## Stringovi (niske) – bilten rešenja nakon takmičenja Shining star 01

1.

**αλφαβητικό καθρέφτη**

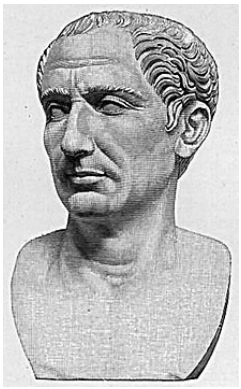
**Katarina je odlučila da svojoj drugarici pošalje šifrovanu poruku, koja sadrži samo slova engleske abecede, cifre i interpunkcijske znake.**
**Svako slovo će šifrovati posebno na osnovu narednih pravila. Mala slova se šifruju velikim slovima tako što se slovo a šifruje slovom Z, slovo b šifruje slovom Y, c slovom X itd., sve do slova y koje se šifruje slovom B i z koje se šifruje slovom A. Velika slova se šifruju potpuno analogno - od A koje se šifruje sa z do Z koje se šifruje sa a. Ostali karakteri se ne menjaju.**
**Ulaz**

**Sa standardnog ulaza unosi se jedna linija teksta, završena karakterom tačka (karakterom .).**
**Izlaz**

**Na standardni izlaz ispisati šifrovani tekst (bez karaktera tačka).**
**Primer**
**Ulaz**
Zdravo svima.

**Izlaz**
aWIZEL HERNZ

Malo istorije:



У криптографији, **Цезарова шифра** је један од најпростијих и најраспрострањенијих начина шифровања. То је тип шифре замењивања у коме се свако слово отвореног текста мења одговарајућим словом азбуке, помереним за одређени број места. На пример, са помаком 3, А се замењује словом Г, Б са Д итд. Овај метод је добио име по Јулију Цезару, који га је користио за размену порука са својим генералима.
*Ако је имао да каже нешто поверљиво, он је то писао шифровано тако што је мењао редослед слова у алфабету и на тај начин постигао да се ниједна реч није могла препознати. Ако би неко то желео да дешифрује и добије значење тога, морао би да замени четврто слово алфабета, дакле* D *са* A *и тако даље за остала.* — Светоније, *Живот Јулија Цезара*
**За размишљање: Који крипто напад предлажете ради дешифровања шифрата добијеног применом Цезарове шифре?**
==========

Rešenje

**Ovaj zadatak rešavamo jednoprolazno: tekst sa ulaza se može učitavati i obrađivati karakter po karakter.**
Dakle, prvi zadatak je da čitamo karaktere jedan po jedan, dok ne dođemo do karaktera '.'. Jedna mogućnost je da učitamo odjednom celu liniju (pomoću getline funkcije, https://en.cppreference.com/w/cpp/string/basic_string/getline).

Ali, druga način učitavanja je da učitamo znake(karaktere) jedan po jedan, u petlji (ciklusu).

Na primer,  takva petlja može da se organizuje tako da se u samom uslovu petlje i pročita karakter i proveri da li je pročitani karakter oznaka kraja reda. U jeziku C++ jedan način je da se napravi petlja oblika

```
char c;
while (cin.get(c) && c != '.') {
   ...
}
```

U telu petlje potrebno je šifrovati svaki karakter. Prvo vršimo klasifikaciju na mala slova, velika slova i ostale karaktera (korišćenjem bibliotečkih funkcija islower, isupper iz zaglavlja cctype u C++, odnosno ctype.h u programskom jeziku C ).

Ako je karakter klasifikovan kao malo slovo, vršimo njegovo šifrovanje i to tako što primetimo da će rastojanje između koda karaktera koji se šifruje od koda karaktera 'a' biti jednako rastojanju između karaktera 'Z' i rezultata. Na primer, ako je u pitanju karakter 'c' njegovo rastojanje od karaktera 'a' je dva, pa je rezultat karakter čiji se kod dobija kada se broj dva oduzme od koda karaktera 'Z' tj. dobija se karakter 'X'. Dakle, mala slova se mogu šifrovati na osnovu veze 'Z' - (c - 'a'), a velika slova na osnovu veze `'z' - (c-'A')`.

