Algoritmi za proste brojeve u programskom jeziku C++

Milobara, Borna

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski studij

ALGORITMI ZA PROSTE BROJEVE U PROGRAMSKOM JEZIKU C++

Završni rad

Borna Milobara

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 21.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Borna Milobara
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	3373, 24.09.2019.
OIB studenta:	48015104439
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Algoritmi za proste brojeve u programskom jeziku C++
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	21.09.2020.
Datum potvrde ocjene Odbora:	23.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 29.09.2020.

Ime i prezime studenta:	Borna Milobara
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	3373, 24.09.2019.
Turnitin podudaranje [%]:	16

Ovom izjavom izjavljujem da je rad pod nazivom**: Algoritmi za proste brojeve u programskom jeziku C++**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Svrha ovoga završnoga rada jest proučavanje kodova u programskom jeziku C++ vezanih za proste brojeve. Prije svega ćemo navesti matematičku definiciju pojmova „prosti brojevi" i „skup P", kao i definiciju tipova i njihove uporabe. Onda ćemo se dotaknuti i samoga programskoga jezika u kojemu radimo i upoznati se s njegovim svojstvima, sintaksom i funkcijama potrebnim za pisanje i razumijevanje napisanog koda na temelju konkretnih primjera.

Nakon toga ćemo se upustiti u proučavanje samih algoritama i kodova napisanih za rad s prostim brojevima u programu C++. Naravno, s obzirom na to da svrha ovoga rada nije rješavanje trenutačno neriješenih matematičkih problema vezanih za proste brojeve niti detaljno proučavanje samog programskog jezika C++, naglasak će biti na samim algoritmima i kodovima vezanima za proste brojeve.

To su algoritmi i kodovi kao provjera brojeva prema specifičnim definicijama, različita sita za ispis prvih n prostih brojeva, pronalazak prostih brojeva unutar zadanog intervala prema različitim postulatima i pretpostavkama, rastavljanje na faktore, ispis specifičnih parova prostih brojeva i slično. Osim same implementacije tih kodova te kodove ćemo analizirati, kategorizirati i komentirati.

1.1. Zadatak završnoga rada

Zadatak ovoga rada jest usporediti recentne algoritme za analizu prostih brojeva, implementirati sve algoritme na računalu u programskom jeziku C++ i kategorizirati ih po brzini i uspješnosti.

2. KRATAK POVIJESNI I AKTUALNI PREGLED PODRUČJA TEME

Zanimljivost primarne teme ovoga rada, prostih brojeva, leži ponajprije u činjenici da trenutačno postoje mnoga neodgovorena pitanja i različite nedokazane pretpostavke u tom području matematike, što ga čini vrlo specifičnom temom za istraživanje.

Dosezi na ovome području nizali su se kroz povijest još od antičkog Egipta, gdje su prvi puta uočene različite forme za proste i složene brojeve[1] na temelju prvih preživjelih zabilježenih spisa u Euklidovim Elementima[2] i prvoga sita nazvanoga prema njegovu izumitelju Eritostanu[3]. Razvojem znanosti stvorio se još veći interes za proste brojeve, otvarajući sve širi spektar pitanja, istraživanja i odgovora na tome području počevši od 17. stoljeća od Fermata[4] i Goldbacha[5] preko Rienmanna[6] i Dirichleta[7] i sve do današnjih znanstvenika kao što su Yitang Zhang[8] i udruženja kao što je GIMPS koji vrše potragu[9] za što većim prostim brojem.

No, ono što je specifično za ovo područje jest to da postoji mnogo važnih neodgovorenih pitanja, kao što su pitanja vezana za razlike između susjednih prostih brojeva[10] (engl. *prime gaps*), i nedokazane pretpostavke, kao što su Goldbachova pretpostavka[11], četiri Landauova problema[12] i Chenov teorem[13].

---

[1] Bruins, Evert Marie, Mathematical Reviews of *Gillings, R.J. (1974). "The recto of the Rhind Mathematical Papyrus. How did the ancient Egyptian scribe prepare it?". Archive for History of Exact Sciences.*
[2] Stillwell, John (2010). Mathematics and Its History. Undergraduate Texts in Mathematics (3rd ed.). Springer. p. 40.
[3] Mollin, Richard A. (2002). "A brief history of factoring and primality testing B. C. (before computers)". Mathematics Magazine.
[4] Sandifer 2007, 8. Fermat's Little Theorem (November 2003)
[5] Yuan, Wang (2002). Goldbach Conjecture. Series In Pure Mathematics. 4 (2nd ed.). World Scientific. p. 21.
[6] Apostol, Tom M. (2000). "A centennial history of the prime number theorem". In Bambah, R.P.; Dumir, V.C.; Hans-Gill, R.J. (eds.). Number Theory. Trends in Mathematics. Basel: Birkhäuser.
[7] Apostol, Tom M. (1976). "7. Dirichlet's Theorem on Primes in Arithmetical Progressions". Introduction to Analytic Number Theory. New York; Heidelberg: Springer-Verlag. pp. 146–156. MR 0434929.
[8] Neale 2017, pp. 18, 47.
[9] Ziegler, Günter M. (2004). "The great prime number record races". Notices of the American Mathematical Society.
[10] Koshy 2002, Theorem 2.14, p. 109
[11] Guy 2013, C1 Goldbach's conjecture, pp. 105–107
[12] Guy 2013, p. vii
[13] Guy 2013, p. 159.

3. PROSTI BROJEVI

3.1. Pojam prostoga broja

Prosti brojevi (ili prim-brojevi) zaokupljuju pažnju matematičara još od davnih vremena. Prvi teoremi o njima sežu u doba starogrčke matematike i sve do danas tema su raznih istraživanja, ispitivanja i analiziranja. Iako su dio nastavnog sadržaja već u ranom stadiju obrazovanja i mogu se čini poprilično jednostavni, i dalje postoji mnogo neodgovorenih pitanja vezanih uz njih. [1]

Najjednostavnija definicija prostih brojeva glasi:

Definicija 3.1.1. *Prost broj jest prirodan broj p>1, koji je djeljiv samo s brojem 1 i sa samim sobom.*

Što znači da prosti broj p nema prirodnih djelitelja, tj. brojeva s kojim se dijeli bez ostatka, osim broja 1 i samoga broja p. Ako broj ima prirodnih djelitelja, naziva se složenim brojem. Uz to uočavamo kako broj 1 nije ni prost ni složen broj, a broj 2 jedini je parni prosti broj.

Osim definicije prostih brojeva, valja spomenuti i pojam relativno prostih brojeva.

Definicija 3.1.2. *Za cijele brojeve a, b različite od 0 kažemo da su relativno prosti ako je njihov najveći zajednički djelitelj jednak 1.*

To zapisujemo kao *NZD(a,b)=1*. Drugim riječima, ti brojevi nemaju zajedničkih faktora. Naprimjer, iako brojevi 10 i 21 nisu sami po sebi prosti, oni su relativno prosti zato što nemaju zajedničkih djelitelja osim broja 1.

Za pronalaženje prostih brojeva u zadanom intervalu danas se upotrebljava mnogo različitih algoritama kojih ćemo se dotaći kasnije. Trenutačno je prije svega potrebno spomenuti metodu sita, s pomoću koje se u teoriji brojeva generalizira principe izbacivanja složenih brojeva iz skupa prirodnih brojeva redom kako bi nakon određenog broja iteracija preostali samo prirodni brojevi.

