Android aplikacija za računanje konjunktivne i disjunktivne normalne forme

Tomić, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

Android aplikacija za rješavanje konjunktivne i disjunktivne normalne forme

Diplomski rad

Matej Tomić

Osijek, 2018.godina

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 13.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Matej Tomić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 885 R, 26.09.2017.
OIB studenta:	82971960861
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Tomislav Keser
Član Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Android aplikacija za računanje konjunktivne i disjunktivn normalne forme
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Student će izraditi aplikaciju za računanje konjunktivne i disjunktivne normalne forme za android platformu. Sumentor: Alfonzo Baumgartner
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 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:	13.09.2018.

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, 01.10.2018.

Ime i prezime studenta:	Matej Tomić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 885 R, 26.09.2017.
Ephorus podudaranje [%]:	15%

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za računanje konjunktivne i disjunktivn normalne forme**

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:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

Ja, Matej Tomić, OIB: 82971960861, student/ica na studiju: Diplomski sveučilišni studij Računarstvo, dajem suglasnost Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek da pohrani i javno objavi moj **diplomski rad**:

Android aplikacija za računanje konjunktivne i disjunktivn normalne forme

u javno dostupnom fakultetskom, sveučilišnom i nacionalnom repozitoriju.

Osijek, 01.10.2018.

SADRŽAJ

1. UVOD

Konstantnim razvojem i napretkom tehnologije život postaje lakši i jednostavniji. Tako je u današnje vrijeme teško zamisliti život bez pametnih telefona koji su postali jedan od osnovnih alata za obavljanje svakodnevnih poslova i obaveza. Koriste se za komunikaciju s drugim osobama, pretreživanje interneta, korištenje društvenih mreža, plaćanje računa i još mnogo toga. Upravo zbog toga se razvijaju aplikacije, i softveri općenito, za razna područja poput matematike, bankarstva, medicine i drugih. Cilj ovog rada je spojiti programiranje i matematičku logiku te napraviti mobilnu aplikaciju za računanje konjunktivne normalne forme (KNF) i disjunktivne normalne forme (DNF). Potrebno je generirati tablicu istinitosti za zadni izraz i na temelju nje izračunati KNF ili DNF. Krajnji proizvod će omogućiti korisniku unos matematičkog izraza i na temelju njega izračun tablice istinitosti te konjunktivne i disjunktivne normalne forme. Glavni dio rada podijeljen je na tri dijela: teorijski dio zadatka, razvojno okruženje i tehnologije i programsko rješenje zadatka. U teorijskom dijelu zadatka analiziran je pojam matematičke logike i objašnjene su sve logičke operacije sa sudovima. Također, definirani su pojmovi konjunktivna i disjunktivna normalna forma. U drugom dijelu opisane su tehnologije i razvojno okruženje koje će se koristiti prilikom izrade zadatka. U zadnjem dijelu prikazano je i opisano programsko rješenje zadatka. Zadatak završnog rada je napraviti aplikaciju za računanje konjunktivne i disjunktivne normalne forme za android platformu.

2. TEORIJSKI DIO ZADATKA

2.1.Matematička logika

Kako bi se moglo govoriti o matematičkoj logici potrebno je prvo reći nešto o pojmu „logika“. Riječ logika proizlazi iz grčke riječi *logos* što znači govor, riječ, um i misao. Na temelju toga logika se može definirati kao grana filozofije koja se bavi ispravnim oblicima mišljena i zaključivanja. Nastala je u staroj Grčkoj, a za osnivača se smatra grčki filozof Aristotel. U novije vrijeme sve je veća povezanost logike i matematike pa je tako i nastala matematička logika.

Matematička logika je grana matematike i logike koja se bavi primjenom formalne logike u matematici.

Početni cilj matematičke logike je bio istražiti pravilna logička zaključivanja, a nakon toga je osnovni cilj postalo ispitivanje osnova matematike. Danas matematička logika predstavljava i teorijsku osnovu računarstva, prema[1].

2.2.Logika sudova i logičke operacije sa sudovima

Logika sudova jedna je od najjednostavnijih formalnih teorija i ona se bavi odnosom među rečenicama (sudovima). Sud je rečenica kojoj je moguće odrediti je li ona istinita ili lažna. Alfabet logike sudova je unija skupova S_1, S_2 i S_3, pri čemu je:

- S_1 = { A, B, C, ... } prebrojiv skup čije elemente nazivamo *propozicionalne varijable*
- S_2 = { ¬, ∧, ∨, →, ↔ } skup *logičkih veznika*
- S_3 = { () , } skup *pomoćnih simbola,* prema[2].

