Datoteke u programskom jeziku C++

Vuk, Antonio

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

ANTONIO VUK

DATOTEKE U PROGRAMSKOM JEZIKU C++

Završni rad

Pula, rujan 2017. godine

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

ANTONIO VUK

DATOTEKE U PROGRAMSKOM JEZIKU C++

Završni rad

JMBAG: 0303054766 redovni student
Studijski smjer: Sveučilišni preddiplomski studij Informatika
Predmet: Napredne tehnike programiranja
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc. Tihomir Orehovački

Pula, rujan 2017. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Antonio Vuk, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan 2017. godine

IZJAVA

o korištenju autorskog djela

Ja, Antonio Vuk dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Datoteke u programskom jeziku C++“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan 2017. godine

Potpis

_____

SAŽETAK

Profesionalan rad programera nije moguć bez poznavanja različitih načina organiziranja datoteka. Nerijetko prilikom izrade različitih programa, igara, *web* stranica, potrebno je pohraniti podatke u datoteku. Odabirom optimalnog načina organizacije podataka unutar datoteke znatno se uvećavaju performanse programa i efikasnost rada. Zato su u ovome radu objašnjene osnovne metode rada s datotekama u programskom jeziku C++. Izrađeni su primjeri za prikaz zamjene sadržaja u tekstualnoj datoteci, rad s različitim organizacijama datoteka, sortiranje datoteke metodom selection sort te je naposljetku prikazano korištenje različitih organizacija datoteka pri izradi aplikacije za fiskaliziranje računa.

***KLJUČNE RIJEČI**: programski jezik C++, datoteke, binarne datoteke, tekstualne datoteke, organizacija datoteke, sortiranje.*

ABSTRACT

File organisation is crucial for proper and professional work of the programmers. It is completely common to store the data in the files while creating different programmes, games, web pages and much more. Choosing the best way for data organization in the files can completely increase program performance and efficiency. Thus, this paper explains basic methods of working with files in C++ programming language. Also, there are examples of content exchange in text file, working with different organization of files, sorting a file by method *selection sort*. Lastly, usage of different organisation files while creating application for fiscalization is shown.

***KEYWORDS**: C++ programming language, files, binary files, text files, file organization, sellection sort.*

SADRŽAJ:

1. UVOD

Svaki program koristi bazu podataka kako bi se sačuvali podatci, no oni se ne moraju pohraniti isključivo u bazu podataka, to se može obaviti i u datoteci. Datoteke su osnovni dio današnjih računalnih sustava. Svaki programer morat će pisati programe u kojima se kreira datoteka, upisuje u datoteku i čita iz datoteke, stoga je ovladavanje tim tehnikama ključno.

Prilikom zatvaranja programa, podatci pohranjeni u radnoj memoriji računala bit će izgubljeni, a kako bi trajno bili sačuvani, oni se moraju zapisati u datoteku te pohraniti u jedan od oblika trajne memorije (tvrdi disk, CD-ROM, USB, …).

Operacijskom sustavu datoteka je predstavljena kao imenovana sekcija za pohranu. Toj se sekciji pristupa preko njenog imena i formata koje je izraženo u obliku: <naziv datoteke>.<format>, format predstavlja tip podataka u datoteci.

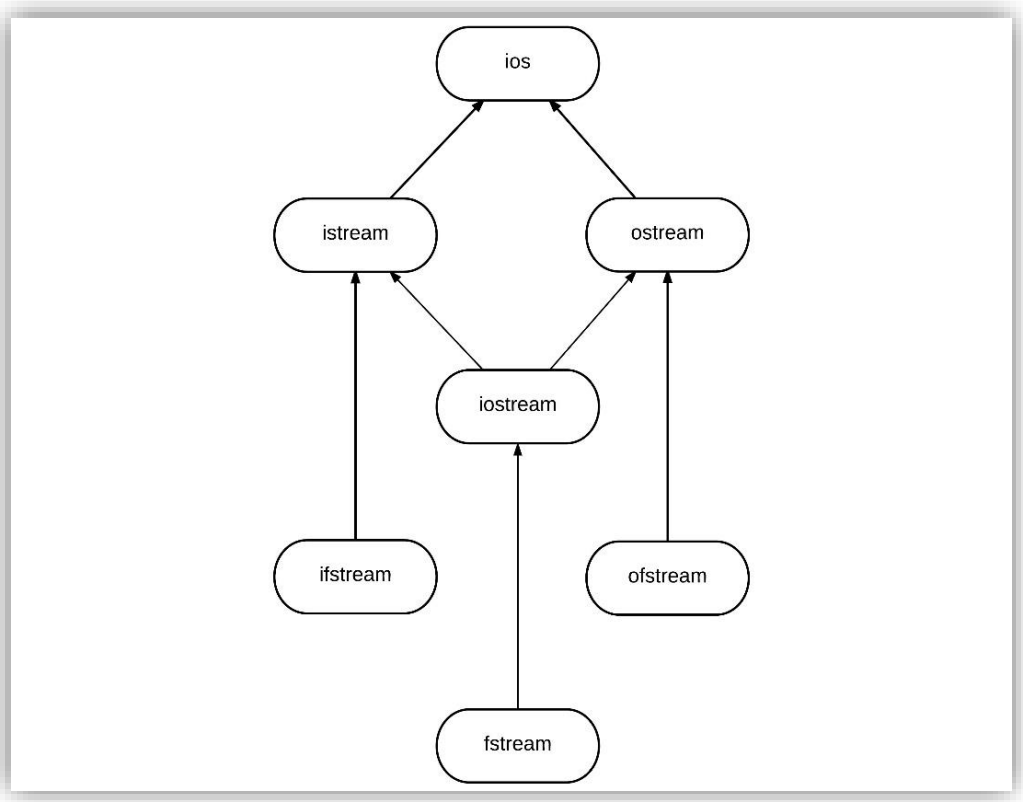
Dva su osnovna tipa datoteka, binarne i tekstualne. U tekstualnim datotekama zapis se vrši formatirano pomoću slijeda ASCII znakova, na isti način kao što se vrši tekstualni ispis na monitoru. Sadržaj tekstualnih datoteka može se pregledati u bilo kojem uređivaču teksta, dok zapis u binarnim datotekama nema taj slučaj. Takav se zapis vrši u istom binarnom obliku, bez konverzije, kao što su kodirane u memoriji računala, a njihov sadržaj može razumjeti samo program, tj. programer koji ih je formirao (Mateljan, 2007).

Datoteke se mogu organizirati na različite načine. Svaki od načina organizacije datoteke ima svoje prednosti i nedostatke. U slijednoj organizaciji datoteke zapis se dodaje na kraj datoteke. Za pristup određenom zapisu potrebno je pročitati datoteku od početka pa sve do traženog zapisa u datoteci. U relativnoj organizaciji datoteke omogućen je direktan pristup zapisu koji se realizira tako da se iz ključa zapisa različitim metodama dođe do adrese zapisa u datoteci. Indeksna organizacija je slična kao i slijedna organizacija datoteke, međutim u indeksnoj organizaciji datoteke pretraživanje je poboljšano stvaranjem indeksne datoteke. Indeksna datoteka sastoji se od ključa zapisa i adrese zapisa u matičnoj datoteci. Pretraživanje u indeksnoj organizaciji datoteka moguće je po različitim atributima zapisa, a za svaki atribut prema kojem se želi pretraživati matična datoteka potrebno je kreirati novu indeksnu datoteku, što zauzima više prostora u memoriji računala.

Prvi se dio rada bavi načinom na koji je implementiran rad s datotekama u programskom jeziku C++. Ono obuhvaća klase za rad s datotekama i datotečne pokazivače, što je detaljnije objašnjeno u drugom i trećem poglavlju. Nadalje, rad s datotekama predočen je u četvrtom poglavlju, a on se sastoji od otvaranja datoteke, provjere otvorenosti datoteke, čitanja i upisa u datoteku te naposljetku zatvaranja iste. Također, u petom se poglavlju na praktičnom primjeru prikazuje zamjena sadržaja u tekstualnoj datoteci. Binarnim se datotekama bavi drugi dio rada, točnije šesto poglavlje. Na početku istoga objašnjene su posebne funkcije za rad s binarnim datotekama, a kako optimalno organizirati datoteku oprimjereno je u nekoliko sljedećih potpoglavlja. U sedmom je poglavlju prikazano sortiranje datoteke u programskom jeziku C++ metodom *selection sort*, a na kraju je prikazan praktični primjer izrade aplikacije za fiskaliziranje računa pomoću različitih organizacija datoteka. On uključuje promjenu stanja na zalihama, izdavanje računa te kreiranje različitih izvještaja.

2. KLASE ZA RAD S DATOTEKAMA

Za razliku od drugih programskih jezika, programski jezik C++ nema definirane naredbe za upis i ispis podataka u datoteku. Naredbe u pojedinim programskim jezicima podržavaju samo ugrađene tipove podataka, no svaki složeniji C++ program sadrži i složenije korisnički definirane tipove podataka. U jeziku C++ upis i ispis podataka realiziran je pomoću ulaznih i izlaznih tokova. Tokovi su zapravo klase definirane u standardnim bibliotekama koje se isporučuju skupa s prevoditeljem. Sadržaj tih klasa je standardiziran i omogućava gotovo sve osnovne ulazno-izlazne operacij, a dodatno se može i proširiti (Motik i Šribar, 1997).



Slika 1. Dijagram klasa za pristup datotekama (Kirch-Prinz i Prinz, 2002, str. 382)

Iz klase ios (početna slova od *Input-Output Stream*) izravno su izvedene klase *istream* i *ostream*. Klasa *istream* koristi se kao osnovna klasa za ulazne tokove dok se klasa *ostream* koristi kao osnovna klasa za izlazne tokove. Klasa *ofstream*, izvedena iz klase *ostream*, podržava upis ugrađenih tipova podataka. Klasa *ifstream*, izvedena iz klase *istream*, podržava učitavanje ugrađenih tipova podataka iz datoteke. Klasa *fstream* podržava čitanje i pisanje istodobno (Motik i Šribar, 1997).

3. DATOTEČNI POKAZIVAČ

Svaka od klasa navedenih u prethodnom poglavlju sadrži datotečni pokazivač koji omogućuje obavljanje operacija čitanja ili upisa u datoteku. Datotečni pokazivač sadrži poziciju unutar datoteke koja je izražena u cijelom broju bajtova od početka datoteke, sve do mjesta upisa u datoteku ili čitanja iz nje. Klasa *ofstream* sadrži datotečni pokazivač za upis u datoteku dok klasa *ifstream* sadrži datotečni pokazivač za čitanje datoteke. Klasa *fstream* sadrži datotečni pokazivač za čitanje datoteke i za upis u datoteku (Radošević, 2007).

3.1. Funkcije za očitavanje vrijednosti datotečnog pokazivača

Za očitavanje vrijednosti datotečnog pokazivača koriste se funkcije:

tellg() – vraća vrijednost pokazivača za čitanje datoteke

tellp() – vraća vrijednost pokazivača za upis u datoteku.

Funkcije za očitavanje vrijednosti pokazivača ne primaju parametre, a vraćaju cjelobrojnu vrijednost tipa pos_type, koja predstavlja trenutnu poziciju pokazivača (Souli, 2007).

3.2. Funkcije za zadavanje vrijednost datotečnog pokazivača

Funkcije za zadavanje vrijednosti datotečnog pokazivača omogućuju da se operacije učitavanja i upisa zapisa u datoteku mogu izvršiti na željenoj poziciji u datoteci, čime je omogućena prekoredna obrada podataka, umjesto podrazumijevane sekvencijalne (Radošević, 2007).

Tablica 1. Preopterećene funkcije za zadavanje vrijednosti datotečnog pokazivača (Stroustrup, 2013, 1085)

NAREDBA	OPIS NAREDBE
tok.seekg(pos);	Vrijednost datotečnog pokazivača za čitanje postavlja se na vrijednost pos, računajući od početka datoteke.
tok.seekp(pos);	Slično kao i prethodna naredba, samo što se mijenja vrijednost pokazivača za upis u datoteku.
tok.seekg(pos, offset);	Vrijednost datotečnog pokazivača za čitanje postavlja se na poziciju pos u smjeru *offset*.
tok.seekp(pos, offset);	Slično kao i prethodna naredba, samo što se mijenja vrijednost pokazivača za upis u datoteku.