U cilju pobližeg pojašnjavanja pojma „sito" prije svega spomenimo Eratostena i njegovo sito, odnosno mehanički postupak s pomoću kojeg se izbacuju svi višekratnici prostih brojeva manjih od *n*. Eratosten je svoje sito osmislio kako bi izbjegao provjeravanje složenosti svakoga broja iz intervala prirodnih od 2 do *n*. U nastavku se nalazi algoritam tog postupka.

Algoritam 3.1.3. *Eratostenovo sito.*

1. Kreirati niz prirodnih brojeva u zadanom intervalu od 2 do n

2. Postaviti p na najmanji prost broj, tj. p=2.

3. Izbaciti iz niza sve višekratnike broja p.

4. Pronaći sljedeći broj niza i postaviti ga za p te ponoviti korak 3. Ako takav ne postoji, stati.

Vrijedno je spomenuti još dva sita koja su kroz povijest bila vrlo zapažena, a to su Sundaramovo sito, osmišljeno 1934. kao poboljšanje Eratostenova s manjom teorijskom složenošću, i Atkinovo sito, osmišljeno 2003., namijenjeno računalima i bazirano na ireducibilnim kvadratnim formama.

3.2. Skup P

Skup svih prostih brojeva označavamo s **P**. Matematičarima je vrlo zanimljivo svojstvo beskonačnosti skupa **P**, pa su se i kroz povijest pojavljivale različite varijacije dokaza upravo toga svojstva.

Iako postoje moderniji dokazi od onoga najstarijega, Euklidova, kao što su Furstenbergerov, Goldbachov, Saidakov i slični, nama je trenutačno dovoljan samo jedan dokaz, pa ćemo spomenuti Euklidov dokaz s pomoću kojeg se to svojstvo dokazuje jednostavnom tvrdnjom u kojoj se pretpostavlja suprotno.

Teorem 3.2.1. *Skup P beskonačan je.*

Dokaz. *Pretpostavimo da je skup **P** konačan, tj. da postoji konačno mnogo prostih brojeva p_1, p_2, p_3, …, p_n. Definirajmo M kao $M = p_1p_2p_3 \ldots p_n+1$ i prosti broj p koji dijeli broj M. S obzirom na to da broj p ne može biti ni jedan od p_1, p_2, p_3, …, p_n jer bi p inače dijelio razliku $M - p_1p_2p_3 \ldots p_n = 1$, što je nemoguće, znači da bi p uistinu bio prost broj koji nije iz skupa p_1, p_2, p_3, …, p_n i upravo tu dolazi do kontradikcije, na temelju čega zaključujemo suprotno, tj. da je skup **P** beskonačan.* ■

Nadalje, s obzirom na to da je jedan od trenutačno najvećih problema skupa **P** određivanje gustoće raspodjele prostih brojeva u skupu N, definirajmo i funkciju s pomoću koje ćemo lakše doći do odgovora, a to je funkcija $\pi\ (x)$ kojom se označava broj prostih brojeva manjih ili jednakih x za svaki $x \in [2, +\infty)$:

Definicija 3.2.2. $\pi(x) = card\{p \in \boldsymbol{P} \mid p \leq x\},\ \pi : [2, \infty) \rightarrow N$

Razumijevanje njezina asimptotskoga ponašanja bio je prvi korak koji su matematičari 19. stoljeća morali napraviti kako bi postavili Teorem o prostim brojevima koji se danas definira na sljedeći način:

Definicija 3.2.3. *Za π(x) vrijedi:* $\lim_{n\to\infty} \frac{\pi(x)}{x/ln\ x} = 1$, *odnosno približno:* $\pi(x) \approx \frac{x}{lnx}$

Kako za potrebe ovoga rada nije nužno detaljnije ulaziti u trenutačno nerješive probleme u pogledu gustoće raspodjele i određivanja n-toga prostoga broja koji postoje u modernoj matematici, u nastavku se nećemo baviti tom temom.

3.3. Tipovi i uporaba

U matematici je kroz povijest postojalo mnogo različitih tipova prostih brojeva, koji uglavnom nose nazive prema svojim svojstvima i međusobnim odnosima ili prema ljudima koji su ih definirali, pa tako primjerice postoje Wilsonov, Woodallov, Cullenov, Wagstaffov tip, kao i tip primbrojeva Sophie-Germain. [2]

Za potrebe ovoga rada definirat ćemo dva određena prema njihovu svojstvu udaljenosti od svojega susjednoga prostoga broja, a to su blizanci (engl. *twin primes*) i rođaci (engl. *cousin primes*) te dva nazvana prema Marinu Mersenneu i Pierreu Fermatu.

Definicija 3.3.1. *Dva prosta broja nazivamo blizancima ako njihova razlika iznosi 2.*

To su npr. (3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), …

Definicija 3.3.2. *Dva prosta broja nazivamo rođacima ako njihova razlika iznosi 4*.

To su npr. (3, 7), (7, 11), (13, 17), (19, 23), (37, 41), (43, 47), …

Definirajmo i Fermatove proste brojeve:

Definicija 3.3.3. *Fermatovi prosti brojevi F_n prosti su brojevi oblika $2^{2^n} + 1$.*

Ubrzo se uočilo da, iako je definicija precizna u startu $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65537$, Euler je već za n = 5 dokazao da se ne radi o prostome broju zbog njegove djeljivosti s brojem 641, tj. $F_5 = 4294967297 = 641 \times 6700417$. Unatoč tomu zbog nekih određenih svojstava Fermatovi brojevi se i danas koriste u određenim istraživanjima te još uvijek nije poznato postoji li beskonačno mnogo Fermatovih brojeva.

Definicija 3.3.4. *Mersennovi prosti brojevi M_n prosti su brojevi oblika $2^n - 1$.*

Iako je Mersenne tvrdio da za brojeve n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257 vrijedi da je M_n prost broj, a za sve druge prirodne brojeve manje od 257 da je složen, no i on je opovrgnut te je dokazano da brojevi M_{67} i M_{257} nisu prosti, a brojevi M_{61}, M_{89} i M_{107} jesu.

Bez obzira na to, njegovi se brojevi i danas upotrebljavaju ponajviše u potrazi za najvećim prostim brojem. Zanimljivo je napomenuti kako trenutačno najveći poznati prosti broj $2^{82589933} - 1$ sadrži 24 862 048 znamenki te da ga je 7. prosinca 2018. godine pronašla udruga Great Internet Mersenne Prime Search (GIMPS), čiji se volonteri bave potragom baš za njegovim brojevima. [3]

Uporaba prostih brojeva prije svega se temelji na osnovnome teoremu aritmetike u okviru kojeg se tvrdi da za svaki prirodni broj $n > 1$ postoji jedinstven rastav na proste faktore. Upravo taj rastav zajedno s ostalim značajkama za programiranje bit će tema u nastavku ovoga rada.

Usto, prosti brojevi imaju bitan utjecaj i u kriptografiji s naglaskom na algoritme za šifriranje poruka i činjenicu da ne postoji vrlo efikasan algoritam za rastavljanje velikih brojeva na proste faktore. Najpoznatiji primjer takve šifre jest šifra RSA.

4. PROGRAMSKI JEZIK C++

4.1. Uvod

C++ jest programski jezik opće namjene autora Bjarne Stroustrupa nastao kao potreba za nadogradnjom proceduralnoga programskoga jezika C autora Dennisa Ritchieja proširivanjem na jezik s podrškom za objektno orijentirano programiranje. [4]

Prije svega, za razumijevanje je potrebno analizirati najjednostavniji primjer koda.