Logički veznici se nazivaju negacija, konjukcija, disjunkcija, kondicional i bikondicional. Prilikom određivanja istinitosti nekog suda postoje prioriteti kod logičkih veznika. Najveći prioritet ima negacija (¬), a nakon nje slijede konjukcija (∧) i disjunkcija (∨) koji imaju jednak prioritet te kondicional (→) i bikondicional (↔) koji također imaju jednak prioritet. Formula logike sudova je svaki niz znakova varijabli, operacija i konstanti (0 ili 1) algebre sudova, a vrijednosti interpretacije na formulama se prikazuju pomoću semantičkih tablica ili tablica istinitosti.

Negacija je operacija nad logičkim vrijednostima koja preslikava istinu u laž i obrnuto. Logički veznik ¬ jednak je značenju *ne* u hrvatskom jeziku. Semantička tablica negacija prikazana je tablicom 2.1.

Tab.2.1 Semantička tablica za negaciju

A	**¬A**
0	1
1	0

Objašnjenje tablice za negaciju pomoću primjera:

- Neka je A sud: „Ivan igra nogomet.“
- Tada je ¬A : „Ivan **ne** igra nogomet.“

Konjukcija je operacija nad logičkim vrijednostima čiji je rezultat istinit samo ako su obje vrijednosti istinite. U suprotnom je rezultat neistinit. Logički veznik ∧ jednak je značenju veznika *i, a, ali, nego, već* u hrvatskom jeziku. Semantička tablica konjukcije prikazana je tablicom 2.2.

Tab.2.2 Semantička tablica za konjukciju

A	**B**	**A ∧ B**
0	0	0
0	1	0
1	0	0
1	1	1

Objašnjenje tablice za konjukciju pomoću primjera:

- Neka je A sud: „Matej svira gitaru.“
- Neka je B sud: „Ivan igra nogomet.“
- Tada je A ∧ B: „Matej svira gitaru **i** Ivan igra nogomet.“

Disjunkcija je operacija nad logičkim vrijednostima čiji je rezultat istinit kada je jedna ili više vrijednosti istinito. Ukoliko niti jedna od vrijednost nije istinita rezultat je neistinit. Logički veznik ∨ jednak je značenju veznika *ili* u hrvatskom jeziku. Semantička tablica konjukcije prikazana je tablicom 2.3.

Tab.2.3 Semantička tablica za disjunkciju

A	**B**	**A ∨ B**
0	0	0
0	1	1
1	0	1
1	1	1

Objašnjenje tablice za disjunkciju pomoću primjera:

- Neka je A sud: „Matej svira gitaru.“
- Neka je B sud: „Ivan igra nogomet.“
- Tada je A ∨ B: „Matej svira gitaru **ili** Ivan igra nogomet.“

Kondicional je operacija nad logičkim vrijednostima čiji je rezultat neistinit samo ako je druga vrijednost neistinita, a prva istinita. Logički veznik → jednak je *ako...onda* skupu riječi u hrvatskom jeziku. Semantička tablica kondicionala prikazana je tablicom 2.4.

Tab.2.4 Semantička tablica za kondicional

A	**B**	**A → B**
0	0	1
0	1	1
1	0	0
1	1	1

Objašnjenje tablice za kondicional pomoću primjera:

- Neka je A sud: „Matej svira gitaru.“
- Neka je B sud: „Ivan igra nogomet.“
- Tada je A → B: „**Ako** Matej svira gitaru, **onda** Ivan igra nogomet.“

Bikondicional je operacija nad logičkim vrijednostima čiji je rezultat istinit samo ako su obje vrijednosti ili istinite ili neistinite. Logički veznik ↔ jednak je skupu riječi *ako i samo ako* u hrvatskom jeziku. Semantička tablica bikondicionala prikazana je tablicom 2.5.