U naredbama iz tablice 1, prvi parametar *pos* predstavlja broj bajtova za koji se pomiče pozicija datotečnog pokazivača. Tip ovog parametra je jednak onome koji vraćaju funkcije *tellg* i *tellp*. Drugi je parametar enumeracija tipa *seekdir,* koji predstavlja poziciju od koje se pomiče datotečni pokazivač za vrijednost pos, a može poprimiti bilo koju od vrijednosti zapisanih u tablici 2.

Tablica 2. Konstante vrijednost tipa seekdir (Stroustrup, 2013, 1090)

REFERENTNA POZICIJA	OBJAŠNJENJE
ios::beg	Referentna pozicija je početak datoteke.
ios::cur	Referentna pozicija je trenutna pozicija.
ios::end	Referentna pozicija je kraj datoteke.

4. RAD S DATOTEKAMA

Osnovni princip rada s datotekama u programskom jeziku C++ jest taj da se za svaku fizičku datoteku na disku definira odgovarajući datotečni objekt koji sadrži sve potrebne podatke i funkcije za rad s datotekom (Radošević, 2007).

Kako bi rad s datotekama bio moguć, prije svega mora se uključiti biblioteka:

`#include <fstream>` - biblioteka za rad s datotekama.

4.1. Otvaranje datoteke

Za otvaranje datoteke potrebno je kreirati datotečni objekt i povezati ga s datotekom. Povezivanje datotečnog objekta s datotekom može se izvršiti na dva načina. Prvi je tako da se konstruktoru datotečnog objekta proslijedi naziv datoteke kao parametar, a drugi je pomoću funkcije open().

Tablica 3. Fstream specifične operacije (Lippman, Lajoie i E.Moo, 2015)

NAREDBA	OPIS NAREDBE
fstream tok;	Kreira datotečni objekt koji nije povezan s datotekom.
fstream tok(s);	Kreira *fstream* objekt i otvara datoteku s imenom s. S mora biti pokazivač na znakovni niz koji završava s *null* znakom ('\0').
fstream tok(s, mode);	Slično kao prethodni konstruktor, samo što se datoteka otvara u zadanom modu (mode).
tok.open(s);	Otvara datoteku po imenu s, i povezuje tu datoteku s datotečnim objektom tok. Početni mod otvaranja ovisi o tipu *fstream* objekta tok.
tok.open(s, mode);	Slično kao prethodna naredba, ali datoteka se otvara u zadanom modu (mode).

4.1.1. Modovi otvaranja datoteke

Mod otvaranja datoteke definira dopušteni način rada s datotekom, početnu poziciju datotečnih pokazivača za čitanje i upis, te hoće li se koristiti postojeća datoteka ili otvoriti nova (Radošević, 2007). U klasi *ios* za tip *openmode* definirane su konstantne vrijednosti za otvaranje datoteka navedene u sljedećoj tablici.

Tablica 4. Modovi otvaranja datoteke (http://www.cplusplus.com/reference/ios/ios_base/openmode/)

Zastavica	Značenje
app	(append) Otvara postojeću datoteku za upis na kraju datoteke. Datotečni pokazivač za upis dobiva vrijednost nula kako bi se zaštitio sadržaj datoteke od promjene.
ate	(at-end) Otvara datoteku i postavlja pokazivače na kraj datoteke, te tako postojeći sadržaj nije zaštićen od promjena kao kod moda app.
binary	(binary) Otvara datoteku u binarnom načinu rada.
in	(input) Otvara datoteku za čitanje.
out	(output) Otvara datoteku za upis. Ova zastavica podrazumijeva ios::trunc ako nije kombinirana sa zastavicom ios::in ili ios::app.
trunc	(truncate) Briše sadržaj postojeće datoteke.

Mod otvaranja datoteke predstavljen je kao bitovna maska. Vrijednost bitovne maske može biti bilo koja ispravna kombinacija od gore navedenih zastavica. Uključenjem više zastavica prilikom otvaranja datoteke postiže se korištenjem bitovnog „ili“ operatora „|“, čime se više bitova u bitovnoj maski postavlja na vrijednost jedan (Radošević, 2007).

4.1.2. Podrazumijevani modovi za otvaranje datoteke

Svaka od kasa za rad s datotekama i funkcija *open* koriste podrazumijevane vrijednosti prilikom otvaranja datoteke navedene u tablici 5.

Tablica 5. Podrazumijevani modovi za otvaranje datoteke (Kirch-Prinz i Prinz, 2002, 386)

KLASA	MOD OTVARANJA DATOTEKE
ifstream	ios::in
ofstream	ios::out | ios::trunc
fstream	ios::in | ios::out

4.2. Provjera otvorenosti datoteke

Prilikom otvaranja datoteke može se dogoditi pogreška, npr. datoteka koju se želi pročitati ne postoji na disku ili korisnik nema pravo pristupa toj datoteci. Radi sigurnosti da se može nastaviti rad s datotekom, potrebno je provjeriti stanje datotečnog objekta.

4.2.1. Stanje toka

Sva su stanja datoteke obuhvaćena pobrojenjem definiranim u klasi *ios*, a imaju nazive *goodbit, badbit, failbit i eofbit*. Stanje toka pohranjeno je u bitovnu masku *iostate* kao kombinacija gornjih vrijednosti. Za čitanje stanja pojedinih bitova definirana su četiri funkcijska člana tipa *bool*:

1. eof () – vraća *true*, ako je iz toka za čitanje izlučen znak za kraj datoteke,
2. bad () – vraća *true*, ako operacija nije uspjela zbog neke nepopravljive pogreške,
3. fail () – vraća *true* ako operacija nije bila uspješno obavljena zbog bilo kojeg drugog razloga,
4. good () – vraća *true* ako niti jedan od gornjih uvjeta nije istinit, tj. ako je operacija potpuno uspješno izvedena (Motik i Šribar, 1997).

Svaka operacija nad tokom gdje nije podignuta zastavica good() neće imati efekta. Stanje zastavica, osim s gore navedenim funkcijama, mogu se provjeriti tako da se objekt *iostream* klase koristi kao uvjet. U tom će slučaju uvjet biti istinit ako je stanje *iostream* objekta good(), u suprotnom rad s objektom nije moguć.

4.2.2. Provjera pomoću funkcije is_open()

Na slici 2. prikazan je programski kod u kojem korisnik unosi naziv datoteke. Nakon unosa naziva datoteke, datoteka se otvara i vrši se provjera. Konstruktor objekta *fstream* kao prvi parametar prima *C-style string*. Prilikom predaje naziva datoteke konstruktoru, potrebno je pomoću funkcije c_str() promijeniti naziv datoteke u *C-style string* s *null* znakom na kraju. Nakon otvaranja datoteke vrši se provjera pomoću funkcije is_open, koja vraća *true,* ako je datoteka uspješno povezana s datotečnim objektom.

```
string naziv_datoteke;

cout<<"Unesite naziv datoteke: "; cin>>naziv_datoteke;
fstream tok(naziv_datoteke.c_str());
cout<<"--------------"<<endl;

if(!tok.is_open()){
    cout<<"Greska prilikom otvaranja datoteke!"<<endl;
    return 0;
}
```

Slika 2. Otvaranje datoteke i provjera pomoću funkcije is_open()

4.3. Čitanje i upis u tekstualnu datoteku

Kada je datotečni objekt povezan s datotekom, te ako je provjera uspješno izvršena, može se započeti rad s datotekom, bilo da se radi o čitanju datoteke ili o upisu podataka u datoteku.

4.3.1. Čitanje iz tekstualne datoteke

Čitanje tekstualne datoteke postiže se metodom *getline* datotečnog objekta. S obzirom na to da tekstualna datoteka sadrži zapise znakovnog niza, sadržaj se učitava u odgovarajući znakovni niz, koji predstavlja pojedini redak iz tekstualne datoteke. Znakovni niz *buffer* koristi se za učitavanje pojedinih zapisa iz tekstualne datoteke, koji su međusobno odvojeni znakom <eoln>. Argumenti metode *getline* su znakovni niz koji se učitava, veličina znakovnog niza te znak za terminiranje reda „\n“. Kontrolni znak „\n“ označava kraj retka u tekstualnoj datoteci. Kontrola kraja datoteke postiže se metodom eof. Metoda eof vraća vrijednost logičke istine tek kad čitanje zapisa iz datoteke ne uspije (Radošević, 2007). Također postoji preopterećena inačica funkcije *getline* za objekte tipa *string*. Poziv te funkcije nešto je drugačiji, a parametri su *istream* objekt iz kojeg se izlučuje znakovni niz i objekt tipa *string* u kojega će se spremiti.

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

int main(){

    ifstream tok("File.txt");

    if(!tok){
        cout<<"Greska!"<<endl;
        return 1;
    }

    char buffer[80];
    while(!tok.eof()){
        tok.getline(buffer, sizeof(buffer), '\n');
        cout<<buffer<<endl;
    }

    system("pause");
    return 0;
}
```

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

int main(){

    ifstream tok("File.txt");

    if(!tok){
        cout<<"Greska!"<<endl;
        return 1;
    }

    string var;

    while(getline(tok, var)){
        cout<<var<<endl;
    }

    system("pause");
    return 0;
}
```

Slika 3. Čitanje tekstualne datoteke pomoću funkcije getline()

4.3.2. Upis u tekstualnu datoteku

Upis u tekstualnu datoteku postiže se jednostavnim umetanjem teksta u datotečni tok pridružen datotečnom objektu i analogan je odgovarajućem konzolnom ispisu teksta pomoću objekta *cout* (Radošević, 2007).

```
ofstream tok("datoteka.txt");

if(!tok){
    cout<<"Greska!"<<endl;
    return 1;
}
int broj = 5;
tok<<"Ovo je prvi redak tekstualne datoteke."<<endl;
tok<<"Ovo je drugi redak tekstualne datoteke."<<endl;
tok<<"Jeli ovo treci redak?";
tok<<endl<<"U tok mozemo ubaciti i brojeve: "<<broj<<"\n";
tok<<"Ovo je zadnji redak tekstualne datoteke.";

tok.clear();
tok.close();

system("pause");
return 1;
```

Slika 4. Upis u tekstualnu datoteku (Radošević, 2007, 63)

```
Ovo je prvi redak tekstualne datoteke.
Ovo je drugi redak tekstualne datoteke.
Jeli ovo treci redak?
U tok mozemo ubaciti i brojeve: 5
Ovo je zadnji redak tekstualne datoteke.
```

Slika 5. Sadržaj tekstualne datoteke nakon upisa

Na slici pet može se vidjeti da su pojedini zapisi različite duljine i da su svi podaci upisani u tekstualnom obliku, uključujući i vrijednost cjelobrojne varijable broj, što je posljedica odgovarajuće izlazne konverzije kojom se binarni zapis vrijednosti varijable konvertira u niz znakova (Radošević, 2007).

4.4. Zatvaranje datoteke

Nakon završetka rada s datotekom potrebno je zatvoriti tok prema datoteci. U slučaju da tok prema datoteci nije zatvoren, a program neočekivano prekine s radom, moguće je da se podatci neće upisati u datoteku. Dakle, nakon prestanka korištenja datoteke potrebno je zatvoriti tok prema datoteci pomoću funkcije *close*(). Ova funkcija ne prima parametre, a njena uloga je pražnjenje memorijskog spremnika i upisivanje njegovog sadržaja u datoteku. Nakon zatvaranja datotečnog objekta, isti taj objekt može se koristiti za otvaranje nove datoteke. Prije zatvaranja toka prema datoteci preporuča se maknuti tok iz greške pomoću funkcije *clear*().

