# Brzi heuristički algoritam za rješavanje logičke zagonetka Sudoku u programskom jeziku C++

**Juhasz, Josip**

**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# BRZI HEURISTIČKI ALGORITAM ZA RJEŠAVANJE LOGIČKE ZAGONETKE SUDOKU U PROGRAMSKOM JEZIKU C++

**Završni rad**

**Josip Juhasz**

**Osijek, 2020.**

## SADRŽAJ

# 1. UVOD

Igra Sudoku predstavlja matematičku zagonetku koja se rješava logičkim razmišljanjem. Klasični Sudoku se sastoji od velikog kvadratnog polja u kojem se nalazi 81 manjih kvadratića. Veliko kvadratno polje podijeljeno je na 9 odjeljaka veličine 3x3 podkvadrata. Cilj zagonetke je popuniti sva polja brojevima od jedan do devet, tako da se jedan broj smije koristiti samo jednom u svakom redu, stupcu i 3x3 podkvadratu. Cilj ovog završnog rada je osmisliti algoritam koji pronalazi rješenje Sudoku zagonetke. Algoritam mora biti heurističan, što znači da rješavanje Sudoku-a neće zahtijevati mnogo vremena.

Prvo poglavlje opisuje zadatak završnog rada.

Sudoku pravila, tipovi i algoritmi koji se koriste u rješavanju Sudoku zagonetke, predstavljeni su u drugom poglavlju.

U trećem poglavlju opisan je programski jezik C++, te Microsoft Visual Studio i pojam heuristički algoritam koji su korišteni u izradi ovog završnog rada.

U četvrtom poglavlju detaljno je prikazan opis i implementacija algoritma za rješavanje Sudoku-a.

## 1.1 Zadatak završnog rada

Zadatak ovog završnog rada je napraviti heuristički algoritam koji će riješiti Sudoku zagonetku. Pronalazak rješenja Sudoku zagonetke neće zahtijevati mnogo vremena.

# 2. SUDOKU

Igra Sudoku predstavlja matematičku zagonetku koja se rješava logikom. Sudoku je igra koja se igra preko 1000 godina, a u današnjem svijetu je popularna zbog američkog arhitekta Howarda Garnsa koji ju spominje u magazinu *Dell Magazines*. U magazinu se pojavio Sudoku dimenzije 9x9 koji se sastoji od 9 manjih kvadrata dimenzije 3x3. Sudoku je skraćenica duže fraze „*Suuji wa dokushin ni kagiru*“, što u prijevodu znači „brojevi moraju ostati jedinstveni“. Sudoku ima više različitih težina, a uglavnom se koristi za zabavu ili testiranje inteligencije. Novozelandski sudac, Wayne Gould, jedna je od najznačajnijih osoba koja je pomogla u populariziranju Sudoku zagonetke. Wayne je šest godina radio na računalnom programu za generiranje Sudoku-a. 2004. godine londonske novine *The Times* koriste njegov program za tiskanje dnevnih Sudoku zagonetka. Sudoku kao igra u posljednje vrijeme značajno dobiva na popularnosti, te se s toga održavaju turniri i svjetska prvenstva. Prvo svjetsko prvenstvo održalo se 2006. godine u talijanskom gradu Lucca. Prvo mjesto osvojila je češka ekonomistica Jana Tylova i tako postala prva osoba koja je osvojila Svjetsko prvenstvo u Sudoku-u. [1]

## 2.1. Matematika logičke igre Sudoku i osnovni rezultati

Analiza Sudoku zagonetke podijeljena je u dva glavna područja: analiza svojstava dovršenih kvadrata i zagonetke. Prva analiza bila je usredotočena na nabrajanje rješenja, a prvi rezultati su se pojavili 2004. godine. [2]

Rješavanje Sudoku-a s gledišta igrača istraženo je u knjizi Denisa Berthiera *„The Hidden Logic of Sudoku“* 2007. godine koja razmatra strategiju poput *„skrivenih XY-lanaca“*. [3]

Opći problem rješavanja Sudoku-a dimenzija od $n^2$ x $n^2$ pripada u NP-potpune probleme. Zagonetka se može izraziti kao problem obojenih grafova. Cilj je konstruirati 9-bojanje određenog grafa, s obzirom na djelomično 9-bojanje. Sudoku graf, zbog 81 kvadratića, sadrži 81 vrh. Vrhovi su označeni uređenim parovima $(x, y)$, gdje $x$ i $y$ predstavljaju brojeve između jedan i devet. Dva različita vrha označena s $(x, y)$ i $(x', y')$ spojena su rubovima ako i samo ako:

$x=x'$ (isti stupac) ili,

$y=y'$ (isti red) ili,

$[x/3] = [x'/3]$ i $[y/3]=[y'/3]$ (isti 3x3 podkvadrat).

Zagonetka se zatim dovršava dodijeljivanjem brojeva između jedan i devet svakom vrhu, tako da vrhovima koji su spojeni rubom nije dodijeljen isti broj. [4]

Postoji nekoliko računalnih algoritama koji će riješiti većinu 9x9 zagonetki u djelićima sekunde, ali rast kompleksnosti se pojavljuje s povećanjem *n*, stvarajući ograničenja svojstvima Sudoku-a koja se mogu konstruirati, analizirati i riješiti kako se *n* povećava. [5]

### 2.1.1. Backtracking

*Backtracking* algoritam pokušava pogoditi točan rezultat u svakome kvadratiću isprobavajući opcije, a ako se tijekom igra ustanovi da broj nije valjan, algoritam se vraća natrag te ga ispravlja. Zbog toga nosi naziv *backtracking*. Glavne prednosti *backtracking* algoritma su što sigurno donosi rješenje Sudoku-a, vrijeme njegovog rješavanja ne ovisi o težini, ali mu je potrebno značajno vrijeme rješavanja kako bi došao do rješenja. *Backtracking* je prikazan slikom 2.10. [5]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 1 | 2 | 7 | 6 | 8 | 9 | 4 |
| 6 | 2 | 4 | 1 | 9 | 5 | 2 | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Sl. 2.10. Rješavanje Sudoku-a backtrackingom

### 2.1.2. Ograničeno programiranje

Sudoku može biti modeliran kao *constraint satisfaction problem* – koriste se algoritmi čiji objekti moraju zadovoljiti ograničenja. U ograničenom programiranju koriste se algoritmi zaključivanja na temelju ograničenja koja se mogu primjeniti na modeliranje i rješavanje problema. Ograničeno programiranje predstavlja najbržu metodu rješavanja, a u kombinaciji sa ostalim metodama rješavanja, riješava svaki Sudoku. [5]

### 2.1.3. Stohastičko pretraživanje / metode optimizacije

Rješenje Sudoku-a se može dobiti koristeći stohastičke algoritme. Algoritam je stohastičan ukoliko je barem jednom neka odluka u njegovom izvršavanju donesena slučajnim odabirom. Prilikom ovakvog rješavanja nasumični brojevi se postavljaju u prazne kvadratiće. Nakon toga se provjerava broj grešaka i zatim se brojevi mjenjaju sve dok je broj grešaka jednak nuli. [5]

