Ararpski u rimski

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Rimski brojevi se zapisuju pomoću slova I, V, X, L, C, D, M. Na primer, broj 15 se zapisuje kao XV, a broj 148 kao CXLVIII. Napiši program koji prevodi uneti arapski broj u rimski.

Ulaz

Sa ulaza se unosi prirodan broj n (1≤n≤3999)

Izlaz

Na standardni izlaz se ispisuje rimski zapis broja n zapisan velikim slovima latinice.

Primer 1

Ulaz

1978

Izlaz

MCMLXXVIII

Primer 2

Ulaz

2017

Izlaz

MMXVII

RESENJE

```
def int_to_Roman(num):

    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
        ]
    syb = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
        ]
    roman_num = ''
    i = 0
    while  num > 0:
        for _ in range(num // val[i]):
            roman_num += syb[i]
            num -= val[i]
        i += 1
    return roman_num

broj=int(input())
print(int_to_Roman(broj))
```

VREME RADA: 0.03s

Ovaj zadatak je dualan zadatku Rimski u arapski, ali im se rešenja prilično razlikuju. Rimski zapis koristi različite simbole za zapis cifara jedinica, cifara desetica i cifara stotica, ali je princip zapisa svake od deset dekadnih cifara od tih simbola isti.
I za jedinice i za desetice i za stotine koristi se po tri karakteristična simbola: simbol s1 koji predstavlja vrednost 1, simbol s5 koji predstavlja vrednost 5 i simbol s10 koji predstavlja vrednost 10.
U slučaju cifre jedinice to su simboli I, V i X, u slučaju cifre desetice to su simboli X, L i C, a u slučaju cifre stotine to su simboli C, D i M.
Cifra 0 se zapisuje praznom niskom, cifra 1 jednim simbolom s1, cifra 2 pomoću dva simbola s1, cifra 3 pomoću tri simbola s1, cifra 4 simbolima s1s5, cifra 5 simbolom s5, cifra 6 simbolima s5s1, cifra 7 simbolima s5s1s1, cifra 8 simbolima s5s1s1s1, a cifra 9 simbolima s1s10. Zato definišemo posebnu funkciju koja za datu cifru i data tri simbola s1, s5 i s10 od tih simbola gradi nisku koja predstavlja zapis te cifre. Ona može izvršiti klasifikaciju kom intervalu pripada data cifra c i ako je ona manja od 4 vratiti nisku dobijenu tako što se c ponovi simbol s1, ako je jednaka 4 vratiti nisku s1s5, ako je manja od 9 vratiti nisku koja se dobije kada se na simbol s5 simbol s1 nadoveže c−5 puta, a ako je jednaka 9 vratiti nisku 1s10.

U jeziku C++, se niska koja sadrži karakter c k puta gradi konstruktorom string(k, c), a u jeziku C# konstruktoromnew string(c, k).

Glavna funkcija konverzije izdvaja cifru po cifru broja n operacijom celobrojnog ostatka i uklanja je operacijom celobrojnog količnika pri deljenju sa 10, krenuvši od cifre jedinica. Svaka tako dobijena cifra se konvertuje u rimski zapis pomoćnom funkcijom, prosleđujući joj simbole s1, s5 i s10 u zavisnosti od težine tekuće cifre i taj zapis se dodaje ispred dosadašnjeg izgrađenog rezultata (koji kreće od prazne niske i proširuje se na levo). Postupak se završava kada se obrade sve cifre broja nn tj. kada se on celobrojnim deljenjem svede na nulu. Ako se to ne desi ni nakon uklanjanja cifre stotine, preostali broj (to je broj hiljada) se konvertuje u rimski tako što se taj broj puta na početak rezultata doda simbol M.

C++ resenje

```
#include <iostream>

#include <string>

using namespace std;

// zapis jedne rimske cifre - simbol1, simbol5 i simbol 10 odredjuju
// da li se radi i cifri jedinica, desetica ili stotina
string rimskaCifra(int c, char simbol1, char simbol5, char simbol10) {
  if (c < 4) // npr. "", "I", "II", "III"
    return string(c, simbol1);
  else if (c == 4) // npr. "IV"
```

```
    return string(1, simbol1) + string(1, simbol5);
  else if (c < 9) // npr. "V", "VI", "VII", "VIII"
    return string(1, simbol5) + string(c-5, simbol1);
  else // npr. "IX"
    return string(1, simbol1) + string(1, simbol10);
}

// prevodi dati broj u rimski zapis
string arapskiURimski(int n) {
  string rezultat = "";
  // cifra jedinica
  rezultat = rimskaCifra(n % 10, 'I', 'V', 'X');
  n /= 10; // uklanjamo je
  // ako nema vise cifara vracamo rezultat
  if (n == 0) return rezultat;
  // cifra desetica
  rezultat = rimskaCifra(n % 10, 'X', 'L', 'C') + rezultat;
  n /= 10; // uklanjamo je
  // ako nema vise cifara vracamo rezultat
  if (n == 0) return rezultat;
  // cifra stotina
  rezultat = rimskaCifra(n % 10, 'C', 'D', 'M') + rezultat;
  n /= 10; // uklanjamo je
  // ako nema vise cifara vracamo rezultat
  if (n == 0) return rezultat;
  // dodajemo hiljade na pocetak rezultata i vracamo ga
  return string(n, 'M') + rezultat;
}


int main() {
  // ucitavamo broj
  int n;
  cin >> n;
  // prevodimo ga u rimski zapis i ispisujemo rezultat
  cout << arapskiURimski(n) << endl;
  return 0;
}
```