5. ZAMJENA SADRŽAJA U TEKSTUALNOJ DATOTECI

Radi demonstracije rada s tekstualnim datotekama prikazat će se primjer u kojem se od korisnika zahtjeva unos naziva datoteke. Nakon unosa, sadržaj datoteke ispisuje se na ekran te se od korisnika zahtjeva unos traženog i zamjenskog znakovnog niza. Pritom se sve pojave traženog znakovnog niza zamjenjuju sa zamjenskim znakovnim nizom. Na kraju, ispisuje se broj izvršenih izmjena u datoteci te ponovno se ispisuje sadržaj datoteke nakon izvršenih izmjena.

5.1. Biblioteke i funkcije korištene u programu

Biblioteke:

1. *iostream* – biblioteka za rad s ulazno izlaznim tokovima,
2. *fstream* – biblioteka za rad s datotekama,
3. *vector* – biblioteka u kojoj je definiran spremnik *vector,*
4. *string* – biblioteka za rad sa znakovnim nizovima,
5. *cstdlib* – biblioteka u kojoj se nalaze funkcije općenite namjene.

Funkcije:

1. provjera (fstream &tok) – funkcija koja provjerava povezanost toka s datotekom, u slučaju greške prekida rad programa,
2. zatvaranje_toka (fstream &tok) – funkcija koja zatvara tok proslijeđen kao parametar.

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <string>
5  #include <cstdlib>
6  using namespace std;
7
8  void provjera(fstream &tok){
9      if(!tok){
10         cout<<"Greska prilikom otvaranja datoteke"<<endl;
11         exit(0);
12     }
13 }
14 void zatvaranje_toka(fstream &tok){
15     tok.clear();
16     tok.close();
17 }
```

Slika 6. Biblioteke i funkcije korištene u programu

5.2. Priprema podataka

Priprema podataka sastoji se od deklariranja svih potrebnih varijabli koje će se koristiti u programu. Nakon unosa naziva datoteke slijedi otvaranje datoteke i provjera. Ako je datoteka uspješno otvorena učitava se redak po redak, koristeći iteraciju tipa *while,* iz datoteke u varijablu „red“ tipa *string,* pomoću funkcije *getline*. Nakon što se učita redak, odmah se sprema u „vektor“ i ispisuje na ekran. Nakon ispisa datoteke, zatvara se datotečni objekt prema datoteci kako bi se poslije mogao ponovno koristiti za upis. Na kraju se od korisnika zahtjeva unos traženog i zamjenskog znakovnog niza.

```
19  int main(){
20
21      string red, naziv_datoteke, trazeni_niz, zamjenski_niz;
22      vector <string> vektor;
23      vector <string>::iterator it;
24
25      cout<<"Unesite naziv datoteke: "; cin>>naziv_datoteke;
26      fstream tok(naziv_datoteke.c_str());
27      provjera(tok);
28
29      cout<<"Ispis datoteke: "<<endl;
30      cout<<"--------------------"<<endl;
31      while(getline(tok, red)){
32          vektor.push_back(red);
33          cout<<red<<endl;
34      }
35      cout<<"--------------------"<<endl;
36      zatvaranje_toka(tok);
37
38      cin.ignore();
39      cout<<"Unesite rijec koju zelite zamjeniti: ";
40      getline(cin, trazeni_niz);
41      cout<<"Unesite novu rijec: ";
42      getline(cin, zamjenski_niz);
```

Slika 7. Priprema podataka za obradu

5.3. Obrada u petlji

Kada je „vektor“ napunjen sa stringovima koji sadrže retke iz datoteke, vanjskom *for* petljom prolazi se kroz svaki element vektora, a unutarnjom *for* petljom učitava se znak po znak iz retka. Zatim, provjerava se odgovara li znak iz retka znaku iz traženog znakovnog niza. Ako su znakovi jednaki, povećava se varijabla brojača za jedan, a u protivnom se postavlja vrijednost brojača na nula. Ako je vrijednost brojača jednaka veličini traženog znakovnog niza, onda se on nalazi u retku, te pomoću funkcije *replace* koja je definirana u biblioteci *string,* zamjenjuje se traženi znakovni niz iz retka sa zamjenskim znakovnim nizom. Prvi parametar funkcije *replace* je pozicija prvog znaka koji će biti zamijenjen, drugi parametar je broj znakova koji se zamjenjuje, a treći parametar je zamjenski niz koji se ubacuje na mjesto traženog niza. Nakon izvršene izmjene varijabla broj_zamjena poveća se za jedan a brojač se postavlja na vrijednost nula, kako bi se omogućile ostale izmjene u trenutnom retku.

```
44      int broj_zamjena = 0;
45      int brojac = 0;
46      for(it = vektor.begin(); it != vektor.end(); it++){
47          for(int i = 0; i < (*it).size(); i++){
48              if((*it)[i] == trazeni_niz[brojac]){
49                  brojac++;
50              }
51              else{
52                  brojac = 0;
53              }
54              if(brojac == trazeni_niz.size()){
55                  (*it).replace(i-trazeni_niz.size()+1,
56                  trazeni_niz.size(), zamjenski_niz);
57                  broj_zamjena++;
58                  brojac = 0;
59              }
60          }
61      }
```

Slika 8. Obrada u petlji

5.4. Završne radnje

Nakon što su sve izmjene izvršene, sadržaj se vektora upisuje u datoteku, ponovno se otvara datoteka, ali ovaj puta samo u modu za upis, kako bi se postojeći sadržaj datoteke obrisao. Nakon upisa zatvara se tok prema datoteci i otvara se tok u modu za čitanje, kako bi se sadržaj datoteke ponovno ispisao na ekranu.

```
62      cout<<"Broj zamjena u datoteci: "<<broj_zamjena<<endl;
63
64      tok.open(naziv_datoteke.c_str(), ios::out);
65      provjera(tok);
66      for(it = vektor.begin(); it != vektor.end(); it++){
67          tok<<*it<<endl;
68      }
69      zatvaranje_toka(tok);
70
71      cout<<"Ispis datoteke nakon izmjene:"<<endl;
72      cout<<"--------------------"<<endl;
73      tok.open(naziv_datoteke.c_str(), ios::in);
74      provjera(tok);
75      while(getline(tok, red)){
76          cout<<red<<endl;
77      }
78      zatvaranje_toka(tok);
79      cout<<"--------------------"<<endl;
80
81      system("pause");
82      return 0;
83  }
```

Slika 9. Završne radnje

```
Unesite naziv datoteke: datoteka.txt
Ispis datoteke:
--------------------
Svaki programer morat ce pisati programe u kojima se kreira datoteka.
Datoteka je mjesto za pohranu podataka.
U ovome redu rijec datoteka pojavljuje se dva puta, datoteka.
--------------------
Unesite rijec koju zelite zamjeniti: datoteka
Unesite novu rijec: dokument
Broj zamjena u datoteci: 3
Ispis datoteke nakon izmjene:
--------------------
Svaki programer morat ce pisati programe u kojima se kreira dokument.
Datoteka je mjesto za pohranu podataka.
U ovome redu rijec dokument pojavljuje se dva puta, dokument.
--------------------
Press any key to continue . . .
```

Slika 10. Demonstracija programa

6. BINARNE DATOTEKE

U binarnim datotekama, bajtovi ne moraju nužno predstavljati znakove kao što je slučaj u tekstualnim datotekama. Skupine bajtova mogu predstavljati i druge tipove podataka, kao što su brojevi, slika ili zvuk. Binarne datoteke nisu podijeljene u retke i u binarnim datotekama nema oznake za kraj retka ili oznake za kraj datoteke, nego se svi bajtovi tretiraju jednako (King. N., 2008).

Svi su zapisi u binarnim datotekama istog, zadanog tipa pa time i veličine te se nižu redoslijedom unosa. Za takve zapise potrebno je kreirati odgovarajući tip podataka, što se u C++ jeziku može napraviti deklaracijom strukture ili klase (Radošević, 2007).

```
struct tstudent{
    int maticni_broj;
    char ime[50];
    char prezime[50];
    char status[20];
};
```

Slika 11. Slog datoteke s podacima o studentu

Na slici 11. prikazana je struktura studenta veličine 124 bajta koja sadrži četiri atributa.

Binarne datoteke kreiraju se i otvaraju isto kao i tekstualne datoteke, no prilikom otvaranja datoteke potrebno je navesti parametar ios::binary. U binarnim datotekama upis podataka pomoću operatora umetanja ili funkcije *getline()* nije efikasan, već su unutar biblioteke *fstream* definirane funkcije za upis i čitanje binarnih podataka.

6.1. Funkcije za slijedno upisivanje i čitanje binarnih podataka

Kada želimo pročitati cijeli blok podataka iz datoteke, tada se koristi funkcija *read*(), koja je članica klase *ifstream* naslijeđene iz klase *istream*. Za upis cijelog bloka podataka koristi se funkcija *write*() koja je članica klase *ofstream* naslijeđene iz klase *ostream*. Objekti klase *fstream* sadrže obje funkcije. Potpis navedenih funkcija može se vidjeti u nastavku;

```
istream& read (char* s, streamsize n);
ostream& write (const char* s, streamsize n);
```

Funkcija *read* izlučuje iz toka memorijski blok veličine n i pohranjuje ga u lokaciju na koju pokazuje pokazivač s. Funkcija *write* upisuje memorijski blok veličine n od adrese na koju pokazuje pokazivač s u datotečni objekt. Prvi parametar u obje funkcije je pokazivač na *char,* dok je drugi parametar *integer* (www.cplusplus.com/reference/ostream).

Nakon izvršenja prethodnih funkcija, datotečni se pokazivač pomiče za jednu memorijsku lokaciju veličine učitanog memorijskog bloka, čime je omogućen različit način organiziranja datoteke.

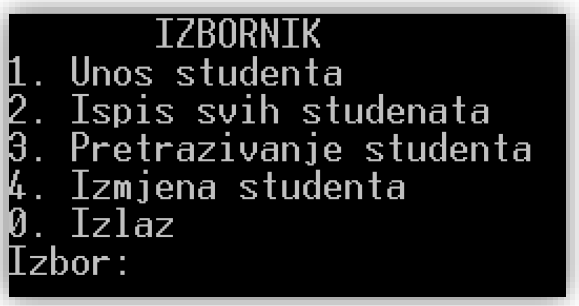
6.2. Organizacija datoteka

Pravilnom organizacijom datoteke dobije se unaprijed definiran i efikasan način kojim se pohranjuje, dohvaća i uređuje zapis.

Postoje različiti načini na koje se podaci u datoteci mogu organizirati, a svaki od njih ima određene prednosti i nedostatke (H. Patil, 2009).

C++ omogućuje slijednu i prekorednu obradu datoteka s fiksnom veličinom zapisa. Indeksne datoteke nisu podržane instrumentarijem jezika C++, ali se mogu implementirati programski (Radošević, 2007).

Za demonstriranje različitih organizacija datoteka u radu je napravljen po jedan program za slijednu, relativnu i indeksnu organizaciju datoteke u kojem su podržane osnovne operacije za rad s datotekama kao što su unos, pretraživanje, izmjena i ispis svih zapisa.



Slika 12. Glavni izbornik programa

6.2.1. Slijedna organizacija datoteka

U slijednoj se organizaciji datoteke zapisi upisuju u datoteku redoslijedom unosa. Novi se zapis uvijek upisuje na kraj datoteke. Zapis koji se nalazi na prvoj poziciji je najstariji zapis, dok zapis na zadnjoj poziciji je zapis koji je posljednji unesen u datoteku. Zapisima u ovoj vrsti datoteke ne može se pristupiti direktno nego, da bi se pristupilo nekom od zapisa mora se pročitati svaki zapis od početka datoteke do traženog zapisa.

6.2.1.1. Dodavanje zapisa u slijedno organiziranu datoteku

Za dodavanje zapisa u slijedno organiziranu datoteku potrebno je učitati zapis od korisnika, otvoriti datoteku koristeći modove *out* i *app* kako bi se pozicija datotečnog pokazivača za upis postavila na prvu praznu poziciju iza kraja datoteke. Nakon uspješnog otvaranja datoteke, pomoću funkcije *write* potrebno je upisati zapis u datoteku te zatvoriti tok prema datoteci.