**REŠENJE 1**

```
#include <iostream>
#include <cctype>
// zbog funkcija islower, isupper

using namespace std;

int main() {
  char c;
  while(cin.get(c) && c != '.') {
    if (islower(c))   //provera da li je c malo slovo
      cout.put('Z' - (c - 'a'));
    else if (isupper(c))    //provera da li je c veliko slovo
      cout.put('z' - (c – 'A'));   //stampa izmenjenjog karaktera na stdout
    else
      cout.put(c);   //stampa neizmenjenjog karaktera na stdout
  }
  return 0;
}
```

**REŠENJE 2**

```
#include <bits/stdc++.h>

using namespace std;
```

```
int main()
{
    string s;
    getline(cin, s);

   //for(int i=0;i<s.length();i++)
    for(auto c : s){    //c=s[i];
        if(islower(c)or isupper(c))    //ako je c slovo, tj. if isalpha(c)
        {
            cout<<char('a'+'Z'-c);
        }
        else if(c!='.') cout<<c;
    }
    return 0;
}
```

**REŠENJE 3**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    getline(cin, s);

    for (int i=0; i<s.size(); i++)
    {
        if (s[i]>='a' && s[i]<='z')
            s[i]= 'Z' - (s[i] - 'a');
        else if (s[i]>='A' && s[i]<='Z')
            s[i]= 'z' - (s[i] - 'A');
    }

    cout<<s.substr(0, s.size()-1);
    return 0;
}
```

Zadatak 2

**Napisati program kojim se proverava da li je data reč sastavljena isključivo od malih slova palindrom. Reč je palindrom ako se jednako čita slevo na desno i sdesna na levo.**

**Ulaz**

**Prva i jedina linija standardnog ulaza sadrži reč.**

**Izlaz**

**Na standardnom izlazu prikazati reč da ako reč predstavlja palindrom inače prikazati reč ne.**

**Primer**

**Ulaz**

madam

**Izlaz**

da

**Rešenje**

**Linearna pretraga za neodgovarajućim karakterom**

Zadatak se može rešiti tako što se porede parovi karaktera iz datog stringa i to prvi karakter i poslednji, drugi i pretposlednji i tako dalje. Jedan način obilaska parova je pomoću dve promenljive ***i , j*** koje predstavljaju pozicije karaktera koji se upoređuju. Indeks i inicijalizujemo na vrednost 0 (početak stringa), a indeks j inicijalizujemo na vrednost n−1 (kraj stringa), a zatim menjamo indekse tako da se i (indeks prvog elementa u paru) uvećava za 1 u svakom prolazu, a j (indeks drugog elementa) smanjuje za 1, sve dok važi da je i<j. Ovakav izbor parova za poređenje u okviru stringa (sa početka i kraja stringa) se može izvršiti i uz pomoć samo jednog indeksa i, s tim što se onda u ciklusu porede elementi niza sa pozicija i i n−1−i, a i se uvećava dok god je strogo manje od vrednosti ⌊n2⌋ tj. dok je i < n/2.
Provera se zasniva na algoritmu linearne pretrage i među parovima se traži da li postoji neki u kojem su karakteri različiti. Pretragu je moguće implementirati u glavnom programu, kao što je prikazano u 1. rešenju.

Pretragu je moguće implementirati i u potptrogramu palindrom (tj.u sklopu funkcije čija je osnova petlja koja prolazi kroz sve parove karaktera koje treba uporediti) kao što je prikazano u 2. rešenju. Prvi put kada naiđemo na različite karaktere, provera se prekida i funkcija može da vrati false, dok vrednost true vraćamo na kraju funkcije, nakon petlje kojoj su provereni svi parovi i u kojoj je utvrđeno da ne postoji različit par karaktera.

**Bibliotečke funkcije**

Kraći kod (ali ne obavezno i efikasno rešenje) moguće je postići primenom bibliotečkih funkcija. Jedna ideja je da se primeti da je string palindrom ako i samo ako je jednak nizu koji se dobija njegovim obrtanjem. Jedna optimizacija ovog pristupa je da se primeti da nije potrebno porediti ceo string, već samo proveriti da li je njegova prva polovina jednaka obratnoj drugoj polovini (pri čemu se u slučaju neparnog broja karaktera središnji karakter ne uračunava ni u jednu od dve polovine). Postoji nekoliko načina da se obrne niska s u jeziku C++. Jedan je da se napravi njegova kopija t, a onda da se upotrebi funkcija reverse(t.begin(), t.end()), međutim, još bolji način je da se upotrebi konstruktor kojem se prosleđuju reverzni iteratori (oni string obilaze sdesna na levo) tj. string(s.rbegin(), s.rend()). Deo niske (podnisku) možemo izdvojiti metodom substr.

***Obrtanje stringa na načine koje smo prikazali uzrokuje izgradnju novog stringa u memoriji, što je memorijski nepotrebno zahtevno i time su prethodna rešenja zasnovana na bibliotečkoj funkcionalnosti neefikasnija nego ona ručno implementirana.***

U jeziku C++ se problem može elegantno razrešiti ako se upotrebi funkcija equal koja proverava da li su dva segmenta elemenata niza jednaka . Argumenti ove funkcije su iteratori koji određuju prvi segment (iterator na njegov prvi element i iterator koji ukazuje neposredno iza njegovog poslednjeg elementa) i iterator koji ukazuje na prvi element drugog segmenta. Drugi iterator može biti i reverzni, čime se postiže da se prilikom poređenja drugi segment obilazi unazad, s desna na

levo. Time se uslov palindroma svodi na equal(s.begin(), s.end(), s.rbegin()) ili još bolje na equal(s.begin(), next(s.begin(), s.size() / 2), s.rbegin()). Ovo rešenje je veoma efikasno.

1. rešenje