Vreme rada: 0.01 s

C# resenje

```
using System;


class Program
{
    // zapis jedne rimske cifre - simbol1, simbol5 i simbol 10
    // odredjuju da li se radi i cifri jedinica, desetica ili
    // stotina
    static string rimskaCifra(int c, char simbol1, char simbol5, char simbol10) {
```

```
        if (c < 4)  // npr. "", "I", "II", "III"
            return new string(simbol1, c);
        else if (c == 4) // npr. "IV"
            return new string(simbol1, 1) + new string(simbol5, 1);
        else if (c < 9) // npr. "V", "VI", "VII", "VIII"
            return new string(simbol5, 1) + new string(simbol1, c-5);
        else // npr. "IX"
            return new string(simbol1, 1) + new string(simbol10, 1);
    }
    // prevodi dati broj u rimski zapis
    static string arapskiURimski(int n) {
        string rezultat = "";
        // cifra jedinica
        rezultat = rimskaCifra(n % 10, 'I', 'V', 'X');
        n /= 10; // uklanjamo je
        // ako nema vise cifara vracamo rezultat
        if (n == 0) return rezultat;
        // cifra desetica
        rezultat = rimskaCifra(n % 10, 'X', 'L', 'C') + rezultat;
        n /= 10; // uklanjamo je
        // ako nema vise cifara vracamo rezultat
        if (n == 0) return rezultat;
        // cifra stotina
        rezultat = rimskaCifra(n % 10, 'C', 'D', 'M') + rezultat;
        n /= 10; // uklanjamo je
        // ako nema vise cifara vracamo rezultat
        if (n == 0) return rezultat;
        // dodajemo hiljade na pocetak rezultata i vracamo ga
        return new string('M', n) + rezultat;
    }

    static void Main(string[] args)
    {
        // ucitavamo broj
        int n = int.Parse(Console.ReadLine());
        // prevodimo ga u rimski zapis i ispisujemo rezultat
        Console.WriteLine(arapskiURimski(n));
    }
}
```

Vreme rada: 0.03 s

Java resenje

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;
import java.lang.*;
import java.io.*;
import java.util.*;
```

```
public class Main
{

enum RomanNumeral {
   I(1), IV(4), V(5), IX(9), X(10),
   XL(40), L(50), XC(90), C(100),
   CD(400), D(500), CM(900), M(1000);

   private int value;

   RomanNumeral(int value) {
      this.value = value;
   }

   public int getValue() {
      return value;
   }

   public static List<RomanNumeral> getReverseSortedValues() {
      return Arrays.stream(values())
       .sorted(Comparator.comparing((RomanNumeral e) -> e.value).reversed())
       .collect(Collectors.toList());
   }
}

public static String arabicToRoman(int number) {

   List<RomanNumeral> romanNumerals = RomanNumeral.getReverseSortedValues();

   int i = 0;
   StringBuilder sb = new StringBuilder();

   while ((number > 0) && (i < romanNumerals.size())) {
      RomanNumeral currentSymbol = romanNumerals.get(i);
      if (currentSymbol.getValue() <= number) {
         sb.append(currentSymbol.name());
         number -= currentSymbol.getValue();
      } else {
         i++;
      }
   }

   return sb.toString();
}

public static void main(String[] args)
 {
   Scanner ulaz = new Scanner(System.in);
    int n = ulaz.nextInt();
```

```
    System.out.println(arabicToRoman(n));
}



}
```

Vreme rada: 0.23 s

Rimski u arapski

Vreme	memorija	ulaz	izlaz
1 s	64 Mb	standardni ulaz	standardni izlaz

Napiši program koji konvertuje rimske u arapske brojeve.

Ulaz

Jedina linija standardnog ulaza sadrži jedan rimski broj iz intervala od 1 do 3999.

Izlaz

Na standardni izlaz ispisati uneti broj zapisan u uobičajenom (arapskom) zapisu.

Primer

Ulaz
MCMLXXVIII
Izlaz
1978

Rešenje

Ovaj zadatak je dualan zadatku Arapski u rimski, ali im se rešenja prilično razlikuju. Vrednost ispravno zapisanog rimskog broja se može odrediti tako što čitamo njegove cifre s leva na desno i na tekući zbir dodeljujemo vrednost svake cifre (M ima vrednost 1000, D vrednost 500, C vredost 100, L vrednost 50, X vrednost 10, V vrednost 5 i I vrednost 1). Jedini izuzetak od ovog pravila je slučaj kada se u zapisu pojavi manja ispred veće cifre i tada se od tekućeg zbira ta cifra oduzima.
Na primer, izračunajmo vrednost zapisa MCMLXXIV. Na osnovu prethodne diskusije, vrednost ovog broja je 1000 - 100 + 1000 + 50 + 10 + 10 – 1 + 5 tj. 1974.
Rezultat se inicijalizuje na nulu. Prva cifra je M iza koje nije veća cifra, pa se rezultat uvećava za 1000. Nakon toga nastupa slučaj u kojem sledi cifra C koja je manja od naredne cifre M, tako da se od zbira oduzima vrednost 100 i dobija se vrednost
900.
Nakon toga se na zbir dodaje 50, pa dva puta 10 i dobija se zbir 1970, dok se ne naiđe na cifru I koja je manja od njoj naredne cifre V. Zato se od zbira oduzima vrednost 1 i dobija se 1969, pre nego što se na kraju doda vrednost poslednje cifre V i dobije konačan rezultat
901.

Prolaz kroz sve rimske cifre datog rimskog broja vršimo u sklopu jedne petlje u kojoj indeksna promenljiva i menja svoju vrednost od nule, pa sve dok je manja od dužine niske kojom je zapisa redni broj. U telu petlje se za svaku cifru proverava da li neposredno iza nje postoji cifra koja je od nje veća (pri tom se mora voditi računa i o tome da tekuća cifra nije poslednja). Ako takva cifra postoji, onda se tekući zbira umanji za vrednost tekuće cifre, u suprotnom se na zbir dodaje vrednost tekuće cifre. U svakom slučaju indeksna promenljiva se uvećava za 1 (preskače se tekuća i prelazi se na narednu cifru, ako ona postoji). Vrednosti cifara možemo čuvati u posebnom asocijativnom nizu (mapi, rečniku) tj. možemo u jeziku C++ upotrebiti bibliotečku strukturu podataka map a u jeziku C# Dictionary. Druga mogućnost je definisanje pomoćne funkcije koja određuje vrednost date cifre i to grananjem (npr. naredbom switch).

Python rešenje