```
void unos(){
    tstudent student;
    cout<<"Odabrali ste unos studenta"<<endl;
    cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
    cin.ignore();
    cout<<"Unesite ime: "; cin.getline(student.ime, 50);
    cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
    cout<<"Unesite status: "; cin.getline(student.status, 20);

    fstream tok("studenti.dat", ios::binary | ios::out | ios::app);
    if(!tok.is_open()) {
        cout<<"Greska prilikom otvaranja datoteke!"<<endl;
    }
    tok.write((char*)&student, sizeof (tstudent));
    cout<<"Student je unesen!"<<endl;
    tok.clear();
    tok.close();
}
```

Slika 13. Dodavanje zapisa u slijedno organiziranu datoteku

6.2.1.2. Pretraživanje u slijedno organiziranoj datoteci

Pretraživanje zapisa vrši se pomoću ključa. Nakon što korisnik unese ključ po kojem se pretražuje datoteka, otvara se datoteka u modu za čitanje. Datotečni pokazivač za čitanje postavljen je na početku datoteke i ima vrijednost nula. U iteraciji tipa *while* slijedno se čitaju zapisi i uspoređuje se ključ svakog zapisa s ključem kojega je upisao korisnik. Ako ključ jednog od pročitanih zapisa odgovara ključu kojeg je upisao korisnik, ispisuju se svi atributi zapisa i zatvara se tok prema datoteci. Ako su svi zapisi iz datoteke pročitani, a pritom se ne pronađe traženi zapis, ispisuje se odgovarajuća poruka i zatvara se tok prema datoteci.

```
void pretrazivanje(){
    ifstream tok("studenti.dat", ios::binary);
    if(!tok.is_open()) {
        cout<<"Greska prilikom otvaranja datoteke!"<<endl;
        return;
    }
    int kljuc;
    cout<<"Odabrali ste pretrazivanje studenata!"<<endl;
    cin.ignore();
    cout<<"Unesite maticni broj studenta: "; cin>>kljuc;

    tstudent student;
    while(tok.read((char*)&student, sizeof (tstudent))){
        if(student.maticni_broj == kljuc){
            cout<<"Student je pronagen: "<<endl;
            cout<<"Maticni broj: "<<student.maticni_broj<<endl;
            cout<<"Ime: "<<student.ime<<endl;
            cout<<"Prezime: "<<student.prezime<<endl;
            cout<<"Status: "<<student.status<<endl;
            tok.clear();
            tok.close();
            return;
        }
    }
    cout<<"Student nije pronagen. "<<endl;
    tok.clear();
    tok.close();
}
```

Slika 14. Slijedno pretraživanje datoteke

6.2.1.3. Izmjena zapisa u slijedno organiziranoj datoteci

Zapis u datoteci je izmijenjen ako jedan od njegovih atributa promijeni vrijednost. Postupak za izmjenu zapisa sličan je postupku za pretraživanje. Pri izmjeni zapisa potrebno je otvoriti datoteku s modovima *out* i *in*. Svaki se zapis čita slijedno od početka datoteke do traženog zapisa. Pri pronalasku odgovarajućeg zapisa, kojega je potrebno izmijeniti, učitavaju se atributi istoga. Zatim, vrijednost datotečnog pokazivača za upis smanjuje se za veličinu jednog zapisa te se pomoću funkcije *write* prepisuje sadržaj starog zapisa sa zapisom kojeg je upisao korisnik.

```
void izmjena(){
    fstream tok("studenti.dat", ios::binary | ios::out | ios::in);
    if(!tok.is_open()) {
        cout<<"Greska prilikom otvaranja datoteke!"<<endl;
        return;
    }
    int kljuc;
    cout<<"Odabrali ste izmjenu studenta!"<<endl;
    cout<<"Unesite maticni broj studenta kojeg zelite izmijeniti: ";
    cin>>kljuc;

    tstudent student;
    while(tok.read((char*)&student, sizeof (tstudent))){
        if(student.maticni_broj == kljuc){
            cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
            cin.ignore();
            cout<<"Unesite ime: "; cin.getline(student.ime, 50);
            cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
            cout<<"Unesite status: "; cin.getline(student.status, 20);
            tok.seekp(-1 * sizeof(tstudent), ios::cur);
            tok.write((char*)&student, sizeof(tstudent));
            tok.clear();
            tok.close();
            cout<<"Student je izmijenjen!"<<endl;
            return;
        }
    }
    cout<<"Student nije pronagen!"<<endl;
    tok.clear();
    tok.close();
}
```

Slika 15. Izmjena zapisa u slijedno organiziranoj datoteci

6.2.2. Relativna organizacija datoteke

Relativne su datoteke dizajnirane kako bi dohvaćanje podataka bilo što jednostavnije i efikasnije. Njihova prednost je u tome što je omogućen direktan pristup zapisu u velikoj količini podataka.

S obzirom na to da se radi o datoteci s fiksnom veličinom zapisa, adresa se svakog zapisa može dobiti množenjem rednog broja zapisa s njegovom veličinom. Kod relativno organizirane datoteke, pojedinom se zapisu može direktno pristupiti ako se zna gdje je u datoteci upisan, što se postiže tako da se adresa pojedinog zapisa izračuna na temelju vrijednosti ključa.

Pristup zapisu u relativnoj datoteci usporediv je s pristupom elementu polja, s tim da se koristi redni broj zapisa umjesto indeksa. Kao i kod polja, postavlja se problem dimenzioniranja, odnosno određivanja potrebnog broja zapisa u datoteci.

U matičnoj datoteci studenata, ključ je matični broj studenta čiji raspon mogućih vrijednosti može biti od jedan do 10 000. Datoteka se može organizirati tako da redni broj zapisa u datoteci bude jednak vrijednosti ključa. Time se dobije datoteka od 10 000 zapisa i velikim brojem praznih mjesta. Da bi se izbjegla ogromna veličina datoteke i smanjio broj praznih mjesta u datoteci potrebno je reducirati broj zapisa uzimajući u obzir da je stvarni broj zapisa daleko manji od maksimalno mogućeg. Zbog toga, potrebno je prije upisa podataka u datoteku odrediti veličinu datoteke izraženu u broju zapisa te kreirati datoteku praznih zapisa, s time da se svim zapisima vrijednost ključa postavi na nula. Kada je kreirana datoteka s praznim zapisima ona zauzima jednaku količinu memorije u svakom trenutku bez obzira koliko zapisa je upisano u datoteku.

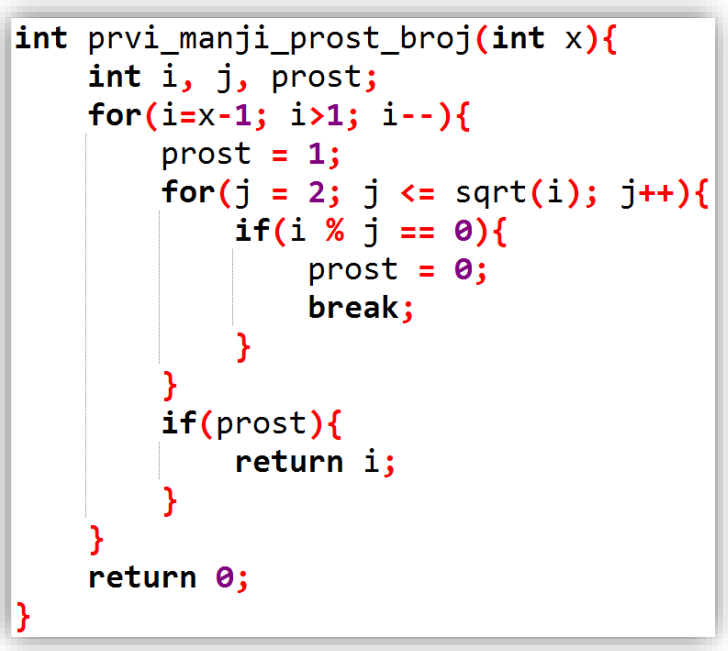
Prilikom unosa podataka, potrebno je izračunati redni broj zapisa. Jedan od postupaka izračunavanja rednog broja zapisa na temelju vrijednosti ključa i veličine datoteke temelji se na upotrebi prostog broja. Redni broj zapisa metodom prostog broja dobije se tako da se izračuna ostatak dijeljenja ključa s prvim manjim prostim brojem od veličine datoteke. Ako je zapis na dobivenoj poziciji zauzet, tada se traži prva prazna pozicija u datoteci. Upotreba prostog broja reducira broj duplikata zbog smanjivanja vjerojatnosti ponavljanja ostatka dijeljenja. U slučaju ostatka dijeljenja s brojevima koji završavaju s nulom, često bi se dobivao ostatak nula, što bi uzrokovalo velik broj duplikata i time kasnije smanjenje efikasnosti pretraživanja (Radošević, 2007).

6.2.2.1. Inicijalizacija datoteke

U slučaju da matična datoteka studenata ne postoji na disku, potrebno ju je kreirati. Nakon što korisnik unese veličinu datoteke u broju zapisa, ovisno o njegovim potrebama, u sve atribute varijable student upisuje se *null* vrijednost. U ključ zapisa, matični broj upisuje se brojčana vrijednost nula čime se poslije prepoznaje prazan zapis. Dok u polja za ime, prezime i status studenta upisuje se prazan znakovni niz. Nakon stvaranja varijable student s *null* vrijednostima, kreira se datoteka otvaranjem u modu za upis, te varijabla student upisuje se u datoteku dok se ne dobije veličina datoteke u broju zapisa koju je unio korisnik. Nakon otvaranja datoteke, pronalazi se prvi manji prosti broj od veličine datoteke pomoću funkcije prikazane na slici 17. te se zatvara tok prema datoteci. Ako je datoteka već inicijalizirana, potrebno je odrediti prost broj koji se koristi za direktan dohvat elemenata iz datoteke. Prost broj dobije se tako da se funkciji prvi_manji_prost_broj preda kao parametar veličina datoteke u broju zapisa, koja se izračunava dijeljenjem veličine datoteke s veličinom jednog zapisa u istoj.

```
fstream tok;
tok.open("studenti_relativna.dat", ios::in | ios::binary);

if(!tok){
    cout<<"Kreiranje datoteke praznih zapisa"<<endl;
    int velicina_datoteke;
    cout<<"Unesite velicinu datoteke u broju zapisa: ";
    cin>>velicina_datoteke;
    tok.open ("studenti_relativna.dat", ios::out | ios::binary);
    tstudent student;
    student.maticni_broj = 0;
    student.ime[0] = '\0';
    student.prezime[0] = '\0';
    student.status[0] = '\0';
    for(int i = 0; i < velicina_datoteke; i++){
        tok.write((char*)&student, sizeof(tstudent));
    }
    prost_broj = prvi_manji_prost_broj(velicina_datoteke);
    tok.clear();
    tok.close();
    cout<<"Datoteka je kreirana!"<<endl;
    system("pause");
}else{
    tok.seekg(0, ios::end);
    prost_broj = prvi_manji_prost_broj(tok.tellg()/sizeof(tstudent));
    tok.clear();
    tok.close();
}
```

Slika 16. Inicijalizacija datoteke



```
int prvi_manji_prost_broj(int x){
    int i, j, prost;
    for(i=x-1; i>1; i--){
        prost = 1;
        for(j = 2; j <= sqrt(i); j++){
            if(i % j == 0){
                prost = 0;
                break;
            }
        }
        if(prost){
            return i;
        }
    }
    return 0;
}
```

Slika 17. Funkcija za pronalaženje prvog manjeg prostog broja (Radošević, 2007, 77)

U funkciji se pronalazi prvi manji prosti broj od broja proslijeđenog kao parametar funkcije. Unutar funkcije testira se svaki pojedini broj i, počevši s vrijednošću x, smanjujući ga do dva. Unutar petlje polazi se od pretpostavke da je broj prost, zatim se u unutarnjoj iteraciji tipa *for* provjerava djeljivost broja i sa svakim pojedinim j u rasponu od dva do korijena od i. Ako se utvrdi da je i djeljiv sa j, tada i nije prost broj i slijedi prijevremeni izlazak iz petlje pomoću *break*. Nakon izlaza iz unutarnje iteracije tipa *for* provjerava se vrijednost varijable *prost*, ako je i prost broj, funkcija vraća vrijednost i (Radošević, 2007).

6.2.2.2. Unos zapisa u relativno organiziranu datoteku

Unos zapisa u relativnu datoteku započinje korisnikovim unosom atributa o zapisu studenta, nakon čega se korištenjem metode temeljene na prostom broju, iz ključa zapisa, pronalazi adresa u datoteci za upis. Adresa se dobije izračunavanjem ostatka dijeljenja matičnog broja studenta s prostim brojem. Moguće je da na odgovarajućoj adresi u datoteci već postoji zapis, pa je to potrebno provjeriti čitanjem datoteke. U tu svrhu datoteka se otvara kao ulazno-izlazna. Pomoću iteracije tipa *do-while* čitamo zapis po zapis sve dok se ne pronađe prvi prazan zapis u datoteci ili dok nije kraj datoteke. Nakon izlaska iz petlje očitava se adresa zadnjeg pročitanog zapisa iz datoteke te se datotečni pokazivač za upis pomjera na poziciju za unos zapisa. Ako je pronađen prazan zapis i uspješno izvršena operacija upisa, ispisuje se odgovarajuća poruka i zatvara se tok prema datoteci. Ako nije pronađen prazan zapis prije kraja datoteke, operacija upisa neće uspjeti i ispisuje se odgovarajuća poruka na ekran (Radošević, 2007).