Tab.2.5 Semantička tablica za bikondicional

A	**B**	**A ↔ B**
0	0	1
0	1	0
1	0	0
1	1	1

Objašnjenje tablice za bikondicional pomoću primjera:

- Neka je A sud: „Matej svira gitaru.“
- Neka je B sud: „Ivan igra nogomet.“
- Tada je A ↔ B: „Matej svira gitaru, **ako i samo ako** Ivan igra nogomet.“

Također, na temelju vrijednosti interpretacija formule (suda) možemo odrediti je li ona ispunjiva, oboriva, tautologija i anitautologija.

Formula (sud) F je ispunjiva ako postoji barem jedna interpretacija I takva da vrijedi I(F) = 1. Formula (sud) F je oboriva ako postoji barem jedna interpretacija I takva da vrijedi I(F) = 0. Formula je tautologija ako je istinita za svaku interpretaciju, a antitautologija ako je neistinita za svaku interpretaciju.

Objašnjenje ispunjivosti, oborivosti, tautologije i antitautologije pomoću primjera:

- Neka je formula $F \equiv (A \wedge B) \leftrightarrow (\neg B \rightarrow \neg C)$
- Tada je semantička tablica za formulu prikazana u tablici 2.6:

Tab.2.6 Semantička tablica formule F

A	**B**	**C**	**(A ∧ B)**	**¬B**	**¬C**	**(¬B → ¬C)**	**F**
0	0	0	0	1	1	1	0
0	0	1	0	1	0	0	1
0	1	0	0	0	1	1	0
0	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
1	0	1	0	1	0	0	1
1	1	0	1	0	1	1	1
1	1	1	1	0	0	1	1

- Kako I(F) = 1 ne vrijedi za svaku interpretaciju kombinacija A, B, C tada F nije tautologija
- Kako I(F) = 0 ne vrijedi za svaku interpretaciju kombinacija A, B, C tada F nije antitautologija
- Kako za barem jednu interpretaciju kombinacija A, B, C vrijedi I(F) = 1 tada je F ispunjiva
- Kako za barem jednu interpretaciju kombinacija A, B, C vrijedi I(F) = 0 tada je F oboriva

Također, u algebri sudova vrijede i razni zakoni koji su prikazani u tablici 2.7.

Tab.2.7 Zakoni algebre sudova

KOMUTATIVNOST	$A \vee B \equiv B \vee A$ $A \wedge B \equiv B \wedge A$
DISTRIBUTIVNOST	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
ASOCIJATIVNOST	$(A \vee B) \vee C \equiv A \vee (B \vee C)$ $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
KOMPLEMENTARNOST	$A \vee \neg A \equiv 1$ $A \wedge \neg A \equiv 0$ $\neg(\neg A) \equiv A$
DE MORGANOVI ZAKONI	$\neg (A \wedge B) \equiv \neg A \vee \neg B$ $\neg (A \vee B) \equiv \neg A \wedge \neg B$

2.3.Konjunktivna normalna forma (KNF)

Konjunktivna normalna forma neke formule algebre sudova je konjunkcija svih njezinih elementarnih disjunkcija, prema[3].

Objašnjenje konjunktivne normalne forme na primjeru:

- Neka je formula $F \equiv (A \rightarrow C) \wedge (\neg C \rightarrow \neg A)$
- Tada je semantička tablica za formulu prikazana u tablici 2.8.

Tab 2.8 Semantička tablica formule F

A	**C**	**A → C**	**¬A**	**¬C**	**¬C → ¬A**	**F**
0	0	1	1	1	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	0
1	1	1	0	0	1	1

- Sada tražimo sve retke tablice za koje je interpretacija I(F) = 0 i pravimo elementarnu disjunkciju. Varijable koje u tom retku imaju vrijednost 1 se negiraju.
- U ovom slučaju drugi i treći redak imaju I(F) = 0 pa su elementarne disjunkcije za zadanu formulu $A \vee \neg C$ i $\neg A \vee C$.
- Tako je KNF od formule F formula $G \equiv (A \vee \neg C) \wedge (\neg A \vee C)$

2.4.Disjunktivna normalna forma (DNF)

Disjunktivna normalna forma neke funkcije algebre sudova je disjunkcija svih njezinih konjunkcija, prema[3].

Određivanje disjunktivne normalne forme na primjeru:

- Neka je formula $F \equiv (A \rightarrow C) \wedge (\neg C \rightarrow \neg A)$
- Tada je semantička tablica za formulu prikazana u tablici 2.9.