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


int main()
{
   cout<<"Hello World!";  // Ovo je komentar
   return 0;
}
```

Kod 4.1.1. *Hello World!*

Proučimo oblik ovoga vrlo poznatoga koda. Prvo uočavamo liniju koda *#include <iostream>*. Ona nam služi kako bi se u program uključila datoteka <*iostream*> koja sadrži osnovne (i nužne) funkcije. Jedna od njih upotrijebljena je upravo u ovome programu, a to je funkcija *cout*. Za potrebe ovoga rada još ćemo upotrebljavati i datoteku <*math.h*> koja sadrži mnogo više matematičkih funkcija od već spomenute datoteke <*iostream*>. Sljedećom linijom koda, *using namespace std;*, označava se tek da će se upotrebljavati klase, funkcije, objekti i varijable iz standardne knjižnice kako se ne bi morali eksplicitno pozivati svaki put. Taj dio koda će u ovome primjeru ostati nepromijenjen.
Sljedeći dio koda, tj. *int main() { ... }* deklaracija je uvijek potrebne glavne funkcije koja može uzimati jedan ili više argumenata i obično vraća neku vrijednost, kao što je u ovome primjeru prikazano linijom koda *return 0;*. Uočimo i da se blok naredbi unutar funkcija piše u vitičastim zagradama { … } te da svaka naredba završava znakom točka sa zarezom (;) kako ne bi došlo do pogreški u izvršavanju.

Za kraj ostaje redak *cout<<"Hello World!";* s pomoću kojeg se ispisuje tekst Hello World!, ali o njemu nešto više u nastavku rada. S pomoću nastavka te linije *// Ovo je komentar* samo se prikazuje jedan od načina pisanja komentara uz kod koji se sami ne izvršavaju.
Treba naglasiti da program može sadržavati više funkcija sličnoga oblika s pomoću kojih je isto tako moguće primati i vraćati varijable. One nisu obavezne kao glavna funkcija (engl. *main*), pa, ako ih želimo upotrijebiti, takve funkcije pozivamo unutar glavnoga programa.

4.2. Tipovi varijabli

U programu C++ postoji više tipova varijabli, ali nas zanimaju samo sljedeće:

int – cijeli brojevi, *float* – decimalni brojevi, *char* – jedan karakter, *string* – tekst i *bool* koji sprema samo dvije vrijednosti, *true* ili *false*.

Deklariramo ih sintaksom: *tip_ varijable naziv_varijable = vrijednost_varijable*. Pogledajmo to i u primjeru.

```
int n = 5, m = 6, k, l;
float x = – 5.43;
char slovo = 'D';
string tekst = "Hello";
bool b = true;
```

Kod 4.2.1. *Primjer deklariranja varijabli*

4.3. Operatori i komunikacija s korisnikom

Popis potrebnih operatora podugačak je, no zato su svi operatori dobro poznati.

Počnimo s već spomenutim operatorom pridruživanja = (jednako) s pomoću kojeg se vrijednost s desne strane pridružuje vrijednosti s lijeve strane i aritmetičkim operatorima koji nam trebaju, a to su:

+ (zbrajanje), – (oduzimanje), * (množenje), / (dijeljenje) te možda i najvažniji % (modulo, tj. ostatak cjelobrojnog dijeljenja).

Naglasimo da te operatore možemo i vezati u nizu kao u primjeru. Upotrebljavat ćemo i operatore inkrementiranja (x++ ili ++x) i dekrementiranja (x- - ili - -x) koji povećavaju, tj. smanjuju vrijednost x za 1.

Za komunikaciju s korisnikom upotrebljavat ćemo samo dvije naredbe, a to su već spomenuta naredba za ispis *cout*<<, s pomoću koje se ispisuje neki tekst ili određena varijabla, i naredba *cin*>>, s pomoću koje se prima vrijednost i sprema u zadanu varijablu. [5]

U nastavku se nalazi primjer koda u koji je potrebno unijeti neki broj, učiniti nekoliko operacija s njime i vrati određeni rezultat.

```
float n = 5.1, m = -2.6, k, l;
cout<< " Unesite broj: ";
cin>>k;
l=4*k-3n+m/2;
l++;
cout<<  "Rezultat je " << l;
```

Kod 4.3.1. *Primjer uporabe operatora i komunikacije s korisnikom*

Npr. za uneseni broj 11 program će izbaciti „Rezultat je 28.4“.

Pogledajmo i osnovne logičke operatore s pomoću kojih formiramo logičke izraze:

< (strogo manje), <= (manje ili jednako), > (strogo veće), >= (veće ili jednako)

i dva operatora jednakosti: == (jednako) i != (različito).

Osim njih, u sljedećim poglavljima upotrebljavat ćemo i malo složenije logičke operatore:

&& (logičko I), || (logičko ili), ! (logičko NE).

4.4. Naredbe uvjetnoga grananja

S pomoću naredbi uvjetnoga grananja omogućuje se izvršavanje pojedinih grupa naredbi u ovisnosti o tome jesu li ispunjeni određeni uvjeti. To su osnovne naredbe kao *if*, *if-else* i *switch*.

Naredba *if* ima sljedeći oblik: *if(uvjet) {blok naredbi}*, gdje program provjerava uvjet i,ako je istinit, izvršava blok naredbi, a ako nije, nastavlja dalje.

Slično tome, naredba *if-else* ima sljedeći oblik: *if(uvjet) {blok1} else {blok2}*. U okviru te naredbe isto se tako provjerava uvjet i, ako je istinit, izvršava se blok naredbi *blok* , a ako nije, umjesto bloka naredbi *blok1* izvršava se blok naredbi *blok2*. U praksi je po potrebi moguće dodati nekoliko *else if* naredbi između naredbi *if* i *else*.

Naredbom *switch* provjeravaju se različite opcije za zadanu varijablu *izraz* i izvršavaju se blokovi kodova ovisno o istinitosti slučaja (engl. *case*). Ta je naredba oblika: s*witch(izraz) { case 1: blok1; break; case 2: blok2; break; ... default: zadani_blok }*. U ovom se slučaju uglavnom upotrebljava naredba *break;* kako bi se prekinulo njezino izvođenje. Ista se naredba upotrebljava i u petljama za izlaz iz njih.

Pogledajmo sljedeći primjer u kojemu korisnik unosi željeni broj te program prvo provjerava je li taj broj manji ili jednak 0. Ako jest manji ili jednak 0, ispisuje prigodnu poruku, a ako nije provjerava je li taj broj veći od 0 i manji od 123 i, ako je, ispisuje drugu poruku. Ako ni jedan od prva dva uvjeta nije ispunjen, izvršava se treći blok tj. ispisuje se treća poruka.

