

IO biblioteka

Objektno programiranje - 4. vježbe

Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

29. ožujka 2023. godine



Zaglavlja i tipovi IO biblioteke

Zaglavlja:

- **iostream** - čitanje/pisanje iz *streamova*
 - čitanje: `istream` (već koristili `cin`)
 - pisanje: `ostream` (već koristili `cout`)
 - pisanje i čitanje: `iostream`
- **fstream** - čitanje/pisanje iz datoteka
 - čitanje: `ifstream`
 - pisanje: `ofstream`
 - pisanje i čitanje: `fstream`
- **stringstream** - čitanje/pisanje iz stringova
 - čitanje: `istringstream`
 - pisanje: `ostringstream`
 - pisanje i čitanje: `stringstream`
- za prošireni skup znakova koristi se tip `wchar_t` - tipovi i objekti za taj tip imaju na početku slovo `w`
- npr. `stringstream` \mapsto `wstringstream`

IO objekti

- zahvaljujući **nasljeđivanju**, možemo ignorirati razlike između različitih vrsta streamova
- primjerice, tipovi `ifstream` i `istringstream` naslijeđeni iz `istream` ⇒ objekte tih tipova koristimo kao `cin` (možemo zvati `getline` i koristiti `>>`)
- posljedica: ono što obradimo za „obične” streamove, vrijedit će i za ostale vrste streamova
- za IO objekte **nema ni pridruživanja ni kopiranja**

Primjer.

```
ofstream out1, out2;  
out1 = out2;      X  
ofstream print(outstream);  X  
out2 = print(out2);  X
```

⇒ funkcije koje rade s IO primaju/vraćaju streamove **kroz reference**

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

istream &ucitaj(istream &ulaz, int &a) {
    ulaz >> a;
    return ulaz;
}

int main() {
    int br1, br2;
    ucitaj(ucitaj(cin,br1), br2);
    cout << endl << br1 << endl << br2 << endl;
    return 0;
}
```

Pitanje. Zašto je u primjeru važno da funkcija vraća referencu na `std::istream`?

Primjer

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      int br, suma = 0;
6      while(cin >> br)
7          suma += br;
8      cout << "Zbroj: " << suma << endl;
9      return 0;
10 }
```

Uočimo: ne razlikujemo unos 2 3 4 abc od 2 3 4 i EOF.

`strm::iostate` tip - provjera stanja streama

- `strm::iostate` tip (`strm` - jedan od IO tipova) - kolekcija zastavica - imamo funkcije za njihovu provjeru

funkcija	true ako postavljen(i):	
<code>s.eof()</code>	<code>eofbit</code>	došli do EOF (to postavlja i <code>failbit</code>)
<code>s.fail()</code>	<code>failbit</code> ili <code>badbit</code>	<code>failbit</code> - greška od koje se možemo oporaviti (npr. umjesto inta dobili char)
<code>s.bad()</code>	<code>badbit</code>	greška na razini sustava (npr. greška čitanja/pisanja od koje nema oporavka)
<code>s.good()</code>	<code>goodbit</code>	sve ok (npr. za razliku od ostalih neće prekinuti naš <code>while(cin >> br)</code>)

- pomoću `s.clear()` / `setstate()` / `rdstate()` čistimo / postavljamo / čitamo zastavice

Primjer. Što radi sljedeći kod:

```
cin.clear(cin.rdstate() & ~cin.failbit &
~cin.badbit);
```



Primjer.

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      int br, suma = 0;
6      while(cin >> br)
7          suma += br;
8      if(cin.eof())
9          cout << "Zbroj: " << suma << endl;
10     else
11         cout << "Doslo je do greske pri unosu!" << endl;
12     return 0;
13 }
```



Pražnjenje *buffera*

```
cout << "Poruka.";
```

- gornja poruka se ne mora ispisati odmah (završi u *bufferu*)

Osim korištenja `endl` (dodaje prelazak u novi red!), imamo:

- `flush` - ne dodaje znakove u izlaz

```
cout << "ABC" << flush;
```

- `ends` - dodaje NULL znak

```
cout << "ABC" << ends;
```

- `unitbuf` - postavlja da stream nakon svakog ispisa radi `flush`
- `nounitbuf` - vraćanje u „normalno” stanje

Primjer.

