سری توانی

در سری توانی نیازمند سه پارامتر n, x, a هستید.

$$x = \sum_{n=1}^{\infty} a_n x^n$$

ابتدا هفت عدد به عنوان ضریب (a) از کاربر بگیرید و تابعی بنویسید که به ازای x = 5 جملات سری توانی را محاسبه کند ومقدار ان را برگرداند.سپس مجموع جملات را ب ازای مقادیر داده شده چاپ کنید

ورودی

هفت عدد double

خروجی

مجموع جملات به ازای هفت عدد داده شده

مثال

در اینجا چند نمونه برای فهم بهتر صورت سوال و قالب ورودی و خروجی تست‌ها داده می‌شود.

ورودی نمونه

```
1 2 3 4 5 6 7
```

$$1 * 5^1 + 2 * 5^2 + 3 * 5^3 + 4 * 5^4 + 5 * 5^5 + 6 * 5^6 + 7 * 5^7 = 659180$$

خروجی نمونه

```
659180
```

مربع

از کاربر عددی مانند a گرفته و مربعی با علامت "*"با ضلع به طول a و دو قطر رسم کنید. توجه: فراموش نکنید که هر دو قطر مربع باید رسم شود .

ورودی

عدد صحیح a

خروجی

مربعی به طول و عرض a با دو قطر

مثال

در اینجا چند نمونه برای فهم بهتر صورت سوال و قالب ورودی و خروجی تست‌ها داده می‌شود.

ورودی نمونه ۱

```
12
```

خروجی نمونه ۱

```
* * * * * * * * * * * *
* *                 * *
*   *             *   *
*     *         *     *
*       *     *       *
*         * *         *
*         * *         *
*       *     *       *
*     *         *     *
*   *             *   *
* *                 * *
* * * * * * * * * * * *
```

ورودی نمونه ۲

```
2
```

خروجی نمونه ۲

```
* *
* *
```

کلاس جفت

در دستگاه مختصات دکارتی دوبعدی از جفت‌مرتب برای نمایش یک نقطه استفاده می‌کنیم:

$$(x, y)$$

حال قصد داریم کلاسی طراحی کنیم که بتواند این ساختمان داده را مدل‌سازی کند.

کلاس شما باید دارای دو field باشد که مقادیر x و y را ذخیره‌سازی کنند. به عنوان مثال کلاس زیر را در نظر بگیرید:

```
public class Person {
    String name;
    String lastName;
}
```

کلاس فوق اطلاعات شخصی یک فرد را مدل می‌کند. برای اینکه بتوانیم به field های این کلاس مقدار بدهیم از توابع setter استفاده می‌کنیم همچنین برای دریافت مقدار field ها از توابع getter استفاده می‌کنیم. مثال زیر برای name از کلاس Person را مشاهده کنید:

```
public class Person {
    String name;
    String lastName;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }
}
```

حال کلاس OrderedPair را طوری طراحی کنید که بتواند مقادیر x و y را ذخیره کند و در صورت لزوم مورد استفاده قرار دهد.

```
public class OrderedPair {
    // put your fields here

    // put your getters and setters here
}
```

جمع دو جفت مرتب

دو جفت مرتب (x, y) و (x', y') در نظر بگیرید. جمع این دو جفت مرتب را به صورت زیر تعریف می‌کنیم:

$$(x, y) + (x', y') = (x + x', y + y')$$

حال می‌خواهیم بازای این عملگر یک تابع در کلاس OrderedPair بنویسیم که کار مشابه انجام دهد:

```
public class OrderedPair {

    public OrderedPair add(OrderedPair second) {
        //TODO
    }
}
```

توجه کنید که جفت مرتب دوم به عنوان پارامتر تابع داده می‌شود و عدد اول مقادیر ذخیره شده در همین کلاس هستند.

به همین ترتیب تابع minus را طوری تعریف کنید که عملیات زیر را انجام دهد:

$$(x, y) - (x', y') = (x - x', y - y')$$

در این حالت اطلاعات موجود در کلاس را به عنوان جفت اول در نظر بگیرید.