## 2.2. Sudoku pravila

Cilj igre Sudoku je popuniti kvadrat veličine 9x9 s brojkama od jedan do devet, tako da se pojedini broj smije pojaviti točno devet puta. Jedan broj se ne smije pojaviti više od jednom u svakom redu, svakom stupcu, te svakom 3x3 podkvadratu. Početak igre otkriva nekoliko brojeva, a rješavač mora rasporediti i popuniti polja s ostalim brojevima. Ukoliko je na početku otkriveno malo brojeva, Sudoku može sadržavati više rješenja, što znači da je rješavanje lakše ukoliko je na početku otkriveno više brojeva. Rješeni Sudoku predstavlja primjer latinskog kvadrata. Glavne strategija za rješavanje Sudoku-a su metoda eliminacije polja u kojoj se pokušava pronaći jedino polje u nekom stupcu, redu ili 3x3 podkvadratu gdje se može postaviti neki broj i metoda eliminacije brojeva u kojoj se pokušava otkriti koji se broj nalazi u nekom polju. Primjer Sudoku-a i njegovo rješenje su prikazani na slikama 2.1. i 2.2. [6]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Sl. 2.1. Primjer Sudoku-a

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

Sl. 2.2. Rješenje Sudoku-a sa slike 2.1.

## 2.3. Vrste Sudoku-a

### 2.3.1. Mini Sudoku

Mini Sudoku je 6x6 kvadratna varijanta Sudoku-a s 3x2 podkvadratima koja se prvi puta pojavila u američkim novinama *USA Today*. Razlika u odnosu na standarni Sudoku je što se u mini Sudoku-u koriste brojevi od 1 do 6. [6]

### 2.3.2. Abecedni Sudoku

Pojavile su se i abecedne vrste Sudoku-a koje se ponekad nazivaju i *Wordoku*. Ne postoji neka funkcionalna razlika u ovoj vrsti Sudoku-a, osim ako kombinacija slova ne otkriva neku riječ. Neke varijante Abecednog Sudoku-a uključuju riječ koja se može pojaviti dijagonalno, u redu ili u stupcu kada je Sudoku riješen. Ukoliko se pokuša unaprijed odrediti riječ, ona može poslužiti kao pomoćno sredstvo za rješavanje. Primjer Abecednog Sudoku-a i njegovo rješenje su prikazani na slikama 2.3. i 2.4. [6]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | P | | K | | R | I | | D |
| D | | | B | | | | | R |
| | B | | E | | | P | A | |
| P | | | | K | W | A | | B |
| | | | | | | R | K | |
| | A | D | | | | | | |
| B | | | | E | | | | P |
| A | | | | | | E | | |
| E | R | | P | | K | B | | |

ABDEIKPRW

Sl. 2.3. Primjer Abecednog Sudoku-a

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| W | P | E | K | A | R | I | B | D |
| D | I | A | B | W | P | K | E | R |
| R | B | K | E | I | D | P | A | W |
| P | E | R | I | K | W | A | D | B |
| I | W | B | D | P | A | R | K | E |
| K | A | D | R | B | E | W | P | I |
| B | K | W | A | E | I | D | R | P |
| A | D | P | W | R | B | E | I | K |
| E | R | I | P | D | K | B | W | A |

ABDEIKPRW

Sl. 2.4. Rješenje Abecednog Sudoku-a sa slike 2.3.

### 2.3.3. Hipersudoku

Hipersudoku koristi 9x9 kvadrat s 3x3 podkvadratima, ali sadrži dodatna četiri 3x3 podkvadrata u kojima se brojevi od jedan do devet smiju pojaviti samo jednom. Peter Ritmeester, osnivač ove vrste Sudoku-a, objavio je prvu inačicu Hipersudoku-a 2005. godine u nizozemskim novinama *NRC Handelsbad.* Rješenje Hipersudoku-a sa slike 2.5. prikazano je na slici 2.6. [6]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | 1 | |
| | | 2 | | | | | 3 | 4 |
| | | | | 5 | 1 | | | |
| | | | | | 6 | 5 | | |
| | 7 | | 3 | | | | 8 | |
| | | 3 | | | | | | |
| | | | | 8 | | | | |
| 5 | 8 | | | | | 9 | | |
| 6 | 9 | | | | | | | |

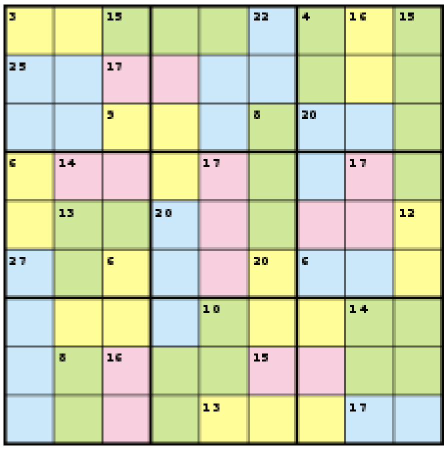
Sl. 2.5. Primjer Hipersudoku-a

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 4 | 6 | 8 | 3 | 2 | 7 | 1 | 5 |
| 1 | 5 | 2 | 6 | 9 | 7 | 8 | 3 | 4 |
| 7 | 3 | 8 | 4 | 5 | 1 | 2 | 9 | 6 |
| 8 | 1 | 9 | 7 | 2 | 6 | 5 | 4 | 3 |
| 4 | 7 | 5 | 3 | 1 | 9 | 6 | 8 | 2 |
| 2 | 6 | 3 | 5 | 4 | 8 | 1 | 7 | 9 |
| 3 | 2 | 7 | 9 | 8 | 5 | 4 | 6 | 1 |
| 5 | 8 | 4 | 1 | 6 | 3 | 9 | 2 | 7 |
| 6 | 9 | 1 | 2 | 7 | 4 | 3 | 5 | 8 |

Sl. 2.6. Rješenje Hipersudoku-a sa slike 2.5.

### 2.3.4. Killer Sudoku

Killer Sudoku je puzzle koji se sastoji od kombinacije Sudoku-a i Kakuro-a. Ovisno o rješavaču, Killer Sudoku može biti lakši od standardne verzije Sudoku-a, a također može zahtijevati sate rješavanja. Razlika Killer Sudoku-a i običnog Sudoku-a je što Killer Sudoku sadrži grupe obojenih kvadratića. Svaka grupa sadrži kvadratić u kojem se na vrhu kvadratića nalazi dodatni broj. Suma svih kvadratića od kojih se grupa sastoji mora biti jednaka dodatnom broju. Primjer Killer Sudoku-a kao i njegovo rješenje su prikazani slikama 2.7. i 2.8. [6]



Sl. 2.7. Primjer Killer Sudoku-a


Sl. 2.8. Rješenje Killer Sudoku-a sa slike 2.7.

