



# **Tema 11**

## **Tokovi podataka, ulaz-izlaz i rad s fajlovima u jeziku C++**

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



# Sadržaj

1. Uvod
2. Pojam toka podataka
3. Rad s fajlovima
4. Formatiranje ulaza i izlaza
5. Formatiranje pomoću funkcija klase ios
6. Upotreba ulazno-izlaznih manipulatora
7. Modul <format>
8. Primeri programa



# 1. Uvod

- Ulaz-izlaz podataka u jeziku C++
- Formatiranje podataka u ulazno-izlaznim operacijama



# Ulaz-izlaz podataka u jeziku C++

- Ulaz i izlaz podataka može biti npr.
  - čitanje i upis podataka u tekstualne datoteke ili bazu podataka
  - prikaz podataka u grafičkom prozoru aplikacije
  - komunikacija putem mreže
- Apstrakcija ulazno-izlaznih uređaja u jeziku C++ omogućava *jedinstveni interfejs* ovih operacija
- Jezik C++ nema ugrađene naredbe za ulaz i izlaz podataka, već se oslanja na ulazno-izlazne operacije iz biblioteke klasa, koje su opisane u zaglavlju **<iostream>** i dostupne kroz imenik std
  - stariji format biblioteke, opisan u zaglavlju [iostream.h](#), bio je dostupan u globalnom imeniku

# Formatiranje podataka u ulazno-izlaznim operacijama

- Biblioteka klasa **<iostream>** omogućava formatirani upis i čitanje (tekstualnih) podataka
- Format podataka u ulazno-izlaznim operacijama može se precizno definisati na više načina
  1. Pomoću *flegova (flags)* i drugih vrednosti pomoću kojih se upravlja formatiranjem podataka određenog toka
  2. Pomoću posebnih *manipulatorskih funkcija* prilikom operacija umetanja i izdvajanja (>> i <<)
  3. Pomoću *regularnih izraza* prilikom čitanja (parsiranja) podataka
  4. Pomoću modula **<format>** koji je uveden u verziji C++20

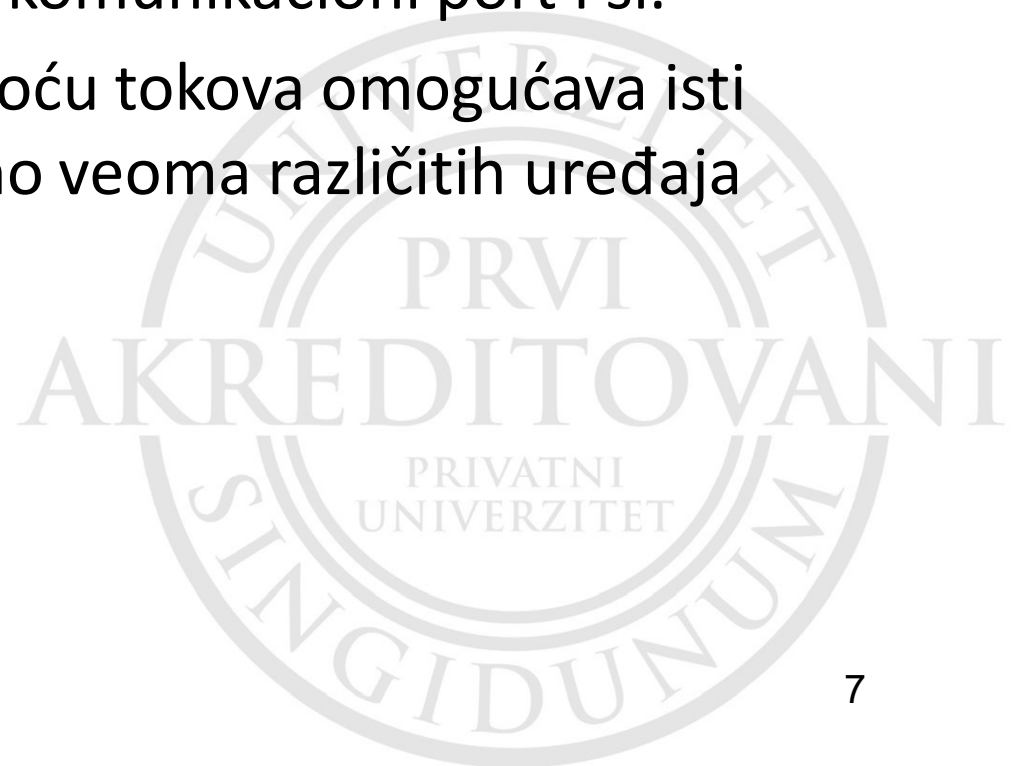
## 2. Pojam toka podataka

1. Pojam toka
2. Vrste tokova
3. Predefinisani tokovi
4. Klase tokova



## 2.1 Pojam toka

- Tok podataka (*stream*) je pogodna apstrakcija ulaznih ili izlaznih uređaja sistema, koja omogućava isti način rada s različitim fizičkim uređajima
- Tok je *logički interfejs* za datoteke, koje fizički mogu biti npr. disk, tastatura, ekran računara, komunikacioni port i sl.
- Jedinstveni prikaz uređaja pomoću tokova omogućava isti način programiranja međusobno veoma različitih uređaja



