

# Regularni izrazi - dva primjera

## Objektno programiranje - 5. vježbe

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,  
Sveučilište u Zagrebu

3. travnja 2024. godine



## Prvi primjer: Popis najpopularnijih filmova

- ▶ popis se nalazi na stranici: [www.imdb.com/chart/moviemeter](http://www.imdb.com/chart/moviemeter)
- ▶ spremi stranicu kao `filmovi.html` datoteku (u istu mapu s `.cpp` datotekom koja će sadržavati kod koji ćemo napisati)

### IMDb Charts

# Most Popular Movies

As determined by IMDb users

100 Titles



Natrag	Alt + Strelica lijevo
Proslijedi	Alt + Strelica desno
Ponovo učitaj	Ctrl + R
<b>Spremi kao...</b>	Ctrl + S
Ispis...	Ctrl + P



## Gdje se nalaze imena filmova s popisa?

- ▶ u datoteci `filmovi.html` (ako ju otvorimo s nekim uređivačem teksta), možemo pronaći da se svaki naziv filma s popisa nalazi ovako:

```
"titleText":{"text":" NAZIV FILMA "}
```

- ▶ primjerice, u datoteci<sup>1</sup> imamo:

```
... "titleText":{"text":"Kung Fu Panda 4", ...
```



<sup>1</sup>Za stranicu preuzetu na dan 2. travnja 2024. godine. 3/28

## Učitavanje sadržaja datoteke `filmovi.html` u string

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

using namespace std;

int main() {
    ifstream dat("filmovi.html");
    string ulaz, linija;
    while(getline(dat, linija))
        ulaz += linija + '\n';
    dat.close();

    // ...tu ide kod sa sljedećih slajdova...

    return 0;
}
```



## Regularni izrazi

- ▶ grubo govoreći, to je način opisa niza znakova koji nas zanima
- ▶ korisimo klasu **regex** - treba nam **regex** zaglavlje:

```
#include<regex>
```

- ▶ regularni izraz koji tražimo spremit ćemo u string kojim zatim inicijaliziramo **regex** objekt:

```
string obrazac =  
    "\"titleText\":\\{\"text\": \"([^\"]*)\"";  
regex regIzraz(obrazac);
```

- ▶ po *defaultu*, za regularne izraze koristi se **ECMAScript** jezik<sup>2</sup>



<sup>2</sup>ECMA = European Computer Manufacturers Association

## Ukratko o prethodnom regularnom izrazu

- ▶ želimo prepoznati:

```
"titleText": {"text": "NAZIV FILMA" }
```

- ▶ no, ne možemo napisati ovako:

```
"titleText": {"text": "NAZIV FILMA" }
```

- ▶ kao prvo, zagrada { ima drugu ulogu, pa za prepoznavanje znaka { moramo ispred dodati \
- ▶ zatim, ne želimo doslovno prepoznati tekst NAZIV FILMA, nego bilo što nakon čega slijedi znak "
- ▶ za bilo koji znak osim " koristimo [^"] - ako želimo proizvoljan broj takvih znakova (možda i niti jedan takav), koristimo \*
- ▶ dobili smo: "titleText": \{ "text": " ([^"]\*) "
- ▶ kako bi u C++ stringu koristili znakove \i " moramo ispred njih dodati (još jedan) znak \
- ▶ time konačno dobivamo:

```
\ "titleText" \": \\ \{ \ "text" \": \ " ([^\ " ]*) \ " "
```



## regex\_search za traženje podudaranja

- ▶ **regex\_search** vraća podataka tipa `bool` - je li regularni izraz prepoznao neki podniz u danom stringu (za podudaranje s cijelim stringom se na isti način koristi `regex_match`)
- ▶ dajemo i jedan objekt tipa **smatch** - sadrži detalje o (**prvom!**) podudaranju (duljinu, poziciju, prefiks i sufiks, tj. sve prije i poslije podudaranja u stringu itd.)
- ▶ **str** metoda objekta vraća podudaranje kao string

