بسمه تعالی

دانشکده مهندسی کامپیوتر

آزمون نرم‌افزار (۴۰۸۲۸) (نیم‌سال ۲ـ۹۹)

دکتر حسن میریان

تمرین اول

امیرحسین کارگران خوزانی (۹۹۲۰۱۱۱۹)
سید سجاد میرزابابایی (۹۹۲۱۰۱۴۲)
رامتین باقری (۹۹۳۰۱۹۳۸)

بخش نظری

۱. در این بخش باید یک برنامه تبدیل اعداد رومی به اعداد دهدهی را مورد بررسی قرار دهید. این برنامه به زبان برنامه‌نویسی جاوا نوشته شده است.

(a) خطا یا خطاهای برنامه در کدام قسمت هستند؟ اصلاح‌شده آنها را بنویسید.

خروجی مورد انتظار این تابع برای ورودی `II` مقدار `2` خواهد بود؛ اما پس از اجرای تابع با ورودی مذکور، مقدار `0` نتیجه می‌شود. این مشکل از آنجا سرچشمه می‌گیرد که افزودن عدد جدید به اعداد قبلی **تنها** در صورتی انجام می‌شود که شرط `currentNumber > next` برقرار باشد، با این حال نیاز است تا برابری این دو متغیر نیز برای افزودن عدد لحاظ شود. چرا که در غیر این صورت در اعدادی مانند `II` که یک کاراکتر پشت‌سرهم تکرار شده است خطا ظاهر می‌شود. با تغییر در خط ۲۳ و اضافه کردن حالت برابری دو متغیر، مشکل حل خواهد شد. نسخه اصلاح‌شده برنامه در کد ۱ موجود است.

نکته‌ای که باید به آن توجه داشت این است که تمام رشته‌های شامل ترکیب حروف استفاده شده در اعداد رومی الزاما صحیح نیستند؛ برای مثال، عدد `8` تنها به فرم `VIII` مورد قبول است و اگر به شکل `IIX` نوشته شود صحیح نیست. از آنجا که دقیقا در توصیف برنامه گفته نشده است که در این شرایط چه باید کرد می‌توان دو فرض داشت:

۱. حتما ورودی به فرم اعداد رومی است و ورودی غیر مجازی داده نمی‌شود. در این صورت تنها خطای برنامه، همان خطای نامساوی بیان شده است.

۲. اگر ورودی به فرم اعداد رومی نباشد آن‌گاه باید برنامه خطای مناسب را در خروجی نشان دهد. در این صورت برنامه یک خطای دیگر نیز دارد چرا که این برنامه برای تمام رشته‌های بدفرم نیز جوابی در سیستم اعداد دهدهی تولید می‌کند.

به منظور جلوگیری از این اتفاق، می‌توان با استفاده از عبارت منظم زیر چک کرد که رشته ورودی معتبر است یا خیر. و تنها در صورتی که معتبر بود برنامه به شیوه‌ای که اصلاح شده است ادامه یابد و در غیر این صورت در خروجی، خطای مناسب تولید شود.

تنها در صورتی متغیر `valid` برابر `1` خواهد شد که رشته مورد نظر با یکی از فرم‌های صحیح عبارت منظم زیر تطبیق یابد:

```
boolean valid = word.matches("M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I}0,3})$");
```

کد ۱: نسخه اصلاح‌شده برنامه تبدیل اعداد رومی به دهدهی

```java
public class RomanNumeral {
   private static Map<Character, Integer> map;
   static {
      map = new HashMap<>();
      map.put('I', 1);
      map.put('V', 5);
      map.put('X', 10);
      map.put('L', 50);
      map.put('C', 100);
      map.put('D', 500);
      map.put('M', 1000);
   }

   public int convert(String s) {
      int convertedNumber = 0;
      for (int i = 0; i < s.length(); i++) {
         int currentNumber = map.get(s.charAt(i));
         int next = 0;
         if (i + 1 < s.length()) {
            next = map.get(s.charAt(i + 1));
         }

         if (currentNumber >= next)
            convertedNumber += currentNumber;
         else
            convertedNumber -= currentNumber;
      }
      return convertedNumber;
   }
}
```

(b) در صورت امکان مورد آزمونی ارائه دهید که خطا را اجرا نکند.

از آنجایی محل خطا در خط شماره ۲۳ قرار داد، باید مورد آزمون را به نحوی طراحی کنیم که این خط از کد اجرا نشود. از این رو مورد آزمون هدف، دارای ورودی `""` (رشته متنی با طول صفر) و خروجی قابل انتظار `0` است. چرا که طول این رشته برابر صفر می‌باشد و شرط `i < s.length()` در حلقه `for` که مقدار اولیه `i` برابر `0` است ارضا نمی‌شود و این حلقه اجرا نمی‌گردد. در بقیه حالات خط شماره ۲۳ که محل خطاست اجرا می‌گردد.