```
float k;
cout<< " Unesite broj: ";
cin>>k;
if (k<=0) { cout<<" Uneseni broj manji je ili jednak nuli. "; }
else if (k>0 && k<=123) { cout<<" Uneseni broj veći je od 0 i manji ili jednak broju 123. ";}
else { cout<<" Uneseni broj veći je od broja 123. ";}
```

Kod 4.4.1. *Primjer uporabe naredbi uvjetnoga grananja*

Npr. ovaj program za unos broja „-34.5“ ispisuje poruku " Uneseni broj manji je ili jednak nuli. "

4.5. Petlje

Petlje su ključne za razumijevanje teksta u nastavku, kao i za programiranje općenito. Petlje koje će se obraditi u ovome radu jesu petlje *while, do/while* i *for*.

Petlja *while* ima oblik: *while (uvjet) { blok }* i radi na jednostavnome principu da izvršuje blok naredbi sve dok je uvjet zadovoljen.

Do/while varijanta je petlje *while* koja ima oblik: *do { blok } while(uvjet);* čije je funkcioniranje veoma slično funkcioniranju petlje *while*, pri čemu je jedina razlika to da prvo svakako jednom odradi zadani blok naredbi, a tek onda provjerava uvjet.

Petlja *for* ima oblik: *for (izraz1; izraz2; izraz3) {blok}* i funkcionira da se *izraz1* izvrši jedanput prije izvršavanja bloka, *izraz2* definira uvjet za izvršavanje bloka, a *izraz3* izvršava se svaki put nakon izvršenja bloka.

Proučimo jednu takvu petlju na sljedećem primjeru. Program traži unos broja, a petlja *for* prvo postavlja brojač, tj. varijablu *i* na vrijednost 0 te nakon toga provjerava uvjet, odnosno je li brojač manji od 10, te kada je uvjet ispunjen, izvršava blok naredbi. Na kraju se brojač povećava za 1 i tako se ispisuje niz od 10 brojeva, od kojih je svaki uvećan za 7 u odnosu na prethodni, počevši od unesenog broja.

```
float k;
cout<< " Unesite broj: ";
cin>>k;
for (int i=0; i<10; i++) {
k=k+7;
cout << k << "  ";
}
```

Kod 3.5.1. *Primjer uporabe naredbi uvjetnoga grananja*

Uzmimo za primjer da za unos broja „-23“ program ispisuje sljedeće: „*-16 -9 -2 5 12 19 26 33 40 47“* .

4.6. Polja

Polja brojeva upotrebljavaju se za spremanje više podataka istoga tipa u jednu varijablu. Za deklaraciju se upotrebljava isti način kao i za standardne varijable uz dodatak kockastih zagrada u koje se upisuje broj elemenata koje se žele u njih spremiti. [6]

Vrijednosti se mogu spremati u polja pri samoj deklaraciji, ali i kasnije, najčešće uporabom petlji. Time se pristupa polju i u njega sprema vrijednost na željeno mjesto unutar polja. Pri tome valja naglasiti da se mjesta u polju kreću od nultog mjesta, a ne od prvog. Petlje su i najčešći način ispisa svih podataka iz polja.

Pogledajmo primjer u kojemu se deklarira niz „auti" od četiri elementa, nakon čega se od korisnika traži unos željene marke i zamjenjuje drugi element niza upravo unesenim tekstom te se nakon toga uporabom petlje *for* ispisuju svi elementi toga polja.

```
string auti[4] = {"Audi", "Ford", "BMW", "Mercedes"};
string k;
cout<< " Unesite marku automobila: ";
cin>>k;
auti[1]=k;
for(int i = 0; i < 4; i++) {
  cout << auti[i] << " ";
}
```

Kod 3.5.2. *Primjer polja*

U ovome primjeru koda će se za uneseni tekst „Opel" ispisati sljedeće: „*Audi Opel BMW Mercedes"*.

C++ programski jezik opširniji je i kompliciraniji od spektra ovdje objašnjenih pojmova. Uz polja obično vežemo i reference i pokazivače, a on podrazumijeva i objektno orijentirana načela i pojmove kao što su klase, konstruktori, enkapsulacija, nasljeđivanje, polimorfizam i slični, ali kako za potrebe ovoga rada nije nužno daljnje proučavanje, u nastavku teksta nećemo se baviti tom temom.

5. PROSTI BROJEVI i C++

5.1. Provjera složenosti, uvrnutosti i ugniježđenosti broja

Prvi program s kojim ćemo se baviti u okviru ovoga rada jest program C++ koji provjerava je li uneseni broj složen ili prost. Uzevši u obzir definiciju prostoga broja, cilj našega programa bit će provjera djeljivosti unesenog prirodnog broja *n* s ijednim prirodnim brojem iz raspona od 2 do *n*.

Naravno, kako bi se skratilo vrijeme rada programa, vrlo je očigledno da nije potrebno analizirati sve brojeve do *n*, nego je dovoljno provjeriti brojeve do *sqrt(n)*.

Navedeno funkcionira iz prilično jednostavnoga razloga: složeni broj ima barem jedan prosti faktor manji ili jednak njegovu korijenu, a do dokaza za tu tvrdnju jednostavno dolazimo primjenom metode kontradikcije:

Dokaz. *Ako pretpostavimo da su a i b prosti faktori broja n, onda slijedi da a*b = n i da su oba faktora veća od* $\sqrt{n}$*, iz čega slijedi a*b >* $\sqrt{n} * \sqrt{n}$*, što je kontradiktorno pretpostavci i time dokazuje tvrdnju.* ■

Sqrt programska je oznaka za matematičku operaciju korijena i nalazi se u biblioteci *<math.h>*, zbog čega je i ona na početku obuhvaćena. Provjeru ćemo odraditi uporabom varijable *bool* i petlje *for* te ćemo provjeravati ostatak cjelobrojnog dijeljenja unutar naredbe *if*, s pomoću koje će se promijeniti vrijednost varijable *bool* i s pomoću naredbe *break* prekinuti petlja *for* ako dođe do uspješnog dijeljenja unesenog broja s bilo kojim brojem iz zamišljenoga spektra. Navedeni kod vidljiv je u sljedećem primjeru.

```
#include <iostream>
#include <math.h>
using namespace std;

int main() {
  int n;
  bool boolProsti = true;
  cout << "Unesite prirodan broj: ";
  cin >> n;
```

```
  for (int i = 2; i <= sqrt(n); ++i) {
    if (n % i == 0) {
      boolProsti = false;
      break;  }
  }
  if (boolProsti)
    cout << n << " jest prost broj";
  else
    cout << n << " nije prost broj";
  return 0;
}
```

Kod 5.1.1. *Provjera složenosti broja*

Sada kada znamo kako provjeriti je li broj prost znamo provjeriti i složenost još jednog interesantnog broja, a to je uvrnuti prosti broj. Uvrnuti prosti broj je broj koji je prost i uz to njegov „obrnuti“ broj je isto prost npr. broj 13 i broj 31. Za to će nam trebati mali dodatak na prošli kod koji će obrnuti uneseni broj uporabom modulo (%) operatora i *while* petlje te istim načinom provjeriti oba i ako su oba prosta to i ispisati.

```
#include <iostream>
#include <math.h>
using namespace std;

int main() {
  int n;
  bool boolProsti = true;
  cout << "Unesite prirodan broj: ";
  cin >> n;
  for (int i = 2; i <= sqrt(n); ++i) {
    if (n % i == 0) {
      boolProsti = false;
      break;
    }
  }
```

```
    if (boolProsti)
   cout << n << " jest prost broj";
 else
   cout << n << " nije prost broj";

  int obrnuti = 0, o;
  while (n > 0) {
     o = n % 10;
     obrnuti = obrnuti * 10 + o;
     n /= 10;
  }
    bool boolObrnuti = true;
 for (int i = 2; i <= sqrt(obrnuti); ++i) {
   if (obrnuti % i == 0) {
     boolObrnuti = false;
     break;
   }
 }
if (boolProsti && boolObrnuti)
   cout << " i uvrnuti prost broj";
 else
   cout << " i nije uvrnuti prost broj";
 return 0;  }
```

Kod 5.1.2. *Provjera složenosti i uvrnutosti broja*

Više od uvrnutosti broja matematičarima je zanimljiva ugniježđenost prirodnoga broja između dvaju prostih brojeva (parova blizanaca). S pomoću sljedećeg koda provjerava se upravo to svojstvo za uneseni prirodni broj *n*, tj. provjerava se jesu li brojevi *(n – 1)* i *(n + 1)* prosti.

Za ovaj program korištena je dodatna funkcija *jeProsti* koja u suštini izvršava funkciju koda 4.1.1., tj. za primljeni broj varijabla *bool* ima vrijednost *true* ako je prost i *false* ako nije. Ta funkcija definirana je izvan glavne funkcije (engl. *main*) i možemo je pozivati unutar glavne funkcije kada nam treba njezina funkcionalnost.