```
def roman_to_int(s):
        rom_val = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
        int_val = 0
        for i in range(len(s)):
            if i > 0 and rom_val[s[i]] > rom_val[s[i - 1]]:
                int_val += rom_val[s[i]] - 2 * rom_val[s[i - 1]]
            else:
                int_val += rom_val[s[i]]
        return int_val

rimski=input()
broj=roman_to_int(rimski)
print(broj)

Vreme rada: 0.03s
```

C++ rešenje

```
#include <iostream>

#include <string>
#include <map>

using namespace std;

int rimskiUArapski(string rimski) {
  // vrednost svake rimske cifre
  map<char, int> cifre = { {'M', 1000}, {'D',  500}, {'C', 100}, {'L', 50}, {'X', 10}, {'V', 5},
{'I', 1} };
  // rezultat - vrednost rimski zapisanog broja
  int arapski = 0;
  // prolazimo kroz sve cifre rimskog broja
```

```
  for (int i = 0; i < rimski.size(); i++) {
    // ako neposredno iza tekuce postoji cifra veca od nje
    if (i + 1 < rimski.size() && cifre[rimski[i]] < cifre[rimski[i+1]])
      // rezultat umanjujemo za vrednost tekuce cifre
      arapski -= cifre[rimski[i]];
    else
      // inace rezultat uvecavamo za vrednost tekuce cifre
      arapski += cifre[rimski[i]];
  }
  // vracamo konacan rezultat
  return arapski;
}

int main() {
  // ucitavamo rimski zapis broja
  string rimski;
  cin >> rimski;
  // odredjujemo mu vrednost i ispisujemo je
  cout << rimskiUArapski(rimski) << endl;
  return 0;
}
```

Vreme rada: 0.02s

C++ resenje bez koriscenje map-e

```
#include <iostream>
#include <string>

using namespace std;

// vrednost svake rimske cifre
int cifra(char c) {
  switch(c) {
  case 'M': return 1000;
  case 'D': return 500;
  case 'C': return 100;
  case 'L': return 50;
  case 'X': return 10;
  case 'V': return 5;
  case 'I': return 1;
  }
}

int rimskiUArapski(string rimski) {
  // rezultat - vrednost rimski zapisanog broja
  int arapski = 0;
  // prolazimo kroz sve cifre rimskog broja
  for (int i = 0; i < rimski.size(); i++) {
```

```
    // ako neposredno iza tekuce postoji cifra veca od nje
    if (i + 1 < rimski.size() && cifra(rimski[i]) < cifra(rimski[i+1]))
      // rezultat umanjujemo za vrednost tekuce cifre
      arapski -= cifra(rimski[i]);
    else
      // inace rezultat uvecavamo za vrednost tekuce cifre
      arapski += cifra(rimski[i]);
  }
  // vracamo konacan rezultat
  return arapski;
}

int main() {
  // ucitavamo rimski zapis broja
  string rimski;
  cin >> rimski;
  // odredjujemo mu vrednost i ispisujemo je
  cout << rimskiUArapski(rimski) << endl;
  return 0;
}
```

Vreme rada: 0.02s

C# resenja

```
using System;
using System.Collections.Generic;

class Program
{
    static int rimskiUArapski(string rimski) {
        // vrednost svake rimske cifre
        Dictionary<char, int> cifre = new Dictionary<char, int>
        { {'M', 1000}, {'D',  500}, {'C', 100}, {'L', 50}, {'X', 10}, {'V', 5}, {'I', 1} };
        // rezultat - vrednost rimski zapisanog broja
        int arapski = 0;
        // prolazimo kroz sve cifre rimskog broja
        for (int i = 0; i < rimski.Length; i++) {
            // ako neposredno iza tekuce postoji cifra veca od nje
            if (i + 1 < rimski.Length && cifre[rimski[i]] < cifre[rimski[i+1]])
                // rezultat umanjujemo za vrednost tekuce cifre
                arapski -= cifre[rimski[i]];
            else
                // inace rezultat uvecavamo za vrednost tekuce cifre
                arapski += cifre[rimski[i]];
        }
        // vracamo konacan rezultat
        return arapski;
    }
```

```
    static void Main(string[] args)
    {
        // ucitavamo rimski zapis broja
        string rimski = Console.ReadLine();
        // odredjujemo mu vrednost i ispisujemo je
        Console.WriteLine(rimskiUArapski(rimski));
    }
}
```

Vreme rada: 0.03s

C# resenje 2 (bez asocijativnog niza tj. recnika)

```
using System;

class Program
{
    // vrednost svake rimske cifre
    static int cifra(char c) {
        switch(c) {
            case 'M': return 1000;
            case 'D': return 500;
            case 'C': return 100;
            case 'L': return 50;
            case 'X': return 10;
            case 'V': return 5;
            case 'I': return 1;
            default: return 0;
        }
    }

    static int rimskiUArapski(string rimski) {
        // rezultat - vrednost rimski zapisanog broja
        int arapski = 0;
        // prolazimo kroz sve cifre rimskog broja
        for (int i = 0; i < rimski.Length; i++) {
            // ako neposredno iza tekuce postoji cifra veca od nje
            if (i + 1 < rimski.Length && cifra(rimski[i]) < cifra(rimski[i+1]))
                // rezultat umanjujemo za vrednost tekuce cifre
                arapski -= cifra(rimski[i]);
            else
                // inace rezultat uvecavamo za vrednost tekuce cifre
                arapski += cifra(rimski[i]);
        }
        // vracamo konacan rezultat
        return arapski;
    }

    static void Main(string[] args)
```

```
    {
        // ucitavamo rimski zapis broja
        string rimski = Console.ReadLine();
        // odredjujemo mu vrednost i ispisujemo je
        Console.WriteLine(rimskiUArapski(rimski));
    }
}
```

Vreme rada: 0.02s

Java resenje