(c) در صورت امکان آزمونی بنویسید که خطا را اجرا کند اما نتیجه حالت میانی و پایانی مشخص کننده حالت اشتباه نباشد.

همانطور که بالاتر گفته شد، با اجرا شدن خط شماره ۲۳، خطا نیز اجرا خواهد شد. مورد آزمون هدف، ورودی `IV` و خروجی قابل انتظار `4` را خواهد داشت. در این صورت خطا اجرا می‌شود ولی تاثیری بر فرآیند محاسبه نخواهد گذاشت.

(d) در صورت امکان آزمونی بنویسید که برنامه در حالت میانی اشتباه است اما در پایان نتیجه شکست نمی‌شود. با مثالی نتیجه مورد انتظار و نتیجه اجرا را نشان دهید.

برنامه زمانی دارای حالت میانی اشتباه می‌شود که هر دو مورد زیر اتفاق بیفتد:

۱. خط شماره ۲۳ اجرا شود
۲. ورودی دارای حداقل دو کاراکتر یکسان متوالی باشد.

همان‌گونه که بحث شد تنها درصورتی که حداقل دو کاراکتر یکسان متوالی وجود داشته باشد، به جای شرط `if` ، شرط `else` اجرا می‌شود. از آنجا که این مورد باعث می‌شود که تنها به غلط از مقدار صحیح کم شود و جای دیگری در برنامه به غلط چیزی اضافه نمی‌شود، پس این حالت نمی‌تواند رخ دهد و نمی‌توان طبیعتا برای آن مورد آزمونی نیز که آن را ارضا کند نوشت.

اما اگر ورودی‌های بدفرم رشته اعداد رومی که شامل کاراکترهای مجاز آن هستند را این تابع به عنوان ورودی بپذیرد ان‌گاه اگر تفسیر رشته‌ای به فرم `IIX` (شامل دو کاراکتر تکراری متوالی قبل از یک کاراکتر پرارزش‌تر) را برابر `10 - 2 = 8` در نظر بگیریم (که این کار اشتباه است، و صرفا اینجا مشاهده را گزارش کرده‌ایم). علی رغم اشتباه بودن حالت میانی در حین محاسبه، نتیجه نهایی برابر `8` خواهد بود.

(e) با استفاده از تحلیل آر.آی.پی شرایطی را تعیین کنید که مورد آزمون تشخیص دهنده خطای این برنامه باید داشته باشد.

نتیجه تحلیل کد ۱ با استفاده از مدل RIP، به شکل زیر است:

دسترسی‌پذیری	$\|S\| > 0$
آلودگی	$\exists i, 0 \leq i < i+1 < \|S\| \wedge S[i] = S[i+1]$
انتشار	$\exists i, 0 \leq i < i+1 < \|S\| \wedge S[i] = S[i+1]$

بنابراین، مشخصات آزمون تشخیص دهنده خطای برنامه عبارت است از:

$$|S| > 0 \ \wedge \left(\exists i, j \ 0 \leq i < j < |S| \wedge j - i = 1 \wedge S[i] = S[j]\right)$$

۲. برنامه داده شده، کوچکترین و بزرگترین عنصر آرایه‌ای از اعداد صحیح را پیدا می‌کند. پیاده‌سازی این برنامه اشتباه است. با استفاده از روش افراز فضای ورودی، موارد آزمونی را طراحی کنید که خطای برنامه را مشخص کند. رویکرد شما باید حداقل دو خصوصیت مبتنی بر عملکرد و یک خصوصیت مبتنی بر واسط داشته باشد.

نسخه اصلاح شده برنامه در کد ۲ آمده است. خروجی مورد انتظار این برنامه بیشترین و کمترین مقدار آرایه‌ی `nums` است. اما اگر ورودی را به نحوی ارائه دهیم که جایگاه بزرگترین داده قبل از جایگاه تمام کوچکترین داده‌ها در لیست باشد، آنگاه مقدار `largest` هیچ‌گاه برابر بزرگترین داده نخواهد بود. برای مثال لیستی مرتب به صورت نزولی با اعضای `[4, 3, 2, 1]` را در نظر بگیرید، آنگاه خروجی کد به ازای این لیست برای متغیر `smallest` مقدار `1` و برای متغیر `largest` مقدار `-2147483648` خواهد بود. مقدار مورد انتظار برای `smallest` همان `1` اما برای `largest` ، مقدار `4` است که توسط این تابع بدست نیامده است.