```
void unos(){
    tstudent student, temp;
    int redni_broj_zapisa;
    cout<<"Odabrali ste unos studenta"<<endl;
    cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
    cin.ignore();
    cout<<"Unesite ime: "; cin.getline(student.ime, 50);
    cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
    cout<<"Unesite status: "; cin.getline(student.status, 20);

    redni_broj_zapisa = student.maticni_broj % prost_broj;
    fstream tok;
    tok.open("studenti_relativna.dat", ios::in | ios::out | ios::binary);
    tok.seekg(redni_broj_zapisa * sizeof(tstudent));
    do{
        tok.read((char*)&temp, sizeof(tstudent));
    }while(temp.maticni_broj > 0 && !tok.eof());
    redni_broj_zapisa = tok.tellg() / sizeof(tstudent) - 1;
    tok.seekp(redni_broj_zapisa * sizeof(tstudent));
    if(tok.write((char*)&student, sizeof(tstudent))){
        tok.clear();
        tok.close();
        cout<<"Student je unesen!"<<endl;
    }else{
        cout<<"Student nije unesen!"<<endl;
    }
}
```

Slika 18. Unos zapisa u relativno organiziranu datoteku (Radošević, 2007, 78)

6.2.2.3. Ispis svih zapisa iz relativno organizirane datoteke

Ispis svih zapisa iz datoteke uvijek se vrši slijedno od početka do kraja datoteke. Za ispis svih zapisa potrebno je otvoriti datoteku za čitanje u modu *ios::in*, nakon što je datoteka otvorena pomoću *while* petlje učitava se zapis po zapis sve dok se ne dođe do kraja datoteke. Ako operacija čitanje ne uspije, funkcija *read* će vratiti vrijednost nula te će *while* petlja završiti s radom. Unutar *while* petlje provjerava se ključ zapisa. U ovom se slučaju provjerava vrijednost matičnog broja studenta. Ako je veći od nula ispisuju se svi atributi zapisa na ekran. Nakon ispisa svih zapisa zatvara se tok prema datoteci.

```
void ispis(){
    fstream tok("studenti_relativna.dat", ios::binary | ios::in);
    if(!tok.is_open()) {
        cout<<"Greska prilikom otvaranja datoteke!"<<endl;
        return;
    }
    tstudent student;
    while(tok.read((char*)&student, sizeof (tstudent))){
        if(student.maticni_broj > 0){
            cout<<student.maticni_broj<<" "<<student.ime;
            cout<<" "<<student.prezime<<" "<<student.status<<endl;
        }
    }
    tok.clear();
    tok.close();
}
```

Slika 19.Ispis svih zapisa iz relativno organizirane datoteke (Radošević, 2007, 80)

6.2.2.4. Pretraživanje relativno organizirane datoteke

Datoteka se pretražuje prema ključu zapisa, odnosno prema matičnom broju studenta. Kao kod upisa podatka u datoteku i ovdje se koristi metoda temeljena na prostom broju za izračun adrese zapisa kako bi mi su moglo direktno pristupiti. Kao i kod upisa podataka u datoteku mora se uzeti u obzir mogućnost postojanja duplikata. Tada se slijedno čitaju zapisi do pronalaska zapisa s odgovarajućim ključem, ili do prvog praznog zapisa u slučaju da traženi zapis nije pronađen (Radošević, 2007).

```
void pretrazivanje(){
    int kljuc;
    cout<<"Odabrali ste pretrazivanje studenata!"<<endl;
    cout<<"Unesite maticni broj studenta: "; cin>>kljuc;
    tstudent student;
    fstream tok("studenti_relativna.dat", ios::in | ios::binary);
    int redni_broj_zapisa = kljuc % prost_broj;
    tok.seekg(redni_broj_zapisa * sizeof(tstudent));
    do{
        tok.read((char*)&student, sizeof(tstudent));
        if(student.maticni_broj == kljuc){
            cout<<"Student je pronagen: "<<endl;
            cout<<"Maticni broj: "<<student.maticni_broj<<endl;
            cout<<"Ime: "<<student.ime<<endl;
            cout<<"Prezime: "<<student.prezime<<endl;
            cout<<"Status: "<<student.status<<endl;
            tok.clear();
            tok.close();
            return;
        }
    }while(student.maticni_broj > 0 && !tok.eof());
    cout<<"Student nije pronagen. "<<endl;
    tok.clear();
    tok.close();
}
```

Slika 20. Funkcija za pretraživanje relativno organizirane datoteke (Radošević, 2007, 82)

Nakon unosa matičnog broja prema kojem se pretražuje datoteka, otvara se datoteka u modu za čitanje. Iz matičnog broja izračunava se adresa zapisa metodom prostog broja i pomiče se datotečni pokazivač. Nakon što je datotečni pokazivač za čitanje postavljen na odgovarajuću adresu slijedi iteracija tipa *do-while*, unutar koje se slijedno čita datoteka u potrazi za traženim matičnim brojem studenta. Ako je matični broj studenta pronađen, ispisuju se svi atributi zapisa, zatvara se tok prema datoteci i pomoću naredbe *return* slijedi izlaz iz funkcije. U slučaju da zapis s odgovarajućim ključem nije pronađen, ispisuje se odgovarajuća poruka i zatvara se datoteka.

6.2.2.5. Zamjena zapisa u relativno organiziranoj datoteci

Postupak za pristupanje elementu koji se želi izmijeniti jednak je postupku za pretraživanje relativno organizirane datoteke. Dakle, kada se pronađe zapis koji je potrebno izmijeniti, postupak za izmjenu zapisa jednak je postupku za izmjenu zapisa u slijedno organiziranoj datoteci.

```
void izmjena(){
    fstream tok("studenti_relativna.dat", ios::in | ios::out | ios::binary);
    if(!tok.is_open()) {
        cout<<"Greska prilikom otvaranja datoteke!"<<endl;
        return;
    }
    int kljuc;
    cout<<"Odabrali ste izmjenu studenta!"<<endl;
    cout<<"Unesite maticni_broj studenta kojeg zelite izmijeniti: ";
    cin>>kljuc;
    tstudent student;

    int redni_broj_zapisa = kljuc % prost_broj;
    tok.seekg(redni_broj_zapisa * sizeof(tstudent));
    do{
        tok.read((char*)&student, sizeof(tstudent));
        if(student.maticni_broj == kljuc){
            cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
            cin.ignore();
            cout<<"Unesite ime: "; cin.getline(student.ime, 50);
            cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
            cout<<"Unesite status: "; cin.getline(student.status, 20);
            tok.seekp(-1 * sizeof(tstudent), ios::cur);
            tok.write((char*)&student, sizeof(tstudent));
            tok.clear();
            tok.close();
            cout<<"Student je izmijenjen!"<<endl;
            return;
        }
    }while(student.maticni_broj > 0 && !tok.eof());
    cout<<"Student nije pronagen. "<<endl;
    tok.clear();
    tok.close();
}
```

Slika 21. Funkcija za izmjenu sadržaja u relativno organiziranoj datoteci

6.2.3. Indeksna organizacija datoteke

Relativna organizacija datoteke omogućuje direktan pristup zapisu, ali ima i mnoge nedostatke. Relativne datoteke mogu se pretraživati samo po ključu, no često postoji potreba za pretraživanjem podataka i po nekom drugim atributu osim ključa. Kada ključ podatka nije brojčana vrijednost teško je dobiti adresu zapisa iz ključa. Relativne datoteke imaju puno praznih zapisa u datoteci te su fiksne veličine. Zbog toga je uveden indeks kao pomoćna struktura koja omogućuje pronalaženje adresa pojedinih zapisa u matičnoj datoteci na temelju vrijednosti ključa. Za svaki zapis u matičnoj datoteci postoji odgovarajući zapis u indeksnoj datoteci koji sadrži dva podatka: vrijednost ključa i adresu zapisa u matičnoj datoteci. Ako je potrebno pretraživati matičnu datoteku i prema drugim atributima, a ne samo prema ključu, za svaki zapis u matičnoj datoteci može se izvesti indeksna datoteka, što omogućuje pretraživanje datoteke po različitim atributima. No, više indeksnih datoteka zahtjeva i više memorije te se kod svake promjene matične datoteke trebaju ažurirati sve indeksne datoteke. Indeks ne mora biti izveden samo kao pomoćna datoteka, indeks se može realizirati i kao indeksno područje unutar matične datoteke ili kao memorijska struktura (polje, vezana lista, binarno stablo) (Radošević, 2007).

6.2.3.1. Tip zapisa u datoteci

Tip zapisa u matičnoj datoteci ostao je nepromijenjen u odnosu na primjere s relativnom i sekvencijalnom organizacijom datoteke. Dok tip zapisa indeksne datoteke sadrži ključ (matični broj studenta) i adresu zapisa u matičnoj datoteci (Radošević, 2007).

```
//matična datoteka
struct tstudent{
    int maticni_broj;
    char ime[50];
    char prezime[50];
    char status[20];
};

//indeksna datoteka
struct tindeks{
    int maticni_broj; //Vrijednost ključa
    int adresa;//Adresa zapisa u matično datoteci
};
```

Slika 22. Tip zapisa u datoteci (Radošević, 2007, 88)

6.2.3.2. Kreiranje praznih datoteka

Prije rada s datotekom potrebno je kreirati istu. Kreiranje datoteke postiže se otvaranjem datoteke u modu za upis (ios::out). Datoteka je u početku prazna, vrijednost datotečnog pokazivača za upis je nula. Svaki puta prilikom pokretanja programa potrebno je provjeriti postojanje matične datoteke na disku. Postojanje datoteke utvrđuje se otvaranjem u modu za čitanje (ios::in), a zatim se provjerava stanje datotečnog objekta. Ako matična datoteka ne postoji na disku, potrebno je kreirati indeksnu i matičnu datoteku i obavijestiti korisnika da su datoteke kreirane te da ne sadrže niti jedan zapis. Nakon kreiranja datoteka korisniku se pojavljuje izbornik prikazan na slici 12.

```
fstream tok, ind;
tok.open("studenti.dat", ios::in | ios::binary);

if(!tok){
    cout<<"Maticna datoteka ne postoji na disku!"<<endl;
    tok.open("studenti.dat", ios::out | ios::binary);
    cout<<"Maticna datoeka je kreirana!"<<endl;
    tok.clear();
    tok.close();
    ind.open("studenti.ind", ios::out | ios::binary);
    cout<<"Indeksna datoeka je kreirana!"<<endl;
    ind.clear();
    ind.close();
    system("pause");
}
```

Slika 23. Kreiranje praznih datoteka

6.2.3.3. Unos zapisa u indeksnu datoteku

Za upis podataka o studentu potrebno je otvoriti indeksnu i matičnu datoteku u modovima za upis i ispis te postaviti datotečne pokazivače na kraj datoteke pomoću moda ios::ate, kako bi se u indeks mogla spremiti adresa novog zapisa. Otvaranjem datoteke pomoću moda ios::app, datotečni pokazivači dobivaju vrijednost nula i nije moguće pročitati adresu unesenog zapisa. Nakon što su datoteke uspješno otvorene unose se svi atributi zapisa. Nakon unosa, prvo se upisuje u matičnu datoteku, te se popunjavaju atributi indeksa. Pomoću funkcije *write* zapisuje se indeks u indeksnu datoteku i zatvaraju se obje datoteke. Programski kod za unos zapisa u indeksnu datoteku prikazan je na slici 24.