## 2.2 Vrste tokova

- Osnovne kategorije tokova definisane su u odnosu na tip podataka, koji mogu biti *binarni* i *tekstualni*
- **Binarni** podaci se zapisuju unutar računara u binarnom obliku bez ikakve transformacije i ljudima nisu direktno čitljivi
- **Tekstualni** podaci su nizovi znakova, ljudima direktno čitljivi, koji se u memoriji računara zapisuju u nekom binarnom formatu, npr. Unicode ili ASCII kodu
  - binarni tok **10100111** može da predstavlja decimalni broj **167** tipa **int**
  - tekstualni tok za zapis broja 167 je niz znakova **"1"**, **"6"** i **"7"**, koji se binarno kodiraju kao **0x31**, **0x36** i **0x37**, odnosno binarno **00110001**, **00110110** i **00110111**
- Binarni podaci su direktno *prenosivi* između različitih sistema



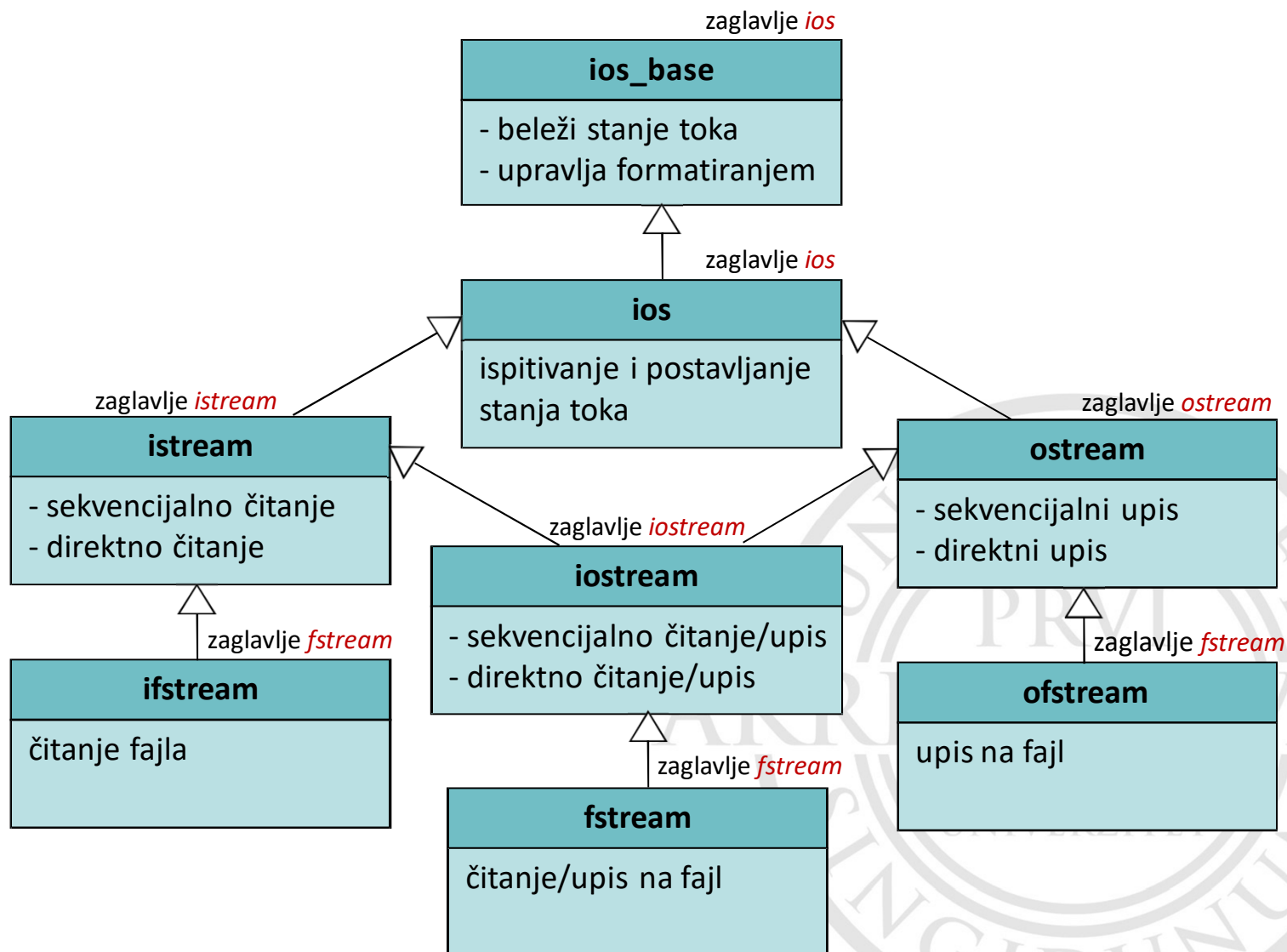
## 2.3 Predefinisani tokovi

- Biblioteka klasa sadrži predefinisane objekte tokova *cin*, *cout*, *cerr* i *clog*, koji su dostupni na početku izvršavanja programa
- Objekt toka *cin* je pridružen standardnom ulazu, uređaju tipa tastature
- Objekt toka *cout* je pridružen standardnom izlazu, ekranu računara
- Objekti toka *cerr* i *clog* takođe su pridruženi standardnom izlazu, ekranu računara i služe za ispis poruka o greškama
  - objekt *cerr* nije baferovan i odmah prikazuje poruke, dok je *clog* baferovan i prikaz informacija se vrši tek po punjenju bafera
- Ovi tokovi se mogu *preusmeravati* na različite uređaje ili datoteke