Tab 2.9 Semantička tablica formule F

A	**C**	**A → C**	**¬A**	**¬C**	**¬C → ¬A**	**F**
0	0	1	1	1	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	0
1	1	1	0	0	1	1

- Sada tražimo sve retke tablice za koje je interpretacija I(F) = 1 i pravimo elementarnu konjunkciju. Varijable koje u tom retku imaju vrijednost 0 se negiraju.

- U ovom slučaju prvi i četvrti redak imaju I(F) = 1 pa su elementarne disjunkcije za zadanu formulu $\neg A \wedge \neg C$ i $A \wedge C$.

- Tako je DNF od formule F formula $G \equiv (\neg A \wedge \neg C) \vee (A \wedge C)$

3. RAZVOJNO OKRUŽENJE I TEHNOLOGIJE

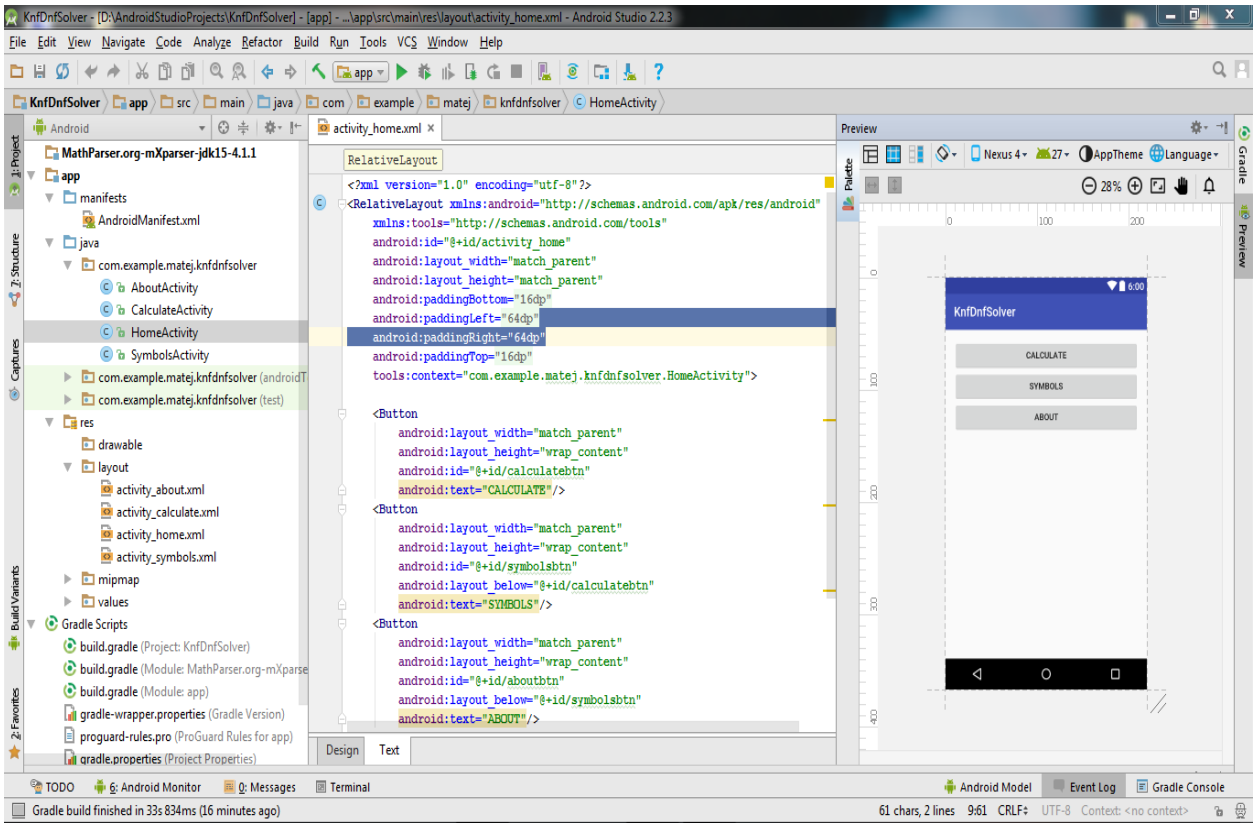
3.1.Android

Android je operacijski sustav otvorenog koda za mobilne uređaje kao što su pametni telefoni, tableti i laptopi temeljen je na Linux jezgri. Početak androida veže se uz tvrtku Android Inc koju su 2003. godine osnovali Andy Rubin, Rich Miner, Nick Sears i Chris White. Nakon dvije godine Google preuzima tvrtku. Open Handset Alliance (OHA) predvođen Google-om 2007. godine razvija prvu beta verziju, a u rujnu 2008. godine i prvu komercijalnu verziju operacijskog sustava.