```
void unos(){
    tstudent student;
    tindeks indeks;
    fstream tok, ind;

    tok.open("studenti.dat", ios::out | ios::in | ios::ate | ios::binary);
    ind.open("studenti.ind", ios::out | ios::in | ios::ate | ios::binary);

    if(!tok || !ind){
        cout<<"Greska prilikom otvaranja datoteka!"<<endl;
        return;
    }

    cout<<"Odabrali ste unos studenta"<<endl;
    cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
    cin.ignore();
    cout<<"Unesite ime: "; cin.getline(student.ime, 50);
    cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
    cout<<"Unesite status: "; cin.getline(student.status, 20);

    tok.write((char*)&student, sizeof(tstudent));
    indeks.maticni_broj = student.maticni_broj;
    indeks.adresa = (int)tok.tellp() - sizeof(tstudent);
    ind.write((char*)&indeks, sizeof(tindeks));

    cout<<"Student je unesen!"<<endl;
    tok.clear(); tok.close();
    ind.clear(); ind.close();
}
```

Slika 24. Unos zapisa u indeksno organiziranu datoteku (Radošević, 2007, 88)

6.2.3.4. Ispis svih zapisa matične datoteke pomoću indeksa

Za ispis svih zapisa iz matične datoteke potrebno je obje datoteke otvoriti u modu za čitanje. Pomoću iteracije tipa *while* slijedno se čita svaki zapis iz indeksne datoteke i sprema u varijablu indeks. Vrijednost datotečnog pokazivača za čitanje iz matične datoteke postavlja se na adresu zapisanu u indeksu te se čita zapis iz matične datoteke. Nakon čitanja zapisa, ispisuju se svi njegovi atributi na ekran. Nakon ispisa svih zapisa zatvaraju se obje datoteke.

```
void ispis(){
    tstudent student;
    tindeks indeks;
    fstream tok, ind;
    tok.open("studenti.dat", ios::in | ios::binary);
    ind.open("studenti.ind", ios::in | ios::binary);
    if(!tok || !ind){
        cout<<"Greska prilikom otvaranja datoteka!"<<endl;
        return;
    }
    while(ind.read((char*)&indeks, sizeof(tindeks))){
        tok.seekg(indeks.adresa);
        tok.read((char*)&student, sizeof(tstudent));
        cout<<student.maticni_broj<<" "<<student.ime;
        cout<<" "<<student.prezime<<" "<<student.status<<endl;
    }
    tok.clear();ind.clear();
    tok.close();ind.close();
}
```

Slika 25. Ispis svih zapisa pomoću indeksne datoteke (Radošević, 2007, 90)

6.2.3.5. Pretraživanje indeksne datoteke

Postupak za pretraživanje sličan je postupku za ispis svih zapisa uz pomoć indeksa, s tim da se prije ispisa provjerava ako je tražena vrijednost ključa koju je upisao korisnik pronađena. U slučaju da traženi zapis nije pronađen, matičnoj se datoteci ne pristupa. U odnosu na slijedno pretraživanje matične datoteke, slijedno pretraživanje indeksa je brže jer je veličina zapisa indeksne datoteke u pravilu manja (Radošević, 2007). Indeksne datoteke su male veličine pa ih je moguće držati u memoriji računala. Sortiranjem indeksa omogućeno je binarno pretraživanje koje je u pravilu puno brže od slijednog pretraživanja.

```
void pretrazivanje(){
    int kljuc;
    tstudent student;
    tindeks indeks;
    fstream tok, ind;
    cout<<"Odabrali ste pretrazivanje studenata!"<<endl;
    cout<<"Unesite maticni broj studenta: "; cin>>kljuc;

    tok.open("studenti.dat", ios::in | ios::binary);
    ind.open("studenti.ind", ios::in | ios::binary);

    while(ind.read((char*)&indeks, sizeof(tindeks))){
        if(indeks.maticni_broj == kljuc){
            tok.seekg(indeks.adresa);
            tok.read((char*)&student, sizeof(tstudent));
            cout<<"Student je pronagen: "<<endl;
            cout<<"Maticni broj: "<<student.maticni_broj<<endl;
            cout<<"Ime: "<<student.ime<<endl;
            cout<<"Prezime: "<<student.prezime<<endl;
            cout<<"Status: "<<student.status<<endl;
            tok.clear(); ind.clear();
            tok.close(); ind.close();
            return;
        }
    }
    tok.clear();ind.clear();
    tok.close();ind.close();
    cout<<"Student nije pronadjen!"<<endl;
}
```

Slika 26. Pretraživanje datoteke pomoću indeksa (Radošević, 2007, 92)

6.2.3.6. Izmjena zapisa u indeksnoj datoteci

Postupak za izmjenu zapisa sličan je postupku za slijedno pretraživanje zapisa uz pomoć indeksa. Nakon pronalaska zapisa, ne ispisuju se svi njegovi atributi na ekran, nego se učitavaju novi atributi od korisnika. Datotečni pokazivač za upis u matičnu datoteku postavlja se na adresu pročitanu iz indeksa i upisuje se zapis u istu. Nakon unosa u matičnu datoteku unosi se novi ključ zapisa u indeks. Vrijednost datotečnih pokazivača za upis iz indeksne datoteke smanjuje se za jednu poziciju te se vrši upis indeksa u indeksnu datoteku.

```
void izmjena(){
    int kljuc;
    tstudent student;
    tindeks indeks;
    fstream tok, ind;
    cout<<"Odabrali ste izmjenu studenta!"<<endl;
    cout<<"Unesite maticni broj studenta kojeg zelite izmijeniti: "; cin>>kljuc;

    tok.open("studenti.dat", ios::in | ios::out | ios::binary);
    ind.open("studenti.ind", ios::in | ios::out | ios::binary);

    while(ind.read((char*)&indeks, sizeof(tindeks))){
        if(indeks.maticni_broj == kljuc){
            cout<<"Unesite maticni broj: "; cin>>student.maticni_broj;
            cin.ignore();
            cout<<"Unesite ime: "; cin.getline(student.ime, 50);
            cout<<"Unesite prezime: "; cin.getline(student.prezime, 50);
            cout<<"Unesite status: "; cin.getline(student.status, 20);
            tok.seekp(indeks.adresa);
            tok.write((char*)&student, sizeof(tstudent));
            indeks.maticni_broj = student.maticni_broj;
            ind.seekp(-1 * sizeof(tindeks), ios::cur);
            ind.write((char*)&indeks, sizeof(tindeks));
            tok.clear(); ind.clear();
            tok.close(); ind.close();
            cout<<"Student je izmijenjen!"<<endl;
            return;
        }
    }
    tok.clear();ind.clear();
    tok.close();ind.close();
    cout<<"Student nije pronadjen!"<<endl;
}
```

Slika 27. Izmjena zapisa u indeksnoj datoteci

7. SORTIRANJE DATOTEKE METODOM SELECTION SORT

Selection sort jedan je od osnovnih algoritama za sortiranje podataka, a samim time i jedan od jednostavnijih. Njegova jednostavnost najbolje se ističe prilikom sortiranja manjih količina podataka. Ovaj algoritam sortira datoteku tako da konstantno pronalazi najmanji zapis iz nesortiranog dijela datoteke i stavlja ga na kraj sortiranog dijela. *Selection sort* započinje sa zapisom na poziciji nula koji predstavlja sortirani dio datoteke. On prolazi kroz cijelu datoteku uspoređujući svaki element s trenutnim zapisom, ako je manji od trenutnog zapisa sprema se njegova pozicija. Jednom kada petlja prođe kroz cijelu datoteku, ako je pronađen zapis koji je manji od zapisa na trenutnoj poziciji, zapis na trenutnoj poziciji zamjenjuje se sa zapisom na poziciji gdje je pronađen najmanji zapis. Pozicija trenutnog zapisa povećava se za jedno mjesto, a samim time i sortirani dio datoteke. Zatim cijeli algoritam kreće ispočetka (http://cforbeginners.com/ssort.html).

Vrijeme izvođenja ovog algoritma ne ovisi o početnom redoslijedu zapisa, nego o količini zapisa u datoteci. Vrijeme izvršavanja *selection sort* algoritma je O(n2).

Na slici 28. prikazan je programski kod za sortiranje datoteke metodom *selection sort*. Prije provođenja ove metode sortiranja potrebno je utvrditi broj zapisa u datoteci. Vanjska iteracija tipa *for* kreće od zapisa na poziciji nula pa sve do zadnjeg zapisa u datoteci. Unutar *for* petlje pozicija trenutnog zapisa sprema se u varijablu „min_poz“. U unutarnjoj iteraciji tipa for, koja ide od sljedeće pozicije trenutnog zapisa vanjske for petlje pa sve do kraja datoteke, učitava se zapis na poziciji „min_poz“ i sprema se u zapis „slog_1“, a zapis na poziciji j sprema se u zapis „slog_2“. Ako je zapis „slog_2“ manji od zapisa „slog_1“, pozicija zapisa „slog_2“ sprema se u varijablu „min_poz“. Nakon izlaska iz unutarnje iteracije tipa for potrebno je učitati zapise na poziciji „min_poz“, te na poziciji i u zapise „slog_1“ i „slog_2“, te zapis !slog_1! zapisati na poziciju i, a zapis !slog_2! na poziciju „min_poz“. Nakon zamjene zapisa sortirani dio datoteke povećava se za jednu poziciju.

```
void selection_sort(){
    fstream tok("studenti.dat", ios::in | ios::out | ios::binary);
    if(!tok){
        cout<<"Greska prilikom otvaranja datoteka!"<<endl;
        return;
    }
    tok.seekg(0, ios::end);
    int broj_zapisa = tok.tellg()/sizeof(tstudent);
    tstudent slog_1, slog_2;

    for(int i = 0; i < broj_zapisa - 1; i++){
        int min_poz = i;
        for(int j = i + 1; j < broj_zapisa; j++){
            tok.seekg(min_poz * sizeof(tstudent));
            tok.read((char*)&slog_1, sizeof(tstudent));
            tok.seekg(j * sizeof(tstudent));
            tok.read((char*)&slog_2, sizeof(tstudent));
            if(slog_2.maticni_broj < slog_1.maticni_broj){
                min_poz = j;
            }
        }
        tok.seekg(min_poz * sizeof(tstudent));
        tok.read((char*)&slog_1, sizeof(tstudent));
        tok.seekg(i * sizeof(tstudent));
        tok.read((char*)&slog_2, sizeof(tstudent));