```
#include<iostream>
#include<math.h>
using namespace std;

bool jeProst(int n)
{
    for (int i = 2; i <=sqrt(n); i++)
        if (n % i == 0)
            return false;
    return true;
}

int main()
{
    int n;
   cout << "Unesite prirodan broj: ";
   cin >> n;



   if (jeProst(n-1) && jeProst(n+1))
      cout<<"Je ugniježđen";
   else
      cout<<"Nije ugniježđen";

    return 0;
}
```

Kod 5.1.3. *Provjera ugniježđenosti broja*

5.2. Eratostenovo sito i specifični tipovi prostih brojeva

Za ispis prostih brojeva od 2 do zadanoga broja *n* može se koristiti standardna petlja *for* i svaki broj redom provjeravati s pomoću prijašnjega koda, a isto je moguće učiniti i poznavanjem algoritma Eratostenova sita i uporabom polja.

Sljedeći će kod prvo tražiti korisnika da unese prirodni broj do kojeg želi ispis prostih brojeva te će generirati polje s varijablama *bool* koje s pomoću jedne petlje *for* sve inicijalno postavlja na vrijednost *true*. Nakon toga će primjenom algoritma i proučavajući mjesta u polju, redom koristeći dvije petlje *for* i jednu naredbu *if*, iz niza „izbacivati“ sve višekratnike brojeva krenuvši od broja 2 mijenjajući vrijednost varijable *bool* iz *true* u *false*. Na kraju program ispisuje sve elemente (sva mjesta) polja čija je vrijednost ostala nepromijenjena (*true*).

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{

  int n;
  cout << "Unesite prirodan broj: ";
  cin >> n;

        bool prosti[n+1];

        for (int i=1; i<=n+1; i++)
        {
           prosti[i]=true;
        }

        for (int i=2; i<=sqrt(n+1); i++)
        {

                if (prosti[i] == true)
                {

                        for (int j=i*i; j<=n+1; j += i)
                                prosti[j] = false;
                }
        }
```

```
        for (int i=2; i<=n+1; i++)
        if (prosti[i])
                cout << i << " ";


   return 0;
}
```

Kod 5.2.1. *Eratostenovo sito*

Ako se pozna princip Eratostenova sita, jednostavnije se ispisuju specifični tipovi prostih brojeva kao što su Mersennovi primbrojevi i primbrojevi Sophie-Germain. Kao što je već definirano, Mersennovi prosti brojevi imaju oblik 2^n-1 pa je jednostavno primjenom prethodno navedenoga koda promijeniti ispis koristeći se upravo tom definicijom kao uvjetom. Za podizanje broja 2 na eksponent *n* primijenit će se ugrađena funkcija *pow(2,n)*.

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    int n;
  cout << "Unesite prirodan broj: ";
  cin >> n;
        bool prosti[n+1];

        for (int i=1; i<=n+1; i++)
        {
           prosti[i]=true;
        }

        for (int i=2; i<=sqrt(n+1); i++)
        {

                if (prosti[i] == true)
                {

                        for (int j=i*i; j<=n+1; j += i)
                                prosti[j] = false;
                }
        }
```

```
  for (int i=2; (pow(2,i)) <= n; i++)
    {
       int merssenov = pow(2,i) - 1;

       if (prosti[merssenov])
          cout << merssenov << " ";
    }
    return 0;
}
```

Kod 5.2.2. *Ispis Mersennovih brojeva*

Primjenom sličnog principa ispisat će se i primbrojevi Sophie-Germain. To su prosti brojevi *p* za koje vrijedi da je i *2p + 1* prost broj. Uz primjenu Eratostenova sita, ovoga je puta potrebno proširiti polje s unesenoga broja *n* na 2*n* + 1.

```
#include<iostream>
#include<math.h>
using namespace std;

int main()
{
    int n;
    cout << "Unesite prirodan broj: ";
    cin >> n;
       bool prosti[2*n+1];

     for (int i=0; i<=2*n+1; i++)
       prosti[i] = true;

    for (int p=2; p<=sqrt(2*n+1); p++)
    {
       if (prosti[p] == true)
       {
          for (int i=p*2; i<=2*n+1; i += p)
             prosti[i] = false;
       }
```

```
}
   for (int i = 2; i <= n; ++i) {
      if (prosti[i] && prosti[2 * i + 1])
         cout << i << " "; }

   return 0;
}
```

Kod 5.2.3. *Ispis primbrojeva Sophie-Germain*

U nastavku se nalazi usporedba „uspješnosti“ već spomenutih dvaju specifičnih tipova prostih brojeva: Mersennovih primbrojeva i primbrojeva Sophie-Germain, odnosno usporedba rezultata s Eratostenovim sitom za isti uneseni broj *n*. Ovo je ispis koda do prvih $n = 200$ prirodnih brojeva:

Eratostenovo sito: *„2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199“*

Mersennovi primbrojevi: *„3 7 31 127“*

Primbrojevi Sophie-Germain: *„2 3 5 11 23 29 41 53 83 89 113 131 173 179 191“*

Odmah je očigledno da Mersennovi primbrojevi i primbrojevi Sophie-Germain definiraju samo manji dio svih prostih brojeva i da su zapravo vrlo neefikasni u toj funkciji. Međutim, njihova „uspješnost“ ne može se gledati isključivo kroz taj spektar, nego i kroz ono za što su oni namijenjeni i za što se upotrebljavaju, kao što je već spomenuta pretraga najvećega prostoga broja, što ih s matematičkog aspekta čini dodatno zanimljivima.

Kada se radi o ispisu matematičarima zanimljivih prostih brojeva, trojke prostih brojeva nezaobilazna su tema. Trojke prostih brojeva niz su od tri prosta broja oblika $(p, p + 2, p + 6)$ ili $(p, p + 4, p + 6)$.

Za ispis će se primjenjivati već poznato Eratostenovo sito, ali i uvjeti iz same definicije trojki prostih brojeva. Pritom je moguće uočiti da raspon za ispis ide do $n – 6$ te da je uvedena naredba endl, s pomoću koje se ispis prenosi u novi red radi organiziranijeg ispisa podataka.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{

   int n;

   cout << "Unesite prirodni broj: ";

   cin >> n;


        bool prosti[n+1];


        for (int i=1; i<=n+1; i++)
        {
           prosti[i]=true;
        }

        for (int i=2; i<=sqrt(n+1); i++)
        {

                if (prosti[i] == true)
                {

                        for (int j=i*i; j<=n+1; j += i)
                                prosti[j] = false;
                }  }

     for (int i = 2; i <= n-6; ++i) {


      if (prosti[i] && prosti[i + 2] && prosti[i + 6])

         cout << i << " " << (i + 2) << " " << (i + 6) << endl;


      else if (prosti[i] && prosti[i + 4] && prosti[i + 6])

         cout << i << " " << (i + 4) << " " << (i + 6) << endl;    }

   return 0;

}
```