```
cout << unitbuf;  
cout << "Poruka.";  
cout << nounitbuf;  
cout << "Poruka.";
```


Čitanje i pisanje u datoteku

- tipovi: `ifstream`, `ofstream`, `fstream` (čitanje/pisanje/oboje)

Općenito, imamo sljedeće korake:

- 1 stvoriti *stream* objekt
- 2 povezati ga s datotekom
- 3 čitati/pisati
- 4 zatvoriti datoteku (prije nego objekt koristi drugu datoteku!)

Primjer. (Otvaranje datoteke)

```
#include <fstream>
...
string naziv = "moja_dat.txt";
ifstream dat; //još nije povezan s datotekom
dat.open(naziv);
```

- ili samo: `ifstream dat("moja_dat.txt");`

Zatvaranje datoteke i provjera otvaranja

- zbog nasljeđivanja, možemo koristiti funkciju `ucitaj` (s jednog od prethodnih slajdova) za `fstream`
- `close` se poziva automatski prilikom uništavanja `fstream` objekta

Primjer.

```
ifstream dat;  
string naziv;  
int br, suma = 0;  
cout << "Unesite ime datoteke (bez .txt): ";  
cin >> naziv;  
dat.open(naziv + ".txt");  
if(dat) {  
    while(ucitaj(dat, br))  
        suma += br;  
    cout << "Suma = " << suma << endl;  
    dat.close();  
} else cout << "Ne mogu otvoriti "  
    << naziv << ".txt!" << endl;
```

Načini korištenja datoteka

- način se iznova određuje prilikom **svakog** otvaranja datoteke
- **in** - za čitanje (ne može za `ofstream` objekte)
- **out** - za pisanje (ne može za `ifstream` objekte)
- **trunc** - briše sadržaj datoteke (samo zajedno s `out` - s njim ide po defaultu)
- **app** - sva pisanja idu na kraj (ne može biti zajedno s `trunc`)
- **ate** - početno se pozicionira na kraj (ali možemo se pozicionirati i negdje nakon toga)

Defaulti: `out` za `ofstream`, `in` za `istream`, a oba za `fstream`

Primjer. Odredite razliku ako je "abc.txt" postojeća datoteka:

```
ofstream dat("abc.txt");  
dat << "Poruka." << endl;
```

```
ofstream dat("abc.txt", ofstream::out | ofstream::app);  
//ili samo: ofstream dat("abc.txt", ofstream::app);  
dat << "Poruka." << endl;
```

Niz .txt datoteka sadrži riječi. Korisnik svako ime datoteke (pri čemu je svako ime jedna riječ bez “.txt”) zadaje kao **argumente komandne linije** (uočite da nije zadano koliko ih ima!). Program treba ispisati statistiku za svaku riječ (sortirano leksikografski): koliko se ukupno puta javila i u kojim se sve datotekama javila.

Ukoliko se neka datoteka nije uspjela otvoriti, ispišite poruku za svaku takvu datoteke (ostale normalno uključite u izračun).

string *streamovi*

- `sstring` zaglavlje
- tipovi: `istringstream` za čitanje stringa, `ostringstream` za pisanje, `stringstream` za čitanje i pisanje

Primjer. U svakoj liniji unosa je: ime učenika (jedna riječ) i niz njegovih ocjena. Primjerice:

John	5	3	4	5
------	---	---	---	---

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

struct osoba{
    string ime;
    vector<int> ocjene;
};
```

(nastavak primjera je na sljedećem slajdu...)



Nastavak primjera (kod u `main` funkciji)

```
string linija;  
int ocj;  
vector<osoba> razred;  
while (getline(cin, linija)) {  
    osoba ucenik;  
    istringstream zapis(linija);  
    zapis >> ucenik.ime;  
    while (zapis >> ocj)  
        ucenik.ocjene.push_back(ocj);  
    razred.push_back(ucenik);  
}
```

- `zapis.str()` - vraća kopiju stringa koji `zapis` sadrži
- `zapis.str(s)` - kopira string `s` u `zapis`

- nastavno na prethodni kod, ispišemo samo one s ocjenama - prije ekrana, sebi složimo (mogući) ispis

```
for (const auto &a : razred) {  
    ostreamstream izvjestaj;  
    izvjestaj << a.ime;  
    for (const auto &o : a.ocjene)  
        izvjestaj << ", " << o;  
    if(a.ocjene.size() != 0)  
        cout << izvjestaj.str() << endl;  
}
```

Primjer - `getline()` za tokenizaciju stringova

Primjer.

```
string dio;  
stringstream in("Dos'o-medo-u-ducan-nije-rek'o");  
while(getline(in, dio, '-'))  
    cout << dio << endl;
```

Ispis:

```
Dos'o  
medo  
u  
ducan  
nije  
rek'o
```

U C-u se slično moglo pomoću [strtok](#) (iz `string.h`).