ضرب داخلی

ضرب داخلی دو جفت مرتب به صورت زیر است:

$$(x, y).(x', y') = xx' + yy'$$

تابع innerProduct را طوری طراحی کنید که عمل مشابه را انجام دهد:

```
public class OrderedPair {

    public double add(OrderedPair second) {
        //TODO
    }
}
```

کلاس Main

کلاس Main باید شامل تابع main باشد:

```
public class Main {

    public static void Main(String[] args) {
        //TODO
    }
}
```

در تابع main دو شیئ دلخواه از کلاس OrderedPair بسازید و آنها را به دلخواه مقداردهی کنید. سپس توابع add، minus و innerProduct را از شیئ اول صدا زده و و شیئ دوم را به آنها پاس دهید. در نهایت مقادیر محاسبه شده را در خروجی چاپ کنید.

نحوه آپلود

می‌توان در یک فایل جاوا بیش از یک کلاس تعریف کرد اما در این صورت تنها یکی از کلاس‌ها می‌تواند public باشد و نام این کلاس public با نام فایل باید یکسان باشد. شما باید یک فایل جاوا با نام Main ایجاد کنید که در آن دو کلاس وجود دارد:

```

public class Main {
    // your code is here
}

class OrderedPair {
    // your code is here
}
```

نکات

- توجه کنید data type اشیاء x و y از نوع double باشد.
- هنگام بارگزاری فایل خود مراقب باشید که در ابتدای فایل خود package تعریف نکرده باشید.

نقطه روی خط

معادله یک خط در فضای دو بعدی به صورت زیر نوشته می‌شود:

$$\mathcal{L} : ax + b = y$$

اگر نقطه(x_0, y_0) روی خط $\mathcal{L}$ قرار داشته باشد در معادله آن صدق می‌کند در غیراینصورت:

$$ax_0 + b \neq y_0$$

حال قصد داریم تابعی بنویسیم که تشخیص دهد آیا نقطه روی خط قرار دارد یا خیر:

```
public class Main {
    public static boolean isPointOnLine(double a, double b, double x, double y) {
        //TODO
    }
}
```

در صورتی که نقطه (x, y) روی خط

$$ax + b = y$$

قرار داشته باشد خروجی این تابع true و در غیر اینصورت false خواهد شد.

کلاس Main همچنین دارای تابع main است که در آن عملیات خواندن ورودی و چاپ خروجی انجام می‌شود:

```
public class Main {
    public static void main(String[] args) {
        //TODO
    }
}
```

ورودی

ورودی شامل چهار عدد در یک سطر است که به ترتیب a و b و x و y را مشخص می‌کنند. هنگام گرفتن ورودی از کاربر x, y را در یک کلاس از جنس OrderedPair که در سوال قبل گفته شد ذخیره کنید.

خروجی

خروجی برنامه شامل یک کلمه است که false یا true می‌باشد.

مثال

ورودی نمونه ۱

```
1 2 0 2
```

خروجی نمونه ۱

```
true
```

ورودی نمونه ۲

```
0.3333333333 0 3 1
```

خروجی نمونه ۲

```
true
```

توجه کنید در این مثال در اصل کاربر قصد وارد کردن عدد $\frac{1}{3}$ را دارد که به علت خطای گرد کردن به این صورت نوشته شده است.

ورودی نمونه ۳

```
2 3 3 8.99
```

خروجی نمونه ۳

```
false
```

نحوه آپلود کردن

با توجه به اینکه از کلاس OrderedPair در طی این تمرین استفاده می‌شود آن را نیز به فایل Main اضافه کنید. جزئیات این کار در تمرین قبل ذکر شد.

نکات

- دقت اندازه‌گیری باید حداقل تا ۱۰ رقم اعشار باشد.
- مراقب باشید تا در اول فایل خود package تعریف نکرده باشید.

رشته بلند

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

علی که در روز پیش توانسته بود یک سیستم برای شبیه سازی لیگ فوتبال طراحی کند، حسابی مغرور شده است و به توانایی‌های خود می‌بالد. حال می‌خواهد یک سیستم دیگر برای اعتبارسنجی رشته‌ها پیاده سازی کند، اما به دلیل خستگی زیاد پروژه‌ی قبلی، نمی‌خواهد سیستمش زیادی پیچیده باشد!

این سیستم در ابتدا یک عدد به عنوان **میزان خوبی** دارد که مقدارش برابر با **صفر** می‌باشد.

همچنین برای شروع کار، یک رشته متشکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` به طول حداکثر ۱۰۰۰ داده می‌شود. همچنین توجه کنید که رشته شامل فاصله (space) **نمی‌باشد**. این رشته را **رشته‌ی مشکوک** می‌نامیم.