## 2.4 Klase tokova

- Ulazno-izlazne operacije se oslanjaju na dve različite hijerarhije klasa tokova:
  - `basic_streambuf` - ulazno izlazne operacije niskog nivoa, klasa koja predstavlja osnovu sistema i retko se koristi direktno
  - `basic_ios` - ulazno izlazne operacije visokog nivoa: formatiranje, provera grešaka i stanja U/I operacija
- Klase izvedene iz klase `basic_ios` su klase ulaznih, izlaznih i ulazno-izlaznih tokova `basic_istream`, `basic_ostream` i `basic_iostream`
- Odgovarajuće klase, koje se najčešće koriste u programima su `streambuf`, `ios`, `istream`, `ostream`, `iostream` i klase za datoteke `ifstream`, `ofstream` i `fstream`

# Ilustracija: Hijerarhija klasa UI tokova visokog nivoa



## 3. Rad s fajlovima

1. Pristup fajlovima
2. Čitanje tekstualnog fajla
3. Upis na tekstualni fajl
4. Objekti i tokovi podataka
5. Preklapanje ulaznih i izlaznih operatora



## 3.1 Pristup fajlovima

- Pristup fajlovima u jeziku C++ ostvaruje se pomoću klasa opisanih u zaglavlju `<fstream>`
- Pristup fajlu ostvaruje se povezivanjem s tokom odgovarajućeg tipa: ulaznim, izlaznim ili ulazno-izlaznim, npr.

```
ifstream in;    // ulazni tok  
ofstream out;   // izlazni tok  
fstream io;     // ulazno-izlazni tok
```

- Povezivanje se vrši odgovarajućom funkcijom *open()*:

```
void ifstream::open(const char *filename, ios::openmode  
mode=ios::in);
```

```
void ofstream::open(const char *filename, ios::openmode  
mode=ios::out);
```

```
void fstream::open(const char *filename, ios::openmode  
mode=ios::in|ios::out);
```

# Otvaranje fajla

- Promenljiva *filename* može biti relativna ili apsolutna putanja fajla, npr.

```
ofstream outfile;
```

```
outfile.open("podaci.dat"); // relativna putanja
```

```
outfile.open("C:\\folder\\podaci.dat"); // apsolutna putanja
```

- Parametar *mode* određuje način otvaranja fajla:

Parametar mode	Značenje
<code>ios::app</code>	Sadržaj se dodaje na kraj postojeće datoteke.
<code>ios::ate</code>	Ako datoteka već postoji, program se pomera direktno na njen kraj. Može se upisivati bilo gde u datoteku (ovaj režim se obično koristi sa binarnim datotekama)
<code>ios::binary</code>	Sadržaj se upisuje u datoteku u binarnom obliku, a ne u tekstualnom (koji je podrazumevani)
<code>ios::in</code>	Sadržaj se čita iz datoteke. Ako datoteka ne postoji, neće biti napravljena.
<code>ios::out</code>	Sadržaj se upisuje u datoteku, a ako ona već ima sadržaj, prepisuje se.
<code>ios::trunc</code>	Ako datoteka već postoji, njen sadržaj će biti prepisan (podrazumevani režim za <code>ios::out</code> )

# Otvaranje i zatvaranje fajla

- Funkcija `open` vraća vrednost koja u logičkim izrazima ima vrednost *false* ako otvaranje ne uspe, npr.

```
if (!tok) {  
    cout << "Greška: fajl se ne može otvoriti!" << endl;  
};
```

- Uspešnost otvaranja može se proveriti i funkcijom `is_open()`

```
if (tok.isopen()) {  
    cout << "Fajl je otvoren!" << endl;  
}
```

- Fajl se zatvara pomoću `tok.close()`. Funkcija `open()` nije obavezna, jer klase *fstream*, *istream* i *ostream* imaju konstruktore, koji automatski otvaraju fajlove, npr.

```
ifstream tok("mojFajl"); // automatsko otvaranje
```



## 3.2 Čitanje tekstualnog fajla

- Upis i čitanje fajlova vrši se pomoću standardnih operatora *umetanja* i *izdvajanja* za tokove povezane s fajlom
- Fajl koji se otvara za čitanje mora prethodno da postoji
- Provera kraja datoteke vrši se pomoću funkcije `eof()`, člana klase `istream`
- Čitanje fajlova vrši se pomoću *operatora izdvajanja* (`>>`) ili funkcije `getline()`
  - operator izdvajanja čita podatke do prvog delimitera (beline, *whitespace*) kao što su razmak, tabulator i sl.
  - funkcija `getline()` čita podatke iz fajla *red po red*



# Funkcija getline

- Funkcija `getline()` je član svih klasa ulaznih tokova
- Prototipovi ove funkcije su

```
istream &getline(char *buf, streamsize num)
istream &getline(char *buf, streamsize num, char delim)
```
- Prva varijanta funkcije učitava podatke dužine do *num-1* znakova ili dok ne naiđe na znak kraja reda ili kraj fajla
- Druga verzija kao delimiter koristi aktuelni argument *delim*
- Znak kraja reda i delimiter se ne učitavaju, već se na kraj učitanoog niza znakova upisuje *nula*