```
import java.util.*;

import java.io.*;

import java.lang.*;


public class Main {

        private static int decodeSingle(char letter) {

                switch(letter) {

                        case 'M': return 1000;

                        case 'D': return 500;

                        case 'C': return 100;

                        case 'L': return 50;

                        case 'X': return 10;

                        case 'V': return 5;

                        case 'I': return 1;

                        default: return 0;

                }

        }

        public static int decode(String roman) {

                int result = 0;

                String uRoman = roman.toUpperCase(); //case-insensitive
```

```
            for(int i = 0;i < uRoman.length() - 1;i++) {
                if (decodeSingle(uRoman.charAt(i)) <
decodeSingle(uRoman.charAt(i+1))) {
                    //oduzmi klasu cifre pocev od M
                    result -= decodeSingle(uRoman.charAt(i));
                } else {
                    //dodaj klasu cifre pocev od D
                    result += decodeSingle(uRoman.charAt(i));
                }
            }
            //zavrsetak broja
            result += decodeSingle(uRoman.charAt(uRoman.length()-1));
            return result;
        }

        public static void main(String[] args) {
            Scanner ulaz = new Scanner(System.in);
         System.out.println(decode(ulaz.toString()));
}

        }
```

Trougao od zvezdica

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Napiši program koji iscrtava trougao kakav je prikazan u primeru.

Ulaz

Sa standardnog ulaza se učitava jedan prirodan broj n ($1 \leq n \leq 30$).

Izlaz

Na standardni izlaz se ispisuje traženi trougao, ispisivanjem karaktera *, razmaka (karaktera blanko) i prelaska u novi red. Posle svake poslednje zvezdice u svakoj vrsti preći u naredni red (ne ispisivati razmake posle zvezdica).

Primer
Ulaz
6
Izlaz

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

Rešenje

Za dati broj nn trougao sadrži n redova.

Svaki red ćemo ispisivati u petlji u kojoj promenljiva ii menja vrednost od 0 do n−1.

Razmotrimo slučaj n=4.

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

U prvom redu (kada je i=0) potrebno je ispisati tri razmaka i jednu zvezdicu, u drugom (kada je i=1) potrebno je ispisati dva razmaka i tri zvezdice, u trećem (kada je i=2) potrebno je jedan razmak i pet zvezdica i u četvrtom (kada je i=3) potrebno je ispisati nula razmaka i sedam zvezdica. Može se zaključiti da je broj razmaka jednak n−i−1, a broj zvezdica jednak 2i+1 (iako je izvođenje ovih relacija prilično očigledno, možemo primetiti da broj razmaka čini aritmetički niz koji kreće od n−1 i čija je razlika susednih elelemenata −1, dok broj zvezdica čini aritmetički niz koji kreće od 1 i čija je razlika susednih elemenata 2 i veze izvesti na osnovu formula za i-ti član aritmetičkog niza). Ispisivanje datog broja pojavljivanja nekog karaktera možemo realizovati jednostavno u petlji. Jednostavnosti radi, tu petlju možemo realizovati u zasebnoj funkciji koju ćemo onda pozvati u različitim kontekstima (jednom za ispisivanje n−i−1 razmaka, a drugi put za ispisivanje 2i+1 zvezdica). Sa druge strane, možemo je dva puta ponoviti kao ugnežđenu petlju. Nakon ispisa zvezdica, kao poslednji korak tela spoljašnje petlje potrebno je da pređemo u naredni red.

Recimo i da broj zvezdica i razmaka nismo morali eksplicitno izračunavati, već je moguće održavati dve posebne promenljive u kojima se pamte ovi brojevi.

```
n=int(input())
for i in range(n):
    for j in range(n-1-i):
        print(" ",sep="",end="")
    for j in range(2*i+1):
        print("*",sep="",end="")
    print("\n")
```

Kvadrat od zvezdica

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Napiši program koji iscrtava kvadrat od karaktera * (kao što je prikazano u primerima).
Ulaz

Sa standardnog ulaza se učitava prirodan broj n (1≤n≤20).
Izlaz

Na standardni izlaz ispisati traženi crtež.
Primer
Ulaz
4
Izlaz
```
****
****
****
****
```

Rešenje

Možemo primetiti da se kvadrat sastoji od n redova, a da se u svakom redu nalazi n zvezdica. Zato program možemo organizovati pomoću ugnežđenih petlji. Spoljna petlja se izvršava n puta i u njenom telu se ispisuje jedan red. To se vrši tako što se u telu unutrašnje petlje koja se izvršava n puta u čijem telu se ispisuje jedna zvezdica, a nakon te unutrašnje petlje se ispisuje prelazak u novi red.
U jeziku C++ se zvezdica ispisuje pomoću cout << '*';, a prelazak u novi red pomoću cout << endl;.
U jeziku C# se zvezdica ispisuje pomoću Console.Write("*"); a prelazak u novi red pomoću Console.WriteLine() (WriteLine nakon ispisanog prelazi u novi red,a Write ne)
U jeziku Python 3 zvezdica ispisuje pomoću print("*", sep="", end="") a prelazak u novi red pomoću print() (dodatnim parametrima sep i end podešavamo šta se ispisuje između navedenih parametara i šta se ispisuje na kraju).

RESENJE1
```
n = int(input())
for i in range(n):
    for j in range(n):
        print("*", sep="", end="")
    print()
```

RESENJE2
```
n=int(input())
for i in range (1,n+1):
        for j in range (1,n+1):
                print("*",end="")
        print("\n")
```

Romb od zvezdica

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Napiši program koji ispisuje romb napravljen od zvezdica (kako je prikazano u primeru).
Ulaz

Sa standardnog ulaza se unosi prirodan broj n (3≤n≤20).
Izlaz

Na standardni izlaz iscrtati romb.
Primer
Ulaz
5
Izlaz

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

Rešenje

Romb se sastoji od n redova, a u svakom redu se prvo ispisuje određeni broj razmaka, a zatim i n zvezdica. Pretpostavimo da ćemo to ostvariti spoljašnjom petljom u kojoj ii uzima vrednosti od 0 do n−1. Obeležimo za svaki red vrednost brojača *i*, broj zvezdica i broj razmaka.