### 2.3.5. Ostale vrste Sudoku-a

Mnoge ostale vrste Sudoku-a su razvijene. Neke varijante Sudoku-a razlikuju se samo u obliku rešetki između kvadratića poput leptira, vjetrenjače ili cvijeta. Ostali su razlikuju u logici rješavanja Sudoku-a. Jedna varijanta je „*Veći od Sudoku*“, u čijim se 3x3 podkvadratima nalaze 12 simbola „veći od (>)“ i „manji od (<)“ između dva broja. Sudoku svake godine dobiva sve više svojih varijanti, te zagonetka postaje sve više popularnija. [6]

# 3. PROGRAMSKA PODRŠKA

## 3.1. Stojni jezik

Strojni jezik je programski jezik kojeg koriste logički sklopovi. Koristi binarne riječi kao instrukcije pomoću kojih logički sklopovi rade i obrađuju podatke. [7]

Program se piše od strane programera pomoću mnemoničkih oznaka u neku datoteku, te koristi editor teksta kako bi to uspješno obavio. Programer zatim poziva program koji stvara binarne instrukcije strojnog jezika prevođenjem mnemoničkih oznaka. Asembler je program koji omogućuje pretvorbu, a asemblerski jezik je programski jezik koji asembler koristi. Program koji je nastao u asemblerskoj jeziku je izvorni program (engl. *Source code*). Izrada izvornog programa sastoji se od dva procesa. Prvi proces jest pisanje izvornog koda, a drugi proces je izrada izvršnog programa prevođenjem izvornog koda. Asemblerski jezik omogućuje upravljanje svim komponentama računala, te je stoga potrebno poznavanje arhitekture računala od strane programera i kontrola operacija koje računalo omogućava. Asemblersko programiranje koristi se uglavnom za procese vezane uz hardware računala. Asemblerski jezik i strojni jezik spadaju u niže programske jezike jer su usko vezani za hardware. Programski jezici na višim razinama koriste se za izradu većine programskih zadaća. [7]

Viši programski jezici razvijeni su kako bi programeru omogućili lakše programiranje. Tipovi viših programskih jezika su: *FORTRAN, Pascal, C++, Java, C.* Korištenje viših programskih jezika ne zahtijeva poznavanje komponenti svakog računala, te je s njima omogućeno prenošenje programa između bilo kojih računala različitih komponenti. Programiranje je značajno brže i jednostavnije korištenjem viših programskih jezika. Prije samog izvođenja, programi prolaze postupak prevođenja u izvršni kod. Prevoditelj (engl. *Compiler*) ispunjava taj zadatak. Glavna razlika prevoditelja i asemblera je što se asembler prilagođava načinu na kojemu rade logički sklopovi, dok su u prevoditelju naredbe i operacije koje programer koristi prilagođene programeru. Prevoditeljev zadatak je prevođenje programa napisanog u višem programskom jeziku u izvršni kod koji se izvršava na zadanom računalu. Viši programski jezici svoje početke dobivaju u 60-im godinama 20. stoljeća. [7]

## 3.2. C++ programski jezik

### 3.2.1. Povijest programskog jezika C++

C++ je objektno-orijentirani programski jezik kojeg je 1979. godine kreirao danski znanstvenik Bjarne Stroustrup. C++ je napravljen kao dodatak na C programski jezik ili popularno zvano „C s klasama“. Novi programski jezik stvorio je na svom radu na doktorskoj disertaciji u kojem su mu problem predstavljali Simula programski jezik koji je koristio za programske projekte koji su bili složeni i BCPL koji kao programski jezik nije bio prikladan za rad jer je bio na nižoj razini programskog jezika. Radeći za *AT&Tove Bell* laboratorije, Stroustrup je počeo analiziriati UNIX operacijski sustav. Svoj problem je riješio tako što je u programski jezik C dodavao produžetke programskoj jezika Simula, jer je bio brz i mogao se lako prenositi na druge platforme. Prvu inačicu ovakvog programskoj jezika nazvao je „C s klasama“. 1983. godine formalno mu daje naziv C++, gdje znak „++“ predstavlja kao povećanje C programskog jezika. U svojoj knjizi *The C++ Programming Language* predstavlja prvu komercijalnu distribuciju jezika krajem listopada 1985. godine. [8]

### 3.2.2. Vrste podataka i operatori

Varijable i konstante su oblici podataka s kojima program radi. Varijable je prvo potrebno definirati, a nakon toga i inicijalizirati. Deklaracijom varijabli govori programu da se koristi varijabla nekog imena. Inicijalizacija je postupak dodijeljivanja vrijednosti varijabli. Konstante su poseban tip varijabli. Konstanti se vrijednost može postaviti samo jednom (inicijalizirati). Konstante se koriste kada se želi zaštiti vrijednost varijable u programu od ostalih programera tako da ju oni ne mogu promijeniti u svom kodu. Riječ „*const*“ se postavlja ispred tipa podatka varijable koja se želi učiniti konstantom. Osnovni tipovi podataka prikazani su tablicom 3.1. [9]

| Tip podatka | Veličina podataka (u bitovima) | Područje vrijednosti |
|---|---|---|
| char | 8 | -128 do 127 |
| int | 16 | -32.768 do 32.767 |
| float | 32 | 3.4E-38 do 3.4E+38 |
| double | 64 | 1.7E-308 do 1.7E+308 |

Tab. 3.1. Osnovni tipovi podataka

C++ sadrži različite vrste operatora. Najvažnije vrste operator su aritmetički operatori i oni se dijele u dvije skupine: unarni i binarni. Tablica 3.2. prikazuje unarne operatore. [9]

| Operator | Objašnjenje |
|---|---|
| +X | Unarni plus |
| -X | Unarni minus |
| ++X | Uvećaj prije |
| --X | Umanji prije |
| X++ | Uvećaj nakon |
| X-- | Umanji nakon |

Tab. 3.2. Unarni operatori

Binarni operatori su poznatija vrsta operatora. Matematičke funckije koje su navedene u tablici ispod su poznate korištenjem s dvije varijable. Tablicom 3.3. prikazani su binarni operatori.

| Operator | Objašnjenje |
|---|---|
| X + Y | Zbrajanje |
| X – Y | Oduzimanje |
| X * Y | Množenje |
| X / Y | Dijeljenje |
| X % Y | Modulo |

Tab. 3.3. Binarni operatori

Modulo operacija je operacija koja kao rezultat vraća ostatak pri cjelobrojnom dijeljenju dva broja. [9]

Simbol operatora pridruživanja je „=“. Operator može uz sebe sadržavati izvedenice kako bi pisanje koda bilo olakšano. Operatori pridruživanja vidljivi su u tablici 3.4. [9]

| Operator | Objašnjenje |
|---|---|
| = | Pridruži vrijednost |
| += | Uvećaj za |
| -= | Umanji za |
| *= | Pomnoži sa |
| /= | Podijeli sa |
| %= | Dodaj vrijednost nakon izvršenja modulo operacije |