## 3.3 Upis u tekstualni fajl

- Upis na *tekstualni* fajl vrši se pomoću standardnih operatora umetanja (<<)
- Upis na fajl je *baferovan*, tako da se podaci fizički upisuju tek kad se interni bafer napuni
- Pošto se prilikom otkaza sistema podaci iz radne memorije mogu izgubiti, upis na fajl može se aktivirati ranije funkcijom `flush()` čiji je prototip  
`ostream &flush();`



# Primer: Upis i čitanje tekstualnog fajla(1/2)

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

int main () {
    char bafer [200];
    cout << "Unesite ime i adresu: ";
    cin.getline(bafer, 200);           // čitanje standardnog ulaza
    cout << endl;

    ofstream outfile("ime.txt"); // konstruktor klase ostream
    if (!outfile) {
        cerr << endl;
        cerr << "Greska: fajl se ne moze otvoriti "
               << "u rezimu dodavanja!" << endl;
        return 1;
    }
}
```

# Primer: Upis i čitanje tekstualnog fajla (2/2)

```
outfile << bafer << endl;           // ispis/dodavanje teksta u outfile
outfile.close();                     // zatvaranje outfile

ifstream infile("ime.txt");          // konstruktor klase istream
while (!infile.eof()) {
    infile.getline(bafer, 200);        // čitanje teksta s fajla istream
    cout << bafer << endl;            // ispis teksta na standardni izlaz
}
infile.close();                      // zatvaranje fajla istream

return 0;
}
```

```
Unesite ime i adresu: Petar Petrović Poenkareova 7a
Petar Petrović Poenkareova 7a
```

```
Unesite ime i adresu: Ivana Ivanović Trešnjina 14
Petar Petrović Poenkareova 7a
Ivana Ivanović Trešnjina 14
```

## 3.4 Objekti i tokovi

- Proces upisa objekta neke klase u tok, tako da može biti ponovo vraćen u memoriju, naziva se *serijalizacija*
- Vraćanje objekta iz toka u memoriju naziva se *deserijalizacija*
- Članovi podaci jedne klase mogu biti osnovni tipovi, klase i pokazivači, tako da su ulazne i izlazne operacije za objekte *specifične* za svaku klasu
- Definišu se nove (nadjačane) verzije operatora *umetanja* i *izdvajanja* za određenu klasu kao prijateljske funkcije klase



## 3.5 Preklapanje ulaznih i izlaznih operatora

- Operatori `<<` i `>>` mogu se preklapati na isti način kao i binarni aritmetički operatori
- Standardna biblioteka obezbeđuje preklopljene definicije ulaznih i izlaznih tokova za standardne tipove vrednosti:

```
ostream& operator<<(ostream& out, tip vrednost);
```

- Upotreba preklopljenih operatora omogućava proširenje ulazno-izlaznih operacija na *nove tipove* podataka, npr.

```
ostream& operator<<(ostream& out, const Razlomak& v) {  
    out << v.brojilac() << "/" << v.imenilac();  
    return out;  
}
```

## 4. Formatiranje ulaza i izlaza

1. Format podataka u ulazno-izlaznim operacijama
2. Upotreba regularnih izraza



## 4.1 Format podataka u ulazno-izlaznim operacijama

- Format podataka u ulazno-izlaznim operacijama može se precizno definisati
  1. Pomoću metoda klase `ios`, koji se koriste za postavljenje flegova i drugih vrednosti kojima se upravlja formatiranjem podataka toka
  2. Pomoću posebnih manipulatorskih funkcija, koje se koriste uz operatore umetanja i izdvajanja (`>>` i `<<`)
  3. Pomoću modula `<format>` od verzije C++20
- Navedeni metodi omogućavaju precizno definisanje formata izlaznih podataka, dok se rad s podacima prilikom njihovog čitanja (parsiranje) može olakšati upotrebom regularnih izraza
  - u mnogim primenama *performanse* upotrebe regularnih izraza znatno zaostaju za standardnim metodima formatiranja



## 4.2 Upotreba regularnih izraza

- U jeziku C++ rad sa stringovima može se pojednostaviti upotrebom *regularnih izraza*, koji omogućavaju skraćeni opis nizova znakova, koji su formirani u skladu s nekim formalnim pravilima (jezikom)
  - standardni zapis regularnih izraza razvijen je 1950-tih i kompletiran 1970-tih za potrebe razvoja Unix programa
  - standardna biblioteka jezika C++ podržava više varijanti sintakse regularnih izraza; podrazumevajuća varijanta je istovremeno i deo standarda jezika *JavaScript* (ECMAScript)
- Regularni izrazi su uključeni u STL biblioteku od verzije C++11
- Upotreba regularnih izraza u jeziku C++ omogućena je preko zaglavlja **<regex>**

# Primer: Upotreba regularnih izraza u jeziku C++

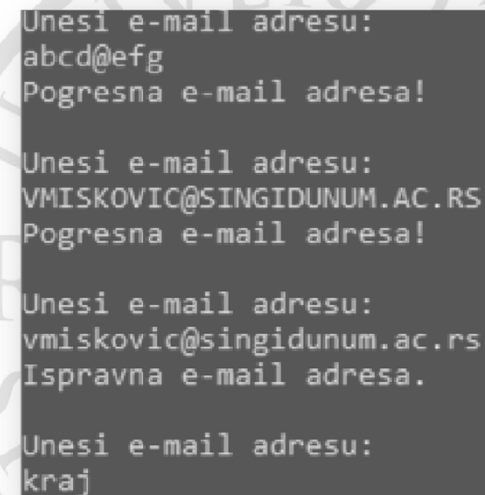
```
#include <iostream>
#include <regex>
#include <string>
using namespace std;

int main() {

    string eadresa;
    regex email("^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,6}$");