```
        i  br  bz
  *****    0   4   5
 *****     1   3   5
*****      2   2   5
*****      3   1   5
*****      4   0   5
```

Možemo primetiti da je u svakom redu zbira broja razmaka i brojača *i* konstantan i jednak vrednosti n−1, pa je stoga broj razmaka uvek jednak n−1−i. Dakle u telu spoljašnje petlje imaćemo unutrašnju petlju koja ispisuje n−1−i razmaka, a zatim i drugu unutrašnju petlju koja ispisuje n zvezdica. Na kraju tela spoljašnje petlje (nakon obe unutrašnje) prelazimo u novi red.
Unutrašnje petlje možemo izdvojiti u funkciju koja dati karakter ispisuje dati broj puta.
Možemo primetiti i da smo spoljašnju petlju mogli realizovati tako da brojač ii kreće od n−1 i spušta se do 0 i tada bi broj razmaka bio jednak *i*.
Još jedno moguće rešenje je da broj razmaka pratim pomoću posebne promenljive koju inicijalizujemo na n−1 i umanjujemo je za 1 na kraju tela unutrašnje petlje.

```
n=int(input())
for j in range(1,n+1):
        for i in range(1,n-j+1):
                print(" ",end="")
```

```
        for i in range(n-j, 2*n-j):
                print("*",end="")
        print("\n")
```

Trougao od reči

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Napiši program koji iscrtava trougao čije su ivice sastavljene od karaktera date reči. Reč se dobija čitanjem slova sa leve i desne ivice trougla naniže, dok se na donjoj ivici nalazi palindrom čija je desna polovina ta data reč.

Ulaz

Sa standardnog ulaza se učitava reč dužine između 3 i 20 karaktera.

Izlaz

Na standardni izlaz ispisati traženi trougao.

Primer

Ulaz

trougao

Izlaz

```
      t
     r r
    o   o
   u     u
  g       g
 a         a
oaguortrougao
```

Rešenje

Možemo primetiti da postoje tri suštinski različita slučaja. Neka je n dužina reči.

U prvom redu se ispisuje n−1 razmaka i prvo slovo reči.

Neka su naredni redovi obeleženi brojevima od 1 do n−2 tj. neka u petlji promenljiva *i* uzima vrednosti od 1 do n−2. U telu te petlje ispisujemo prvo n−i−1 razmaka, zatim slovo reči na poziciji *i*, nakon toga 2·i−1 razmaka i ponovo slovo na poziciji *i*.

Na kraju, u poslednjem redu ispisujemo slova reči na pozicijama od n−1 do 0 i zatim od 1 do n−1.

```
s=(input())
n=len(s)
for i in range(n-1):
    for j in range(1,2*n):
        if(j==n-i or j==n+i):
            print(s[i],end="")
        else:
```

```
            print(" ",end="")
    print("")
for i in range(n-1,0,-1):
    print(s[i],end="")
print(s)
```

Serija 123

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Za dato n ispisati članove niza 1,2,2,3,3,3,4,4,4,4,...,n,...,n (niz sadrži jednu jedinicu, dve dvojke, tri trojke itd.).
Ulaz
Sa standardnog ulaza se učitava prirodan broj n (1≤n≤30).
Izlaz
Članove traženog niza ispisati na standardni izlaz, svaki u posebnom redu.
Primer
Ulaz
4
Izlaz
1
2
2
3
3
3
4
4
4
4

Rešenje

Prvo je potrebno ispisati jedan broj 1, zatim dva broja 2, zatim tri broja 3 i na kraju nn brojeva n. Dakle, za svaku vrednost promenljive *i* od 1 do n, potrebno je *i* puta ispisati vrednost promenljive *i*. Pošto se ispisivanje broja *i* i puta može realizovati uz pomoć petlje u kojoj promenljiva jj uzima redom vrednosti od 1 do i, a u čijem se telu nalazi naredba za ispis broja i, zadatak se može rešiti ugnežđenim ciklusima. Ako je n učitana vrednost, spoljašnjim ciklusom u kome promenljiva ii uzima redom vrednosti od 1 do n se obezbeđuje n ponavljanja unutrašnjeg ciklusa u kome se promenljiva spoljašnjeg ciklusa *i* ispisuje *i* puta.
Drugo rešenje izbegava korišćenje ugnežđenih ciklusa, tako što održava dve promenljive - jednu u kojoj pamtimo tekući broj koji se ispisuje, a drugu u kojoj pamtimo koliko puta do sada je taj broj ispisan. Broj se inicijalizuje na 1 i petlja se izvršava sve dok je broj manji ili

jednak od vrednosti n. U svakom koraku petlje ispisuje se broj i uvećava promenljiva kojom se broji koliko puta je ispisan. Kada ta vrednost postane jednaka vrednosti tekućeg broja (to je njegov željeni broj ispisivanja), prelazi se na sledeći broj, tako što se vrednost broja uveća za 1, a brojač koji broji koliko je puta broj ispisan vrati na nulu (jer uvećani broj još nismo ispisali).