```
smatch rezultat;  
if(regex_search(ulaz, rezultat, regIzraz))  
    cout << rezultat.str() << endl;
```

- ▶ ovdje u stringu `ulaz` tražimo podudaranje s `regIzraz` i, ako ga pronađemo, ispišemo ga (kao string)
- ▶ primjer ispisa:

```
"titleText":{"text":"Road House"}
```



## Opcije za regex objekt

- ▶ ako u stringu `obrazac` umjesto `title` stavimo `Title`, neće doći do ispisa
- ▶ ako želimo ignorirati razliku velikih i malih slova, možemo iskoristiti zastavicu (*flag*) `regex::icase`
- ▶ uz sljedeću promjenu ponovo dobivamo ispis:

```
regex regIzraz(obrazac, regex::icase);
```



## Što ako u regularnom izrazu napravimo grešku?

- ▶ C++ kompajler neće provjeriti sintaksu regularnog izraza
- ▶ ukoliko imamo sintaksnu grešku u izrazu, pri pokretanju će se baciti iznimka tipa **regex\_error** (pomoću `what` metode, kao i kod ostalih iznimaka, dobivamo opis nastale greške)
- ▶ primjer - iz našeg izraza uklonimo znak `)`:

```
"\"TitleText\":\\{\"text\": \" ([^\" ])*\""
```

(nastavak primjera je na idućem slajdu...)



## Nastavak primjera s prethodnog slajda

- ▶ sljedeći kod daje ispis `Parenthesis is not closed.`

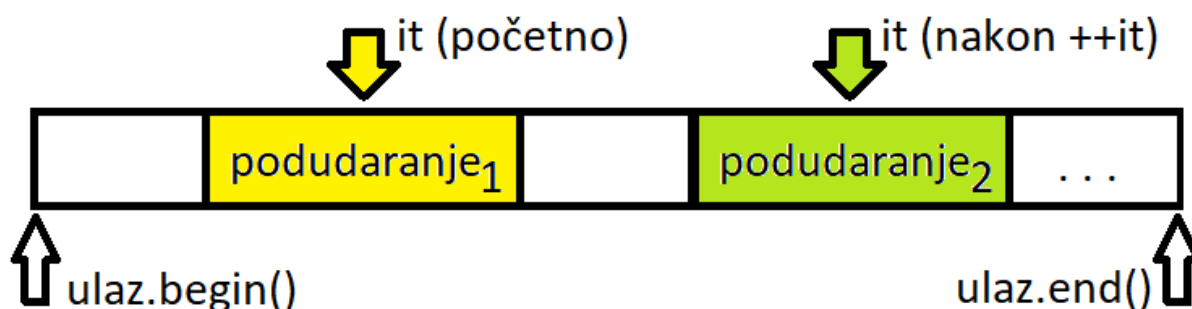
```
try {
    regex regIzraz(obrazac, regex::icase);
    smatch rezultat;
    if (regex_search(ulaz, rezultat, regIzraz))
        cout << rezultat.str() << endl;
} catch (regex_error e) {
    cout << e.what() << endl;
}
```

**VAŽNO:** Kako bi daljnji kod ispravno radio, potrebno je vratiti znak zagrade `)` koji smo maloprije uklonili iz našeg regularnog izraza!



## Kako pronaći sva podudaranja za naš regularni izraz?

- ▶ pomoću trenutnog koda možemo naći samo prvo podudaranje
- ▶ sva podudaranja nalazimo pomoću `sregex_iterator` adaptera za iterator
  - ▶ za dani tekst i izraz, nalazi prvo podudaranje
  - ▶ dereferenciranjem dobivamo odgovarajući `smatch` objekt
  - ▶ inkrementiranjem pronalazimo iduće podudaranje
  - ▶ „prazan” iterator (`end_it` u sljedećem primjeru) ponaša se kao „jedan-iza-zadnjeg” iterator



Ilustracija za `sregex_iterator`

```
it (ulaz.begin(), ulaz.end(), regIzraz);
```



## Pronalaz svih podudaranja u našem primjeru

- ▶ u našem primjeru zamijenimo `if(...)` ... - dio koji smo imali sa sljedećim kodom:

```
sregex_iterator end_it;
for (sregex_iterator it (ulaz.begin(),
    ulaz.end(), regIzraz); it != end_it; ++it)
    cout << it->str() << endl;
```



## Primjer ispisa dobiven prethodnim kodom

```
"titleText":{"text":"Road House"}
"titleText":{"text":"Dina: Drugi dio"}
"titleText":{"text":"Uboga stvorenja"}
"titleText":{"text":"Damsel"}
"titleText":{"text":"Dina"}
"titleText":{"text":"Ghostbusters: Frozen Empire"}
"titleText":{"text":"Alien: Romulus"}
"titleText":{"text":"Beetlejuice Beetlejuice"}
"titleText":{"text":"Oppenheimer"}
"titleText":{"text":"Madame Web"}
```

- ▶ želimo „izvaditi” iz tih podudaranja samo nazive filmova, tj. sljedeće dijelove između dvostrukih navodnika: Road House, Dina: Drugi dio, Uboga stvorenja, Damsel itd.



## Upotreba podizraza (*subexpressions*)

- ▶ podizrazi predstavljaju dijelove regularnog izraza u zagradama ()
- ▶ u našem regularnom izrazu:

```
"\"TitleText\":\\{\\\"text\":\\\"([^\"]*)\\\"}
```

imamo sljedeći podizraz:

```
[^\"]*
```

- ▶ ukoliko je bilo podudaranja, dobiveni `smatch` objekt daje pristup i svakom podizrazu za to podudaranje - to dobivamo pomoću `str` i nekog od brojeva 1, 2, 3, ... za prvi, drugi, treći, ... podizraz (`str(0)` je isto kao `str()`)
- ▶ podizraz koji odgovara nazivu filma koji tražimo je prvi po redu pa koristimo `str(1)`



# Konačno rješenje

- ▶ također smo dodali ispis rednog broja filma i malo poravnanja (pa trebamo `#include <iomanip>`):