Kod 5.2.4. *Ispis trojki prostih brojeva*

5.3. Ispis iz raspona, Bertrandov postulat i Legrendeova pretpostavka

Za ispis prostih brojeva iz unesenoga raspona upotrijebit će se dio koda za provjeru prostih brojeva iz pretprošloga poglavlja, kao i dodatna petlja *for*, koja će se vrtjeti od donje do gornje unesene granice unutar koje provjeravamo svaki element i, ako je broj prost, ispisujemo ga.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
   int n, m;
   cout << "Unesite donju granicu: ";
   cin >> n;
   cout << "Unesite gornju granicu: ";
   cin >> m;

   for(int i = n+1; i < m; ++i)
   {
   bool boolProsti = true;
  for(int j = 2; j <= sqrt(i); ++j)
   {
      if (i%j == 0)
      {
         boolProsti = false;
         break;
      }  }
      if(boolProsti)
         cout << i << " ";
   }
   return 0;
}
```

Kod 5.3.1. *Ispis iz raspona*

Nakon ispisa iz raspona potrebno je posvetiti se dvjema zanimljivim temama vezanima za ispis iz određenih raspona, a to su Bertrandov postulat i Legrendeova pretpostavka.

Bertrandov postulat tvrdnja je kako postoji prosti broj u rasponu od n do $2n-2$, gdje je n prirodan broj veći ili jednak 4. Kod naveden u nastavku teksta izvršava tu funkciju, odnosno za uneseni broj n traži sve proste brojeve u definiranom rasponu i ispisuje ih.

```
#include<iostream>
#include<math.h>
using namespace std;

bool jeProst(int n)
{
   for (int i = 2;  i <=sqrt(n); i++)
      if (n % i == 0)
         return false;
   return true;
}

int main()
{
   int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;

      cout << "Prosti brojevi u rasponu od " << n << " do "
       << 2 * n - 2 << ": ";
   for (int i = n + 1; i < 2 * n - 2; i++)
      if (jeProst(i))
         cout << i << " ";

   return 0;
}
```

Kod 5.3.2. *Bertrandov postulat*

Osim Bertrandova postulata valja spomenuti i Legrendeovu pretpostavku u okviru koje se tvrdi da postoji barem jedan prosti broj između kvadrata dvaju uzastopnih prirodnih brojeva. Iako je to samo pretpostavka temeljena na ograničenom broju podataka, što znači da nije dokazana, još uvijek nije pronađen ni jedan prirodni broj za koji ona ne vrijedi.

Sljedeći kod odnosi se upravo na tu pretpostavku i za uneseni prirodni broj *n* radi ispis svih prostih brojeva iz raspona n^2 i $(n + 1)^2$ uporabljujući slične principe kao i kod naveden prethodno u radu.

```
#include<iostream>
#include<math.h>
using namespace std;

bool jeProst(int n)
{
   for (int i = 2; i <=sqrt(n); i++)
       if (n % i == 0)
           return false;
   return true;
}
 int main()
{
   int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;
   cout << "Prosti u rasponu od "<<n*n  << " do "<<(n+1)*(n+1)  <<" su: ";

  for (int i = n*n; i <= ((n+1)*(n+1)); i++)

     if (jeProst(i))
         cout << i << " ";

  return 0;
}
```

Kod 5.3.3. *Legrendova pretpostavka*

Zanimljivo je uočiti kako se za brojeve unesene u kodove za Bertrandov postulat i Legrendovu pretpostavku (pa i Goldbachovu koja se nalazi u nastavku rada) uspješno dobivaju rezultati. Time se postiže dodatna sigurnost u istinitost tih nedokazanih pretpostavki s obzirom na to da i današnja računala ne mogu pronaći broj za koji se ne bi dobio rezultat.

5.4. Goldbachova pretpostavka

Jedna od najstarijih i najpoznatijih neriješenih problema u teoriji brojeva jest Goldbachova pretpostavka. Radi se o tvrdnji da se svaki parni prirodni broj može zapisati kao suma dvaju prostih brojeva. Program u nastavku teksta ima funkciju da za uneseni parni prirodni broj ispisuje sve kombinacije dvaju prostih brojeva čija je suma jednaka unesenome broju.

Isto je tako moguće pojednostavniti kod i ispisati tek prvu kombinaciju dodavanjem naredbe *break;* na kraju tijela zadnjega uvjetnoga grananja *if*. Program prvo primjenjuje Eratostenovo sito nakon čega s pomoću petlje *for* koja se izvršava do granice *n*/2 za svaki prost broj u polju redom provjerava složenost razlike unesenoga broja i brojača petlje funkcijom *jeProst* te ih, ako je i razlika prost broj, ispisuje u obliku sume.

```
#include <iostream>
#include <math.h>
using namespace std;

bool jeProst(int n)
{
   for (int i = 2; i <=sqrt(n); i++)
      if (n % i == 0)
         return false;
   return true;
}


int main()
{

            int n;
   cout << "Unesite parni prirodni broj: ";
   cin >> n;
```

```
        bool prosti[n+1];

        for (int i=1; i<=n+1; i++)
        {
           prosti[i]=true;
        }


      for (int i=2; i<=sqrt(n+1); i++)
        {

                if (prosti[i] == true)
                {

                        for (int j=i*i; j<=n+1; j += i)
                                prosti[j] = false;
                }
        }


                for (int i=0 ; i <= n/2; i++)
  {

  if (prosti[i])  {

     int razlika = n - i;

     if (jeProst(razlika))
     {

        cout << i << " + " << razlika << " = "
            << n << endl;

     }
  }
}

  return 0;
}
```

Kod 5.4.1. *Goldbachova pretpostavka*

Ako se uzme u obzir da je Golbachova pretpostavka točna, slijedom logike moguće je doći i do pretpostavke da se svaki prirodni broj može zapisati kao suma najviše triju prostih brojeva. Upravo će se ta logika primjenjivati u sljedećem programu koji za uneseni prirodni broj vrši ispis prema toj pretpostavci, u skladu s koracima u nastavku.

Prvo se vrši provjera je li uneseni broj *n* sam po sebi prost, a ako je, ispisuje se. Ako *n* nije prost, provjerava se složenost (– 2) te, ako je taj broj prost, ispisuje se u obliku: 2 + (*n* –2). Ako *n* i (*n* – 2) nisu prosti, (*n* – 3) mora biti paran broj, pa se prema Goldbachovoj pretpostavci ispisuje kao suma broja 3 i dvaju prostih brojeva dobivenih na temelju Goldbachove pretpostavke, tj. prema ideji koda 4.4.1. Važno je naglasiti da se kod 4.4.1. može i drugačije realizirati, tj. petljom *for*, naredbom *if* i funkcijom *jeProst* bez uporabe polja, kako je i prikazano u sljedećem kodu.