```
n=int(input())
for i in range (1,n+1):
      for j in range(i):
            print(i)
```

Serije parni neparni

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Niz prirodnih brojeva se može razložiti na segmente (podnizove uzastopnih elemenata) koje čine elementi iste parnosti. Na primer, niz 3,5,2,4,6,7,1 se može razložiti na segmente 3,5, zatim 2,4,6 i na kraju 7,1. Napiši program koji za dati niz prirodnih brojeva izračunava zbirove svih tako dobijenih segmenata.
Ulaz
Sa standardnog ulaza se učitava broj n (0≤n≤50000) a zatim i n prirodnih brojeva, svaki u posebnom redu.
Izlaz
Na standardni izlaz ispisati tražene zbirove, svaki u posebnom redu.
Primer
Ulaz
7
1
3
4
6
8
5
1
Izlaz
4
18
6

Najjednostavniji način da se zadatak uradi je čitanje jednog po jednog elementa (u jednoj petlji), uz održavanje ukupnog broja pročitanih elemenata, tekućeg elementa, prvog elementa u tekućoj seriji parnih tj. neparnih elemenata (umesto ovoga, moguće je održavati prethodni element u seriji) i zbira elemenata u toj tekućoj seriji. Pre početka

petlje učitavamo prvi element, inicijalizujemo zbir tekuće serije na njegovu vrednost, i broj pročitanih elemenata na jedan, pri čemu moramo voditi računa o specijalnom slučaju kada je ulazni niz prazan (u tom slučaju je potrebno jednostavno izaći iz programa, bez čitanja i jednog elementa). Nakon toga, u petlji koja se izvršava sve dok nisu učitani svi elementi čitamo tekući element (uvećavajući pri tom broj učitanih elemenata) i provervamo da li je iste parnosti kao prvi element tekuće serije (provera da li su dva broja iste parnosti može se izvršiti upoređivanjem njihovih ostataka pri deljenju sa 2). Ako jeste, tada taj element produžava tekuću seriju i dodaje se na tekući zbir. U suprotnom, prethodna serija je završena i ispisuje se njen zbir. Broj koji je pročitan započinje novu seriju, tako da se prvi element serije i suma postavljaju na njegovu vrednost. Nakon čitanja svih elemenata promenljiva suma sadrži vrednost poslednje serije i potrebno je ispisati nakon petlje.

Drugi moguć pristup rešavanju ovog zadatka je da se algoritam realizuje kroz dve ugnežđene petlje: spoljašnju koja obrađuje jednu po jednu seriju i unutrašnju čiji je zadatak da učita i sabere elemente pojedinačne serije. Iako je ovo veoma prirodna ideja, pokazaće se da se u ovom konkretnom zadatku ona malo teže realizuje. Svejedno, neke tehnike koje prikazujemo u nastavku mogu biti korisne u širem kontekstu i drugim zadacima, tako da ih ima smisla proučiti.

Kada bi svi elementi bili istovremeno učitani u memoriju (u niz ili vektor), program bi se mogao organizovati na sledeći način.

```
// broj ucitanih elemenata
int i = 0;
while (i < N) {
  // obradjujemo seriju koja pocinje od elementa a[i] i izracunavamo
  // njenu sumu elemenata
  int suma = a[i];
  // citamo naredne elemente sve dok ne stignemo do kraja ili do
  // elementa razlicite parnosti i dodajemo ih na sumu
  int j;
  for (j = i + 1; j < N && isteParnosti(a[i], a[j]); j++)
    suma += a[j];
  // ispisujemo sumu
  cout << suma << endl;
  // naredna serija, ako postoji, pocinje na poziciji j
  i = j;
}
```

Međutim, ako pokušamo da ovo (veoma prirodno i jednostavno) rešenje prilagodimo slučaju kada se brojevi unose jedan po jedan i ne čuvaju svi istovremeno u memoriji, primetićemo da se kraj tekuće serije može ustanoviti tek ako je učitan prvi element naredne serije, što dalje znači da taj broj ne treba čitati ponovo na početku naredne iteracije. Potrebno je, dakle, da nekako rešimo problem ispitivanja vrednosti narednog elementa na ulazu, bez njegovog stvarnog čitanja sa ulaza. Biblioteke programskog jezika ne daju obično direktnu podršku za ovo, tako da je neophodno da je sami isprogramiramo. Jedna mogućnost je da se problem reši tako što će se omogućiti da se

nakon čitanja određeni element vrati nazad na ulaz, dok je druga mogućnost da se obezbedi "gvirkanje" na ulaz tj. provera vrednosti sledećeg elementa koji i dalje ostaje na ulazu. U nastavku ćemo opisati obe.

U prvom rešenju se nakon čitanja broja koji prekida tekuću i započinje novu seriju petlja prekida i taj broj se vraća nazad u ulaz, da bi se onda ponovo pročitao u sledećoj iteraciji, tokom čitanja elemenata naredne serije.

```
while (brojProcitanih < N) {
  int prvi = citaj();
  int suma = prvi;
  while (brojProcitanih < N) {
    int tekuci = citaj();
    if (!isteParnosti(prvi, tekuci)) {
      vrati(tekuci);
      break;
    }
    suma += tekuci;
  }
  // ispisi sumu
}
```

Moguća realizacija ove tehnike koristi pomoćni prihvatnik (bafer) - u ovom slučaju jednu promenljivu koja privremeno čuva broj koji se naredni čita i pomoćnu promenljivu kojom beležimo da li je prihvatnik trenutno pun ili prazan. Funkcija za čitanje se realizuje tako da čita i vraća naredni broj sa ulaza ako je prihvatnik prazan, odnosno broj upisan u prihvatnik, ako je prihvatnik pun (beležeći nakon toga da je prihvatnik ispražnjen). Funkcija kojom broj vraćamo na ulaz, zapravo će broj upisati u prihvatnik, da bi naredni poziv funkcije za čitanje vratio baš taj broj. Uz ovu funkcionalnost, glavni algoritam se može izraziti na sledeći način. U implementaciji koju prikazujemo koristimo globalne promenljive, dok bi se u naprednijim implementacijama ovo moglo izolovati u zaseban modul (na primer, klasu).

U drugom rešenju uvodimo mogućnost "gvirkanja" tj. provere narednog elementa na ulazu bez njegovog stvarnog čitanja. Uz ovakvu funkcionalnost, glavni algoritam se može implementirati na sledeći način:

```
napreduj();
while (brojProcitanih < N) {
  int prvi = citaj();
  int suma = prvi;
  while (brojProcitanih < N) {
    if (!isteParnosti(prvi, gvirni()))
      break;
    suma += citaj();
  }
  // ispisi sumu
}
```

I realizacija ovog rešenja koristi prihvatnik, ali na malo drugačiji način. Pretpostavićemo da se svaki pročitan broj sa ulaza smešta u prihvatnik (na samom početku programa prvi broj čitamo u prihvatnik, funkcijom `napreduj`). Ovo nam omogućava da realizujemo dve različite operacije nad ulazom. Funkcijom `citaj` vršićemo čitanje narednog elementa -

funkcija element preuzima iz prihvatnika, uvećava broj pročitanih elemenata i učitava naredni element u prihvatnik (ako na ulazu ima još elemenata). Sa druge strane, fukcija `gvirni` samo vraća element koji se nalazi u prihvatniku, omogućavajući nam da vidimo koji je naredni element na ulazu, bez njegovog stvarnog čitanja.
Dakle, baferisani ulaz (ulaz sa prihvatnikom) omogućava vraćanje pročitanih elemenata i pregled još nepročitanih elemenata. U rešenjima koje prikazujemo dovoljno je bilo koristiti jednočlane prihvatnike, dok se u praksi nekada koriste i prihvatnici koji mogu istovremeno da čuvaju i više elemenata.
Na kraju, opišimo još jedno rešenje sa dve ugnežđene petlje u kojem se ne koriste posebne tehnike rada sa ulazom. Tokom rada održavaćemo element kojim počinje tekuća serija i njegov redni broj (brojanja ćemo započinjati od nule). Na samom početku programa, pre petlje, učitaćemo prvi element prve serije i njegov redni broj ćemo inicijalizovati na nulu. Spoljna petlja se izvršava sve dok se ne završi sa obradom svih serija, što će biti slučaj kada je redni broj prvog elementa tekuće serije manji od ukupnog broja elemenata. U telu spoljašnje petlje održavaćemo naredni element tekuće serije i njegov redni broj, koji ćemo inicijalizovati na vrednost rednog broja prvog elementa tekuće serije uvećanu za jedan. Unutrašnju petlju ćemo izvršavati sve dok je redni broj narednog elementa manji od ukupnog broja elemenata ili dok se u njenom telu ne ustanovi da je pročitan element koji ne pripada tekućoj seriji. Na početku tela unutrašnje petlje učitavamo naredni element (jer je na osnovu uslova petlje ustanovljeno da on postoji) i proveravamo da li on pripada tekućoj seriji (da li je iste parnosti kao prvi element tekuće serije). Ako ne pripada, unutrašnja petlja se prekida (naredbom `break`), a ako pripada, on se dodaje tekućoj seriji tako što se uvećava njen zbir elemenata i redni broj narednog elementa. Po završetku unutrašnje petlje se ispisuje zbir elemenata tekuće serije, i redni broj prvog elementa naredne serije se postavlja na redni broj narednog elementa. Ako je taj redni broj manji od ukupnog broja elemenata, onda je naredni element već učitan i ustanovljeno je da je različite parnosti od prvog elementa upravo završene serije, tako da prvi element naredne serije postavljamo na njega. Ako se došlo do kraja, ovaj korak je suvišan (naredni element, zapravo nije ni učitan), ali, s obzirom na to da se neće vršiti naredne iteracije spoljašnje petlje, ovaj korak ništa ne kvari.
Primetimo da je ovaj pristup prilično sličan pristupu sa nizom elemenata istovremeno učitanih u memoriju i da promenljiva `i` u tom kodu odgovara rednom broju prvog elementa u seriji, a promenljiva `j` rednom broju narednog elementa u seriji.

```
n = int(input())

niz=[]
for i in range(n):
    x=int(input())
    niz.append(x)
```

```
zbir = niz[0]


for i in range(1,n):
    if niz[i] % 2 == niz[i-1]%2:
        zbir+=niz[i]
    else:
        print(zbir)
        zbir=niz[i]

print(zbir)
```

Dinari i pare

Vreme	memorija	ulaz	izlaz
1 s	64 Mb	standardni ulaz	standardni izlaz

Napiši program koji za dati realni broj dinara određuje odgovarajući ceo broj dinara i ceo broj para.

Ulaz

Sa standardnog ulaza se unosi jedan realan broj zaokružen na dve decimale koji predstavlja iznos novca.

Izlaz

Na standardni izlaz napisati dva cela broja, svaki u posebnom redu, koji predstavljaju broj dinara i broj para.

Primer

Ulaz
123.45
Izlaz
123
45

Rešenje

Broj dinara d se može dobiti tako što se uneti iznos k zaokruži naniže ($d=\lfloor k \rfloor$). Broj para p može se dobiti tako što se od iznosa kkoduzme broj dinara i rezultat pomnoži sa 100 ($p=(k-\lfloor k \rfloor)\cdot 100$).
Zaokruživanje realnog broja naniže vrši se bibliotečkom funkcijom floor tj. Math.Floor.
Međutim, ako se broj dinara i para predstavi celobrojnim promenljivama može doći do neočekivanih rezultata. Naime prilikom svake dodele realne vrednosti celobrojnoj promenljivoj vrši se njena konverzija zaokruživanjem naniže. Iako deluje da vrednost $(k-\lfloor k \rfloor)\cdot 100$ mora biti celobrojna, jer broj k po pretpostavci

ima samo dve decimale, zbog nepreciznosti rada sa brojevima u pokretnom zarezu moguće je da se desi da ta vrednost ponekad bude malo manja nego što je to očekivano (na primer, da umesto vrednosti 45 ima vrednost 44.99999994321). U tom slučaju će se prilikom dodele celobrojnoj promenljivoj dobiti vrednost koja je za jedan manja nego što treba da bude. Da bi se stvar popravila, potrebno je pre dodele celobrojnoj promenljivoj izvršiti zaokruživanje na najbližu celu vrednost, što se može uraditi bibliotečkom funkcijom round tj. Math.Round.
Recimo i da je razdvajanje realnog broja na ceo i razlomljen deo moguće izvršiti i bibliotečkom funkcijom modf.

Resenje 1

```
import math

iznos = float(input())
dinari = math.floor(iznos)
pare = round((iznos - math.floor(iznos)) * 100.0);
print(dinari)
print(pare)
```

Resenje 2

