Objektno orijentisano programiranje 1

Ulaz i izlaz

Uvod

- Ulaz i izlaz podataka – komunikacija programa sa spoljnim svetom
  - ulaz – unos podataka preko ulaznih uređaja ili čitanje iz memorije
  - izlaz – prikaz podataka na izlaznim uređajima ili smeštanje u memoriju
- Memorija u koju se podaci smeštaju može biti trajna ili privremena
  - trajna memorija – magnetni diskovi, *flash* memorija, ...
  - privremena memorija – operativna memorija (RAM)
- Ulaz i izlaz podataka nisu deo jezika *C++*
  - ne postoje zasebne naredbe za ulaz/izlaz
  - postoje odgovarajuće bibliotečke klase za ulaz/izlaz
- Prilikom U/I podataka može (a ne mora) da se obavlja U/I konverzija

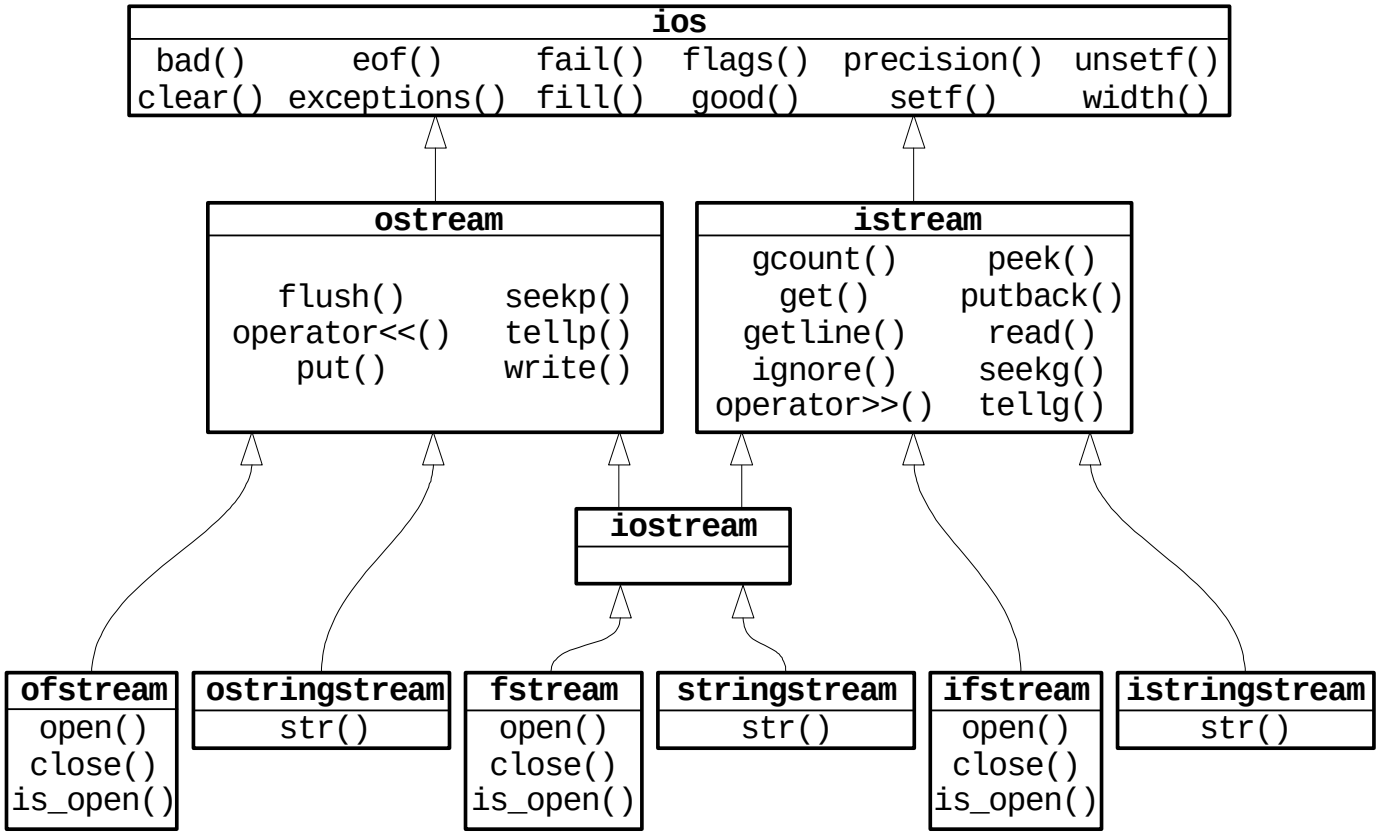
Tokovi

- U C++ (kao i u C) U/I podataka se obavlja preko **tokova** (*stream*)
  - tok je (dugačka) serija bajtova
- Postoji razlika (ali nije suštinska) između pojmova:
  - datoteka - tok u trajnoj memoriji (npr. disku) ili na U/I uređaju
  - string - tok u privremenoj (operativnoj) memoriji
- Većina radnji sa tokovima je ista, bez obzira gde su oni smešteni
- Rad sa tokovima u jeziku *C++* realizuje se odgovarajućim klasama
  - konkretni tokovi predstavljaju se pomoću primeraka tih klasa
  - klase za tokove su deo stanadardne biblioteke
- Osnovna klasa za U/I je `ios` u zaglavlju `<iostream>`

Biblioteke klasa za rad sa tokovima

- Prenos podataka (sa ili bez U/I konverzije) – klase u `<iostream>`
  - `ostream` – samo za izlazne tokove
  - `istream` – samo za ulazne tokove
  - `iostream` – za ulazno/izlazne tokove
- Iz ovih klasa su izvedne klase za rad sa datotekama i stringovima
- Za rad sa datotekama (tokovima na diskovima) - klase u `<fstream>`
  - `ofstream` – samo za izlazne datoteke
  - `ifstream` – samo za ulazne datoteke
  - `fstream` – za ulazno/izlazne datoteke