    // Provera ispravnosti e-mail adrese (mala slova)
    while(true) {
        cout << "Unesi e-mail adresu: " << endl;
        cin >> eadresa;
        if (eadresa=="kraj") // ako korisnik unese kraj
            break;

        if (regex_match(eadresa,email))
            cout << "Ispravna e-mail adresa.\" << endl;
        else {
            cout<< "Pogresna e-mail adresa!\" <<endl;
        }
    }
    return 0;
}
```



```
Unesi e-mail adresu:
abcd@efg
Pogresna e-mail adresa!

Unesi e-mail adresu:
VMISKOVIC@SINGIDUNUM.AC.RS
Pogresna e-mail adresa!

Unesi e-mail adresu:
vmiskovic@singidunum.ac.rs
Ispravna e-mail adresa.

Unesi e-mail adresu:
kraj
```

## 5. Formatiranje pomoću funkcija klase ios

1. Formatiranje pomoću funkcija člana klase ios
2. Upotreba flegova za formatiranje
3. Preciznije podešavanje formata



## 5.1 Formatiranje pomoću fukcije člana klase ios

- Formatiranje metodima klase *ios* koristi skup flegova `fmtflags`, kojim se upravlja formatiranjem podataka toka, npr.
  - `left` - označava levo poravnavanje izlaza (`right` je desno)
  - `oct`, `hex`, `dec` - prikaz vrednosti u različitim brojnim sistemima
  - `showbase` - flag za prikaz oznake brojnog sistema
  - `uppercase` - prikaz se vrši velikim slovima
  - `boolalpha` - logičke vrednosti se mogu unositi i ispisivati simbolički kao *true* i *false*
- Kada nije postavljen nijedan fleg, format bira prevodilac

## 5.2 Upotreba flegova za formatiranje

- Postavljanje flegova vrši se metodom `setf()` klase `ios`, npr.

```
tok.setf(ios::showpos); // ispis znaka + za poz. vred.
```

- Nazivi flegova su definisani kao konstante u klasi `ios`, pa se za pristup koristi operator dosega
- U jednoj naredbi moguće je istovremeno definisati više flegova, npr.

```
cout.setf(ios::scientific | ios::showpos);
```

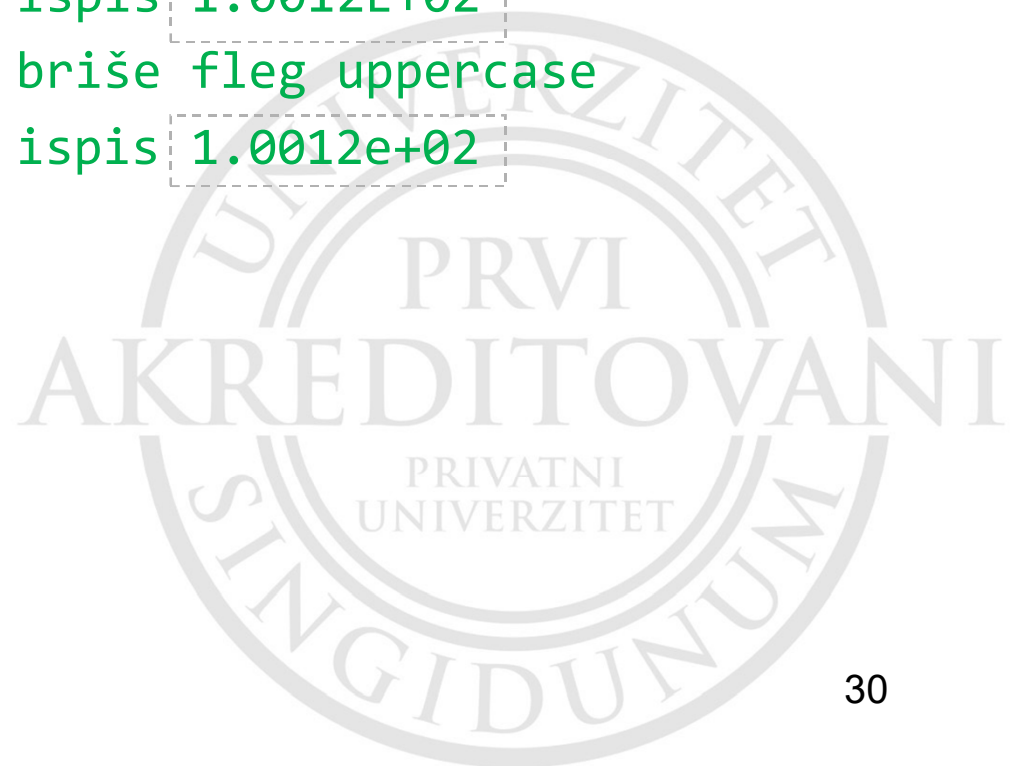
- Brisanje flega vrši se metodom `unsetf()`, a trenutne vrednosti pojedinih flegova dobijaju se pomoću funkcije `flags()` definisane kao