Tab. 3.4. Operatori pridruživanja

Operatori usporedbe omogućuju da se usporede vrijednosti dviju varijabli. Operatori usporedbe se uglavnom koriste kod grananja i petlji. Tablica 3.5. prikazuje operatore usporedbe. [9]

| Operator | Objašnjenje |
|---|---|
| > | Striktno veće |
| < | Striktno manje |
| >= | Veće ili jednako |
| <= | Manje ili jednako |
| == | Potpuno jednako |
| != | Potpuno različito |

Tab. 3.5. Operatori usporedbe

### 3.2.3. Kontrola toka i petlja

Kontrole toka omogućuju da se operacije izvršavaju ukoliko je neki uvjet zadovoljen. Kontrole toka korištene u programskoj jeziku C++ su: *if*, *if-else* i *switch*. Jedan od najčešće korištenih načina za kontrolu toka je *if*, koja izvršava dio koda tek kada se zadovolji neki uvjet. Kod 1. prikazuje opći oblik *if* kontrole toka. [9]

*if (uvjet) {*

*kod koji se izvršava ukoliko je uvjet zadovoljen*

*}*

Kod 1. Opći oblik *if* kontrole toka

Druga vrsta kontrole toka je *if-else* kontrola toka. Ova vrsta kontrole toka izvršava određen kod ukoliko if uvjeti nisu zadovljeni. Pisanje koda je značajno olakšano korištenjem *else* kontrole toka. Kod 2. prikazuje sintaksu *if-else* kontrole toka. [9]

*if (uvjet) {*

*kod koji se izvršava ukoliko je uvjet zadovoljen}*

*else {*

*kod koji se izvršava ukoliko if uvjet nije zadovoljen }*

Kod 2. Sintaksa *if-else* kontrole toka

*Switch* kontrola toka omogućava jednostruko grananje koje ovisi o vrijednosti postavljenog uvjeta. Uvjet predstavlja cjelobrojna varijabla ili cjelobrojni izraz. Provjerava se je li vrijednost uvjeta jednaka nekoj od niza zadanih cjelobrojnih konstanti, te ukoliko je, izvršava se kod koji je pridružen toj konstanti. *Break* naredba zaustavlja izvršavanje *switch* kontrole toka, te program izvršava naredbu koja je nakon *switch* bloka. Default naredba izvršava određeni kod ukoliko nijedan od uvjeta nije jednak ponuđenim konstantama. Kod 3. prikazuje sintaksu *switch* kontrole toka. [10]

*switch(uvjet) {*

*case (konstanta1): kod koji se izvršava ako je uvjet jednak konstanti1*

*break;*

*case (konstanta2): kod koji se izvršava ako je uvjet jednak konstanti2*

*break; default: ): kod koji se izvršava ako je uvjet nije jednak niti jednoj konstanti*

*}*

Kod 3. Sintaksa *switch* kontrole toka

Petlje omogućavaju da se linija nekog koda ili skup linija nekog koda izvršavaju određeni broj puta. Većinom se ne zna točno unaprijed koliko se puta izvršava nešto u petlji. Petlja omogućava izvršavanje nekog koda sve dok je neki uvjet zadovoljen. Petlje se mogu koristiti jedna unutar druge. Petlje koje se koristi u C++ programskom jeziku su *for*, *do-while* i *while* petlja. [9]

*For* petlja se koristi kada se određen kod ponavlja unaprijed poznati broj puta. *For* petlja sadrži kontrolnu varijablu kojoj se mijenja vrijednost svakim prolaskom kroz petlju. Prije uporabe *for* petlje, potrebno je deklarirati kontrolnu i zadati joj početnu vrijednost. Također, *for* petlja sadrži i uvjet čiji rezultat mora biti logički podatak. Petlja se izvršava sve dok vrijednost uvjeta logička istina. Ukoliko vrijednost postane neistina, petlja prekida sa radom. Zadnji podatak *for* petlje je prirast, koji predstavlja iznos za koliko se mijenja vrijednost kontrolne varijable. Kod 4. prikazuje opći oblik *for* petlje. [10]

*for (kontrolna_varijabla; uvjet; prirast)*

*{*

*kod koji se izvršava*

*}*

Kod 4. Opći oblik *for* petlje

U *while* petlji, određen kod se izvršava sve dok je uvjet petlje zadovoljen. Razlika između *for* i *while* petlje je što u *for* petlji znamo koliko će se puti određeni kod izvršiti, dok u *while* petlji ne znamo tu informaciju. Kod 5. prikazuje opći oblik *while* petlje. [9]

*while ( uvjet ) {*

*kod koji se izvršava dok je uvjet ispunjen }*

Kod 5. Opći oblik *while* petlje

Kod *do-while* petlje, prvo se navodi kod koji je potrebno izvršiti, a tek onda se navodi uvjet koji treba biti zadovoljen. *Do-while* petlja je jedina petlja koja će se sigurno jednom izvršiti jer će se prvo izvršiti kod petlje, a tek nakon toga se dolazi do uvjeta petlja. Kod 6. prikazuje opži oblik *do-while* petlje. [9]

*do {*

*kod koji se izvršava*

*} while ( uvjet );*

Kod 6. Opći oblik *do-while* petlje

#### 3.2.4. Polja i pokazivači

Polja (nizovi) predstavljaju skupove podataka koji su istoga tipa sklopljeni u jednu cjelinu. Polja omogućavaju da se neki skup podataka pohrani pod nekakvim zajedničkim imenom, a indeks označava redni broj nekog podatka u cjelokupnom skupu. Polja mogu biti jednodimenzionalna i višedimenzionalna. [10]

U jednodimenzionalnom polju podaci se nalaze jedan iza drugoga (u nizu su), a položaj (indeks) članova čine njihove udaljenosti od početnog člana. Prvi član u nizu sadrži indeks 0, dok je indeks posljednjeg člana za jedan manji od duljine polja. Kod 7. prikazuje deklaraciju polja.

*tip_podataka_polja ime_polja [veličina polja];*

Kod 7. Opći oblik deklaracije polja

Ukoliko se polju žele dodijeliti neke vrijednosti (inicijalizacija polja), vrijednosti se unose u vitičaste zagrade i odvajaju zarezom. Kod 8. prikazuje inicijalizaciju polja.

*double ime_polja [3] = {1.2, 2.3, 4.5};*

Kod 8. Inicijalizacija polja

Duljina polja se ne može mijenjati tijekom izvođenja programa, te ona mora biti jednaka ili veća broju članova polja. Ukoliko je duljina polja veća od broja njegovih članova, preostalim članovima se pridružuje 0. [10]

Pokazivači u C++ programskom jeziku predstavljaju varijable s kojima se pamte memorijske adrese. Pokazivači se uglavnom koriste za rad s poljima, omogućuju povezivanje objekata i klasa, rezerviraju i oslobađaju memoriju u vremenu izvođenja programa. Vrijednost adrese uzima se upotrebom operatora „*“, a adresa neke vrijednosti uzima se pomoću operatora &, koji bi u prijevodu značio „adresa od“. Kod 9. prikazuje definiciju pokazivača.