حال عملیات‌هایی وجود دارند که بر روی این رشته اعمال می‌شوند و با آن می‌توان میزان اعتبار رشته را تشخیص داد.

دستور copy

```
copy key count
```

در این دستور، `key` یک رشته مشتکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` می‌باشد و `count` یک عدد طبیعی می‌باشد.

با استفاده از این دستور، شما باید رشته‌ی `key` را به اندازه‌ی `count` مرتبه به خود بچسبانید (یعنی اگر `key` برابر با `ab` باشد و `count` برابر با ۳ باشد، رشته‌ی به دست آمده برابر با `ababab` می‌شود) و سپس اگر رشته‌ی نهایی، طولش k باشد، باید k حرف اول (سمت چپ) رشته‌ی مشکوک را حذف کنید و این رشته‌ی جدید را به ابتدای رشته‌ی مشکوک بچسبانید.

همچنین تضمین می‌شود در صورتی که `key` را به اندازه‌ی `count` مرتبه به خودش بچسبانیم، طولش کمتر مساوی طول رشته‌ی مشکوک باشد.

برای مثال اگر رشته‌ی مشکوک برابر با `aaabbbccc` باشد و دستور زیر داده شود:

```
copy zx 2
```

رشته‌ی مشکوک برابر با `zxzxbbccc` می‌شود.

دستور compare

```
compare key
```

در این دستور، `key` یک رشته مشتکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` می‌باشد.

شما باید رشته‌ی مشکوک را با رشته‌ی `key` مقایسه کنید و در صورتی که این دو رشته با یکدیگر برابر باشند، **میزان خوبی** به اندازه‌ی **یک واحد** افزایش پیدا می‌کند و در غیر اینصورت تغییری نمی‌کند.

دستور substr

```
substr key count
```

در این دستور، `key` یک رشته مشتکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` می‌باشد و `count` یک عدد طبیعی می‌باشد.

در صورتی که رشته‌ی `key` **دقیقا** `count` مرتبه به عنوان زیررشته در رشته‌ی مشکوک ظاهر شده باشد، **میزان خوبی** به اندازه‌ی **یک واحد** افزایش پیدا می‌کند و در غیر اینصورت تغییری نمی‌کند.

دستور attach

```
attach key count str
```

در این دستور، `key` و `str` یک رشته مشتکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` می‌باشند و `count` یک عدد طبیعی می‌باشد.

شما باید رشته‌ی `str` را به انتهای رشته‌ی `key` بچسبانید و در صورتی که رشته‌ی بدست آمده **دقیقا** `count` مرتبه به عنوان زیررشته در رشته‌ی مشکوک ظاهر شده باشد، **میزان خوبی** به اندازه‌ی **یک واحد** افزایش پیدا می‌کند و در غیر اینصورت تغییری نمی‌کند.

برای مثال اگر رشته‌ی مشکوک برابر با `abc` باشد و دستور زیر داده شود، یک واحد به میزان خوبی افزوده می‌شود:

```
attach a 1 b
```

دستور length

```
length count
```

در این دستور `count` یک عدد طبیعی می‌باشد. در صورتی که طول رشته‌ی مشکوک **دقیقا** برابر با `count` باشد، **میزان خوبی** به اندازه‌ی **یک واحد** افزایش پیدا می‌کند و در غیر اینصورت تغییری نمی‌کند.

انتهای برنامه

همچنین هنگامی که تمامی دستورها داده شوند، در انتها یک دستور به صورت `Is it right or not?` داده می‌شود که از شما می‌پرسد آیا رشته‌ی مشکوک دارای اعتبار می‌باشد یا خیر.

در صورتی که **میزان خوبی** بیشتر و یا مساوی **نصف** تعداد دستورات داده شده باشد (دستورات `copy` و `compare` و `substr` و `attach` و `length`)، رشته دارای اعتبار است و باید `Eyval` را چاپ کنید و در غیر اینصورت دارای اعتبار نمی‌باشد و باید `HeifShod` را چاپ کنید.

ورودی

در خط اول ورودی، رشته‌ی مشکوک داده می‌شود که متشکل از حروف **کوچک و بزرگ انگلیسی** و کاراکترهای `#` و `!` و `?` و `.` می‌باشد و طول آن حداکثر ۱۰۰۰ است.

در خطوط بعدی، در هر خط یکی از دستوراتی که در صورت سوال آمده‌اند داده می‌شود. تعداد این دستورها کمتر از ۱۰۰۰ می‌باشد.