```
void unsetf(fmtflags flags);  
fmtflags flags();
```

# Primer: Formatiranje izlaza pomoću flegova

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

```
int main () {
    cout.setf(ios::uppercase | ios::scientific);
    cout << 100.12;           // ispis 1.0012E+02
    cout.unsetf(ios::uppercase); // briše fleg uppercase
    cout << endl << 100.12;   // ispis 1.0012e+02
    return 0;
}
```



# Preciznije podešavanje formata

- Klasa `ios` poseduje funkcije `width()` za preciznije podešavanje širine polja ispisa, `precision()` za preciznost ispisa brojeva i `fill()` za izbor znak za popunu, čiji su prototipovi:

```
streamsize width(streamsize w); // minimalna širina  
streamsize precision(streamsize p); // p inicijalno 6  
char fill(char ch); // ch je novi, a vraća stari znak
```

- Funkciju `width()` je potrebno pozvati pre svake izlazne operacije
- Ako je ispis kraći, popunjava se znakom za popunu, koji je inicijalno razmak
- Funkcijom `precision()` definiše se broj cifara u prikazu brojeva u pokretnom zarezu

# Primer: Preciznije podešavanje formata

```
#include <iostream>
using namespace std;
int main () {
    cout.precision(4);
    cout.width(10);
    cout << 10.12345 << endl; // prikaz 10.12
    cout.fill('*');
    cout.width(10);
    cout << 10.12345 << endl; // prikaz *****10.12
    cout.width(10);
    cout << "Zdravo!" << endl; // prikaz ***Zdravo!
    cout.width(10);
    cout.setf(ios::left); // levo pravnanje ispisa
    cout << 10.12345 << endl; // prikaz 10.12 *****
    return 0;
}
```



## 6. Upotreba ulazno-izlaznih manipulatora

1. Ulazno izlazni manipulatori
2. Manipulatorske funkcije
3. Argumenti manipulatorskih funkcija
4. Definicija manipulatorske funkcije



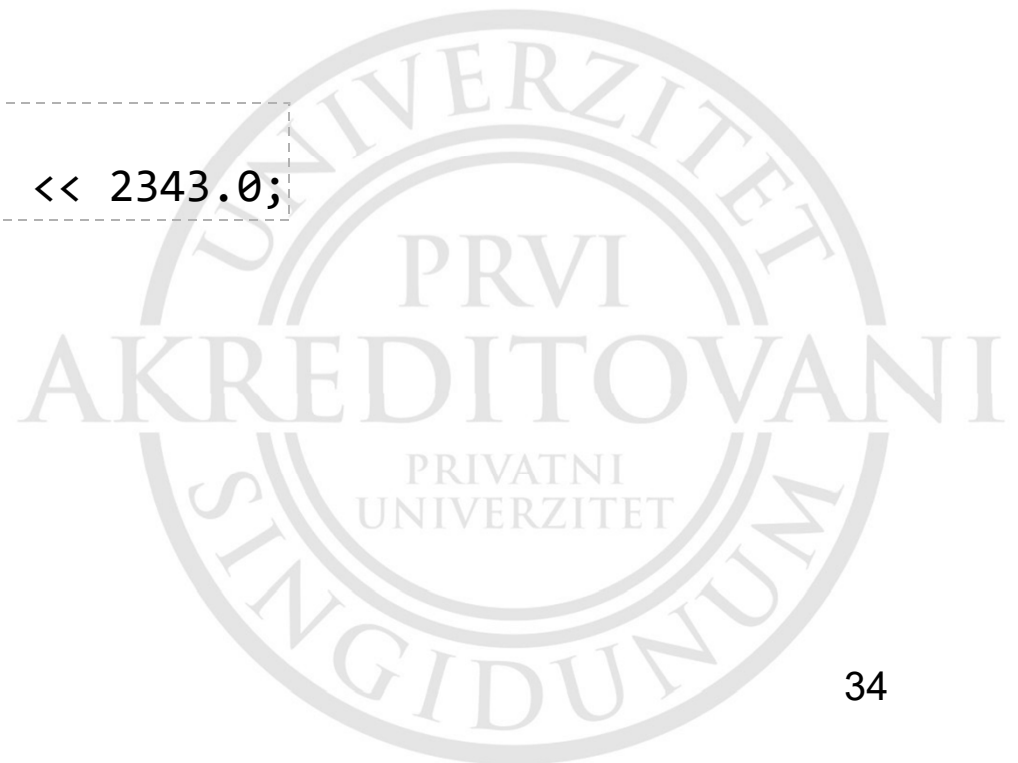
## 6.1 Ulazno izlazni manipulatori

- Ulazno-izlazni format se može definisati pomoću posebnih *manipulatorskih* funkcija definisanih u zaglavlju **<iomanip>**
- Npr. rezultat izvršavanja ulazno-izlaznih naredbi programa

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
    cout << hex << 100 << endl;
    cout << setfill('?') << setw(10) << 2343.0;
    return 0;
}
```

je ispis

```
64
??????2343
```



## 6.2 Manipulatorske funkcije

- Manipulatorske funkcije su posebne funkcije namenjene upotrebi uz operator umetanja << i izdvajanja >>
  - menjaju parametre formatiranja tokova i umeću ili izdvajaju određene specijalne znakove
- Mogu se koristiti i kao standardne funkcije, čiji je argument objekt tipa toka, npr.  
`boolalpha(cout);`