```
int br = 0;
for(... kao prije ...)
    cout << setw(3) << ++br << ".  "
         << it->str(1) << endl;
```

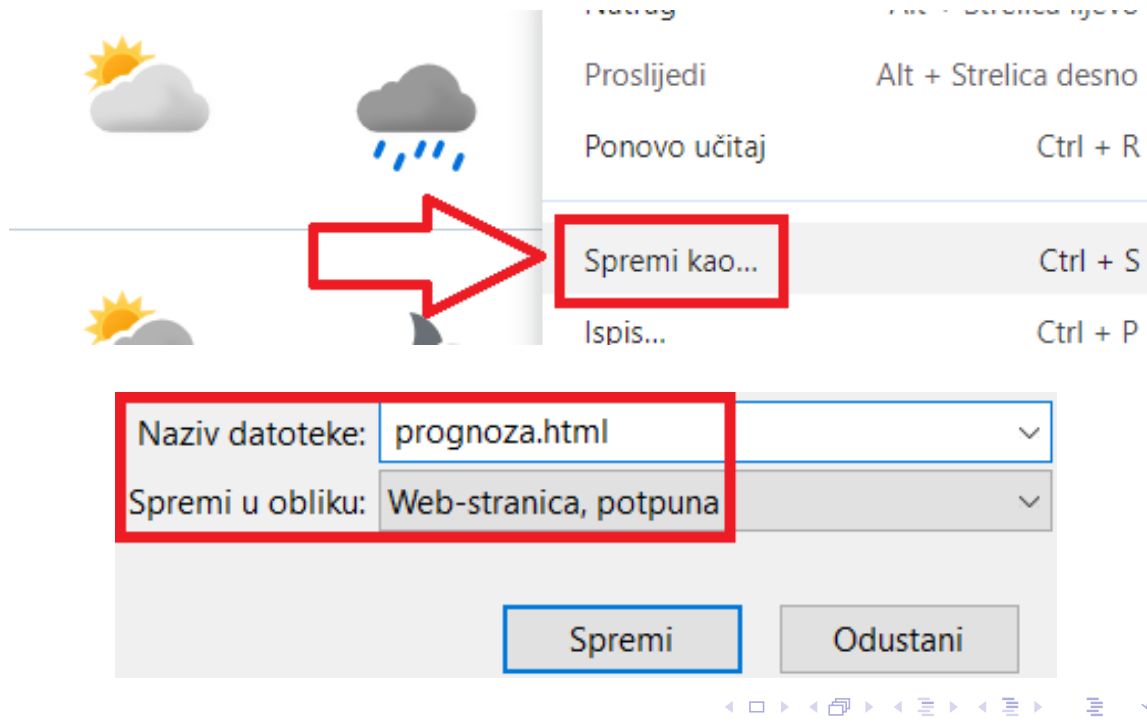
## Dobiven ispis

1. Road House
2. Dina: Drugi dio
3. Uboga stvorenja
4. Damsel
5. Dina
6. Ghostbusters: Frozen Empire
7. Alien: Romulus
8. Beetlejuice Beetlejuice
9. Oppenheimer
10. Madame Web
- ...
100. Kungfu panda



## Drugi primjer: Datumi u vremenskoj prognozi

- ▶ otići na [www.yr.no/en](http://www.yr.no/en)
- ▶ pronaći prognozu za odabrani grad (primjerice, Zagreb) i spremiti tu stranicu kao prognoza.html (odabrati spremanje potpune web-stranice)



## Što želimo napraviti?

- ▶ želimo datume poput Today 1 Apr. zamijeniti datumima poput 01.04.2024.
- ▶ primjer - iz sljedećeg:

	Night	Morning	Afternoon	Even
Today 1 Apr.				
Tuesday 2 Apr.				
Wednesday 3 Apr.				

želimo dobiti:

	Night	Morning	Afternoon	Even
01.04.2024.				
02.04.2024.				
03.04.2024.				

## Gdje se nalaze datumi koji nas zanimaju?

- ▶ pregledom datoteke `prognoza.html` možemo uočiti da datumima na web-stranici poput `Today 1 Apr.`

	Night	Morning	Afternoon	Even
Today 1 Apr.				
Tuesday 2 Apr.				
Wednesday 3 Apr.				

odgovaraju dijelovi u `.html` datoteci poput

```
<time datetime="2024-04-01">Today 1 Apr.</time>
```

- ▶ plan: napraviti novu datoteku u kojoj je tekst kao u datoteci `prognoza.html`, ali tako da je tekst unutar tzv. *time taga* (oznake) **zamijenjen** datumom koji je dobiven iz vrijednosti atributa `datetime`



## Prvi dio koda

- ▶ iz datoteke `prognoza.html` prebacivat ćemo sadržaj u (novu) datoteku `prognoza-hr.html` (uz poneku zamjenu)

```
#include <iostream>
#include <fstream>
#include <string>
#include <regex>

using namespace std;

int main() {
    ifstream ulaz("prognoza.html");
    ofstream izlaz("prognoza-hr.html");

    // ... nastavak na sljedećim slajdovima ...

    ulaz.close();
    izlaz.close();
    return 0;
}
```



# Kako izgleda naš regularni izraz?

```
<time datetime="2024-04-01">Today 1 Apr.</time>
```

- ▶ regularni izraz koji ćemo koristiti:

```
(<time datetime="\d{4}-\d{2}-\d{2}">)([<]*)(</time>)
```

- ▶ string (dodani su potrebni znakovi '\') kojim ćemo ga zapisati u kodu (kako bi stao na ekran, razlomljen je u dva dijela):