- Za rad sa stringovima (tokovima u OM) – klase u `<sstream>`
  - `ostringstream` – samo za smeštanje podataka u tokove
  - `istringstream` – samo za uzimanje podataka iz tokova
  - `stringstream` – za uzimanje i smeštanje podataka u tokove
- U tokove (i/o)`stringstream` se primarno uskladištavaju tekstovi

Dijagram klasa



Standardni UI tokovi

- Standardni tokovi su primerci klasa `ostream` ili `istream` koji se automatski stvaraju na početku izvršavanja svakog programa
  - `cin` – glavni (standardni) ulaz tipa `istream`
    - predstavlja tastaturu dok se ne izvrši skretanje glavnog ulaza unutar programa ili u komandi operativnog sistema za izvršavanje programa
  - `cout` – glavni (standardni) izlaz tipa `ostream`
    - predstavlja ekran dok se ne izvrši skretanje glavnog izlaza unutar programa ili u komandi operativnog sistema za izvršavanje programa
    - koristi se za ispisivanje podataka koji čine rezultate izvršavanog programa
  - `cerr` – (standardni) izlaz za poruke tipa `ostream`
    - predstavlja ekran dok se ne izvrši skretanje izlaza za poruke unutar programa
    - koristi se obično za ispisivanje poruka o greškama
  - `clog` – (standardni) izlaz za beleške tipa `ostream`
    - predstavlja ekran dok se ne izvrši skretanje izlaza za beleške unutar programa
    - koristi se obično za vođenje dnevnika o događajima za vreme izvršavanja programa

C funkcije za U/I

- Funkcije za ulaz i izlaz korišćene u jeziku *C*
  - opisane su u zaglavlju `<cstdio>`, odnosno `<stdio.h>`
  - i dalje stoje na raspolaganju i mogu i dalje da se koriste u jeziku C++
- Klase za ulaz i izlaz koje se nude u jeziku *C++*
  - omogućavaju efikasnije izvođenje U/I operacija
  - preporučuje se da se koriste one umesto C funkcija
- Nikako se ne preporučuje
  - da se u nekom programu istovremeno koriste obe vrste U/I operacija