*double *pokazivac;*

Kod 9. Definiranje pokazivača

Definiran je pokazivač koji pokazuje na varijablu tipa double. Pokazivači u memoriji uvijek zauzimaju istu količinu memorije. Pokazivaču se dodjeljuje vrijednost (pokazuje na lokaciju neke varijable tipa double).

*double *pokazivac;*

*double n = 7.49;*

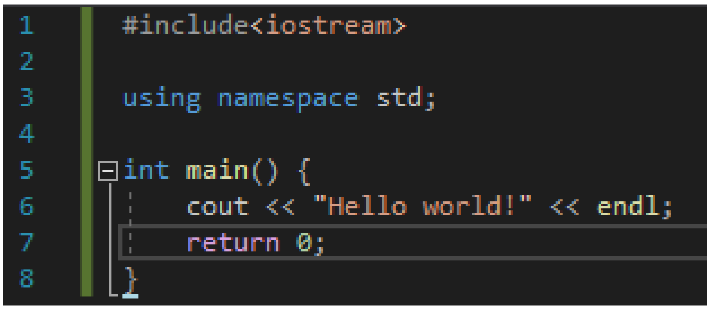
*pokazivac = &n;*

Kod 10. Definiranje pokazivača i dodjela vrijednosti

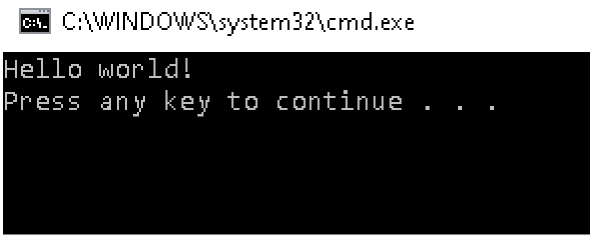
Pokazivač *pokazivac* je inicijaliziran, te on sada sadrži podatak koji predstavlja adresu varijable na koju pokazivač pokazuje. [11]

### 3.2.5. Primjer programa napisanog u C++ programskom jeziku

Primjer programa koji ispizuje niz znakova. Slikom 3.1. prikazan je kod primjera programa, a izlaz programa prikazan je slikom 3.2.



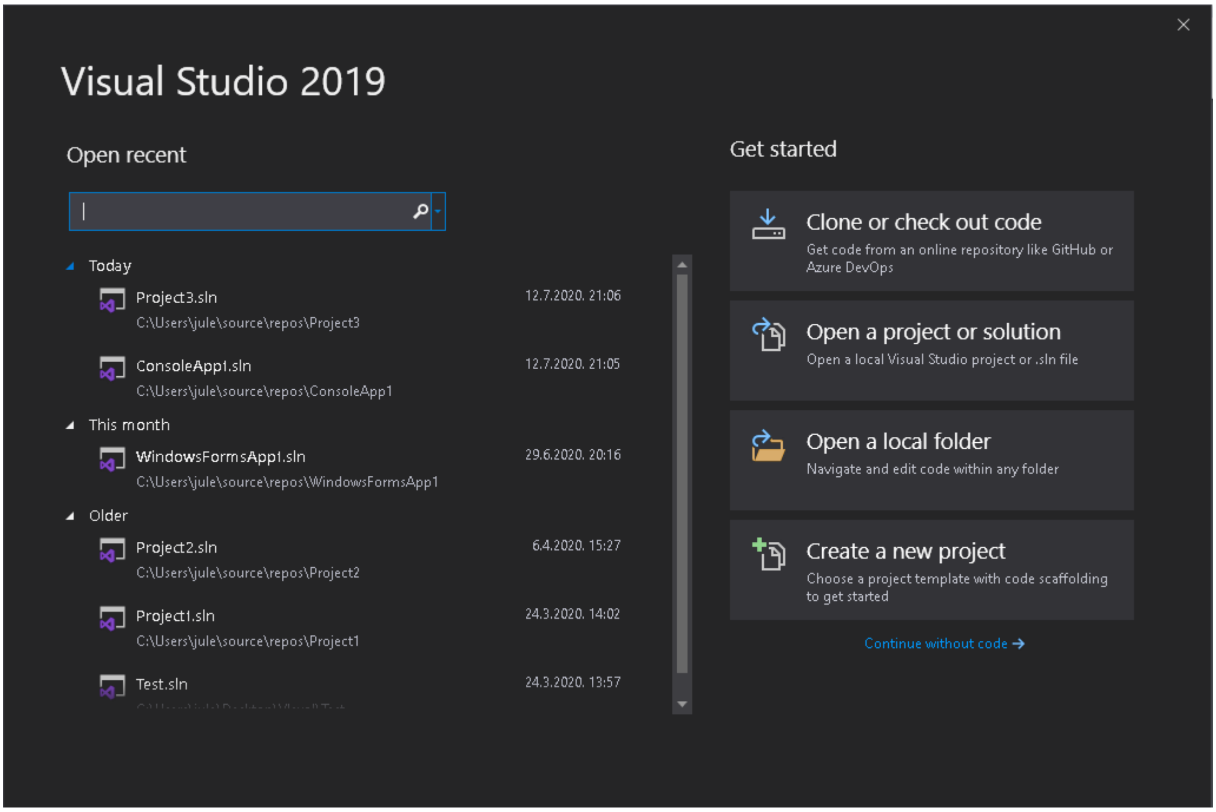

Sl. 3.1. Blok naredbi programa




Sl. 3.2. Izlaz programa

## 3.3. Microsoft Visual Studio

Microsoft Visual Studio predstavlja razvojno okruženje čiji je izdavač Microsoft. Glavna namjena Visual Studio je izrada mobilnih i web aplikacija, računalnih programa i web servisa. Visual Studio je kompatibilan sa različitim programskim jezicima. Jedini uvjet je da postoji usluga za određeni jezik. Jezici koji su ugrađeni u Visual Studio su C, C++, C#, VB.NET i F#. Ostalim programskim jezicima poput Pyhton-a i Ruby-a, podrška je dostupna instalacijom jezičkih servisa. Visual Studio podržava i HTML, CSS i JavaScript. Izgled razvojnog okruženja Microsoft Visual Studia nalazi se na slici 3.3. [12]




Sl. 3.3. Izgled razvojnog okruženja Microsoft Visual Studio

## 3.4. Heuristički algoritam

U području računarstva, pojam heurističan se odnosi na vještinu konstruiranu za rješavanje problema puno brže kada su klasične metode spore ili za pronalazak približnog rješenja kada klasične metode ne mogu pronaći točno rješenje. Heuristička funkcija je funkcija koja rangira alternative u algoritmima pretraživanja na svakom koraku grananja, te na temelju tih dostupnih informacija odlučuje koju će granu slijediti. Primjerice, može približiti točno rješenje. Cilj heuristike je pronaći rješenje u vremenskom okviru koji je dovoljno dobar za rješavanje problema. Kriteriji za odlučivanje hoće li se heuristika koristiti za rješavanje zadanog problema su: optimalnost, potpunost, točno i preciznost, te vrijeme izvršavanja. [13]