```
" (<time datetime= \" (\d{4}) "  
"- (\d{2}) - (\d{2}) \"> ) ([<]* ) (</time> ) "
```



## regex\_replace

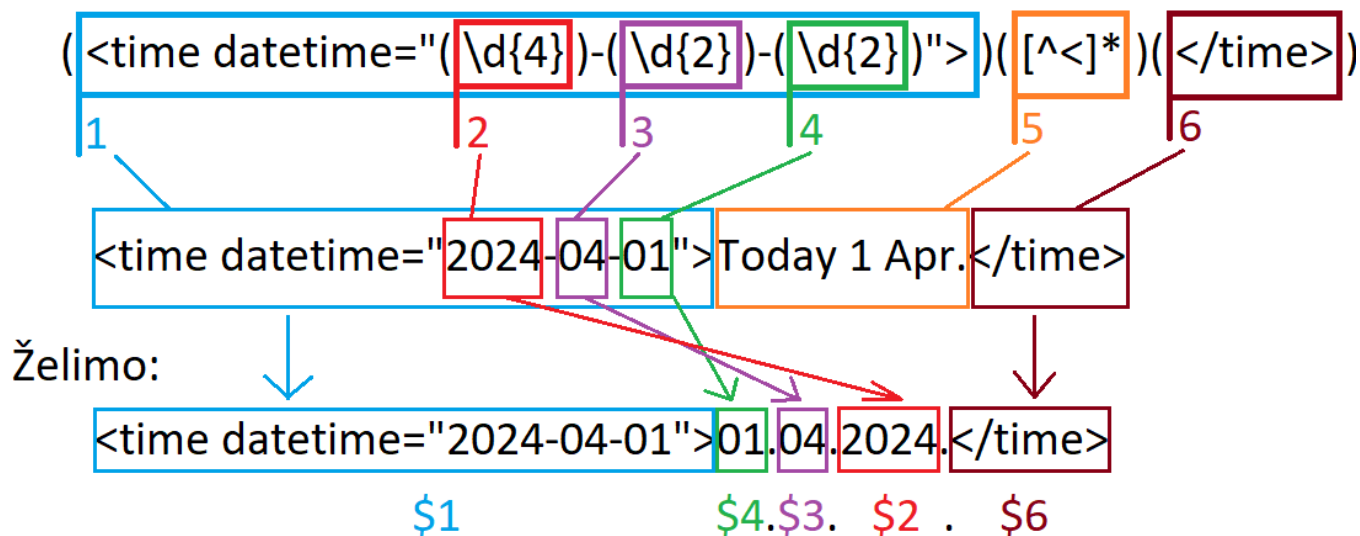
- ▶ kao i `regex_search` prima ulazni niz znakova i regularni izraz (`regex` objekt)
- ▶ navodimo i `string` koji predstavlja izraz kojim želimo zamijeniti **svako** pronađeno podudaranje
- ▶ u tom „zamjenskom” izrazu možemo navesti i podizraze iz pronađenog podudaranja - na određeni podizraz referiramo se navođenjem broja (nakon simbola `$`) koji predstavlja njegov redni broj u podudaranju
- ▶ u našem slučaju tražni „zamjenski” izraz je sljedeći:

```
$1$4.$3.$2.$6
```

- ▶ objašnjenje se nalazi na sljedećem slajdu...



## Objašnjenje za izraz s prethodnog slajda

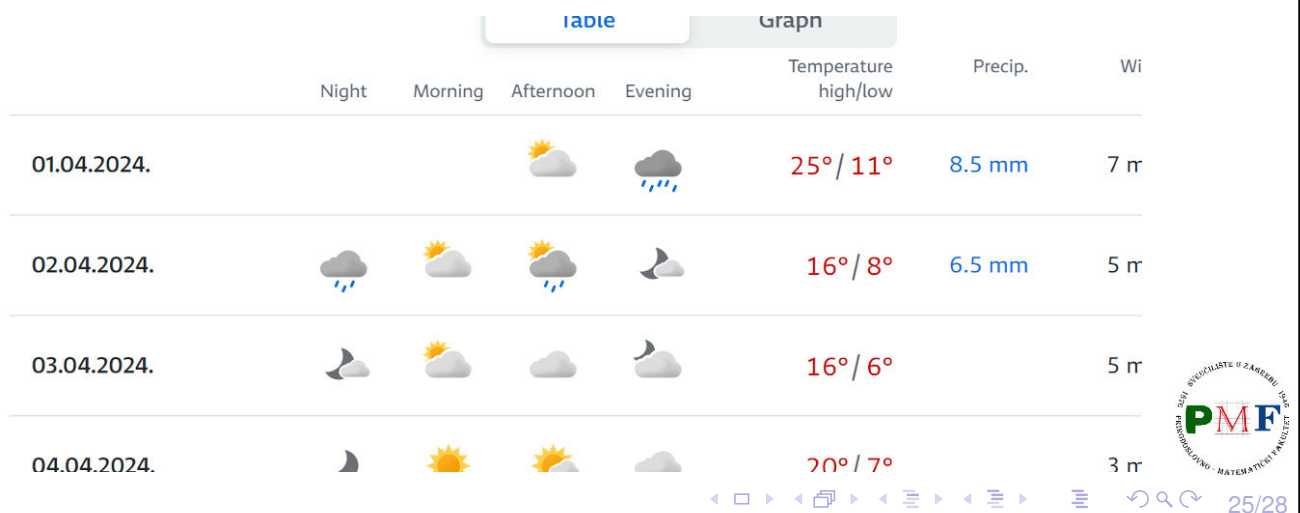


## Drugi dio koda

```
string obrazac = "<time datetime=\\\"(\\d{4}) \"  
    \"-(\\d{2})-(\\d{2})\\\"> ([^\<]*) (</time>)\";  
regex regIzraz(obrazac);  
string format = \"$1$4.$3.$2.$6\";  
  
string linija;  
while(getline(ulaz, linija)) {  
    izlaz << regex_replace(linija, regIzraz, format)  
    << \"\n\";  
}
```

## Testiranje napisanog koda

- ▶ nakon kompiliranja i pokretanja našeg programa, pronaći u istoj mapi upravo stvorenu datoteku **prognoza-hr.html** (ta datoteka mora biti u istoj mapi kao i datoteka `prognoza.html` s obzirom da smo s njom, zbog odabira spremanja potpune web-stranice, preuzeli još i hrpu datoteka koje utječu na izgled pri otvaranju te datoteke s nekim web-preglednikom poput *Chromea*, *Firefox* i sl.)
- ▶ primjer (otvoreno pomoću programa *Chrome*):



## Preusmjeravanje `clog-a`

- ▶ u prethodni kod bismo mogli dodati poruke o tome što pokrenuti program trenutno radi, poput:

```
clog << "Otvaranje datoteke \"prognoza.html\"..."  
    << endl;
```

- ▶ tako dobivamo ispis na ekran:

```
...$ ./prog  
Otvaranje datoteke "prognoza.html"..  
Otvaranje datoteke "prognoza-hr.html"..  
Zamjena datuma iz datoteke..  
Zatvaranje datoteka..  
Kraj programa.
```



## Upotreba `rdbuf`

- ▶ `rdbuf` metoda *streama* bez argumenata vraća pokazivač na *stream buffer* objekt povezan s tim *streamom*
- ▶ `rdbuf` metoda s jednim argumentom čisti *error* zastavice tog *streama* i postavlja njegov *stream buffer* na onaj na koji pokazuje dani argument
- ▶ možemo u kodu otvoriti neku datoteku (npr. "log.txt"), spremiti pokazivač na *buffer* koji `clog` ima, zatim postaviti da `clog` upisuje u tu otvorenu datoteku, a onda na kraju `clog`-u vratiti *buffer* koji je imao na početku (i naravno zatvoriti datoteku koju smo otvorili) → kod je na sljedećem slajdu...



## Promjene u `prognoza.cpp` (unutar `main` funkcije)

```
ofstream novilog("log.txt");
stringstream *backup = clog.rdbuf();
clog.rdbuf(novilog.rdbuf());

clog << "Otvaranje datoteke \"prognoza.html\"..."
    << endl; // ispis u datoteku log.txt

...

clog.rdbuf(backup);
novilog.close();

return 0;
```