Tokovi-datoteke

- Tokovi-datoteke se koriste za trajno uskladištavanje podataka
  - najčešće na magnetnim diskovima
  - može i na trakama, *flash* memoriji i drugim medijima
  - u širem smislu datoteke su i ulazno-izlazni uređaji:
    - tastatura, ekran, štampač itd.
  - datoteke su vidljive u fajl-sistemu operativnog sistema
- Datoteke mogu da se podele na:
  - binarne - za kompaktno uskladištavanje podataka
  - tekstualne - za uskladištavanje podataka u obliku čitljivom za čoveka

Stvaranje tokova-datoteka

- Podrazumevani konstruktori

```
fstream();
ofstream();
ifstream();
```

  - pozivaju se automatski kad se definiše objekat odgovarajuće klase bez inicijalizatora
  - rezultat je stvaranje objekta bez otvaranja datoteke
    - posledica: ulazno-izlazne operacije, do daljnjeg, nisu dozvoljene za taj objekat
- Ostali konstruktori

```
fstream(const char* imedat, int režim=ios::in|ios::out);
ofstream(const char* imedat, int režim=ios::out);
ifstream(const char* imedat, int režim=ios::in);
```

  - pozivaju se automatski kad se definiše objekat toka sa odgovarajućim inicijalizatorom
  - *imedat*
    - predstavlja naziv datoteke koja treba da se otvori u toku inicijalizacije objekta u navedenom režimu
  - *režim* (rada sa datotekom)
    - može da se označi jednom od simboličkih konstanti ili njihovom kombinacijom (pomoću operatora |)

Režimi otvaranja datoteka

- U klasi `ios` definisane su konstante koje određuju režime otvaranja:
  - `ios::in` – otvaranje datoteke za ulaz
  - `ios::out` – otvaranje datoteke za izlaz
  - `ios::ate` – pozicioniranje na kraj datoteke posle otvaranja
  - `ios::app` – pozicioniranje na kraj datoteke pre svakog upisivanja
  - `ios::binary` – binarna datoteka
  - `ios::trunc` – uništavanje sadržaja datoteke, ukoliko postoji na disku
- Podrazumevano:
  - ako se ne navodi `ios::out`, datoteka mora da postoji na disku
  - bez `ios::binary` - tekstualna datoteka (znak `'\n'` može da bude CR+LF)
  - za izlaz (`ios::out`) se podrazumeva `ios::trunc`, izuzev kad se navede i jedan od režima `ios::ate` ili `ios::app`
- Primeri kreiranja i otvaranja datoteka:

```
fstream podaci;                              // bez otvaranja
ifstream uldat  ("podaci.ul");               // za čitanje
ofstream izldat ("podaci.izl", ios::app); // za dopisivanje
```

Otvaranje i zatvaranje datoteka

- Otvaranje i zatvaranje datoteka omogućavaju da isti objekat tipa nekog od tokova za datoteke u raznim trenucima predstavlja različite datoteke
- Metode:

  ```
  void open (const char* imedat, int režim);
  ```
  - metoda otvara datoteku *imedat* u navedenom režimu pridružujući je tekućem objektu
  - tip tekućeg objekta može da bude `fstream`, `ofstream` ili `ifstream`

  ```
  void close ();
  ```
  - metoda vrši zatvaranje datoteke koja je pridružena tekućem objektu
  - posle zatvaranja datoteke, za isti objekat može da se otvori druga datoteka
  - zatvaranje datoteke ne uništava objekat

  ```
  bool is_open ();
  ```
  - metoda ispituje da li je datoteka koja je pridružena tekućem objektu otvorena
- Primer: otvaranje i zatvaranje datoteke

```
fstream podaci;
podaci.open ("podat.dat", ios::in | ios::out);
podaci.close ();
```

Tokovi u memoriji

- Tokovi u memoriji omogućavaju jednostavnu konverziju podataka iz binarnog u tekstualni oblik i obrnuto
  - postiže se efikasno pisanje programa za obradu teksta kada:
    - u tekst treba ugraditi neku brojnu vrednost
    - iz teksta treba izdvojiti neku brojnu vrednost
- Za spoljašnje prikazivanje sadržaja tokova u memoriji koriste se objekti tipa `string`
- Sadržaj toka u memoriji ne mora biti tekst
  - može da bude i niz binarnih podataka, kao što je to slučaj kod binarnih datoteka