```
import math

iznos = float(input())
dinari = math.floor(iznos)
pare = round((iznos - math.floor(iznos)) * 100.0);
print(dinari)
print(pare)
```

Mali loto

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

U jednom odeljenju su odlučili da u sklopu novogodišnje priredbe organizuju malo izvlačenje igre loto. Da bi povećali šanse za dobitak, odlučili su da se izvlače samo tri kuglice. Napiši program koji ispisuje koje sve kombinacije mogu biti izvučene, ako se zna da u bubnju ima n različitih kuglica obeleženih brojevima od 1 do n.
Ulaz

Sa standardnog ulaza se unosi broj n (4≤n≤20).
Izlaz

Na standardni izlaz ispiši sve kombinacije, pri čemu su brojevi u svakoj kombinaciji sortirani rastuće, a kombinacije su leksikografski sortirane.
Primer
Ulaz
5
Izlaz
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

Rešenje

Već je u tekstu zadatka naglašeno da se u ovom zadatku traži ispisivanje svih *kombinacija* (pošto su u igri loto svi brojevi različiti, radi se o *kombinacijama bez ponavljanja*).
Najjednostavnije rešenje podrazumeva tri ugnežđene petlje, po jednu za svaku od tri loptice. Spoljašnja petlja nabrajaće redom brojeve na prvoj loptici, srednja na drugoj, a unutrašnja na trećoj, pri čemu su loptice sortirane rastući, kako se traži u tekstu zadatka. Stoga brojač srednje petlje uvek mora biti veći od brojača spoljašnje, a brojač unutrašnje petlje uvek mora biti veći od brojača srednje petlje. Zato granice za prvu brojačku promenljivu b1 možemo postaviti na 1 do n, srednju b2 na b1+1 do n, a unutrašnju b3 na b2+1 do n. Zapravo, gornje granice su malo manje (za spoljašnju je to n−2, a srednju n−1), međutim, i za granice n program će raditi korektno, jer će neka od unutrašnjih petlji biti prazna (na primer, za b1=n, b2 kreće od n+1 i traje dok je b2≤n, što je u samom startu narušeno).

```
n=int(input())

for i in range(1,n+1):
    for j in range (i+1,n+1):
        for k in range(j+1,n+1):
            print(i,j,k)
```

Broj kolone u tabeli

Vremensko ograničenje	Memorijsko ograničenje	Ulaz	izlaz
0,5 s	64 MB	standardni ulaz	standardni izlaz

U programima za tabelarna izračunavanja (Microsoft Excel, LibreOffice Calc i slično) kolone su obeležene slovima i to kao A, B, ..., Z, AA, AB, ..., AZ, BA, BB, ..., ZZ, AAA, ... Napiši program koji omogućava konverziju tekstualne oznake kolone u redni broj kolone (od 1 pa naviše).

Ulaz

Sa standardnog ulaza se unosi tekstualna oznaka kolone koja sadrži velika slova engleske abecede (njih najviše 5).

Izlaz

Na standardni izlaz ispisati broj koji odgovara toj koloni.

Primer 1

Ulaz

D

Izlaz

4

Primer 2

Ulaz

AB

Izlaz

28

Primer 3

Ulaz

ZZZZZ

Izlaz

12356630

Ako slovu A pridružimo vrednost 1, slovu B vrednost 2, slovu C vrednost 3 i tako sve do slova Z kojem pridružujemo vrednost 26, možemo smatrati da se ovde radi o zapisu broja u osnovi 26, sa razlikom da se ne koristi nula. Dakle, broj se zapisuje u obliku $a_n \cdot 26^n + a_{n-1} \cdot 26^{n-1} + \ldots + a_1 \cdot 26 + a_0$, pri čemu su sve cifre a_i između 1 i 26. Zaista, kolona AA ima redni broj $1 \cdot 26 + 1 = 27$, dok, na primer, kolona CBA ima redni broj $3 \cdot 26^2 + 2 \cdot 26 + 1 = 2081$. Imajući ovo u vidu, slovnu oznaku kolone možemo prevesti u njen redni broj korišćenjem Hornerove sheme.

Brojevna vrednost svake cifre se može dobiti tako što se od njenog koda (ASCII ili Unicode) oduzme kôd karaktera 'A' i na to doda 1.

U obratnom smeru, malo ćemo modifikovati algoritam određivanja cifara u datoj brojevnoj osnovi koji, podsetimo se, određuje poslednju cifru operacijom ostatka pri deljenju brojevnom osnovom i uklanja je operacijom celobrojnog deljenja.

Redni broj se može razložiti na $a_n \cdot 26^n + a_{n-1} \cdot 26^{n-1} + \ldots + a_1 \cdot 26 + a_0$, pri čemu je vrednosti cifara a_i između 1 i 26. Ako se od broja oduzme 1, poslednja cifra je između 0 i 25 i može se odrediti određivanjem ostatka pri deljenju brojem 26. Ako se na taj ostatak doda kôd karaktera 'A', dobija se kod poslednjeg karaktera u slovnoj oznaci kolone. Određivanjem celobrojnog količnika pri deljenju sa 26 uklanja se poslednja cifra i dobija se broj $a_n \cdot 26^n + a_{n-1} \cdot 26^{n-1} + \ldots + a_1$, pri čemu je vrednost svih cifara a_i između 1 i 26. Ovo je problem iste

strukture kao polazni i određivanje njegovih cifara a_i može se vršiti na isti način (oduzimanjem broja 1, određivanjem ostataka pri deljenju i celobrojnim deljenjem osnovom 26).

Recimo da je izgradnja string uzastopnim dodavanjem karaktera na njegov početak generalno veoma neefikasna operacija (složenost joj je kvadratna). Ipak, pošto se u zadatku radi sa veoma kratkim stringovima, nećemo obraćati pažnju na to.

```
s = input()

br = 1
res=0
for c in s[::-1]:
    res+= (ord(c)-ord('A')+1)*br
    br*=26

print(res)
```