# 4. PROGRAMSKO RJEŠENJE

## 4.1. Uvod u rješenje

Cilj ovog algoritma je popuniti dani Sudoku određenim brojevima kako bi se zadovoljili svi kriteriji za rješavanje Sudoku-a. Prolazeći kroz Sudoku pretragom u dubinu (engl. *Depth-first search)*, algoritam pronalazi prazni kvadratić s najmanjim brojem mogućih rješenja kvadratića (kandidata). Kandidat predstavlja potencijalni broj koji može biti rješenje kvadratića sukladno Sudoku pravilima. U kvadratić se postavlja vrijednost prvog kandidata, te se zatim rekurzivno pronalazi sljedeći prazni kvadratić s najmanjim brojem kandidata i popunjava. Ukoliko algoritam dođe do praznog kvadratića koji nema kandidata za njegovo rješavanje, vraća se na prethodni riješeni kvadratić i postavlja mu vrijednost njegovog drugog kandidata. Razlog zbog kojeg se popunjavaju prazni kvadratići s najmanjim brojem kandidata je ukoliko dođe do nemogućnosti popunjavanja kvadratića (izostanak kandidata praznog kvadratića zbog popunjava prethodnih kvadratića), to će biti rana faza rješavanja Sudoku zagonetke i algoritam će pokušati pronaći novo rješenje. Ovakvom strategijom omogućeno je smanjenje koraka u potrazi za točnim vrijednostima Sudoku-a, a time će i vrijeme izvršavanja programa biti kraće.

## 4.2. Implementacija rješenja

### 4.2.1. Funkcija za dohvaćanje kandidata

Funckija *dohvatiKandidate* je funkcija koja vraća vektor čiji su elementi cjelobrojni brojevi. Slika 4.1. prikazuje funkciju *dohvatiKandidate*. Vektor predstavlja niz podataka jednakog tipa. Funkcija kao argumente prima zadani Sudoku, te red i stupac kvadratića za koji se traže moguća rješenja (kandidati). Prvom *for* petljom deklarira se i inicijalizira varijabla *broj* čija se vrijednost inkrementira od broja jedan do devet, odnosno brojevima koji su dozvoljeni prema Sudoku pravilima. Kako pravila Sudoku-a naglašavaju da se jedan broj ne smije ponoviti u istom redu, stupcu i 3x3 podkvadratu, pomoću druge *for* petlje, prolazeći kroz red, stupac i 3x3 podkvadrat danim u argumentima funkcije, provjerava se postojanje broja jednake vrijednosti kao i varijabla *broj*. Bool varijabla *moguceRjesenje* predstavlja „zastavu" koja signalizira postojanje takvog broja. Ukoliko broj postoji, varijabli se postavlja vrijednost na *true* i prekida se daljnje izvođenje petlje. Ukoliko broj ne postoji, *moguceRjesenje* zadržava početnu vrijednost *false*. Naredbom *if* provjerava se vrijednost varijable *moguceRjesenje*, te ukoliko je uvjet istinit vrijednost varijable *broj* dodaje se u niz *int* podataka *ukupnoKandidata*, u suprotnom vrijednost se ne dodaje. Nakon toga vrijednost varijable *broj* se inkrementira, te se ponovno prolazi kroz red, stupac i 3x3 podkvadrat i ponovno provjerava postojanja broja jednakog promijenjenoj vrijednosti varijable *broj*, te se postupak se ponavlja dok svi broji od jedan do devet nisu provjereni kao mogući kandidati. Funkcija vraća niz *int* podataka koji predstavljaju moguća rješenja za kvadratić čiji je položaj određen argumentima funkcije.

```
vector<int> dohvatiKandidate(int Sudoku[velicina][velicina], int red, int stupac){

    vector<int> ukupnoKandidata;

    for (int broj = 1; broj <= 9; broj++)
    {

            bool moguceRjesenje = false;

            for (int i = 0; i < 9; i++)
            {
                if (Sudoku[red][i] == broj ||
                    Sudoku[i][stupac] == broj ||
                    Sudoku[(red - red % 3) + i / 3][(stupac - stupac % 3) + i % 3] == broj)
                {
                    moguceRjesenje = true;
                    break;
                }
            }

            if (!moguceRjesenje)
                ukupnoKandidata.push_back(broj);

    }

    return ukupnoKandidata;

}
```

Sl. 4.1. Funkcija koja vraća niz *int* podataka koji predstavljaju moguća rješenja

### 4.2.2. Funkcija za rješavanje Sudoku-a

Funkcija *rijesiSudoku* kao parametar prima zadani Sudoku te vraća bool vrijednost. Deklarirane su *int* varijable *red* i *stupac* kao pomoćne varijable koje će sadržavati položaj praznog kvadratića s najmanjim brojem kandidata. Bool varijabla *zastava* se koristi kao vrijednost koju će funkcija vraćati ovisno o rješenosti Sudoku-a. Vektor cjelobrojnih brojeva *kandidati* će sadržavati brojeve koji su moguća rješenja za određeni prazni kvadratić. Prvi dio funkcije *rijesiSudoku*, pomoću *for* petlji prolazi kroz Sudoku i traži prvi prazni kvadratić. Nakon što je pronađen, kreira se vektor *noviKandidati* u koji se, pozivom funkcije *dohvatiKandidate*, spremaju kandidati (moguća rješenja) kvadratića. Uvjet *„red < 0“* omogućuje da varijable *red, stupac* i *kandidati* sadrže položaj prvog praznog kvadratića, odnosno njegove kandidate. Zatim, ponovnim prolaskom kroz petlje se traži sljedeći prazni kvadratić. Ponovno se kreira vektor *noviKandidati* u koji se, pozivom funckije *dohvatiKandidate*, spremaju kandidati tog praznog kvadratića. Nakon toga se uvjetom *„noviKandidati.size() < kandidati.size()“* provjerava je li veličina vektora *noviKandidati* manja od veličine vektora *kandidati*. Ukoliko je manja, varijablama *red, stupac* i *kandidati* se vrijednosti mjenjaju u vrijednosti drugog praznog kvadratića. Ponovnim prolaskom kroz petlje postupak se ponavlja, te na kraju varijable *red* i *stupac* imaju vrijednosti položaja praznog kvadratića u Sudoku-u sa najmanjim brojem kandidata, a vektor *kandidati* sadrži vrijednosti tih kandidata.

```
bool rijesiSudoku(int Sudoku[velicina][velicina])
{
    int red = -1;
    int stupac = -1;
    vector<int> kandidati;

    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
            if (Sudoku[i][j] == 0)
            {
                vector<int> noviKandidati = dohvatiKandidate(Sudoku, i, j);
                if (red < 0 || noviKandidati.size() < kandidati.size())

                {
                    red = i;
                    stupac = j;
                    kandidati = noviKandidati;
                }
            }

    if (red < 0)
    {
        return true;
    }
    else

        for (int i = 0; i < kandidati.size(); i++)
        {
            Sudoku[red][stupac] = kandidati[i];

            if (rijesiSudoku(Sudoku))
            {
                return true;

            }
            Sudoku[red][stupac] = 0;

        }

    return false;

}
```