        tok.seekp(i * sizeof(tstudent));
        tok.write((char*)&slog_1, sizeof(tstudent));
        tok.seekp(min_poz * sizeof(tstudent));
        tok.write((char*)&slog_2, sizeof(tstudent));
    }
    cout<<"Datoteka je sortirana!"<<endl;
}
```

Slika 28. Sortiranje datoteke metodom selection sort

8. PRIMJER RADA S DATOTEKAMA: BLAGAJNA

U programskom primjeru ilustriran je rad blagajne. Za pohranu podatka koriste se različite organizacije datoteka s fiksnom veličinom zapisa, a ispis računa i potrebnih izvještaja vrši se pomoću tekstualnih datoteka.

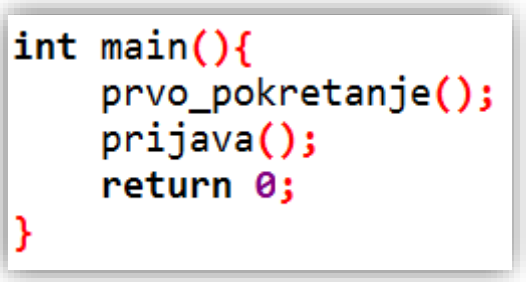
Tablica 6. Biblioteke korištene u programu

BIBLIOTEKA	OPIS
`#include <iostream>`	Rad s ulazno izlaznim tokovima.
`#include <cstdlib>`	Sadrži funkcije općenite namjene.
`#include <ctime>`	Upravljanje datumima i vremenom.
`#include <fstream>`	Rad s datotekama.
`#include <cmath>`	Omogućava matematičke operacije.
`#include <iomanip>`	Omogućava manipuliranje ulazno izlaznim tokovima.
`#include <cstring>`	Uključuje funkcije za manipuliranje stringovima.
`#include <vector>`	Definiran je spremnik vector.
`#include <algorithm>`	Sadrži funkcije koje se koriste nad rasponom elemenata, npr. „sortiranje elemenata“.

Tablica 7. Strukture podataka korištene u programu

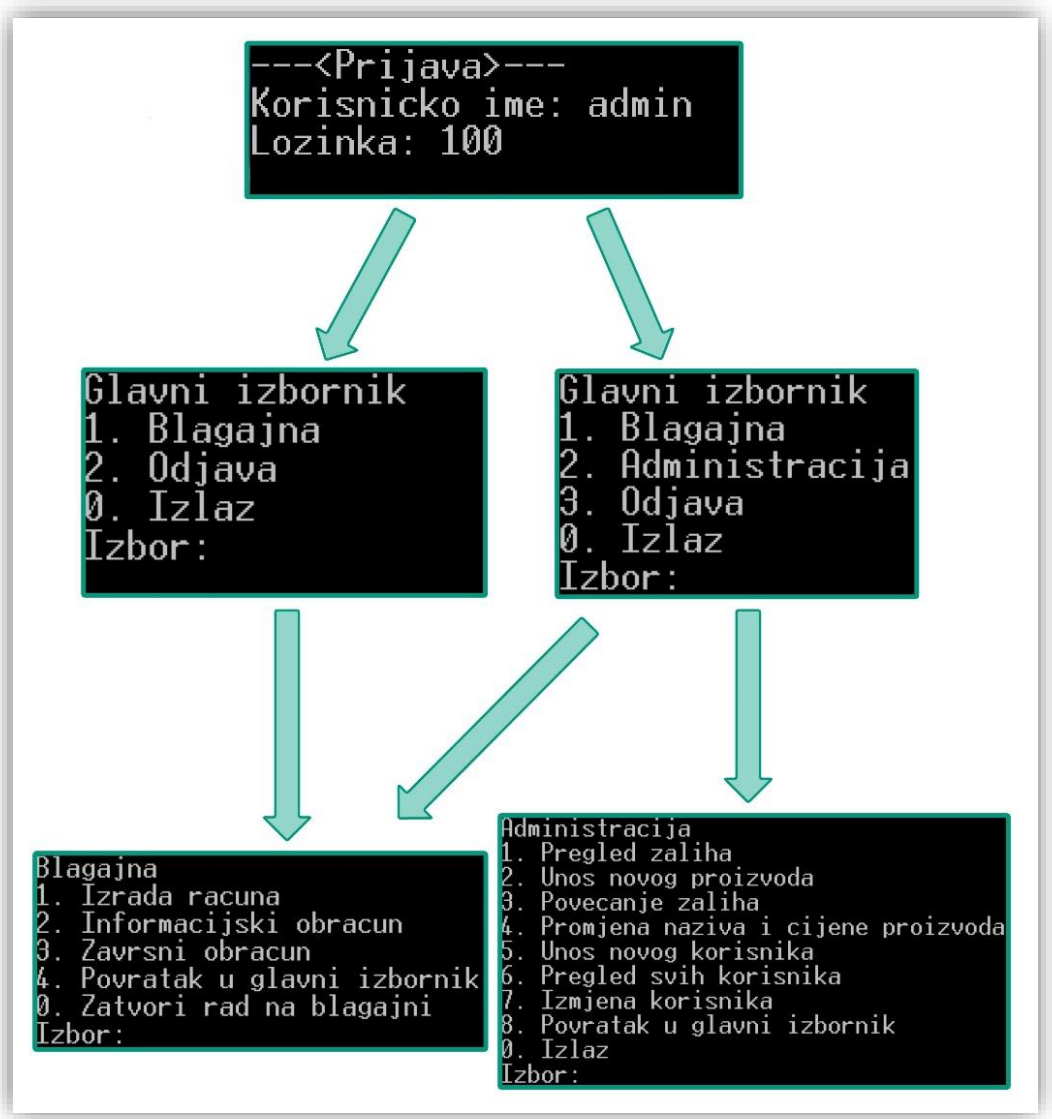
NAZIV STRUKTURE	ZA ŠTO SE KORISTI
`tindeks`	Za pohranu podataka o korisniku u indeksnu datoteku.
`tkorisnik`	Za pohranu podataka o korisniku u matičnu datoteku.
`tproizvod`	Za pohranu podataka o proizvodu u datoteku zaliha.
`tartikl`	Zadrži podatke o artiklu, koristi se prilikom izdavanja računa.
`tobracun`	Zadrži potrebne podatke za izradu izvještaja.

Prilikom pokretanja programa poziva se funkcija „prvo_pokretanje()“ u kojoj se provjerava postojanje matične datoteke s korisnicima. Ako datoteka s korisnicima ne postoji, kreira se novi korisnik kojem se dodjeljuje status administratora. Za pohranu podatak o korisnicima koristi se indeksna organizacija datoteke kako bi se postigla što veća efikasnost prilikom pretraživanja. Nakon funkcije „prvo_pokretanje()“ slijedi funkcija „*prijava()*“.



Slika 29. Glavna funkcija programa

U navedenoj funkciji korisnik se prijavljuje u sustav te ovisno o statusu istoga prikazuje se glavni izbornik. Ako je korisnik administrator sustava, u glavnom izborniku prikazuje se dodatna mogućnost „Administracija“. Izgled izbornika prikazan je na slici 30.



Slika 30. Dijagram s izbornicima programa

Glavni izbornik uključuje mogućnosti u kojima se može otvoriti izbornik „blagajne“ i „administracije“, odjaviti se te izaći iz programa. Poziv svake od mogućnosti navedenih u izborniku „administracija“ i „blagajna“ poziva odgovarajuću funkciju. Glavna funkcionalnost blagajne je izrada računa, no prije izrade računa potrebno je kreirati datoteku zaliha. Za kreiranje zaliha potrebno je pozvati funkciju „unos_novog_proizvoda()“. Prilikom dodavanja stavke računa potrebno je pretražiti datoteku zaliha te nakon svakog izdanog računa potrebno je ažurirati istu. Zbog navedenih razloga poželjno je omogućiti direktan pristup zapisu, što se postiže relativnom organizacijom datoteke. Nakon unosa proizvoda, količina proizvoda na zalihi jednaka je nuli. Kako bi se omogućilo izdavanje računa potrebno je povećati stanje proizvoda na zalihama što se postiže funkcijom „povecanje_zaliha()“.

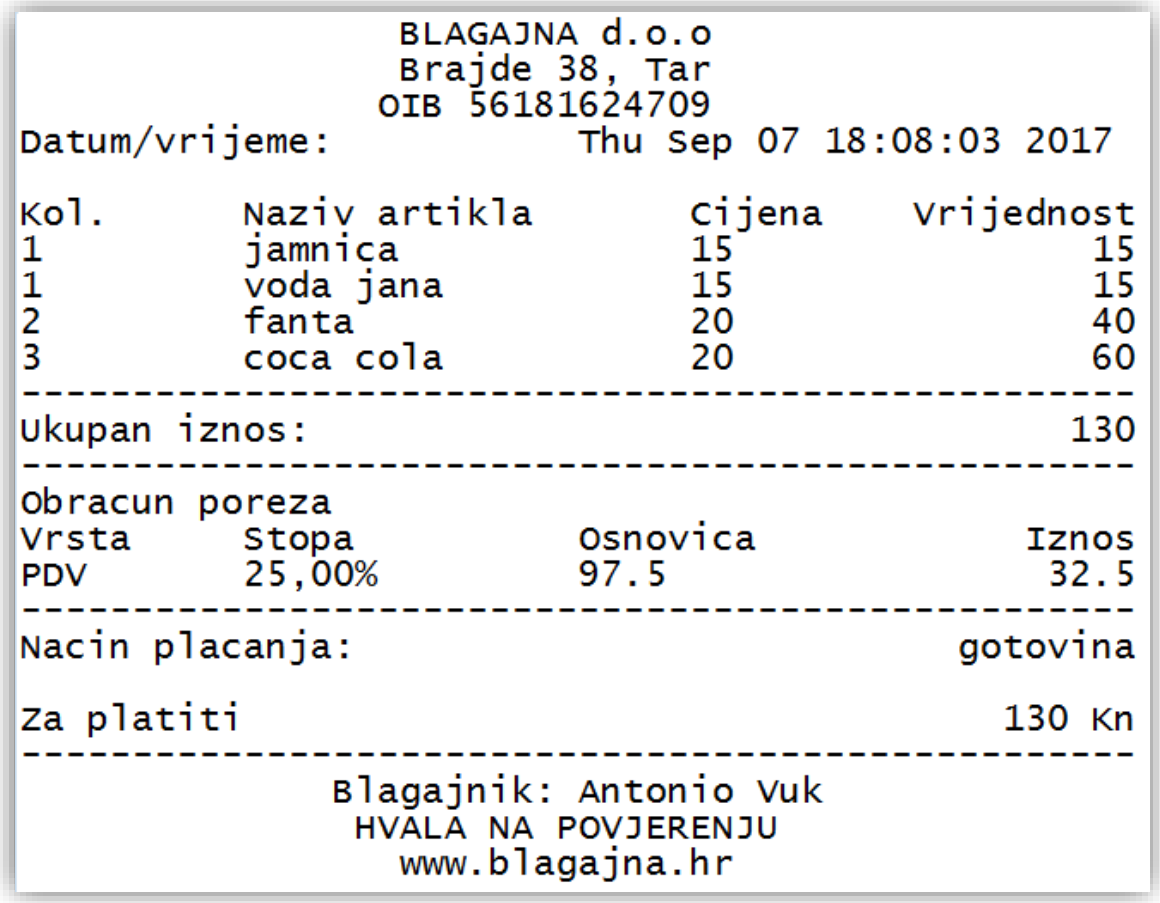
```
void izrada_racuna(tkorisnik korisnik){
    fstream tok("proizvodi.dat", ios::in | ios::binary);
    if(!tok){
        cout<<"Datoteka sa zalihama ne postoji na disku!"<<endl;
        return;
    }
    tproizvod proizvod;
    tartikl artikl, temp;
    vector <tartikl> vektor;
    vector <tartikl>::iterator it;
    cout<<"Odabrali ste izradu racuna!"<<endl;
    char izbor;
    do{
        system("cls");
        if(!popis_proizvoda()){
            cout<<"Nema proizvoda na zalihama!"<<endl;
            cout<<"Za povratak pritisnite enter"<<endl;
            return;
        }
        do{
            cout<<"Unesite sifru artikla: "; unos(artikl.sifra);
            proizvod = pronagi(artikl.sifra);
            if(!proizvod.sifra){
                cout<<"Unijeli ste krivu sifru!"<<endl;
                return;
            }
            if(proizvod.kolicina <= 0){
                cout<<"Trazenog proizvoda nema na zalihi!"<<endl;
                return;
            }
        }while(!proizvod.sifra);

        strcpy(artikl.naziv, proizvod.naziv);
        artikl.cijena = proizvod.cijena;
        int iskoristeno = provjeri_stavku(vektor, artikl.sifra);

        do{
            if(proizvod.kolicina-iskoristeno <= 0){
                cout<<"Ovog proizvoda nema na zalihi!"<<endl;
                artikl.kolicina = 0;
                break;
            }
            cout<<"Unesite kolicinu: ";
            unos(artikl.kolicina);
            if(artikl.kolicina<= 0){
                cout<<"Kolicina mora biti veca od nule!"<<endl;
                return;
            }
            if(proizvod.kolicina < artikl.kolicina + iskoristeno){
                cout<<"Nedostatak zaliha u skladistu: "<<proizvod.naziv<<endl;
                cout<<"Stanje na zalihama: "<<proizvod.kolicina<<endl;
                cout<<"Iskoristeno stanje: "<<iskoristeno<<endl;
                cout<<"Raspolozivo stanje: "<<proizvod.kolicina - iskoristeno<<endl;
            }
        }while(proizvod.kolicina < artikl.kolicina + iskoristeno);

        if(iskoristeno){
            for(it = vektor.begin(); it != vektor.end(); it++){
                tartikl temp = *it;
                if(temp.sifra == artikl.sifra){
                    it = vektor.erase(it);
                    temp.kolicina+=artikl.kolicina;
                    vektor.push_back(temp);
                    break;
                }
            }
        }else{
            vektor.push_back(artikl);
        }
        cout<<"Zelite li dodati novi artikl (y / n): "; unesi_izbor(izbor);
    }while(izbor == 'y' || izbor == 'Y');
```

Slika 31. Programski kod za izradu računa - 1. dio

