v_ > MAX_VALUE) throw std::out_of_range("range error");
    s_ = scale;
}

Mantissa128 operator *(Mantissa128 a, Mantissa128 b)
{
    __int128 res;
    if (__builtin_mul_overflow(a.v_, b.v_, &res)) throw std::overflow_error("Mantissa128::operator *");
    return Mantissa128(res);
}

Fixed64 operator *(Fixed64 a, Fixed64 b)
{
```

```
    if (a.s_ != SCALE) throw std::invalid_argument("invalid a scale");
    if (b.s_ != SCALE) throw std::invalid_argument("invalid b scale");
    auto s = Fixed64::reducetail((a.m128() * b.m128()).to_string(), SCALE);
    /*
    std::cout << ":" << s << std::endl;
    auto m = Mantissa64(s);
    std::cout << "::" << m.to_string() << std::endl;
    auto f = Fixed64(m, SCALE);
    std::cout << ":::" << f.to_string() << std::endl;
    */
    return Fixed64(Mantissa64(Fixed64::reducetail((a.m128() * b.m128()).to_string(), SCALE)), SCALE);
}

Mantissa64 operator << (Mantissa64 a, uint32_t b)
{
    int64_t aa = a.v_;
    for (size_t i = 0; i < b; ++i) {
        if (__builtin_mul_overflow(aa, int64_t(BASE), &aa))
            throw std::overflow_error("Mantissa64::operator <<");
    }
    if (aa < MIN_VALUE || aa > MAX_VALUE) throw std::out_of_range("range error");
    return Mantissa64(aa);
}

Mantissa128 operator << (Mantissa128 a, uint32_t b)
{
    __int128 aa = a.v_;
    for (size_t i = 0; i < b; ++i) {
        if (__builtin_mul_overflow(aa, __int128(BASE), &aa))
            throw std::overflow_error("Mantissa128::operator <<");
    }
    return Mantissa128(aa);
```

```
}

Mantissa128 operator / (Mantissa128 a, int64_t b)
{
   return Mantissa128(a.v_ / b);
}

Fixed64 operator / (Fixed64 a, Fixed64 b)
{
   if (a.s_ != SCALE) throw std::invalid_argument("invalid a scale");
   if (b.s_ != SCALE) throw std::invalid_argument("invalid b scale");
   return Fixed64(Mantissa64(Fixed64::reducetail(((Mantissa128(a.v_) << (SCALE + SCALE)) /
b.v_).to_string(), SCALE)), SCALE);
}

Fixed64 operator + (Fixed64 a, Fixed64 b)
{
   if (a.s_ != SCALE) throw std::invalid_argument("invalid a scale");
   if (b.s_ != SCALE) throw std::invalid_argument("invalid b scale");
   int64_t res;
   if (!__builtin_add_overflow(a.v_, b.v_, &res)) {
      if (res < MIN_VALUE || res > MAX_VALUE) throw std::overflow_error("add");
      return Fixed64(res, a.s_);
   }
   throw std::overflow_error("add");
}

Fixed64 operator - (Fixed64 a, Fixed64 b)
{
   if (a.s_ != SCALE) throw std::invalid_argument("invalid a scale");
   if (b.s_ != SCALE) throw std::invalid_argument("invalid b scale");
   int64_t res;
   if (!__builtin_sub_overflow(a.v_, b.v_, &res)) {
```

```
        if (res < MIN_VALUE || res > MAX_VALUE) throw std::overflow_error("add");
        return Fixed64(res, a.s_);
    }
    throw std::overflow_error("add");
}

std::ostream & operator << (std::ostream &f, Fixed64 a)
{
    f << a.to_string();
    return f;
}

std::mt19937_64 rnd(3838383);

Mantissa64 randmant1()
{
    switch (rnd() % 6) {
    case 0:
        return Mantissa64(int64_t(rnd() % 19) - 9);
    case 1:
        return Mantissa64(int64_t(rnd() % 199) - 99);
    case 2:
        return Mantissa64(int64_t(rnd() % 1999) - 999);
    case 3:
        return Mantissa64(int64_t(rnd() % 19999) - 9999);
    case 4:
        return Mantissa64(int64_t(rnd() % 199999) - 99999);
    case 5:
        return Mantissa64(int64_t(rnd() % 1999999) - 999999);
    default:
        abort();
    }
```

```
}

// fractional part
Fixed64 randmant2()
{
   uint32_t len = rnd() % SCALE + 1;
   std::string buf(SCALE, '0');
   for (uint32_t i = 0; i < len; ++i) {
      switch (rnd() % 5) {
      case 0: buf[i] = '0'; break;
      case 1: buf[i] = '1'; break;
      case 2: buf[i] = '2'; break;
      case 3: buf[i] = 'B'; break;
      case 4: buf[i] = 'A'; break;
      }
   }
   return Fixed64(Mantissa64(buf), SCALE);
}

Fixed64 randomint()
{
   return Fixed64(Mantissa64(randmant1()) << SCALE, SCALE);
}

