

# Stringovi

## Objektno programiranje (C++) - 1. vježbe (2. dio)

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,  
Sveučilište u Zagrebu

6. ožujka 2024. godine



## Stringovi

- ▶ uz osnovne tipove (`int`, `char`, ...), C++ pruža mehanizam za definiranje vlastitih tipova
- ▶ biblioteke to koriste za definiranje složenijih tipova
- ⇒ imamo bogate biblioteke **apstraktnih tipova podataka**
- ▶ među najvažnijima **stringovi** (nizovi znakova varijabilne duljine) i **vektori** (nizovi objekata zadanog tipa varijabilne duljine)
- ▶ za pristup elementima stringa i vektora koristimo **iteratore**
- ▶ tipovi string i vektor definirani u biblioteci su apstrakcije ugrađenih **polja** (*array*)

Primjer 1.

```
#include <string>
#include <vector>
...
string s = "abc";
vector<int> v = {1, 2, 3};
```



## Stringovi - Primjer 2. (definicija i inicijalizacija)

```
#include <iostream>
#include <string> ← treba uključiti (iako možda neizravno uključeno)
using namespace std; ← inače za string treba using std::string;
int main() {
    string s1, ← defaultna inicijalizacija na prazan string
        s2 = s1,
        s3 = "pmf",
        s4(10, 'c');
    cout << "Ispis zadanih stringova (sv"
        "aki u svoj red):" << endl;
    cout << s1 << endl << s2 << endl
        << s3 << endl << s4 << endl;
    return 0;
}
```



## Ispis za prethodni primjer

```
Ispis zadanih stringova (svaki u svoj red):
```

```
pmf
cccccccccc
```

Uočite:

- ▶ s2 je **kopija** od s1
- ▶ u s3 iskopirani svi (osim null znaka) iz literala "pmf"
- ▶ s4 sadrži 10 kopija znaka 'c'
- ▶ uzastopni string literali "Ispis zadanih stringova (sv" i "aki u svoj red):" spojeni (konkatenirani) u jedan literal



# Inicijalizacije

- ▶ **direktne** inicijalizacije (bez korištenja =)

```
string s("matka");  
string t(10, 'c');
```



- ▶ inicijalizacije **kopiranjem** (uz korištenje =)

```
string p = "pmf";
```

## Primjer 3. Neizravna inicijalizacija kopiranjem:

```
string s = string(10, 'c');
```

ekvivalentna je stvaranju privremenog objekta za kopiranje:

```
string temp(10, 'c');  
string s = temp;
```



## Primjer 4. (Ostali konstruktori) - samostalno proučiti!

- ▶ Definirajmo dva polja znakova (samo prvo završava s '\0'):

```
const char *cp = "abeceda";  
char neNull[] = {'a', 'b'};
```

- ▶ Ne možemo kopirati drugo polje (jer ne završava s '\0'):

```
string s1(cp);           ✓  
string s2(neNull);      ✗
```

- ▶ No, možemo kopirati prva 2 znaka:

```
string s3(neNull, 2);    ✓
```

Greška ako > 2 znaka!

- ▶ Za stringove, 2 bi značio poziciju od koje se kopira:

```
string s4(s1, 2);        ✓ //kopirali "eceda"  
string s5(s1, 10);      ✗ //out_of_range iznimka
```

- ▶ Pozicija od koje se kopira + koliko znakova:

```
string s6(s1, 2, 4);     ✓ //kopirali "eced"  
string s7(s1, 2, 10);   ✓ //"eceda"
```

- ▶ u posljednjem primjeru, ako je br. znak. prevelik, kop. se do kraja



# Miješanje C++ stringova i C-ovskih stringova

C++ mora imati sučelje prema C programima!

Lippman, Lajoie, Moo, *C++ Primer*, 5. izdanje, str. 122:



Although C++ supports C-style strings, they should not be used by C++ programs. C-style strings are a surprisingly rich source of bugs and are the root cause of many security problems. They're also harder to use!