کد ۲: برنامه یافتن بزرگترین و کوچکترین اعداد آرایه

```java
public class NumFinder {
   private int smallest = Integer.MAX_VALUE;
   private int largest = Integer.MIN_VALUE;

   public void find(int[] nums) {
      for (int n : nums) {
         if (n < smallest)
            smallest = n;
         if (n > largest)
            largest = n;
      }
   }

   public int getSmallest() {
      return smallest;
   }

   public int getLargest() {
      return largest;
   }
}
```

خصوصیات مبتنی بر واسط

A. علامت متغیر `smallest`

1. مثبت 2. منفی 3. صفر

B. علامت متغیر `largest`

1. مثبت 2. منفی 3. صفر

C. اندازه لیست `nums`

1. صفر 2. یک 3. بیشتر از یک

D. آیا لیست `nums` به ترتیب صعودی مرتب شده است؟

1. بله 2. خیر

E. آیا لیست `nums` به ترتیب نزولی مرتب شده است؟

1. بله 2. خیر

F. آیا تمامی اعضای `nums` از جنس `int` (در بازه مجاز) هستند؟

1. بله 2. خیر

G. آیا `n` `null` است یا خیر؟

1. بله 2. خیر

H. آیا `n` نوع `int` است؟

1. بله 2. خیر

I. آیا `n` در بازه ی مجاز `int` است یا خیر؟

1. بله 2. خیر

خصوصیات مبتنی بر عملکرد

J. آیا بزرگترین عنصر قبل از تمام عنصرهای کوچکتر در لیست ظاهر شده است؟

1. بله 2. خیر

K. آیا کوچکترین عنصر قبل از تمام عنصرهای بزرگتر در لیست ظاهر شده است؟

1. بله 2. خیر

«محل قرارگیری جدول موارد آزمون»

بخش عملی

۳. هدف این بخش آن است که یک برنامه پشته را با استفاده از رویه کاری ایجاد آزمون رانه گشترش داده و مورد آزمون قرار دهید. شما باید برای این برنامه که زبان برنامه‌نویسی جاوا نوشته شده است، به شیوه ایجاد آزمون رانه قابلیت جست‌وجو در یک پشته را فراهم کنید. به بیان دقیق‌تر، باید یک تابع با واسط زیر را پیاده‌سازی کنید.

ویژگی‌های معمول برای پشته به صورت زیر مورد پرسش قرار می‌گیرند:

A. آیا پشته خالی است؟

1. بله 2. خیر

B. اندازه پشته چقدر است؟

1. صفر 2. یک 3. بیشتر از یک

C. آیا پشته شامل عنصرهای `null` است؟

1. بله 2. خیر

همچنین ویژگی‌های مربوط به عنصر که قصد جستجوی آن را داریم به زیر هستند:

D. آیا عنصر `i` ، `null` است؟

1. بله 2. خیر

و همچنین سولاتی که پیرامون رابطه پشته و عنصری که قصد جستجوی آن در پشته را داریم قابل طرح هستند، نظیر:

E. آیا عنصر `i` در پشته وجود دارد؟

1. بله 2. خیر

F. آیا عنصر `i` ، المان اول در پشته است؟

1. بله 2. خیر

G. آیا عنصر `i` ، المان آخر در پشته است؟

1. بله 2. خیر

حال با توجه به خصوصیات مطرح شده و افراز بلاک‌های انجام شده، به روش EC[1] اقدام به ساخت موارد آزمون می‌کنیم که لیست این موارد در جدول ۱ آورده شده است. در این روش کافی است از هر بلاک حداقل یک مورد در آزمون استفاده شود.

جدول ۱: موارد آزمون ساخته شده به روش EC

	TC 1	TC 2	TC 3	TC 4	TC 5	TC 6	TC 7	TC 8
A	A2	A2	A2	A1	A2	A2	A2	A2
B	B3	B3	B3	B1	B2	B3	B3	B3
C	C2	C2	C2	C2	C2	C1	C2	C2
D	D2	D2	D2	D2	D2	D2	D1	D2
E	E1	E1	E2	E2	E1	E1	E2	E1
F	F1	F2	F2	F2	F1	F2	F2	F2
G	G2	G1	G2	G2	G1	G2	G2	G2

توجه کنید که با توجه به اینکه زبان جاوا به صورت strict-type است، امکان push کردن عنصر null به پشته وجود ندارد و همچنین در صورت ارسال آن به تابع search، با خطای compile روبرو خواهیم شد. به همین دلیل، موارد آزمون ۶ و ۷ در عمل پیاده‌سازی نمی‌شوند. شکل ۱ نتایج اعمال موارد آزمون را توسط چارچوب JUnit نمایش می‌دهد.

[1] Each Choice

شکل ۱: نتایج موارد آزمون اعمال شده