Stvaranje tokova u memoriji

- Podrazumevani konstruktori:

```
explicit  stringstream (int režim=ios::in|ios::out);
explicit ostringstream (int režim=ios::out);
explicit istringstream (int režim=ios::in);
```

  - stvaraju se prazni tokovi u memoriji
- Ostali konstruktori:

```
explicit stringstream (const string& s, int režim=ios::in|ios::out);
explicit ostringstream(const string& s, int režim=ios::out);
explicit istringstream(const string& s, int režim=ios::in );
```

  - stvaraju se tokovi u memoriji čiji početni sadržaj kopija argumenta *s*
- Primeri za stvaranje objekata za tokove u memoriji:

```
stringstream tekst;                          // prazan objekat
istringstream ultekst  ("Dobar dan!");       // za čitanje
ostringstream izltekst (ios::out | ios::app);  // za dopisivanje
```

Pristup sadržaju tokova u memoriji

- Metode:

```
string& str ();
```

  - vrednost je trenutni sadržaj toka za koji je pozvana metoda

```
void str (const string& s);
```

  - metoda postavlja vrednost kopije argumenta s kao novi sadržaj toka za koji je pozvana

- Primer za postavljanje i dohvatanje sadržaja toka u memoriji:

```
tekst.str ("Danas je divan dan.");
string rez = ultekst.str ();
```

Tekstualni tokovi – bez konverzije (1)

- Metode:
  - kod onih kod koje je tip rezultata `ostream&` ili `istream&`
    - vrednost rezultata je upućivač na ulazni/izlazni tok za koji je pozvana
    - omogućava uklapanje poziva metode u složenije izraze

```
ostream& put (char znak);
```

  - smešta *znak* u izlazni tok za koji je pozvana

```
ostream& flush ();
```

  - zahteva se da se isprazni bafer, smeštanjem svih znakova u njemu u izlazni tok
  - korisno je da se uradi kad se duže vremena ne očekuje novi pristup posmatranom toku

```
int get ();
```

  - uzima sledeći znak iz ulaznog toka za koji je pozvana
  - vrednost metode je kod dohvaćenog znaka ili `EOF` ako se stiglo do kraja toka

```
istream& get (char& znak);
```

  - uzima sledeći znak iz ulaznog toka i smešta ga u argument *znak*

Tekstualni tokovi – bez konverzije (2)

- Metode (nastavak):

```
istream& get     (char* niz, int max, char kraj='\n');
istream& getline (char* niz, int max, char kraj='\n');
```

  - uzimaju najviše *max*−1 znakova iz ulaznog toka i smeštaju ih u *niz*
  - prenos se završava kada se naiđe na završni znak *kraj* ili kada je preneto *max*−1 znakova
  - u *niz*, iza poslednjeg prenetog znaka, uvek se dodaje znak `'\0'`, a znak *kraj* nikada
    - u slučaju `get()` završni znak *kraj* ostaje u ulaznom toku
      - biće isporučen kao prvi znak prilikom prvog sledećeg uzimanja znakova iz toka
    - u slučaju `getline()` završni znak *kraj* se izbacuje iz ulaznog toka, ali se ne stavlja u *niz*

```
int peek ();
```

  - dohvata sledeći znak iz ulaznog toka, s tim da sam znak ostaje u ulaznom toku
    - biće isporučen kao prvi znak prilikom prvog sledećeg uzimanja znakova iz toka
  - vrednost metode je kod dohvaćenog znaka iz ulaznog toka

Tekstualni tokovi – bez konverzije (3)

- Metode (nastavak):
  `istream& putback (char znak);`
  - vraća *znak* u ulazni tok
    - vraćeni znak biće isporučen kao prvi znak prilikom prvog sledećeg uzimanja znakova iz toka