## ▶ C → C++:

```
string s("niz znakova"); ✓
```

## ▶ obratno nema izravnog načina, nego preko `c_str`:

```
char *str = s; ✗
```

```
const char *str = s.c_str(); ✓
```

## ▶ posljednje vraća samo pokazivač ⇒ promjenom stringa `s` može se dogoditi da ne možemo više koristiti `str` (zato bi za daljnje korištenje trebalo kopirati C-string `str`)



## Operacije na stringovima: `string` IO operatori

### Primjer 5.

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

using namespace std;

int main() {
    string s1, s2;
    cin >> s1 >> s2;
    cout << "Prvi: " << s1 << endl
         << "Drugi: " << s2 << endl;
    return 0;
}
```



# Primjer unosa i ispisa za prethodni kod



```
sehorva@DESKTOP-S92RB8H:~$ ./prog
      neki      tekst
Prvi:  neki
Drugi: tekst
sehorva@DESKTOP-S92RB8H:~$
```

Pri čitanju:

- ▶ zanemaruju se vodeće bjeline
- ▶ čitaju se znakovi do sljedeće bjeline

Operatori vraćaju lijevi operand  $\Rightarrow$  možemo ih ulančavati ✓



## getline funkcija

- ▶ argumenti su istream objekt i string
- ▶ čita do **uključivo** prelaska u novi red (ali sam **prelazak se ne sprema nego odbacuje!**) i to sprema u string
- ▶ ako imamo samo prelazak u novi red, dobiveni string je prazan

**Pitanje.** Što rade sljedeći dijelovi koda:

```
string rijec;
while (cin >> rijec)
    cout << rijec << endl;
```

```
string linija;
while (getline(cin, linija))
    cout << linija << endl;
```



## Primjer 6. Operacije sa stringovima

```
string linija;
while (getline(cin, linija)) {
    if (!linija.empty()) {
        auto d = linija.size();
        string poruka = ": " + linija + '\n';
        cout << "Duljina " << d << poruka;
    }
}
```

- ▶ korištene su sljedeće funkcije članice `string` klase: `empty` (vraća `bool` - je li string prazan) i `size` (vraća duljinu stringa)
- ▶ `d` je tipa `string::size_type` (`auto` ⇒ odredio kompajler)
- ▶ `=` (kopiranje), `+` (konkatencija - bar jedan operand je `string`)
- ▶ usporedbe: `==`, `!=`, `<`, `<=`, `>`, `>=`

## Pitanja: Koje konkatencije su ispravne?

- (1.) `string s1 = "pmf", s2 = "mo";`  
`s1 += s2;`
- (2.) `string s3 = s1 + "-" + s2 + '\n';`
- (3.) `string s4 = "pmf" + "-mo"`
- (4.) `string s5 = s1 + "-" + "mo";`
- (5.) `string s6 = "pmf" + "-" + s2;`



## Primjer 7. (Uspoređivanje stringova)

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

using namespace std;

int main() {
    string s1 = "pmf", s2 = "pmf-bo",
           s3 = "PMF-BO", s4 = "PMF-MO";
    cout << (s1 < s2) << (s2 < s3)
         << (s3 < s4) << endl;
    return 0;
}
```



**Ispis:** 101

**Napomena:** Zbog prioriteta su zagrade u gornjem primjeru nužne



## Rad s pojedinim znakovima u stringu

► biblioteka *naziv.c* iz C-a može se koristiti u C++-u kao *cnaziv*

⇒ C: `#include<ctype.h>` ⇔ C++: `#include<ctype>`

Kratki pregled `cctype` funkcija:

<code>isalnum(c)</code>	true akko je c slovo ili znamenka
<code>isalpha(c)</code>	true akko je c slovo
<code>iscntrl(c)</code>	true akko je c kontrolni znak
<code>isdigit(c)</code>	true akko je c znamenka
<code>isgraph(c)</code>	true akko je c nije razmak ali je ispisiv
<code>islower(c)</code>	true akko je c malo slovo
<code>isprint(c)</code>	true akko je c ispisiv
<code>ispunct(c)</code>	true akko je c interpunkcija
<code>isspace(c)</code>	true akko je c bjelina
<code>isupper(c)</code>	true akko je c veliko slovo
<code>isxdigit(c)</code>	true akko je c heksadecimalna znamenka
<code>tolower(c)</code>	samo ako je c veliko slovo vraća malo slovo
<code>toupper(c)</code>	samo ako je c malo slovo vraća veliko slovo



## Zadatak 1.

U stringu sva mala slova zamijeniti velikima i ispisati broj promijenjenih znakova.

- ▶ pojedinom elementu stringa možemo pristupiti pomoću **indeksa** (npr. prvi znak stringa `s` je `s[0]`) ili pomoću **iteratora** (kasnije...)
- ▶ **Range-Based for** petlja - prolazi svim elementima niza:
  - ▶ `for ( varijabla za pristup elementima : niz )`  
...nešto radimo s tim elementima...