- **manipulatori** - funkcije ili objekti koji mijenjaju stanje *streama*
- vraćaju objekt na koji su primjenjeni \Rightarrow mogu se kombinirati u istoj naredbi s podacima
- ako promijene formatiranje *streama*, ono ostaje takvo za sav daljnji IO

Primjer. **boolalpha** manipulator ispisuje `bool` vrijednosti kao “true” i “false” - povratak na *default* pomoću **noboolalpha**

```
cout << (2 == 2) << endl;  
cout << boolalpha << (2 == 2);  
cout << (3 < 2) << noboolalpha  
    << (3 < 2) << endl;
```

Promjena baze (za cijele brojeve!)

- baze: decimalna (**dec**), oktalna (**oct**), heksadecimalna (**hex**)
- ispis i baze (prefiks "0x" za hex, "0" za okt) postavljamo pomoću **showbase**, a isključujemo pomoću **noshowbase**
- ako želimo velika slova u hex. zapisu: **(no)uppercase**

Primjer.

```
cout << 16 << " " << 26 << endl;
cout << oct << 16 << " " << 26 << endl;
cout << hex << 16 << " " << 26 << endl;
cout << showbase << 26 << endl;
cout << uppercase << 26 << endl;
```

- za učitavanje samo jednog hex. broja:

```
int a;
cin >> hex >> a >> dec;
```

Format brojeva s pomičnom točkom

- **setprecision(n)** manipulator - postavlja preciznost (ukupan broj znamenaka) na n
- kao i ostali manipulatori koji primaju argumente treba **#include<iomanip>**
- određivanje zapisa (mijenjaju značenje preciznosti!):
 - **scientific** - znanstveni zapis
 - **fixed** - fiksni decimalni zapis
 - **hexfloat** - u heksadecimalnom zapisu
 - **defaultfloat** - povratak na *defaultni* zapis
- **(no)showpoint** - ako je broj cijeli treba li ispisati točku

Primjer. (za `sqrt` treba `#include<cmath>`)

```
cout << setprecision(5) << sqrt(2) * 100 << endl  
      << scientific << sqrt(2) * 100 << endl  
      << hexfloat << sqrt(2) * 100 << endl;
```

Manipulatori za poravnanje

- u sljedećim primjerima `□` predstavlja prazninu (tj. ' ')
- **setw** - minimalan broj mjesta koja varijabla pri ispisu zauzima

```
double br = -3.14;  
cout << setw(10) << br; → Ispis: □□□□□ - 3.14
```

10 mjesta

- po *defaultu* je **right** - možemo postaviti na **left**:

```
cout << left << setw(10) << br; → Ispis: -3.14□□□□□
```

10 mjesta

VAŽNO: Poput `endl` manipulatora **setw ne mijenja stanje streama - odnosi se samo na sljedeću vrijednost** koju ispisujemo!

- **internal** - broj desno, a predznak - (ako postoji) je lijevo

```
cout << internal << setw(9) << br; → Ispis: -□□□□3.14
```

9 mjesta

- **setfill** - „popunjavanje” ispisa određenim znakom (npr. '#')

```
cout << setfill('#') << setw(9) << br; → -#####3.14
```

9 mjesta

(*defaultnu* prazninu vraćamo korištenjem `setfill(' ', ' ')`)



Čitanje bjelina pri unosu

Primjer. Sljedeći dio koda:

```
char c;  
while(cin >> c)  
    cout << c;
```

za unos ispisuje

- **ignoriraju se bjeline pri unosu** (razmaci, prelasci u novi red i sl.)
- **(no)skipws** (is/u)ključuje čitanje bjelina

Primjer. Sljedeći dio koda:

```
char c;  
cin >> noskipws;  
while(cin >> c)  
    cout << c;  
cin >> skipws; //vratimo na default
```

za unos ispisuje

- rad sa *streamom* kao nizom neinterpretiranih bajtova

Primjer. Sljedeći dio koda radi isto što i kod iz prethodnog primjera:

```
char ch;  
while (cin.get(ch))  
    cout.put(ch);
```

- ponekad moramo pročitati znak ulaza da bi shvatili da još nismo spremni za njega - taj jedan (!) znak možemo vratiti u *stream* za sljedeće čitanje:
 - **unget** - vraća zadnji pročitani znak u *stream*
 - **peek** - vraća kopiju sljedećeg znaka iz *streama* (ne mijenja *stream* - znak je samo iskopiran!)
 - **putback** - kao `unget`, ali prima argument koji mora biti jednak zadnjem pročitanoj znaku