Sl. 4.2. Funkcija *rijesiSudoku*

U *for* petlji, kvadratiću, s najmanjim brojem kandidata, se pridružuje vrijednost prvog njegovog kandidata. Zatim se ponovno poziva funkcija *rijesiSudoku*, te se Sudoku pokušava rekurzivno popuniti. Ukoliko se uspješno prođe kroz sve rekurzije, funkcija vraća *true* vrijednost i Sudoku je uspješno riješen. Ukoliko prolazak kroz rekurzije nije uspješan, netočan broj je postavljen u nekom od kvadratića. Vrijednost trenutnog kvadratića postavlja se na nulu, te mu se zatim postavlja vrijednost njegovog drugog kandidata. Ukoliko su testirani svi kandidati, te rješenje i dalje nije pronađeno, netočan broj je postavljen u nekom od prethodnih riješenih kvadratića. Vraća se vrijednost *false*, te se funkcija vraća na prethodni riješeni kvadratić i postavlja vrijednost drugog njegovog kandidata. Pomoću *if* naredbe i njezinog uvjeta *„red < 0“* , provjerava se sadrži li Sudoku prazne kvadratiće, te ukoliko sadrži, traže se točne vrijednosti, u suprotnom funkcija vraća *true* vrijednost. Slikom 4.2. prikazana je funkcija *rijesiSudoku*.

### 4.2.3. Testiranje algoritma

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | | 8 | | | | | | 4 |
| 1 | 3 | | | | 2 | | | |
| 2 | | 4 | | 6 | 9 | | | |
| 8 | 7 | | | | | 4 | | |
| | 6 | | | | | | 3 | |
| | | 2 | | | | | 8 | 7 |
| | | | 9 | 5 | | 2 | | 3 |
| | | | 4 | | | | 9 | 5 |
| 9 | | | | | | 1 | | 8 |

Sl. 4.4. Primjer Sudoku zagonetke

Slika 4.4. prikazuje primjer klasične 9x9 Sudoku zagonetke koja se koristi kao test za provjeru ispravnosti algoritma. Zagonetka je primjer Sudoku-a koji pripada razini težine „teško“.

```
#include<iostream>
#include<vector>

using namespace std;

#define velicina 9
```

Sl. 4.5. Biblioteke korištene pri kreiranju algoritma

Slika 4.5. prikazuje biblioteke koje se koriste pri kreiranju algoritma, te se pomoću naredbe *„define velicina 9“* definira konstanta naziva *velicina* s vrijednošću 9.

```cpp
int main()
{

    int Sudoku[velicina][velicina] = {
                     { 7, 0, 8, 0, 0, 0, 0, 0, 4 },
                     { 1, 3, 0, 0, 0, 2, 0, 0, 0 },
                     { 2, 0, 4, 0, 6, 9, 0, 0, 0 },
                     { 8, 7, 0, 0, 0, 0, 4, 0, 0 },
                     { 0, 6, 0, 0, 0, 0, 0, 3, 0 },
                     { 0, 0, 2, 0, 0, 0, 0, 8, 7 },
                     { 0, 0, 0, 9, 5, 0, 2, 0, 3 },
                     { 0, 0, 0, 4, 0, 0, 0, 9, 5 },
                     { 9, 0, 0, 0, 0, 0, 1, 0, 8 } };

    ispisiSudoku(Sudoku);

    cout << "---------------------------" << endl;


    if (rijesiSudoku(Sudoku))
        ispisiSudoku(Sudoku);
    else
        cout << "Nema rješenja.";

    return 0;
}
```

Sl. 4.6. *main* funkcija

U *main* funkciji (prikazana slikom 4.6.) deklarirano je polje cjelobrojnih brojeva pod nazivom *Sudoku.* Polju su dodijeljenje vrijednosti zagonetke sa slike 4.4. gdje nule predstavljaju prazan kvadratić. Pozvana je funkcija *ispisiSudoku* koja ispisuje početni Sudoku, odnosno Sudoku koji još nije riješen. Nakon toga se poziva funkcija *rijesiSudoku*, te ukoliko se vrati vrijednost *true*, ponovno se poziva funkcija *ispisiSudoku* koja ispisuje vrijednosti riješenog Sudoku-a. Ukoliko funkcija *rijesiSudoku* vraća vrijednost *false*, ispisuje se poruka *„Nema rješenja.“*.

```
void ispisiSudoku(int Sudoku[velicina][velicina])
{
    for (int red = 0; red < velicina; red++) {
        for (int stupac = 0; stupac < velicina; stupac++) {
            cout << Sudoku[red][stupac] << " ";
            if ((stupac+1) % 3 == 0) {
                cout << "  ";
            }
        }
        if ((red + 1) % 3 == 0) {
            cout <<  endl;
        }
        cout << endl;
    }
}
```

Sl. 4.7. Funkcija za ispis Sudoku-a

Funkcija *ispisiSudoku* je vrsta *void* funkcije koja kao paramater prima polje cjelobrojnih brojeva koje predstavlja Sudoku zagonetku. Pomoć *for* petlji, funkcija ispisuje vrijednosti Sudoku-a. Ukoliko je uvjet *if* naredbe *„(stupac+1) % 3 == 0“* vrijednosti *true*, funkcija će nakon svakog trećeg stupca dodati još jedan razmak između vrijednosti zbog bolje preglednosti. Također, ukoliko je uvjet *if* naredbe *„(red + 1) % 3 == 0“* vrijednosti *true*, funkcija će omogućiti prijelaz u novi red zbog bolje preglednosti.

```
Microsoft Visual Studio Debug Console
7 0 8   0 0 0   0 0 4
1 3 0   0 0 2   0 0 0
2 0 4   0 6 9   0 0 0

8 7 0   0 0 0   4 0 0
0 6 0   0 0 0   0 3 0
0 0 2   0 0 0   0 8 7

0 0 0   9 5 0   2 0 3
0 0 0   4 0 0   0 9 5
9 0 0   0 0 0   1 0 8

--------------------------
7 9 8   5 1 3   6 2 4
1 3 6   7 4 2   8 5 9
2 5 4   8 6 9   3 7 1

8 7 3   2 9 5   4 1 6
4 6 9   1 7 8   5 3 2
5 1 2   6 3 4   9 8 7

6 8 7   9 5 1   2 4 3
3 2 1   4 8 6   7 9 5
9 4 5   3 2 7   1 6 8


C:\Users\jule\source\repos\Project1\Debug\Project1.exe (process 11508) exited with code 0.
```

Sl. 4.8. Sudoku i njegovo rješenje

Slika 4.8. prikazuje sadržaj koji se ispisuje na ekran nakon izvođenja *main* funckije. Prvo je prikazan početni Sudoku kojemu se pokušava pronaći rješenje, te se nakon toga prikazuje i riješeni Sudoku s točnim vrijednostima.