# Manipulatorske funkcije (pregled)

Manipulator	Namena	U/I
boolalpha	Postavlja fleg boolalpha	ulazno/izlazni
dec	Postavlja fleg dec	ulazno/izlazni
endl	Prelazi u novi red i prazni tok	izlazni
ends	Prikazuje null	izlazni
fixed	Uključuje fleg fixed	izlazni
flush	Prazni tok	izlazni
hex	Postavlja fleg za heksadecimalni prikaz	izlazni
internal	Postavlja fleg internal	izlazni
left	Postavlja fleg left	izlazni
noboolalpha	Isključuje fleg noboolalpha	ulazno/izlazni
noshowbase	Isključuje fleg noshowbase	izlazni
noshowpoint	Isključuje fleg noshowpoint	izlazni
noshowpos	Isključuje fleg noshowpos	izlazni
noskipws	Isključuje fleg noskipws	ulazni
nounitbuf	Isključuje fleg nounitbuf	izlazni
nouppercase	Isključuje fleg nouppercase	izlazni
oct	Postavlja fleg za oktalni prikaz	ulazno/izlazni
resetiosflags (fmtflags f)	Resetuje flegove navedene u f	ulazno/izlazni
right	Postavlja fleg right	izlazni
scientific	Uključuje eksponencijalni prikaz	izlazni
set base(int base)	Postavlja brojnu osnovu na base	ulazno/izlazni
setfill(int ch)	Postavlja znak za popunu na ch	izlazni
setiosflags(fmtflags f)	Postavlja flegove navedene u f	ulazno/izlazni
setprecision(int p)	Postavlja broj znakova za preciznost	izlazni
setw(int w)	Postavlja širinu polja na w	izlazni
showbase	Postavlja fleg showbase	izlazni
showpoint	Postavlja fleg showpoint	izlazni
showpos	Postavlja fleg showpos	izlazni
skipws	Postavlja fleg skipws	ulazni
unitbuf	Postavlja fleg unitbuf	izlazni
uppercase	Uključuje fleg uppercase	izlazni
ws	Preskače vodeće razmake	ulazni

## 6.3 Argumenti manipulatorskih funkcija

- Upotrebom manipulatorskih funkcija dobija se kompaktniji i pregledniji kod ulazno-izlaznih izraza
- Funkcije mogu biti s argumentima ili bez argumenata, kada se u izrazima izostavljaju zagrade, npr. za `boolalpha()` i `endl()`

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
    bool b;
    b = true;
    cout << b << " " << boolalpha << b << endl;
    cout << "Unesite logicku vrednost: ";
    cin >> boolalpha >> b;
    cout << "Uneli ste " << b << endl;
    return 0;
}
```

```
1 true
Unesite logicku vrednost: false
Uneli ste false
```

## 6.4 Definicija manipulatorske funkcije

- Manipulatorske funkcije su funkcije čiji je argument *referenca* na tok koje se koriste za promenu parametara formatiranja ili umetanje ili izdvajanje specijalnih znakova
- Opšti oblik manipulatorske funkcije je

```
ostream &naziv_funkcije(ostream &stream) {  
    // telo manipulatorske funkcije  
    return stream;  
}
```

- Argument se u pozivu manipulatorske funkcije ne navodi kad se ona poziva za tok

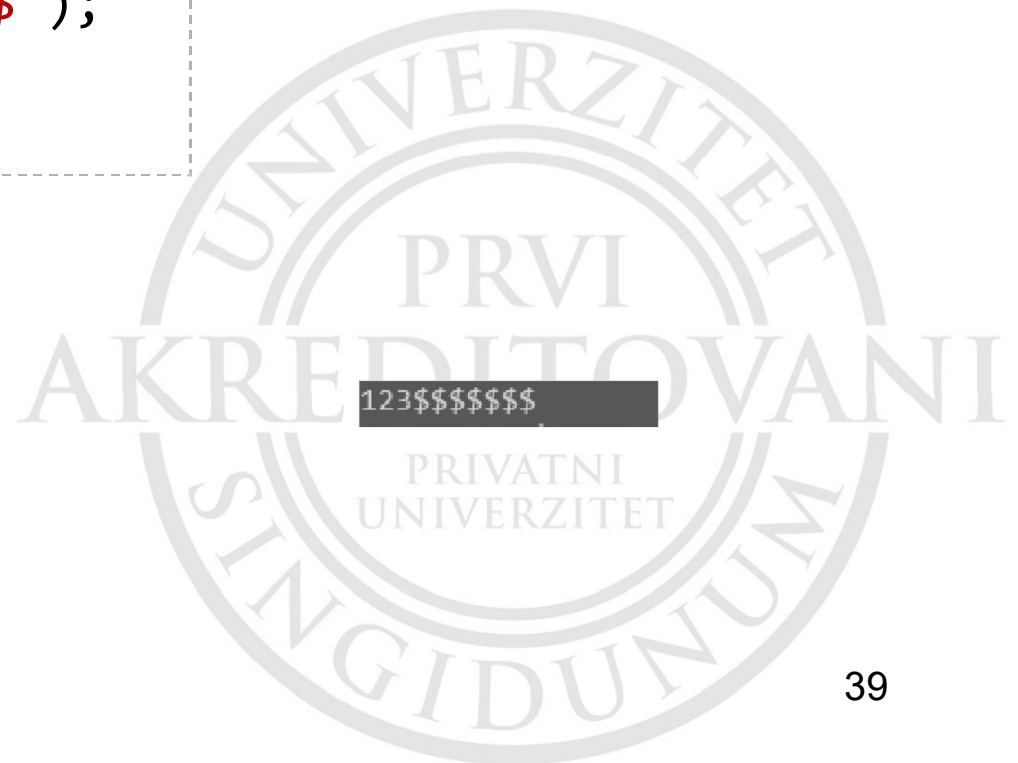


# Primer: Definisanje sopstvene manipulatorske funkcije

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