Povratna vrijednost `int`

- `peek` i verzija funkcije `get` bez argumenata vraćaju znak *streama* kao `int` (a ne `char`)
- razlog: mogućnost vraćanja EOF markera

Primjer.

```
int ch;    //ovdje ne koristiti char!  
while ((ch = cin.get()) != EOF)  
    cout.put(ch);
```

- za operacije s više bajtova možemo koristiti: `get` (primjer na sljedećem slajdu), `getline`, `read`, `gcount`, `write`, `ignore`

Operacije s više bajtova

Primjer.

```
ifstream dat("abc.txt");
char* str = new char[11];
for(int i = 1; i <= 3; ++i) {
    dat.get(str, 10, ',');
    cout << "Ucitano:  " << str << endl;
}
delete str;
```

- **get** čita najviše 10-1 znakova (ili do EOF ili do zarez - kojeg ostavlja u streamu za sljedeće čitanje!) i sprema to u polje charova str
- ako je sadržaj datoteke "abc.txt": Ovo je neki, tekst. dobivamo ispis:

```
Ucitano:  Ovo je ne
Ucitano:  ki
Ucitano:
```


Random pristup *streamu*

- koristimo za `sstream` i `fstream` tipove - imaju **marker** koji određuje gdje počinje sljedeće čitanje/pisanje
 - `seek` ga premješta, a `tell` daje njegov trenutni položaj
 - za *input streamove* te funkcije imaju sufiks `g` (*getting*), a za *output streamove* sufiks `p` (*putting*)

<code>tellg()</code> <code>tellp()</code>	vraća trenutnu poziciju markera
<code>seekg(poz)</code> <code>seekp(poz)</code>	premješta marker na danu apsolutnu adresu <i>streama</i>
<code>seekp(off, od)</code> <code>seekg(off, od)</code>	premješta marker za <code>off</code> znakova prije/poslije <code>od</code>

- `od` je jedan od sljedećeg:
 - `fstream::beg` - početak *streama*
 - `fstream::cur` - trenutni položaj u *streamu*
 - `fstream::end` - kraj *streama*

Primjer: Pisanje i čitanje za istu datoteku

- neka je dana datoteka “abc.txt” sa sljedećim sadržajem:

```
abcd  
efg  
hi  
j
```

- u tu** datoteku treba dodati novu liniju na njen kraj u kojoj će pisati pozicije početka svake (osim prve!) linije u izmijenjenoj datoteci
- novi sadržaj datoteke “abc.txt”:

```
abcd  
efg  
hi  
j  
5 9 12 14
```

(Komentar: Broje se i prelasci u novi red!)

Prvi dio - otvaranje datoteke i pozicioniranje

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

int main() {
    fstream dat("abc.txt", fstream::ate |
                fstream::in | fstream::out);
    if (!dat) {
        cerr << "Ne mogu otvoriti dat!" << endl;
        return -1;
    }
    auto end_mark = dat.tellg();
}
```

- otvorili datoteku “abc.txt” za čitanje i pisanje
- početno smo se pozicionirali na kraj datoteke
- zapamtili originalnu poziciju kraja

Drugi dio

```
dat.seekg(0, fstream::beg);
size_t cnt = 0;    //broj procitanih B/znakova
string line;
while (dat && dat.tellg() != end_mark
        && getline(dat, line)) {
    ...
}
dat.seekp(0, fstream::end);
dat << "\n";
return 0;
}
```

- postavimo se na početak datoteke
- tri uvjeta u `while` petlji:
 - nije se još dogodila greška
 - još uvijek čitamo **originalne** podatke
 - možemo dobiti sljedeću liniju ulaza
- na kraju dodamo prelazak u novi red (na kraj izmijenjene `dat.`)

```
cnt += line.size() + 1; // + 1 zbog '\n'  
auto mark = dat.tellg();  
dat.seekp(0, fstream::end);  
dat << cnt;  
if (mark != end_mark)  
    dat << " ";  
dat.seekg(mark);
```

- **zapamtimo gdje smo stali s čitanjem**, **pozicioniramo se na kraj i tamo zapišemo** trenutno stanje brojača (i razmak prije sljedećeg broja ako to nije zadnji broj koji ćemo zapisati)
- zatim se **vratimo tamo gdje smo stali s čitanjem**