Fixed64 randomfrac()
{
   auto a = randomint();
   auto b = randmant2();
   auto c = a + b;
   //std::cout << ":" << a << "," << b << "," << c << std::endl;
   return c;
}
```

```
void gen3()
{
   std::ofstream os("003.dat");
   os << randomint();
   for (int i = 0; i < 100; ++i) {
      os << " " << randomint() << " +";
   }
   os << std::endl;
}
void gen4()
{
   std::ofstream os("004.dat");
   os << randomint();
   for (int i = 0; i < 10000; ++i) {
      os << " " << randomint();
   }
   for (int i = 0; i < 10000; ++i) {
      os << " +";
   }
   os << std::endl;
}
void gen5()
{
   // 1000 random multiply samples
   std::ofstream os("005.dat");
   for (int i = 0; i < 1000;) {
      try {
         auto a = randomint();
         auto b = randomint();
         auto c = a * b - ONE;
         os << " " << a << " " << b << " * " << c << " - ";
```

```
            if (i > 0) os << " + ";
            ++i;
        } catch (...) {
        }
    }
    os << std::endl;
}
void gen6()
{
    // 1000 random div samples
    std::ofstream os("006.dat");
    for (int i = 0; i < 1000;) {
        try {
            auto a = randomint();
            auto b = randomint();
            if (!b) continue;
            auto c = a * b;
            os << " " << c << " " << b << " / " << (a - ONE) << " - ";
            if (i > 0) os << " + ";
            ++i;
        } catch (...) {
        }
    }
    os << std::endl;
}
void gen7()
{
    std::ofstream os("007.dat");
    os << randomfrac();
    for (int i = 0; i < 1000; ++i) {
        os << " " << randomfrac();
    }
```

```
    for (int i = 0; i < 1000; ++i) {
        os << " +";
    }
    os << std::endl;
}
void gen8()
{
    std::ofstream os("008.dat");
    os << randomfrac();
    for (int i = 0; i < 6000; ++i) {
        os << " " << randomfrac();
    }
    for (int i = 0; i < 6000; ++i) {
        os << " -";
    }
    os << std::endl;
}
void gen9()
{
    // 1000 random multiply samples
    std::ofstream os("009.dat");
    for (int i = 0; i < 1000;) {
        try {
            auto a = randomfrac();
            auto b = randomfrac();
            auto c = a * b - ONE;
            os << " " << a << " " << b << " * " << c << " - ";
            if (i > 0) os << " + ";
            ++i;
        } catch (...) {
        }
    }
```

```
    os << std::endl;
}
void gen10()
{
    // 1000 random multiply samples
    std::ofstream os("010.dat");
    for (int i = 0; i < 1000;) {
        try {
            auto a = randomfrac();
            auto b = randomfrac();
            auto c = a / b - ONE;
            os << " " << a << " " << b << " / " << c << " - ";
            if (i > 0) os << " + ";
            ++i;
        } catch (...) {
        }
    }
    os << std::endl;
}

void gentests()
{
    gen3();
    gen4();
    gen5();
    gen6();
    gen7();
    gen8();
    gen9();
    gen10();
}
```

```
int main(int argc, char *argv[])
{
   /*
     64:   divisor 48828125
     max:  76293945312.49999998976     3725290298461914062
     min: -76293945312.49999998976    -3725290298461914062

     32:   divisor 3125
     max:  195312.49984   610351562
     min: -195312.49984  -610351562
    */
   if (argc == 2 && !strcmp(argv[1], "gen")) {
      gentests();
      return 0;
   }

   try {
      std::vector<Fixed64> fstk;
      std::string w;
      while (std::cin >> w) {
         if (w == "+") {
            if (fstk.size() < 2) throw std::runtime_error("stack underflow");
            auto f2 = fstk.back(); fstk.pop_back();
            auto f1 = fstk.back(); fstk.pop_back();
            fstk.push_back(f1 + f2);
         } else if (w == "-") {
            if (fstk.size() < 2) throw std::runtime_error("stack underflow");
            auto f2 = fstk.back(); fstk.pop_back();
            auto f1 = fstk.back(); fstk.pop_back();
            fstk.push_back(f1 - f2);
         } else if (w == "*") {
            if (fstk.size() < 2) throw std::runtime_error("stack underflow");
```

```
            auto f2 = fstk.back(); fstk.pop_back();
            auto f1 = fstk.back(); fstk.pop_back();
            fstk.push_back(f1 * f2);
        } else if (w == "/") {
            if (fstk.size() < 2) throw std::runtime_error("stack underflow");
            auto f2 = fstk.back(); fstk.pop_back();
            auto f1 = fstk.back(); fstk.pop_back();
            fstk.push_back(f1 / f2);
        } else {
            auto fval = Fixed64(w, SCALE);
            fstk.push_back(fval);
        }
    }
    if (fstk.size() > 1) throw std::runtime_error("stack contains many elements");
    auto f1 = fstk.back(); fstk.pop_back();
    std::cout << f1.m64().to_string(WIDTH) << std::endl;
} catch (const std::out_of_range &x) {
    std::cout << "ERROR" << std::endl;
} catch (const std::overflow_error &x) {
    std::cout << "ERROR" << std::endl;
}