Slika 3.1 Logo Android operacijskog sustava

S obzirom da je izvorni kod Android operacijskog sustava otvorenog tipa, aplikacijama je pomoću posrednika (eng. middleware) omogućeno komuniciranje i pokretanje drugih aplikacija poput ostvarivanja poziva, slanja SMS poruka i pokretanja kamere. Android aplikacije uglavnom su pisane u Java programskom jeziku koristeći Android SDK (Software Development Kit). Android SDK je skup razvojnih alata koji pruža API (Appliction Programming Interface) biblioteke te alate za izgradnju, testiranje i ispravljanje aplikacija. Android aplikacije moguće je pisati i u C/C++ programskom jeziku, ali tada se koristi Android NDK (Native Code Development Kit), prema[4].

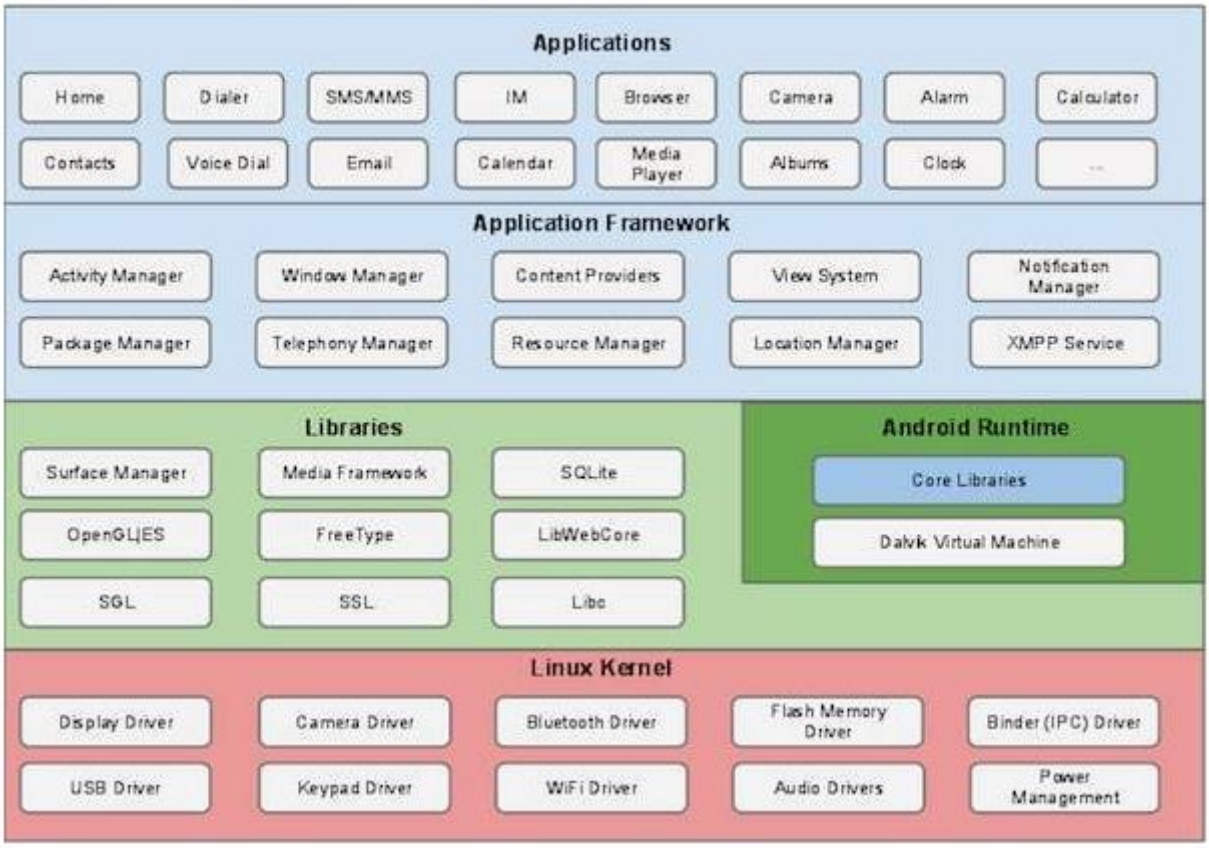
Prilikom izrade aplikacije korišteno je Android Studio razvojno okruženje. Android studio je službeni IDE (Integrated Development Environment) za Android operacijski sustav temeljen na IntelliJ IDEA softveru, prema[5]. Izgled razvojnog okruženja prikazan je na slici 3.2.