```
#include <iostream>
#include <string>
using namespace std;

int main() {
string s;
cin >> s;
for (int i = 0, j = s.size() - 1; i < j; i++, j--)
if (s[i] != s[j])
{cout << "ne" << endl; return 0;}
cout << "da" << endl;
return 0;
}
```

2. rešenje

```
#include <iostream>
#include <string>
using namespace std;

bool palindrom(string s) {
  for (int i = 0, j = s.size() - 1; i < j; i++, j--)
    if (s[i] != s[j])      return false;

  return true;
}


int main() {
  string s;
  cin >> s;
  if (palindrom(s))
    cout << "da" << endl;
  else

    cout << "ne" << endl;

  return 0;
}
```

3.rešenje

```
#include <iostream>
#include <string>
using namespace std;
```

```
bool palindrom(string s) {
  int n = s.size();
  for(int i = 0; i < n/2; i++)
    if (s[i] != s[n-1-i])
      return false;

  return true;
}

int main() {
  string s;
  cin >> s;
  if (palindrom(s))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

4. rešenje

```
#include <iostream>

#include <string>
#include <algorithm>
using namespace std;


int main() {
  string s;
  cin >> s;
  if (equal(s.begin(), s.end(), s.rbegin()))
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

5. rešenje

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main() {
  string s;
  cin >> s;
  if (equal(s.begin(), next(s.begin(), s.size() / 2), s.rbegin()))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
```

```
}
```

6. rešenje

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main() {
  string s;
  cin >> s;
  if (s == string(s.rbegin(), s.rend()))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

7. rešenje

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main() {
  string s;
  cin >> s;
  if (string(s.begin(), next(s.begin(), s.size()/2)) ==
      string(s.rbegin(), next(s.rbegin(), s.size()/2)))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

---

## Zadatak 03

Žuti mravi žele da odrede površinu livade po kojoj se kreću. Pošto su oni jako mali, dužina i širina livade su izraženi u mikrometrima.

### Ulaz

Sa standardnog ulaza se unose dva cela broja a i b ($0 \leq a,b \leq 4 \cdot 10^9$) koji predstavljaju dimenzije livade u mikrometrima.

### Izlaz

Na standardni izlaz ispiši površinu livade u mikrometrima kvadratnim.

### Primer 1

Ulaz

```
1234567890

3216549870
```

Izlaz
`3971049186085674300`

Primer 2

Ulaz
`4000000000`

`4000000000`

Izlaz
`16000000000000000000`

Rešenje

Algoritam koji treba primeniti u ovom zadatku je jednostavan, jer je potrebno pomnožiti dva cela broja.
Naime, površina pravougaonika je jednaka $P=a \cdot b$.
Ali u ovom zadatku su ulazni brojevi koji se množe veoma veliki (gornje ograničenje ulaznih parametara je $4 \cdot 10^9$). Za njihovo predsstavljanje u memoriji potrebno je koristiti 32-bitni, neoznačeni zapis brojeva
(jer je $2^{31}<4 \cdot 10^9<2^{32}$) i to je moguće uraditi pomoću tipa `unsigned int` u jeziku C++.
Maksimalna vrednost njihovog proizvoda je $16 \cdot 10^{18}$, a nju je moguće ispravno reprezentovati samo pomoću neoznačenih 64-bitnih brojeva - tipa `unsigned long long` u jeziku C++ (označeni 64-bitni brojevi dostižu samo $2^{63}-1$, što je manje od $16 \cdot 1018$).
Jednostavnosti radi, moguće je i ulazne podatke predstaviti 64-bitni neoznačenim brojevima.

```
#include <iostream>
using namespace std;

int main() {
  unsigned long long a, b;
  cin >> a >> b;
  cout << a * b << endl;
  return 0;
}
```

## Zadatak 05

Napisati program kojim se određuje tekst koji se dobija brisanjem iz datog teksta svih pojavljivanja podreči iz datog skupa. Brišu se prvo sva pojavljivanja prve reči, zatim druge, treće i tako do kraja. Taj postupak se ponavlja sve dok je se tekst njime menja. Prilikom brisanja svih pojavljivanja reči postupak se iscrpno ponavlja sve dok je brisanje moguće.