```
ostream &setup(ostream &stream) {
    stream.setf(ios::left);
    stream << setw(10) << setfill('$');
    return stream;
}
```

```
int main () {
    cout << setup << 123 << endl;
    return 0;
}
```



## 7. Modul <format>

- U verziji jezika C++20 uvode se *moduli*, koji se koriste pomoću deklaracije: `import <naziv_modula>;`
- Modul <format> omogućava jednostavnije formatiranje izlaza, npr. za podatke

```
double r {1.5};  
double p {r * r * 3.14}; // 7.065
```

umesto dosadašnjeg načina formatiranja prikaza rezultata

```
cout << "Povrsina kruga r=" << r << " je "  
      << setprecision(2) // broj značajnih cifri  
      << p << "\n";      // 7.1
```

može se koristiti noviji način

```
cout << format("Povrsina kruga r={} je {:.2} \n", r, p);
```



## 8. Primeri programa

### 1. Lista prostih brojeva



## 8.1 Lista prostih brojeva

- Prosti brojevi
  - Generisanje i ispis zadanog broja  $n$  prostih brojeva
    - za generisanje liste prostih brojeva manjih od zadanog broja koristi se algoritam *Eratostenovo sito*
    - prosti brojevi se generišu u asocijativnom kontejneru tipa *set*
  - Spisak prostih brojeva upisuje se na tekstualni fajl [prosti.txt](#)



# Prosti brojevi (1/3)

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

using namespace std;
typedef set<int>::iterator s_iter;

void stampaj_spisak(const set<int>& s) {
    const int BR_PO_LINIJI = 10;

    // Štampanje spiska prostih brojeva s
    int brojStamp = 0;    //brojač stampanih brojeva u jednoj liniji
    for (s_iter i = s.begin(); i != s.end(); i++) {
        cout << *i << " ";
        brojStamp++;
        if (brojStamp % BR_PO_LINIJI == 0 || brojStamp == s.size()) {
            cout << endl;
        }
    }
}
```

# Prosti brojevi (2/3)

```
void sito(set<int>& s, int n) {  
    // Kreiranje spiska celih brojeva 1..n  
    for (int i = 1; i <= n; i++)  
        s.insert(i);  
    // Prosejavanje: brisanje brojeva koji nisu prosti  
    for (int m = 2; m < *s.rbegin(); m++) {  
        for (int k = m; k <= n; k++)  
            s.erase(m*k); // brisanje iz skupa proizvoda dva broja  
    }  
    stampaj_spisak(s);  
}
```

```
void sacuvaj_spisak(const set<int>& s, string putanja) {  
    ofstream save_file(putanja); // konstruktor klase ostream  
    if (!save_file) {  
        cerr << endl << "Greska: fajl se ne moze otvoriti za upis!" << endl;  
    } else {  
        for (s_iter i = s.begin(); i != s.end(); i++)  
            save_file << *i << " "; // upis/prepisivanje teksta u outfile  
        save_file << endl;  
        save_file.close();  
    }  
}
```

# Prosti brojevi (3/3)

```
int main() {  
    int n;           // gornja granica intervala  
    set<int> s;       // skup (prostih) brojeva  
  
    cout << "Spisak svih prostih brojeva manjih od n\n"  
          << "Unesite n: ";  
    cin >> n;  
  
    sito(s, n);  
    sacuvaj_spisak (s, "prosti.txt");  
  
    return 0;  
}
```

```
Spisak svih prostih brojeva manjih od n  
Unesite n: 50  
1 2 3 5 7 11 13 17 19 23  
29 31 37 41 43 47
```

prosti.txt

1.2.3.5.7.11.13.17.19.23.29.31.37.41.43.47.CRLF

Napomena: oznaka kraja linije tekstualnog fajla je:

- *Windows* CRLF
- *Linux/Unix* LF
- *Mac OS* CR

# Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Horton I., *Beginning C++*, Apress, 2014
5. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. O'Dwyer A., *Mastering the C++17 STL*, Packt, 2017
8. Horstmann C., *Big C++*, 2nd Ed, John Wiley&Sons, 2009
9. Web izvori
  - <http://www.cplusplus.com/doc/tutorial/>
  - <http://www.learncpp.com/>
  - <http://www.stroustrup.com/>
10. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