```
    cout<<"Zelite li izdati racun (y / n): "; unesi_izbor(izbor);
    if(izbor == 'y' || izbor == 'Y'){
        sort (vektor.begin(), vektor.end(), funkcija_usporedi);
        fstream dat, tok;
        tobracun obracun;

        dat.open("pomocna_datoteka.dat", ios::in | ios::out | ios::binary);
        dat.read((char*)&obracun, sizeof(tobracun));
        if(!obracun.potvrgeno){
            string s = asctime(localtime(&ctt));
            strcpy(obracun.pocetak_obracuna, s.c_str());
            obracun.potvrgeno = true;
            obracun.ukupan_broj_izdanih_racuna++;
            obracun.broj_dokumenta++;
        }else{
            obracun.ukupan_broj_izdanih_racuna++;
        };
        char broj_racuna[5];
        itoa(obracun.ukupan_broj_izdanih_racuna, broj_racuna, 10);
        string racun = "C:/Users/VUK/Desktop/Blagajna/racuni/racun_";
        racun+=broj_racuna;
        racun+=".txt";
        tok.open(racun.c_str(), ios::out);

        system("cls");
        tok<<setw(31)<<internal<<"BLAGAJNA d.o.o"<<endl;
        tok<<setw(31)<<internal<<"Brajde 38, Tar"<<endl;
        tok<<setw(31)<<internal<<"OIB 56181624709"<<endl;
        tok<<setw(25)<<left<<"Datum/vrijeme:"<<setw(25)<<right<<asctime(localtime(&ctt))<<endl;
        tok<<setw(10)<<left<<"Kol."<<setw(20)<<left<<"Naziv artikla"<<setw(10)<<left<<"Cijena";
        tok<<setw(10)<<right<<"Vrijednost"<<endl;

        int suma;
        for(it = vektor.begin(); it != vektor.end(); it++){
            tartikl temp = *it;
            azuriraj_zalihu(temp.sifra, temp.kolicina);
            suma+=temp.kolicina*temp.cijena;
            tok<<setw(10)<<left<<temp.kolicina<<setw(20)<<left<<temp.naziv<<setw(10)<<left;
            tok<<temp.cijena<<setw(10)<<right<<temp.cijena*temp.kolicina<<endl;
        }
        tok<<"-------------------------------------------------"<<endl;
        tok<<setw(25)<<left<<"Ukupan iznos: "<<setw(25)<<right<<suma<<endl;
        tok<<"-------------------------------------------------"<<endl;
        tok<<"Obracun poreza"<<endl;
        tok<<setw(10)<<left<<"Vrsta"<<setw(15)<<left<<"Stopa"<<setw(15)<<left<<"Osnovica";
        tok<<setw(10)<<right<<"Iznos"<<endl;
        tok<<setw(10)<<left<<"PDV"<<setw(15)<<left<<"25,00%"<<setw(15)<<left;
        tok<<suma - ((float)suma*25/100)<<setw(10)<<right<<(float)suma*25/100<<endl;
        tok<<"-------------------------------------------------"<<endl;
        tok<<setw(25)<<left<<"Nacin placanja: "<<setw(25)<<right<<"gotovina"<<endl<<endl;
        tok<<setw(25)<<left<<"Za platiti"<<setw(22)<<right<<suma<<" Kn"<<endl;
        tok<<"-------------------------------------------------"<<endl;
        tok<<setw(25)<<internal<<"Blagajnik: "<<korisnik.ime_i_prezime<<endl;
        tok<<setw(34)<<internal<<"HVALA NA POVJERENJU"<<endl;
        tok<<setw(32)<<internal<<"www.blagajna.hr"<<endl;
        obracun.ukupan_iznos+=suma;
        dat.seekp(0, ios::beg);
        dat.write((char*)&obracun, sizeof(tobracun));
        dat.clear();
        dat.close();
        tok.clear();
        tok.close();
        cout<<"Racun je uspjesno izragen!"<<endl;
        cout<<"-------------------------------------------------"<<endl;
        ispis_txt_datoteke(racun);
    }else{
        cout<<"Racun je ponisten!"<<endl;
        cout<<"Slijedi povratak u glavni izbornik!"<<endl;
    }
}
```

Slika 32. Programski kod za izradu računa - 2. dio

Prilikom poziva funkcije „izrada_računa()“ prikazuje se popis proizvoda čije je stanje na zalihama veće od nula. Odabir artikla vrši se unosom šifre, nakon čega je potrebno unijeti i željenu količinu. Zatim, nudi se izbor korisniku za unos novog artikla. U slučaju potvrdnog odgovora, ponavlja se cijeli postupak. Nakon odabira artikala, pojavljuje se novi izbornik u kojem se korisniku nudi opcija izdavanja računa. U slučaju potvrdnog odgovora kreira se datoteka računa te se njen sadržaj ispisuje na ekran. U suprotnom slijedi povratak u glavni izbornik. Sadržaj datoteke računa prikazan je na slici 33.



```
                  BLAGAJNA d.o.o
                  Brajde 38, Tar
                 OIB 56181624709
Datum/vrijeme:           Thu Sep 07 18:08:03 2017

Kol.      Naziv artikla       Cijena    Vrijednost
1         jamnica             15                15
1         voda jana           15                15
2         fanta               20                40
3         coca cola           20                60
-------------------------------------------------
Ukupan iznos:                                  130
-------------------------------------------------
Obracun poreza
Vrsta     Stopa          Osnovica             Iznos
PDV       25,00%         97.5                  32.5
-------------------------------------------------
Nacin placanja:                           gotovina

Za platiti                                  130 Kn
-------------------------------------------------
              Blagajnik: Antonio Vuk
               HVALA NA POVJERENJU
                 www.blagajna.hr
```

Slika 33. Sadržaj datoteke računa

Nakon izdavanja računa, u svakom je trenutku moguće vidjeti stanje blagajne pozivom mogućnosti informacijski obračun. Pozivom mogućnosti završni obračun kreira se nova datoteka „završni_obračun“ u kojoj je zapisan zaključno stanje prometa, tko je i kada izradio obračun, te početak obračuna. Nakon izrađenog završnog obračuna stanje blagajne postavlja se na nula.

9. ZAKLJUČAK

Glavna je uloga današnjih računala rješavanje problema i procesiranje podataka. U svakoj računalnoj aplikaciji glavni je entitet podatak, koji se zapisuje u datoteku. Tekstualne se datoteke mogu obrađivati samo slijedno, dok se datoteke s fiksnom veličinom zapisa mogu organizirati na različite načine.

Pravilnom organizacijom datoteke moguće je uvelike povećati performanse i efikasnost korištenja programa. Organizacijom se dobije unaprijed definiran i efikasan način kojim se pohranjuje, dohvaća i uređuje zapis u datoteci.

U slijednoj organizaciji datoteke dohvaćanje zapisa zahtjeva čitanje svih zapisa od početka do traženog zapisa. U relativno organiziranim datotekama, omogućeno je direktno pristupanje zapisu, no one su fiksne veličine i imaju puno praznih zapisa, a pretraživanje je moguće samo po ključu zapisa. Nadalje, u indeksnim datotekama pristupanje zapisima je efikasnije nego u slijednim datotekama. Za razliku od relativnih datoteka, indeksne datoteke nisu fiksne veličine i pretraživanje je moguće po različitim atributima zapisa. Najveći nedostatak indeksnih datoteka je česta potreba za stvaranjem više od jedne indeksne datoteke što zauzima puno prostora na disku.

Cilj je ovog rada u potpunosti savladati tehnike rada s datotekama u programskom jeziku C++, kao i principe rada s datotekama, ukazati na prednosti i nedostatke različitih organizacija datoteka i pružiti konkretne primjere rada s datotekama.

Odabir najprikladnije organizacije datoteke trebao bi ovisiti o različitim čimbenicima, kao što su broj zapisa u datoteci, potreba za pretraživanjem zapisa po različitim atributima, učestalost dohvaćanja zapisa i sl.

U ovom su radu objašnjene osnovne metode rada s datotekama u programskom jeziku C++, rad s datotečnim pokazivačem i osnovne metode klase *fstream* zadužene za rad s datotekama. Kroz primjere prikazana je zamjena sadržaja u tekstualnoj datoteci, rad s različitim organizacijama datoteka, sortiranje datoteke metodom *selection sort* te je naposljetku prikazano korištenje različitih organizacija datoteka pri izradi aplikacije za fiskaliziranje računa. Iako je iznesena aplikacija izvedena samo pomoću konzole te su svi podaci pohranjeni u datoteku umjesto u bazu podataka, temeljni princip razvoja sličan je rješenjima koji sadrže grafičko korisničko sučelje.

LITERATURA

1. Cforbeginners, (nema datuma), dostupno na: cforbeginners.com/ssort.html [10.8.2017.].
2. Cplusplus, (2000), dostupno na: www.cplusplus.com/reference/ [20.6.2017.].
3. Geeksforgeeks, (nema datuma), dostupno na: http://www.geeksforgeeks.org/selection-sort/ [10.8.2017.].
4. H. Patil, Varsha. (2009) *Data Structures Using C++*, *Cengage Learning*. Dostupno na: http://file.allitebooks.com/20160524/Data Structures using C++.pdf [12.7.2017.].
5. King. N., K. (2008) 'C Programming: A Modern Approach'. Dostupno na: https://www.scribd.com/doc/39933932/C-Programming-a-Modern-Approach-2nd-Edition-K-N-King [11.6.2017.].
6. Kirch-Prinz, Ulla. i Prinz, Peter. (2002) *A Complete Guide to Programming in C++.* Dostupno na: http://www.lmpt.univ-tours.fr/~volkov/C++.pdf [20.7.2017.].
7. Lippman, Stanley. B., Lajoie, Josee. i Babara Moo (nema datuma) *C++ Primer*. Peto izdanje, [20.7.2017.].
8. Mateljan, I. (2007) '10.6. Rad s datotekama', pp. 1–15. Dostupno na: http://marjan.fesb.hr/~mateljan/cpp/slides10-2-files.pdf [1.8.2017.].
9. Motik, Boris. i Šribar, Julijan. (1997) 'Demistificirani C++'. Dostupno na: http://personal.oss.unist.hr/~jvrlic/dokumenti/OOP/demistificirani_c++.pdf, [17.7.2017.].
10. Radošević, Danijel. (2007) *Programiranje 2*. Varaždin, [12.7.2017.].
11. Souli, Juan. (2007) 'C ++ Language Tutorial'. Dostupno na: http://www.cplusplus.com/files/tutorial.pdf, [30.6.2017.].
12. Stroustrup, Bjarne. (nema datuma) *The C ++ Programming*. Dostupno na: https://github.com/BestSonny/materials/blob/master/The C%2B%2B Programming Language %5B4th Edition%5D - Bjarne Stroustrup.pdf, [17.7.2017.].

POPIS SLIKA

POPIS TABLICA