```
#include <iostream>
#include <math.h>
using namespace std;

bool jeProst(int n)
{
   for (int i = 2; i <=sqrt(n); i++)
      if (n % i == 0)
         return false;
   return true;
}

int main()
{
            int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;

      if (jeProst(n))
      cout << n << endl;

   else if (jeProst(n - 2))
      cout << 2 << " + " << n - 2 << endl;

   else
   {
```

```
        cout << 3 << " ";
        n = n - 3;
        for (int i = 2; i < n; i++) {
           if (jeProst(i) && jeProst(n - i)) {
              cout << i << " " << (n - i);
              break;
           }
}
}
   return 0;  }
```

Kod 5.4.2. *Prirodan broj kao suma prostih brojeva*

5.5. Faktorizacija i susjedi

Sljedeći program imat će funkciju rastavljanja prirodnih brojeva na proste faktore. Sljedeći kod standardno će primati jedan prirodni broj te prvo provjeravati djeljivost s 2 i dijeliti s 2 te ispisivati dok može. Nakon toga se pretpostavlja da će broj koji je preostao biti neparan, što znači da će, kada se bude upotrebljavala u nastavku, petlja *for* kretati od vrijednosti 3, a korak će joj biti 2. Na kraju se, naravno, ispisuje ostatak. Primjerice, za uneseni broj 611940 program će ispisati: „*2 2 3 5 7 31 47*“.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
   int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;
   while (n % 2 == 0)
   {
      cout << 2 << " ";
      n = n/2;    }
```

```
   for (int i = 3; i <= sqrt(n); i = i + 2)
   {  while (n % i == 0)
      {
         cout << i << " ";
         n = n/i;
      }  }
   if (n > 2)
      cout << n << " ";
   return 0;
}
```

Kod 5.5.1. *Faktorizacija*

Jednom kada se svladaju osnove traženja brojeva iz određenoga raspona i jednodimenzionalnih polja, moguće je napisati kod koji će prvo pretražiti brojeve iz raspona i spremiti ih u niz, a onda ispisivati one koji su međusobno udaljeni za korisnikovu željenu razliku. Ako korisnik bude želio pronaći parove blizance, vrijednost razlike postavit će na 2, a ako bude želio pronaći parove rođake, ta će vrijednost iznositi 4.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
      int n, m, r, b=0;
      bool boolProsti;

   cout << "Unesite donju granicu: ";
   cin >> n;
   cout << "Unesite gornju granicu: ";
   cin >> m;
   cout << "Unesite razliku: ";
   cin >> r;
```

```
for(int i = n+1; i < m; ++i)
{
   boolProsti = true;
for(int j = 2; j <= sqrt(i); ++j)
{
   if (i%j == 0)
   {
      boolProsti = false;
      break;
   }
}
  if(boolProsti) {
      b++; }
}
int Prosti[b];
b=0;

for(int i = n+1; i < m; ++i)
{
boolProsti = true;

for(int j = 2; j <= i/2; ++j)
{
   if (i%j == 0)
   {
      boolProsti = false;
      break;
   }
}
          if(boolProsti) {
      Prosti[b]=i;
      b++; }
}
```

```
   for (int i=0; i<b; i++) {
   if (Prosti[i+1]-Prosti[i]==r)
   {
   cout << "( " << Prosti[i] << "," << Prosti[i+1] << " ) ";
   }
   }

   return 0;
}
```

Kod 5.5.2. *Susjedi*

Ako se navedeni kod pomnije promotri, prvo se primijeti da su, kao i do sada, tražene granice, ali sada se uz njih od korisnika traži da unese željenu razliku. Uočavamo da su dvaput napravljene iste petlje za pretraživanje brojeva kako bi se „zaobišla“ uporaba naprednih koncepata ovoga programskoga jezika. U nekim drugim programskim jezicima, poput jezika PHP, gdje se jednostavno može dodati element na kraj niza, to ne bi ni bilo potrebno. Primjenom prve petlje prije svega se s pomoću brojača *b* provjerava količina prostih brojeva u zadanom intervalu. To se radi kako bi se niz *Prosti* što jednostavnije inicijalizirao znajući koliko je potrebno mjesta u memoriji. Nakon toga se ponavlja petlja, pri čemu se u niz spremaju svi prosti brojevi od najmanjega prema najvećemu. Treća petlja obavlja zadnji dio posla i provjerava razliku svakoga element niza i, ako je jednaka zadanoj razlici, petlja ispisuje taj par. Primjerice, s pomoću funkcije za unos donje granice 50 i gornje 150 te razlike 2 dobit ćemo sljedeće: *„(59, 61) (71, 73) (101, 103) (107, 109) (137, 139“).*

5.6. Sundaramovo i Atkinovo sito

Kao što je već spomenuto, Sundaramovo i Atkinovo sito naprednije su verzije sita u odnosu na Eratostenovo. Na temelju koda u nastavku analizirat će se algoritam Sundaramova sita. Za ispis prostih brojeva do unesenoga broja *n* s pomoću Sundaramova sita potrebna je nova granica do koje će se definirati polja i vršiti petlje. Tu granicu nazvat ćemo *nNovi* i zadati joj vrijednost (*n* – 1)/2. Slično kao u slučaju Eratostenova sita, nakon toga se radi polje brojeva i primjenjuje se petlja *for* za postavljanje početne vrijednosti na *true*.

Nakon toga slijedi ključni dio: s pomoću dviju petlje *for* s brojačima *i* i *j* označuju se svi elementi polja oblika *i + j + 2ij* kao *false*, gdje je 1 <= *i* <= *j*. Preostali prosti brojevi imaju oblik 2*i* + 1. Nakon toga slijedi ispis. Prvo se ispisuje 2 kao prvi prosti broj, nakon čega se ispisuju svi *true* „elementi" polja u zadanom obliku 2*i* + 1.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
   int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;

   int nNovi = (n-1)/2;
   bool prosti[nNovi + 1];

         for (int i=1; i<=nNovi+1; i++)
                  prosti[i]=true;

   for (int i=1; i<=nNovi; i++)
      for (int j=i; (i + j + 2*i*j) <= nNovi; j++)
         prosti[i + j + 2*i*j] = false;

   if (n > 2)
      cout << 2 << " ";

   for (int i=1; i<=nNovi; i++)
      if (prosti[i] == true)
         cout << 2*i + 1 << " ";

   return 0;  }
```

Kod 5.6.1. *Sundaramovo sito*

Iako se čini da ima više koraka, Atkinovo sito teorijski je jednostavnije od prijašnjih dvaju sita i temelji se na trima specifičnima oblicima prostih brojeva s uvjetima. Kao i kod Eritostenova sita, prvo se uzima unesena granica *n*, inicijalizira polje s *n* članova i postavlja vrijednost na *false*.

Za potrebe ovoga sita najlakše je najprije ispisati prva dva prosta broja, odnosno 2 i 3. Nakon toga dolazi glavni dio iza logike Atkinova sita koji se sastoji u promjeni vrijednosti elemenata polja na *true*, ali u nekoliko koraka, tj. uvjeta. To se postiže primjenom dviju petlji *for* s brojačima *x* i *y* te odrađivanjem zadanih koraka.

1. korak: postaviti vrijednost *true* za brojeve oblika $n=4x^2+y^2$ za koje vrijedi *n % 12 == 1* ili *n % 12 == 5*

2. korak: postaviti vrijednost *true* za brojeve oblika $n=3x^2+y^2$ za koje vrijedi *p % 12 == 7*

3. korak: postaviti vrijednost *true* za brojeve oblika $n=3x^2-y^2$ kada je $x > y$ za koje vrijedi *p % 12 == 11*

4. korak: s pomoću dvije petlje *for* ukloniti sve one složene brojeve u vidu višekratnika kvadrata kojima je u prva tri koraka greškom pripisana vrijednost *true* i postaviti im vrijednost *false*.