  `istream& ignore (int max=1, int kraj=EOF);`
  - preskače najviše `max` znakova u ulaznom toku
    - znakovi neće nikada da se isporuče korisniku toka
    - preskakanje se završava pre vremena ako naiđe završni znak `kraj`
- Primer - sadržaj jedne tekstualne datoteke se prepisuje u drugu:

```
int zn;
while ((zn = uldat.get ()) != EOF)
   izldat.put (zn);
```

Tekstualni tokovi – sa konverzijom (1)

- Za potrebe ulazne i izlazne konverzije podataka operatori >> i <<
  - preklopljeni su za sve standardne tipove podataka
  - za klasne tipove podataka korisnik može da vrši preklapanje ovih operatora
- Podaci tipova **`short`**, **`int`** i **`long`**
  - prenose se konverzijom za cele brojeve koja odgovara %d u jeziku *C*
- Podaci tipova **`float`**, **`double`** i **`long double`**
  - prenose se konverzijom za realne brojeve koja odgovara %g u jeziku *C*
- Podaci tipa **`char`**
  - prenose se konverzijom koja odgovara %c u jeziku *C*
    - sa razlikom da se prilikom čitanja preskoče beli znakovi
- Podaci tipa **`char`*** smatraju se znakovnim nizovima u stilu jezika *C*
  - prenose se konverzijom koja odgovara %s u jeziku *C*
- Pokazivači na objekte proizvoljnih tipova (**`void`***)
  - prenose se konverzijom koja odgovara %p u jeziku *C*

Tekstualni tokovi – sa konverzijom (2)

- Kod izlaznih konverzija primenjuju se podrazumevane konverzije funkcije `printf()` jezika *C*
- Odstupanje može da se postigne
  - korišćenjem manipulatora ili
  - pozivanjem sledećih metoda:

```
long setf (long maska);
long setf (long maska, long vrsta);
```

    - vrše uključivanje (postavljanje na 1) svih indikatora načina konverzije za koje odgovarajući bitovi u *`maski`* imaju vrednost 1
    - preostali indikatori zadržaće svoje vrednosti
    - kod metode sa dva argumenta, *`vrsta`* označava jednu ili više grupa indikatora koje treba isključiti (postaviti na 0) pre uključivanja indikatora predviđenih *`maskom`*
    - vrednost metode je maska vrednosti indikatora načina konverzije koja je važila pre pozivanja
- Za označavanje grupa indikatora za isključivanje pomoću argumenta `vrsta` može da se koristi kombinacija (pomoću operatora `|`) sledećih simboličkih konstanti:

`ios::adjustfield`	– isključivanje indikatora `left`, `right` i `internal`.
`ios::basefield`	– isključivanje indikatora `dec`, `oct` i `hex`.
`ios::floatfield`	– isključivanje indikatora `scientific` i `fixed`.

Tekstualni tokovi – sa konverzijom (3)

- *Maska* indikatora načina konverzije može da kombinuje (pomoću operatora |):
  - `ios::skipws` – preskakanje belih znakova u toku ulaza
  - `ios::left` – ravnanje podataka uz levu ivicu izlaznog polja
  - `ios::right` – ravnanje podataka uz desnu ivicu izlaznog polja
  - `ios::internal` – ravnanje predznaka i/ili oznake brojevnog sistema uz levu ivicu, a cifara broja uz desnu ivicu izlaznog polja
  - `ios::boolalpha` – predstavljanje logičkih podataka u tekstualnom obliku
  - `ios::dec` – korišćenje decimalne izlazne konverzije celih brojeva
  - `ios::oct` – korišćenje oktalne izlazne konverzije celih brojeva
  - `ios::hex` – korišćenje heksadecimalne izlazne konverzije celih brojeva
  - `ios::showbase` – prikazivanje oznake baze brojnog sistema (0 ili 0x) pri izlazu celih brojeva
  - `ios::showpoint` – obavezno prikazivanje decimalne tačke pri izlazu realnih brojeva
  - `ios::showpos` – obavezno prikazivanje predznaka (čak i +) pri izlazu
  - `ios::uppercase` – korišćenje velikih slova za prikazivanje heksadecimalnih vrednosti i slova E u eksponencijalnom obliku realnih brojeva
  - `ios::scientific` – prikazivanje realnih brojeva u eksponencijalnom obliku
  - `ios::fixed` – prikazivanje realnih brojeva bez eksponenta