**Primjer 8.** Ispis svih znakova stringa:

```
string s = "Pmf-mo";
for(auto c : s){
    cout << c << endl;
}
```

- ▶ `auto` ⇒ kompajler će odrediti da je tip od `c` ovdje `char`



## Prvi pokušaj rješenja zadatka 1.

```
string s = "Pmf-mo";
decltype(s.size()) br = 0;
for(auto c : s){
    if(islower(c))
        br += 1;
    c = toupper(c);
}
cout << "Ukupno promijenjeno " << br
     << " znakova: " << s << endl;
```

- ▶ tip od `br` je povratni tip od `s.size` (tj. `string::size_type`)
- ▶ **String `s` nije promijenjen!** (Zašto?)





## Popravak: korištenje reference

```
for(auto &c : s) {  
    if(islower(c))  
        br += 1;  
    c = toupper(c);  
}
```

- ▶ **referenca** = drugo ime za objekt
- ▶ koristimo referencu kao kontrolnu varijablu - vezana uz svaki element niza (možemo mijenjati taj element)

## Kratko o referencama i pokazivačima

- ▶ **referenca** = drugo ime za objekt
  - ⇒ nije objekt
  - ⇒ mora se inicijalizirati (ne literalom!) i tipovi se moraju podudarati
  - ⇒ ostaje do kraja vezana za objekt

**Primjer 9.** Odredite jesu li sljedeće naredbe ispravne:

```
int br = 123;  
int &r1 = br; ✓  
int &r2; ✗  
r1 = 5; //mijenjamo br ✓  
double &r3 = br; ✗  
int &r4 = 10; ✗
```

- ▶ **pokazivači** (pointeri) su objekti, ne moraju biti inicijalizirani i mogu se mijenjati tako da pokazuju na druge objekte

**Primjer 10.**

```
int *p1 = &br, p2;
```

← p2 nije pokazivač!

## Operacije za modificiranje stringova

`s.insert(p, n, c)`

- ▶ ubacuje  $n$  znakova  $c$  u string  $s$  prije indeksa  $p$
- ▶ razne mogućnosti - primjerice, `s.insert(p, str)` - ubacivanje stringa  $str$

### Primjer 11.

```
string s = "pmfmo";
s.insert(3, 5, '+'); //pmf++++mo
s.erase(3, 5); //pmfmo
s.insert(3, " - "); //pmf - mo
s.append(" 2024"); //pmf - mo 2024
```

`s.erase(p, n)`

- ▶ briše  $n$  znakova u stringu  $s$  počevši od indeksa  $p$

`s.append(str)`

- ▶ dodavanje na kraj stringa  $s$



## Operacije za modificiranje stringova - replace

### Primjer 12.

```
string s = "Knjiga - 1. izdanje";
s.replace(9, 2, "drugo");
cout << s << endl;
```

- ▶ počevši od indeksa 9 prvo obrišemo dva znaka, a zatim se od indeksa 9 ubacuje zadani string
- ▶ ispis: Knjiga - **drugo** izdanje



# Podstringovi - **substr** operacija



## **s.substr(p, n)**

- ▶ vraća `string` koji sadrži **n znakova** iz `stringa s` počevši od **indeksa p** (koji ne smije biti veći od veličine `stringa`)
- ▶ *default* za **p** je **0**, a *default* za **n** je ona vrijednost za koju se kopiraju svi znakovi **do kraja** `stringa` (isto dobivamo za **n** koji je veći od duljine `stringa`)

## Primjer 13.

```
string s1("praktikum dva");  
string s2 = s1.substr(0, 5); // "prakt"  
string s3 = s1.substr(6); // "kum dva"  
string s4 = s1.substr(6, 15); // "kum dva"  
string s5 = s1.substr(20); X // out_of_range
```



## Operacije za **pretraživanje** stringova

- ▶ sve imaju povratni tip `string::size_type` - indeks gdje smo pronašli traženo ili `string::npos`<sup>1</sup>
- ▶ `find` vraća indeks **prvog** podudaranja
- ▶ `find_first_of` - indeks prvog znaka iz zadanog `stringa`
- ▶ prvi koji nije iz zadanog `stringa` - `find_first_not_of`
- ▶ verzije traženja unatrag: `rfind`, `find_last_of`, `find_last_not_of`
- ▶ opcionalan drugi argument predstavlja poziciju od koje počinje potraga

<sup>1</sup>Konstanta inicijalizirana na **-1**, no to je `unsigned` ⇒ zapravo je jednako najvećoj mogućoj veličini nekog `stringa`. Zato je loše koristiti neki `signed` tip za povratnu vrijednost (primjerice, tip `int`).