Na primer, za tekst `babrarkadabrabbrr` i skup reči {`br`, `ka`, `aa`}, prvo se iscrpno briše `br` i dobija se `baarkadaa`, zatim se briše `ka` i dobija se `baardaa`, zatim se briše `aa` i dobija se `brd`. Nakon toga se kreće iz početka, iscrpno se briše `br` i dobija `d`, pokušava se sa brisanjem `ka` i `aa` koje ne uspeva, prolazi se kroz treći krug u kojem reč ostaje ista i prijavljuje se rezultat `d`.

# Ulaz

U prvoj liniji standardnog ulaza je tekst, dužine najviše $5 \cdot 10^5$ karaktera iz kojeg se brišu reči koje se učitavaju iz narednih linija. Reči se učitavaju do kraja standardnog ulaza i ima ih najviše $10^5$, a svaka je dužine najviše 10 karaktera.

# Izlaz

Tekst koji se dobija nakon brisanja podreči koje sadrži.

Primer

Ulaz
`babrarkadabrabbrr`

`br`

`ka`

`aa`

Izlaz
`d`

## Rešenje

Ključni element rešenja je potprogram koji uklanja sva pojaviljivanja datog podstringa (podniske) iz datog stringa (niske).

U jeziku C++ na raspolaganju nam stoji metoda `erase` koja kao parametar prima poziciju i broj karaktera i briše podstring koji počinje na toj poziciji i ima taj broj karaktera.
Da bismo obrisali konkretnu podnisku prvo treba da pronađemo njeno pojavljivanje što možemo da uradimo korišćenjem metode `find` .
U petlji vršimo pretragu pamteći pronađenu poziciju sve dok ona ne dobije vrednost `string::npos`. U uslovu petlje istovremeno dodeljujemo vrednost promenljivoj koja čuva poziciju i proveravamo da li je ta dodeljena vrednost različita od `string::npos` . U telu petlje pozivamo funkciju `erase` i vršimo brisanje. Pošto funkcija mora da vrati podatak o tome da li je izvršena neka zamena, uvodimo logičku promenljivu koju pre petlje inicijalizujemo na vrednost `false` (ništa do tada još nije obrisano), a koju u telu petlje (u kome se vrši brisanje) postavljamo na vrednost `true` (nešto jeste obrisano). Na kraju funkcije vraćamo vrednost te promenljive.

U glavnom programu učitavamo liniju teksta koja će se menjati, a zatim i niz podniski koje će biti brisane. Pošto ne znamo koliko ih je, smestićemo ih u dinamički niz (vektor u jeziku C++). Nakon toga, u petlji ćemo prolaziti kroz kolekciju podniski, brisati jednu po jednu (pozivom pomoćne funkcije), čuvajući, pri tom, u logičkoj promenljivoj informaciju da li je došlo do nekog brisanja. Postupak ćemo ponavljati (pomoću spoljašnje petlje) sve dok se ne desi da se prođe kroz celu kolekciju podniski, a da ne dođe ni do jednog brisanja (što ćemo znati na osnovu vrednosti logičke promenljive). Primetimo da je najpogodnija petlja za ovo petlja `do-while`, koja se, kako smo videli, malo ređe koristi od petlji sa proverom uslova na početku (pettlji `while` i njoj veoma srodne petlje `for`).

### 1. rešenje

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// brise sva pojavljivanja stringa s u stringu str i vraca podatak
// o tome da li je nesto obrisano
bool obrisi(string& str, const string& s) {
  size_t poz;
  // u pocetku nije nista obrisano
  bool obrisano = false;
  // dok god uspevamo da pronadjemo podstring s na nekoj poziciji poz
  while ((poz = str.find(s)) != string::npos) {
    // brisemo podstring s sa pozicije poz
    str.erase(poz, s.length());
```

```
    // registrujemo da je nesto obrisano
    obrisano = true;
  }
  // vracamo podatak o tome da li je nesto obrisano
  return obrisano;
}

int main() {
  // ucitava znake u string niska do znaka za kraj linije
  string tekst;
  getline(cin, tekst);

  // ucitavamo sve stringove u niz (vektor)
  vector<string> reci;
  string rec;
  while (cin >> rec)
    reci.push_back(rec);

  // brisemo delove dok god ima promena
  bool obrisano;
  do {
    obrisano = false;
    // brisemo iscrpno pojavljivanja svih podniski redom
    for (auto rec: reci)
      if (obrisi(tekst, rec))
        // belezimo da je nesto bilo obrisano
        obrisano = true;
  } while (obrisano);

  // ispisujemo konacni rezultat
  cout << tekst << endl;
}
```