Tekstualni tokovi – sa konverzijom (4)

- Metode:

```
long unsetf (long maska);
```

  - vrši isključivanje (postavljanje na 0) svih indikatora načina konverzije za koje odgovarajući bitovi u maski imaju vrednost 1
  - preostali indikatori zadržaće svoje vrednosti
  - *maska* se obrazuje na isti način kao i kod metode `setf()`
  - vrednost metode je maska vrednosti indikatora načina konverzije pre poziva

```
long flags (long maska);      // postavljanje
long flags ();                // citanje
```

  - metoda sa jednim argumentom postavlja sve indikatore načina konverzije na odgovarajuće vrednosti bitova u *maski*
  - *maska* se obrazuje na isti način kao i kod metode `setf()`
  - vrednost obe metode je maska vrednosti indikatora načina konverzije pre poziva

```
int width (int širina);       // postavljanje
int width ();                 // citanje
```

  - metoda sa jednim argumentom vrši podešavanje širine polja na *širina* znakova
  - vrednost obe metode je širina polja pre poziva

Tekstualni tokovi – sa konverzijom (5)

- Metode (nastavak):

```
char fill (char znak);          // postavljanje
char fill ();                   // citanje
```

  - metoda sa jednim argumentom služi za zadavanje *znaka* kojim treba da se popuni izlazno polje ukoliko tekući podatak nije dovoljno dugačak
  - vrednost obe metode je znak za popunjavanje koji se koristio pre poziva

```
int precision (int tačnost);    // postavljanje
int precision ();               // citanje
```

  - metoda sa jednim argumentom služi za podešavanje broja prikazanih cifara realnih brojeva na *tačnost* značajnih cifara
  - vrednost obe metode je tačnost koja je važila pre pozivanja metode

- Primer za podešavanje parametara izlazne konverzije

```
int i=123;
cout.fill('*'); cout.width(6); cout << i;
```

  - ispisuje se:

```
***123
```

Binarni tokovi

- Metode:
  `ostream& write (const char* niz, int broj);`
  - smešta *broj* bajtova iz *niza* u izlazni tok
  - prenose se svi bajtovi i nijedan bajt nema posebno značenje (na primer: `'\n'` ili `'\0'`)
  - vrednost metode je upućivač na izlazni tok za koji je pozvana
    - omogućava uklapanje poziva metode u složenije izraze

  `ostream& flush ();`
  - isto kao kod tekstualnih tokova

  `istream& read (char* niz, int broj);`
  - uzima *broj* bajtova iz ulaznog toka u *niz*
  - broj prenetih bajtova može da bude manji od traženog broja

  `int gcount ();`
  - vrednost metode je broj bajtova koji su uzeti iz ulaznog toka u toku poslednjeg pristupa
- Primer za uzimanje niza bajtova iz ulaznog toka uz proveru uspeha:
  `greska = podaci.read (vekt, n).gcount () < n`

Pozicioniranje unutar toka

- Metode:

```
istream& seekg (long pozicija);
ostream& seekp (long pozicija);
```

  - vrše pozicioniranje na bajt sa rednim brojem *pozicija* unutar ulaznog (seekg) ili izlaznog (seekp) toka
  - sledeći prenos podataka počeće od tog bajta
  - vrednost obe metode je upućivač na ulazni odnosno izlazni tok za koji su pozvane

```
istream& seekg (long pomeraj, seek_dir reper);
ostream& seekp (long pomeraj, seek_dir reper);
```

  - vrše pomeranje pozicije unutar ulaznog (seekg) ili izlaznog (seekp) toka za *pomeraj* bajtova relativno u odnosu na navedenu *repernu* tačku
  - sledeći prenos podataka počeće od tako dobijene nove pozicije unutar toka
  - reperna tačka može da se naznači jednom od sledećih simboličkih konstanti:
    - `ios::beg` – početak toka
    - `ios::cur` – trenutna pozicija u toku
    - `ios::end` – kraj toka
  - vrednost obe metode je upućivač na ulazni odnosno izlazni tok za koji su pozvane