Na kraju, kao i uvijek, ispisuju se prosti brojevi.

```
#include <iostream>
#include <math.h>
using namespace std;
 int main()
{
   int n;
   cout << "Unesite prirodni broj: ";
   cin >> n;
      bool prosti[n];
   for (int i = 0; i < n; i++)
      prosti[i] = false;
   if (n > 2)
      cout << 2 << " ";
   if (n > 3)
      cout << 3 << " ";
```

```
  for (int x = 1; x < sqrt(n); x++) {
        for (int y = 1; y < sqrt(n); y++) {
          //1. korak
           int p = (4 * x * x) + (y * y);
           if (p <= n && (p % 12 == 1 || p % 12 == 5))
              prosti[p] = true;
           //2. korak
           p = (3 * x * x) + (y * y);
           if (p <= n && p % 12 == 7)
              prosti[p] = true;
           //3. korak
           p = (3 * x * x) - (y * y);
           if (x > y && p <= n && p % 12 == 11)
              prosti[p] = true;
        }
    }
    // 4. korak
    for (int i = 5; i < sqrt(n); i++) {
        if (prosti[i]) {
           for (int j = i * i; j < n; j += i * i)
              prosti[i] = false;
        }
    }

    for (int i = 5; i < n; i++)
        if (prosti[i])
           cout << i << " ";

    return 0;
}
```

Kod 5.6.2. *Atkinovo sito*

Kada je riječ o teorijskoj kompleksnosti algoritama Sundaramova, Atkinova i Eratostenova sita, nužno je usporediti ih s brzinom izvođenja implementiranih kodova. Teorijska kompleksnost Eratostenova sita iznosi $O_{Era}(NloglogN)$, Sundaramova $O_{Sun}(NlogN)$, a Atkinova $O_{Atk}(N)$, na temelju čega se može zaključiti da je Atkinovo sito teorijski najučinkovitije.

Međutim, ako se proučava kompleksnost kodova za sita, uočava se da se podudaraju s teorijskim kompleksnostima Eratostenova i Sundaramova sita, ali veća složenost koda Atkinova sita u odnosu na njegovu teorijsku odmah je uočljiva na temelju broja operacija, i to unutar dviju petlji *for* , što implementaciju Atkinova sita čini teorijski najsporijim kodom.

To se i potvrđuje mjerenjem prosječnog vremena izvršavanja upotrebom biblioteke <*time.h*> i funkcije *clock()*, na računalu s 2,30 GHz Intel Core i5-2410M procesorom i 4 GB RAM-a.

n	**Eratosten**	**Sundaram**	**Atkin**
$\mathbf{10^3}$	0.00004	0.00009	0.00016
$\mathbf{10^4}$	0.00018	0.00067	0.00094
$\mathbf{10^5}$	0.00177	0.00591	0.01217
$\mathbf{10^6}$	0.04543	0.09134	0.10596
$\mathbf{10^7}$	0.37891	1.16548	1.38253

Tablica 5.6.3. *Usporedba brzine izvođenja kodova sita u sekundama*

Iz rezultata jasno je uočljivo kako za unesene brojeve *n*, naspram Sundaramovog i Atkinovog sita, Eratostenovo sito mnogo efikasnije obavlja zadatak.

6. ZAKLJUČAK

U ovome radu objašnjeni su osnovni matematički pojmovi vezani uz proste brojeve i osnove pisanja kodova u programskom jeziku C++, pa može poslužiti kao podsjetnik ili rubrika „tko želi znati više“ za one koji su možda samo upoznati s osnovnom idejom prostih brojeva, ali žele proširiti svoje znanje ili se žele okušati u programiranju.

Ideje iza njegovih kodova nisu nužno ograničene na isti, tako da mogu biti primjenjive i u drugim programskim jezicima. Usto, može poslužiti kao ušteda vremena programerima, pogotovo onima manje upoznatima s prostim brojevima, kojima treba upravo takvo znanje za posao, gotov kod, dio koda ili kao ideja za njihov rad, potrebe učenja ili nešto treće.

Ipak najvažniji aspekt ovoga rada jest proučavanje matematičarima zanimljivih i aktualnih algoritama za analizu prostih brojeva kroz prizmu računalnog programskog jezika C++. U njemu su obrađeni, analizirani i komentirani kodovi za različite provjere prostih brojeva prema njihovim svojstvima i tipovima, za faktorizaciju složenih brojeva, razna sita i drugi ispisi iz određenih raspona i prema specifičnim kriterijima i pretpostavkama. Svi ti algoritmi i kodovi su bili i biti će korišteni u analizi područja s još mnogo neodgovorenih pitanja i nedokazanih tvrdnji pod nazivom prosti brojevi.

LITERATURA

[1] N. Elezović, Diskontna matematika, Element, Zagreb, 2017.

[2] http://primes.utm.edu [18.7.2020.]

[3] https://www.mersenne.org/ [14.8.2020.]

[4] B. Motik, J. Šribar, Demistificirani C++, Element, Zagreb, 1997

[5] https://www.w3schools.com/cpp/ [24.7.2020.]

[6] https://www.geeksforgeeks.org [12.9.2020.]

SAŽETAK

Algoritmi za proste brojeve u programskom jeziku C++

Glavna ideja ovoga rada jest proučavanje dviju tema. Prva tema koja se obrađuje jesu prosti brojevi. Obrađuje se pojam, skupovi, tipovi i uporaba prostih brojeva. Druga tema koja se obrađuje jest programski jezik C++. Proučava se općenita sintaksa funkcija, vrste varijabli, operatori, komunikacija s korisnikom, uvjetno grananje, petlje i polja programskog jezika C++. Nakon toga proučava se tema prostih brojeva kroz aspekt programskog jezika C++. Pišu se, proučavaju, kategoriziraju, analiziraju i komentiraju aktualni i zanimljivi algoritmi i kodovi za provjeru složenosti broja, različita sita, ispis prostih brojeva iz zadanoga raspona, za rastavljanje složenoga broja na proste faktore i za ispis prostih brojeva prema njihovim svojstvima.

Ključne riječi: programski jezik C++, prosti brojevi

ABSTRACT

Algorithms for prime numbers in the C++ programming language

The main idea of this paper was to study two topics. The first topic covered are prime numbers. The concept, sets, types and use of prime numbers are covered. Another topic that is covered is the C++ programming language. The general syntax of functions, types of variables, operators, communication with the user, conditional branching, loops and fields of the C++ programming language were studied. After that, comes the topic of prime numbers as seen through the aspect of the C++ programming language. Current and interesting algorithms and codes for checking the complexity of a number, different sieves, printing prime numbers from a given range, for breaking a complex number into prime factors, and for printing prime numbers according to their mutual properties are written, studied, categorized, analysed and commented on.

Keywords: C++ programming language, prime numbers

ŽIVOTOPIS

Borna Milobara rođen je 19.04.1993. u Osijeku. Nakon završene osnovne škole s odličnim uspjehom 2007. upisuje Isusovačku klasičnu gimnaziju s pravom javnosti u Osijeku. Završava srednjoškolsko obrazovanje polaganjem državne mature te, u akademskoj godini 2011/12-toj upisuje Elektrotehnički fakultet, smjer Računarstvo. Tokom studiranja obavlja razne studentske poslove preko Studentskog centra i volontira pripremajući maturante za državnu maturu iz predmeta matematike. 2019. upisuje i završava tečaj za programera web aplikacija na Pučkom otvorenom učilištu Algebra.

Potpis:

_____