# 5. ZAKLJUČAK

Zadatak je bio osmisliti algoritam za rješavanje Sudoku puzzle. Algoritam pomoću funkcije *dohvatiKandidate*, sprema u niz *int* podataka (vektor), sve moguće brojeve koje je moguće postaviti u prazni kvadratić tako da se ne krše osnovna Sudoku pravila. Funkcija *rijesiSudoku* pronalazi prazni kvadratić u predanom Sudoku-u s najmanjim brojem kandidata. Kvadratiću se postavlja vrijednost prvog kandidata, te se pomoću rekurzije traži vrijednost za idući prazni kvadratić s najmanjim brojem kandidata. Ukoliko su rekurzije uspješne, pomoću njih se popunjavaju prazni kvadratići Sudoku-a, u suprotnom se algoritam vraća na prethodni riješeni kvadratić, te odabire njegovog drugog kandidata. Nakon toga se opet koristi rekurzija kako bi se pronašle točne vrijednosti preostalih praznih kvadratića.

Bjarne Stroustrup tvorac je programskog jezika C++. Priložena su sva znanja i funkcinalnosti programskog jezika C++ korištena pri izradi heurističkog algoritma. Najbitnije svojstvo programskog jezika C++ pri izradi algoritma je rekurzija.

Stvoreni algoritam je unatražne prirode, te je time omogućeno sigurno rješenje zagonetke Sudoku. Algoritam se kreće kroz kvadratiće zagonetke primjenjujući pravila za smještanje određenih brojeva u određene kvadratiće.

Algoritmom je omogućeno rješavanje 9x9 Sudoku zagonetke za manje od 20 milisekundi, te je tako cilj završnog rada uspješno ostvaren.

# SAŽETAK

**Naslov: Brzi heuristički algoritam za rješavanje logičke zagonetke Sudoku u programskom jeziku C++**

Tema završnog rada bila je napraviti algoritam za rješavanje Sudoku-a u programskom jeziku C++. Programsko okruženje korišteno pri izradi algoritma je Microsoft Visual Studio. Algoritam je realiziran pomoću dvije funkcije koje su međusobno povezane. Glavna odlika algoritma je njegovo brzo izvođenje. Tehnika *backtracking-a* najkorištenija je metoda za rješavanje zagonetke. Ulaz programa predstavlja Sudoku čije podatke korisnik unosi, a na izlazu programa prikazano je rješenje Sudoku zagonetke. Algoritam je osmišljen da riješava dimenzije svih Sudoku zagonetki po svim razinama težine: *easy, medium* i *hard.* Završni rad sadrži detaljan opis algoritma i njegove same strukture. Rad sadrži nekoliko poglavlja s detaljnim opisom određenih segmenata radi lakšeg shvaćanja i obrazloženja samog algoritma. Kako bi algoritam za rješavanje Sudoku-a imao smisla, detaljno je opisana igra Sudoku, kao i sve njene vrste. Sudoku s vremenom sve više dobiva na popularnosti, pa s toga postoji i sve više algoritama za njegovo rješavanje.

**Ključne riječi: C++, polje, Sudoku, heuristika, rekurzija**

# ABSTRACT

**Title: Fast heuristic algorithm for solving logic puzzle Sudoku in C++ programming language**

The topic of the final paper is to create an algorithm for solving Sudoku in the C++ programming language. Programming environment used for creating the algorithm is Microsoft Visual Studio. The algorithm is realized using two functions that are interconnected. The main feature of the algorithm is its fast execution. The *backtracking* technique is the most used method for solving the puzzle. Algorithm is designed for solving all dimensions of the puzzle at all difficulty levels: *easy, medium* and *hard.* The final paper contains a detailed description of the algorithm and its structure. The paper contains several chapters with a detailed description of certain segments for easier understanding and explanation of the algorithm itself. In order to make se for the algorithm to solve Sudoku, the Sudoku game is described in detail, as well as all its types. As Sudoku is getting more and more popular, there are also and more algorithms to solve it.

**Keywords: C++, array, Sudoku, heuristics, recursion**

# ŽIVOTOPIS

Josip Juhasz rođen je 31.07.1998. godine u Đakovu. Živi u Selcima Đakovačkim na adresi Bana Josipa Jelačića 97a. Osnovnoškolsko obrazovanje započinje 2005. godine u OŠ Đakovački Selci u Selcima Đakovačkim. Nakon osnovnoškolskog obrazovanja, 2013. godine upisuje Gimnaziju Antuna Gustava Matoša u Đakovu. Nakon srednjoškolskog obrazovanja upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, sveučilišni studij, smjer Računarstvo. Posjeduje znanje u govoru, čitanju i pisanju engleskog jezika, te vozačku dozvolu B kategorije. Osim navedenog posjeduje i osnovno znanje programskih alata, odnosno tehnologija: HTML, CSS, C, C++, C# i JAVA te Microsoft Office alatima.

# LITERATURA


[1] Nezavisne novine: Sudoku

https://www.nezavisne.com/zabava/sudoku [16.9 2020.]

[2] K. Y. Lin: Number of Sudokus, Journal of Recreational Mathematics, 33(2): 120-24, 2004

[3] D.Bethier, The Hidden Logic of Sudoku, Lulu.com, 2017

[4] T.Yato, T.Seta: Complexity and Completeness of Finding Another Solution and Its Application to Puzzles

http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/SIGAL87-2.pdf [18.9.2020]

[5] Wikipedia, Sudoku solving algorithms

https://en.wikipedia.org/wiki/Sudoku_solving_algorithms [13.7. 2020.]

[6] Wikipedia: Sudoku,

https://en.wikipedia.org/wiki/Sudoku [13.7. 2020.]

[7] Wikipedia, Programming language

https://en.wikipedia.org/wiki/Programming_language [13.7. 2020.]

[8] Wikipedia, C++,

https://en.wikipedia.org/wiki/C%2B%2B [13.srpnja 2020.]

[9] Udruga Carpe Diem, Publikacije Carpe Diem, C++ programiranje,

http://carpediem.hr/PublikacijeCarpeDiem/Publikacije/C++%20programiranje.pdf

[13.srpnja 2020.]

[10] Sanda Šutalo, Osnove programiranje u jeziku C++,

https://sites.google.com/site/sandasutalo/osnove-programiranja [13.7. 2020.]

[11] Tehnička škola Čakovec, Nastavni materijali, Pokazivači

http://www.ss-tehnicka-ck.skole.hr/wp-content/uploads/2016/10/Pok_1.pdf [13.7.2020.]

[12] Wikipedia, Microsoft Visual Studio,

https://bs.wikipedia.org/wiki/Microsoft_Visual_Studio [13.7. 2020.]

[13] Wikipedia, Heuristic (computer science)

https://en.wikipedia.org/wiki/Heuristic_(computer_science) [13.7. 2020.]