در خط نهایی، یک عبارت با عنوان `Is it right or not?` می‌آید که توضیح آن در صورت سوال داده شده است.

خروجی

در تنها خط خروجی، در صورتی که رشته‌ی مشکوک دارای اعتبار باشد باید عبارت `Eyval` و در غیر اینصورت عبارت `HeifShod` را چاپ کنید.

مثال

ورودی نمونه

```
eyval!inTamrinkheiliSadast.Hooorrraaaaa
copy hi 3
compare hihihiinTamrinkheiliSadast.Hooorrraaaaa
substr aaa 3
attach hi 2 in
length 39
Is it right or not?
```

خروجی نمونه

```
Eyval
```

در این مثال، فقط در هنگام اجرای دستور `attach` میزان خوبی اضافه نمی‌شود.

تکرار الگو

اطلاعات عملکرد یک سلول بر روی یک مولکول به نام DNA ثبت شده است. مولکول DNA یک پلیمر هست که مونومرهای سازنده اون چهار حالت A, C, G, T را دارا می‌باشند. این مولکول دارای یک ساختار خطی است به این معنی که هر مونومر آن تنها به مونومر قبلی و بعدی خود وابسته است و در نتیجه تنها اطلاعات مهم برای ما توالی این مونومرها است. به عنوان مثال به توالی زیر توجه کنید:

```
ACGAACGGGCCCACGGACGGACCTTAGG
```

با کمی توجه بر روی این توالی متوجه می‌شویم که در بین زیررشته‌های با طول 3 از این توالی، ACG به تعداد قابل توجه تکرار شده است:

```
ACGAACGGGCCCACGGACGGACCTTAGG
```

این حقیقت ما را کنجکاو می‌کند که شاید این توالی تکرار شده معنایی داشته باشد. معنا داشتن این زیررشته به این معنی است که ممکن است ACG جایگاهی برای نشستن سایر مولکول‌های زیستی مانند پروتئین‌ها باشد.

حال گاهی پیش می‌آید که در این مونومرها جهش ایجاد می‌شود به این معنی که یک مونومر به مونومر دیگر تبدیل می‌شود:

```
ACG -> AGG
```

با وجود این جهش‌های کوچک ممکن است که عملکرد زیر رشته تغییر نکند و به عنوان مثال همچنان پروتئین مورد نظر خود را به سمت خود جذب کند. در نتیجه زیررشته‌هایی که تعداد اندکی جهش در آنها رخ داده‌اند نیز برای ما ارزشمند هستند. به عنوان مثال توالی اولیه را در نظر بگیرید:

```
ACGAACGGGCCCACGGACGGACCTTAGG
```

حال زیست شناسان از ما درخواست کرده‌اند تا تابعی برای آنها بسازیم بطوریکه بتواند تعداد تکرار یک زیررشته در یک توالی بزرگتر را (با احتساب جهش‌های کوچک) پیدا کند:

```java
public class Main {

    public static int countOccurrences(String sequence, String pattern, int d) {
        //TODO
    }
}
```

در تابع فوق sequence دنباله‌ی اصلی است ، pattern زیردنباله‌ای است که می‌خواهیم تعداد وقوع آن در sequence را پیدا کنیم و d حداکثر جهش قابل قبول است.

برای پیاده‌سازی تابع فوق ابتدا تابع زیر را پیاده‌سازی کنید:

```java
public class Main {

    public static int distance(String s1, String s2) {
        //TODO
    }
}
```

این تابع دو رشته هم طول را می‌گیرد و اختلاف آن‌ها را خروجی می‌دهد. به عنوان مثال تفاوت دو رشته ACGT و ACCC، دو می‌باشد. حال روی sequence حرکت کنید و توالی‌های هم طول با pattern را مشاهده کنید و اگر فاصله آنها از pattern کمترمساوی d بود تعداد وقوع را یک واحد افزایش دهید.

کلاس Main علاوه بر توابع فوق دارای تابع main نیز می‌باشد که گرفتن ورودی و نمایش خروجی در آن انجام می‌شود.

مثال

ورودی نمونه ۱

```
ACGAACGGGCCCACGGACGGACCTTAGG ACG 1
```

خروجی نمونه ۱

```
6
```

رنگ‌ها برای فهم بهتر هستند.

ورودی نمونه ۲

```
ACCGGAGG CG 1
```

خروجی نمونه ۲

```
5
```