/*
std::vector<double> dstk;
std::vector<Fixed64> fstk;

std::string w;
while (std::cin >> w) {
    if (w == "+") {
        std::cout << " " << w;
        auto d2 = dstk.back(); dstk.pop_back();
        auto d1 = dstk.back(); dstk.pop_back();
```

```
    auto f2 = fstk.back(); fstk.pop_back();
    auto f1 = fstk.back(); fstk.pop_back();
    dstk.push_back(d1 + d2);
    fstk.push_back(f1 + f2);
  } else if (w == "-") {
    std::cout << " " << w;
    auto d2 = dstk.back(); dstk.pop_back();
    auto d1 = dstk.back(); dstk.pop_back();
    auto f2 = fstk.back(); fstk.pop_back();
    auto f1 = fstk.back(); fstk.pop_back();
    dstk.push_back(d1 - d2);
    fstk.push_back(f1 - f2);
  } else if (w == "*") {
    std::cout << " " << w;
    auto d2 = dstk.back(); dstk.pop_back();
    auto d1 = dstk.back(); dstk.pop_back();
    auto f2 = fstk.back(); fstk.pop_back();
    auto f1 = fstk.back(); fstk.pop_back();
    dstk.push_back(d1 * d2);
    fstk.push_back(f1 * f2);
  } else if (w == "/") {
    auto d2 = dstk.back(); dstk.pop_back();
    auto d1 = dstk.back(); dstk.pop_back();
    auto f2 = fstk.back(); fstk.pop_back();
    auto f1 = fstk.back(); fstk.pop_back();
    dstk.push_back(d1 / d2);
    fstk.push_back(f1 / f2);
    std::cout << " " << w;
  } else {
    double val = std::stod(w);
    dstk.push_back(val);
    auto fval = Fixed64(val, SCALE);
```

```
        fstk.push_back(fval);
        std::cout << " " << fval.to_string() << " [" << fval.value() << "]";
    }
}
std::cout << std::endl;
auto d1 = dstk.back(); dstk.pop_back();
auto f1 = fstk.back(); fstk.pop_back();
std::cout << "double result: " << d1 << std::endl;
std::cout << "fixed result: " << f1.to_string() << " [" << f1.value() << "]" << std::endl;
double err = fabs(f1.to_double() - d1);
std::cout << "error: " << err << std::endl;;
*/
/*
double a, b;
while (std::cin >> a >> b) {
    int64_t i64a = dtof64(a), i64b = dtof64(b);
    double a2 = f64tod(i64a, SCALE), b2 = f64tod(i64b, SCALE);
    std::string a3 = i64tos5s(i64a), b3 = i64tos5s(i64b);
    std::cout << "value1: " << a << "," << i64a << "," << a3 << "," << a2 << std::endl;
    std::cout << "value2: " << b << "," << i64b << "," << b3 << "," << b2 << std::endl;
    __int128 c = (__int128) i64a * (__int128) i64b;
    std::string abs1 = i128tos5s(c);
    std::cout << "a * b: " << abs1 << std::endl;
    std::string abs2 = reducetail(abs1, SCALE);
    std::cout << "a * b: " << abs2 << std::endl;
    int64_t ab3 = s5stoi64(abs2);
    std::cout << "a * b: " << ab3 << std::endl;
    std::cout << "a * b result: " << f64tod(ab3, SCALE) << std::endl;
    double ab4 = a * b;
    std::cout << "a * b check: " << ab4 << std::endl;
    std::cout << "a * b error: " << fabs(f64tod(ab3, SCALE) - ab4) << std::endl;
}
```

```
*/
/*
std::string s;
while (std::cin >> s) {
  try {
    auto [ value, scale ] = s5stof64(s5trim(s));
    std::cout << value << "," << scale << "," << f64tod(value, scale) << "," << i64tos5sw(rescale(value, scale, SCALE), WIDTH) << std::endl;
  } catch (const std::exception &ex) {
    std::cout << ex.what() << std::endl;
  }
}
*/
/*
std::string s;
while (std::cin >> s) {
  try {
    std::cout << trim(s) << std::endl;
  } catch (const std::exception &ex) {
    std::cout << ex.what() << std::endl;
  }
}
*/
/*
for (int64_t val = -1000000; val < 1000000; ++val) {
  std::string vs = i64tos5s(val);
  int64_t val2 = s5stoi64(vs);
  if (val2 != val) abort();
}
*/
/*
std::random_device hwrnd;
std::mt19937_64 rnd(((uint64_t) hwrnd() << 32) | (uint64_t) hwrnd());
```

```
    for (int i = 0; i < 1000000; ++i) {
        int64_t val = (int64_t) rnd();
        std::string vs = i64tos5s(val);
        int64_t val2 = s5stoi64(vs);
        if (val2 != val) abort();
    }
    */
    /*
    int64_t value;
    while (std::cin >> value) {
        std::cout << i64tos5s(value) << std::endl;
    }
    */
}
```