## Primjer 14. Za zadane stringove, što bi se ispisalo?

```
string s1("abcAbcAbc"), brojevi("0123456789"),  
      s2("r2d3"), s3("03a714p3");  
auto p = s1.find("Abc");
```

<code>cout &lt;&lt; ... &lt;&lt; endl;</code>	ispis
<code>p</code>	3
<code>s2.find_first_of(brojevi)</code>	1
<code>s3.find_first_not_of(brojevi)</code>	2
<code>s2.find_last_of(brojevi)</code>	3
<code>s3.find_last_not_of(brojevi)</code>	6
<code>s1.find("aBc")</code>	18446744073709551615
<code>s1.rfind("bc")</code>	7
<code>s1.find("bc",2)</code>	4
<code>s3.find_last_not_of(brojevi,4)</code>	2



## Numeričke konverzije

- ▶ `to_string(a)` - vraća `string` koji predstavlja broj `a`
- ▶ `stoi[?](s, p, b)` - vraća početni podstring od `s` kao broj
  - ▶ broj tipa `[?]` = `i`, `l`, `ul`, `ll`, `ull` (int, long, unsigned long, ...)
  - ▶ `p` = pokazivač na `size_t` objekt u koji se sprema indeks prvog nenumeričkog znaka u `s` (default je 0 → nema spremanja)
  - ▶ `b` = baza (default je 10)
- ▶ `stof[?](s, p)`, `[?]` = `f`, `d`, `ld` - za float, double i long double

**Primjer 15.** Dobiveni broj se ne može prikazati ⇒ `out_of_range`;  
ako se ne može pretvoriti u broj ⇒ `invalid_argument`.

```
cout << stof("-0.34e5");    ✓    //-34000  
cout << stof("-0.a");      ✓    //-0  
cout << stof("0x46");     ✓    //70  
cout << stof("1.23e100"); ✗
```

```
sehorva@DESKTOP-S92RB8H:~$ ./prog  
terminate called after throwing an instance of 'std::out_of_range'  
what(): stof  
Aborted (core dumped)
```



## Primjer 16. Što sljedeći dio koda ispiše?

```
int a = 123;
string s1("234.12m"), s2("pi = 3.14");
size_t p;
cout << to_string(a) << endl
      << stoi(s1,&p) << endl
      << p << endl
      << stoi(s1,&p,5) << endl
      << stod(s1,&p) << endl
      << p << endl
      << stod(s2.substr(s2.find_first_of
                      ("+- .0123456789"))) << endl;
```

U svaki red posebno: 123, 234, 3, 69, 234.12, 6, 3.14



## Zadatak 2.

Napišite program koji od korisnika učitava jednu liniju teksta. Ispišite poruku nalazi li se u toj liniji riječ paprika. Ukoliko se nalazi, ispišite i koliko puta se nalazi.

### Primjeri unosa i ispisa.

- ▶ Jane je na Dolcu kupila tri paprike.  
→ Ispis: paprika se ne javlja u unosu :(
- ▶ Alocirana paprika naziva se punjena paprika, a ne samo paprika.  
→ Ispis: paprika se javlja 3 puta



## Zadatak 3. (Riješite bez upotrebe regularnih izraza!)

Trgovac John je u liniju teksta upisao podatke o nekom proizvodu i točno jednu cijenu. Neposredno nakon cijene (i samo tamo!) piše HRK. Ispišite cijenu proizvoda (u EUR).<sup>2</sup>

### Primjeri unosa i ispisa.

- ▶ Slanutak u konzervi 11.99HRK za 240g.  
→ Ispis: Cijena je 1.59 EUR.
- ▶ Sok 100% jabuka 1L 15.74HRK  
→ Ispis: Cijena je 2.09 EUR.
- ▶ 7.53HRK je cijena beskvasnog kruha.  
→ Ispis: Cijena je 1.00 EUR.
- ▶ Cijena karte je 50HRK.  
→ Ispis: Cijena je 6.64 EUR.

**Napomena.** Za ispis broja *a* na točno dvije decimale (uz upotrebu `#include <iomanip>`):

```
cout << setprecision (2) << fixed << a;
```

---

<sup>2</sup>Fiksni tečaj konverzije: 7,53450 kuna za 1 euro.