

Datoteke

Objektno programiranje - 9. vježbe (2. dio)

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

21. svibnja 2025. godine

Pamćenje *highscorea* za igru *Zmija*

- želimo pamtit do sad najveći postignut broj bodova (tzv. *highscore*) u igri *Zmija*
- no, kad bi to pamtili u nekoj varijabli u programu, ponovnim pokretanjem programa izgubili bi vrijednost *highscorea* iz prethodnih igri (tj. prethodnih pokretanja programa)
- pamtit ćemo ga u **datoteci** *najrezultat.txt*
- kad se program pokrene, ispišemo vrijednost učitane iz te datoteke na ekran
- tek pri **zatvaranju programa**, spremamo novi *highscore* koji je igrač ostvario (ili stari *highscore* ako igrač nije ostvario veći broj bodova)



Pregled zaglavlja i tipova IO biblioteke

Zaglavlja:

- **istream** - 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`
- **sstream** - 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**
- primjer: `stringstream` $\mapsto$ **w**`stringstream`

IO objekti

- zahvaljujući **nasljeđivanju**, možemo ignorirati razlike između različitih vrsta streamova
- primjerice, tipovi `ifstream` i `istreamstringstream` 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**

Primjer.

```
#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`? (Ulančavanje poziva funkcije `ucitaj`!)

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)) //(dat << br) je false ako  
        suma += br; // ne moze vise učitati int iz dat  
    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 `ifstream`, 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).

Rješenje (1. dio - includeovi)

```
1  #include<iostream>
2  #include<string>
3  #include<map>
4  #include<set>
5  #include<utility> //za pair
6  #include<fstream>
7  using namespace std;
8
9  int main(int argc, char* argv[]) {
```

Rješenje (2. dio - čitanje riječi iz datoteka)

```
ifstream dat;
map<string,pair<int,set<string>>> stat;
for(auto p = argv + 1; p != argv + argc; ++p){
    dat.open(string(*p) + ".txt");
    if(dat){
        string s;
        while(dat >> s){
            stat[s].first++;
            stat[s].second.insert(string(*p) + ".txt");
        }
        dat.close();
    } else cout << "Ne mogu otvoriti " << *p
              << ".txt." << endl;
}
```

Rješenje (3. dio - ispis podataka)

```
25 //ispis svih podataka
26 for(auto a : stat) {
27     cout << a.first << ", br: " << a.second.first
28         << ", u datotekama: ";
29     for(auto i : a.second.second)
30         cout << i << ";";
31     cout << endl;
32 }
33
34 return 0;
35 }
```

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`.)

Treći dio - tijelo `while` petlje

```
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**

Vođenje evidencije o *highscoreu* u igri *Zmija*

- pri pokretanju programa (u konstruktoru klase `Igra`), otvorimo datoteku `najrezultat.txt` (mora biti u istoj mapi kao i naš projekt!) te učitamo broj iz nje koji će biti početna vrijednost varijable *highscore* (koju smo dodali kao privatan dio klasi `Igra`)

```
class Igra {  
    ...  
    private:  
        ...  
        int highscore;  
};
```

```
Igra::Igra() : ... {  
    ...  
    std::ifstream dat("najrezultat.txt");  
    if (dat) {  
        dat >> highscore;  
        (... nastavak na idućem slajdu ...)
```

Nastavak koda (slučaj uspješnog otvaranja datoteke)

- ako je datoteka uspješno otvorena (te smo učitali *highscore*), dodamo poruku u `textbox` o trenutnoj vrijednosti *highscorea*
- također ćemo zatvoriti otvorenu datoteku

```
textbox.Dodaj("Trenutni highscore: "  
              + std::to_string(highscore));  
dat.close();  
}  
(... nastavak na idućem slajdu ...)
```

Slučaj neuspješnog otvaranja datoteke

```
else {  
    std::cout << "Highscore nije dostupan!"  
              << std::endl;  
    highscore = 0;  
}  
}
```

- ako datoteka ne postoji (pa nije uspješno otvorena za čitanje), postavimo *highscore* na 0
- pri prvom pokretanju programa ta datoteka još ne postoji (osim ako smo ju „ručno” napravili), pa ćemo na konzoli vidjeti ispis poruke

Highscore nije dostupan!

Spremanje *highscorea* pri zatvaranju programa

- nećemo svaki put kad igrač izgubi spremati *highscore* (ako je igrač tada postigao veći *highscore*), nego samo pri zatvaranju programa
- to znači da ćemo spremanje rezultata obaviti u destrukturu klase Igra:

```
Igra::~Igra() {  
    std::ofstream dat("najrezultat.txt");  
    if (dat) {  
        dat << highscore;  
        dat.close();  
    }  
    else { // neuspješno otvaranje datoteke  
        std::cout << "Ne mogu otvoriti najrezultat.txt!"  
                  << std::endl;  
    }  
}
```

Ažuriranje *highscorea*

- kad igrač izgubi, provjerimo je li ostvario više bodova no što je trenutni *highscore*
- ako jest, onda samo promijenimo vrijednost varijable *highscore* (podsjetnik: spremanje u datoteku se radi u destrukturu)

```
void Igra::update() {  
    ...  
    if (zmija.JelIzgubio()) {  
        if (zmija.DohvatiBodove() > highscore) {  
            highscore = zmija.DohvatiBodove();  
            textbox.Dodaj("Novi highscore: "  
                + std::to_string(highscore));  
        } else {  
            textbox.Dodaj("Nije ostvaren highscore!");  
            textbox.Dodaj("(Podsjetnik: trenutni je "  
                + std::to_string(highscore) + ")");  
        }  
        ...  
    }  
}
```