Pozicija unutar toka

- Metode:

```
long tellg ();
long tellp ();
```

  - vrednost ovih metoda je trenutna pozicija unutar ulaznog (`tellg`) ili izlaznog (`tellp`) toka kao redni broj bajta u odnosu na početak datoteke

- Primer upisa novog sadržaja u *`zapis`* sa rednim brojem *k* (*k*=1,2,...) relativne datoteke *`podaci`* sa zapisima dužine *duz* bajtova :

```
podaci.seekp ((k-1)*duz).write ((char*)&zapis, duz);
```

Provera grešaka

- Kada se objekti tipa tokova koriste kao logičke vrednosti
  - **`true`** označava da je tok u ispravnom stanju
  - **`false`** označava da se desila neka greška u toku pristupanja datom toku
- Metode za proveru grešaka u toku čitanja i pisanja tokova:

  `bool` `good ();`
  - **`true`** označava da je tok ispravan, prethodni pristup je bio uspešan
  - sledeće čitanje može da bude uspešno
  - ako tok nije u ispravnom stanju, ulazno-izlazne operacije nemaju efekta

  `bool` `eof ();`
  - **`true`** označava da se stiglo do kraja toka, prethodni pristup je bio uspešan
  - sledeće čitanje neće uspeti (osim ako se trenutna pozicija prethodno ne promeni)

  `bool` `fail ();`
  - **`true`** označava da je prethodni pristup bio uspešan i nijedan bajt nije izgubljen, ali da naredni pristup neće biti uspešan

  `bool` `bad ();`
  - **`true`** označava da je tok u neispravnom stanju
  - ne može ništa da se predvidi o uspehu narednih pristupa toku

Signalizacija grešaka

- Problem sa proverom grešaka - svaki čas se ispituje da li je promenjeno stanja toka
- Moguće je podesiti prijavljivanje promene stanja toka izuzetkom tipa `ios::failure`
- Metode:

```
void exceptions (int maska);
int exceptions ();
```

  - *maska* određuje promene stanja toka za koje se traži prijavljivanje izuzetka
  - vrednost druge metode označava koje promene stanja toka se trenutno prijavljuju izuzecima
  - *maska* može da bude kombinacija (pomoću operatora `|`) sledećih simboličkih konstanti:
    - `ios::eofbit` – izuzetak se prijavljuje kad vrednost metode `eof()` postane **`true`**
    - `ios::failbit` – izuzetak se prijavljuje kad vrednost metode `fail()` postane **`true`**
    - `ios::badbit` – izuzetak se prijavljuje kad vrednost metode `bad()` postane **`true`**
    - `ios::goodbit` – isključuje se prijavljivanje izuzetaka; ne može da se kombinuje sa drugim konstantama

```
void clear (int maska=ios::goodbit);
```

  - postavlja stanje toka na osnovu vrednosti argumenta *maska*
  - vrednost argumenta može da bude `ios::goodbit` ili kombinacija preostala tri bita
  - prijaviće izuzetak tipa `ios::failure` ako je to metodom `exceptions()` ranije zatraženo za novopostavljeno stanje toka

- Primer kojim se sadržaj jedne tekstualne datoteke prepiše u drugu, uz prekid ako se desi bilo kakva neispravnost (`!good()`) kod bilo kog toka:

```
char zn; while (izldat && uldat.get (zn)) izldat.put (zn);
```

Bafer toka

- Klase izvedene iz `ios` podržava drugi sistem klasa (bafer toka) za elementarne radnje vezane za rukovanje fizičkim uređajima
- Sastoji se od:
  - osnovne klase `streambuf`
  - klasa `filebuf` i `stringbuf` (izvedene iz `streambuf`)
- Njihov zadatak je rukovanje ulazno-izlaznim baferima i neposredni pristup do samih ulazno-izlaznih uređaja
- Sistem klasa bafera tokova nije u nasledničkom odnosu sa `ios` sistemom klasa
- Usluge klasa bafera tokova se dobijaju pozivanjima njihovih javnih metoda