Slika 3.2 Android Studio razvojno okruženje

Arhitektura Android operacijskog sustava dijeli se na pet dijelova prikazanih na slici 3.3:

1. Linux jezgra
2. Android Runtime
3. Biblioteke
4. Aplikacijski okvir
5. Aplikacije



Slika 3.3 Arhitektura Android operacijskog sustava

Na samom dnu se nalazi Linux jezgra koja zapravo predstavlja apstraktni sloj između hardvera i softvera. Linux jezgra sadržava sve bitne *driver-e* poput drivera za kameru, tipkovnicu, ekran i drugih. Također, jezgra se koristi i za sigurnost te upravljenje procesima i memorijom.

Iznad Linux jezge nalazi se skup biblioteka napisanih u C/C++ programskom jeziku. Neke od bibilioteka koje se nalaze u ovom dijelu su biblioteke za nadziranje iscrtavanja grafičkog sučelja (Surface Manager), upravljanje bazama podataka (SQLite), sigurnosnu komunikaciju internetom (SSL) i druge.

Nakon biblioteka slijedi Android Runtime koji pruža virtualni stroj Dalvik Virtual Machine (Dalvik VM). Dalvik VM sličan je Java Virtual Machineu i dizajniran je specijalno za Android. Dalvik VM omogućava optimiziranje korištenja memorije te višenitnost. Pomoću Dalvika se svaka aplikacija pokreće u svom vlastitom procesu i ima vlastitu instancu Dalvik VM-a. Također, Android Runtime pruže set osnovnih biblioteka koje omogućavaju developerima da razvijaju Android aplikacije pomoću Java programskog jezika.

Aplikacijski okvir omogućava nekoliko višerazinskih servisa koje developeri mogu koristiti prilikom razvoja aplikacija. Neki od najbitnijih servisa su:

- View System – skup elemenata prikaza koji služe za izradu korisničkog sučelja kao što su gumbi, polja za unos i drugi
- Activity Manager – upravitelj aktivnosti koji služi za upravljanje životnim ciklusom aplikacije
- Content Providers – pružatelji sadržaja koji omogućavaju dijeljenje podataka s drugim aplikacijama
- Resource Manager – upravitelj resursima koji omogućava pristup resursima poput stringova, boja, slika i drugih koji određuju izgled korisničkog sučelja

Na samom vrhu se nalazi skup osnovnih ugrađenih aplikacija poput kontakta, telefona, kalendara i internet preglednika, ali i aplikacija koje se nalaze u GooglePlay-u. Ovaj dio je vidljiv krajnjem korisniku, prema[6].

3.2. Java

Java je objektno orijentirani programski jezik koji je razvio James Gosling iz tvrtke Sun Microsystems. Razvoj je započeo 1991. godine, a prva verzija objavljena je u studenom 1995. godine. U odnosu na ostale programske jezike, Java kod se izvršava u Java Virtual Machine (JVM), virtualnom stroju koji se za svaku platformu razvija posebno. Upravo zbog JVM-a programski jezik Java je neovisan o platformi na kojoj se izvodi. Java prevoditelji (eng. *compiler*) prevode Java kod u *bytecode*, a JVM zatim interpretira taj *bytecode* i izvršava program. Postoje dva paketa u kojima se javlja Java:

- Java Runtime Environment (JRE) – sadrži JVM, biblioteke Java klasa i funkcionalnost za pokretanja Java programa
- Java Development Kit (JDK) – uz sve ono što sadržava JRE sadrži i razvojne alate za razvoj Java programa

U Java programskom jeziku postoji automatsko upravljanje memorijom pa programer ne mora sam rezervirati i oslobađati memoriju prilikom kreiranja i uništavanja objekata, prema[7].

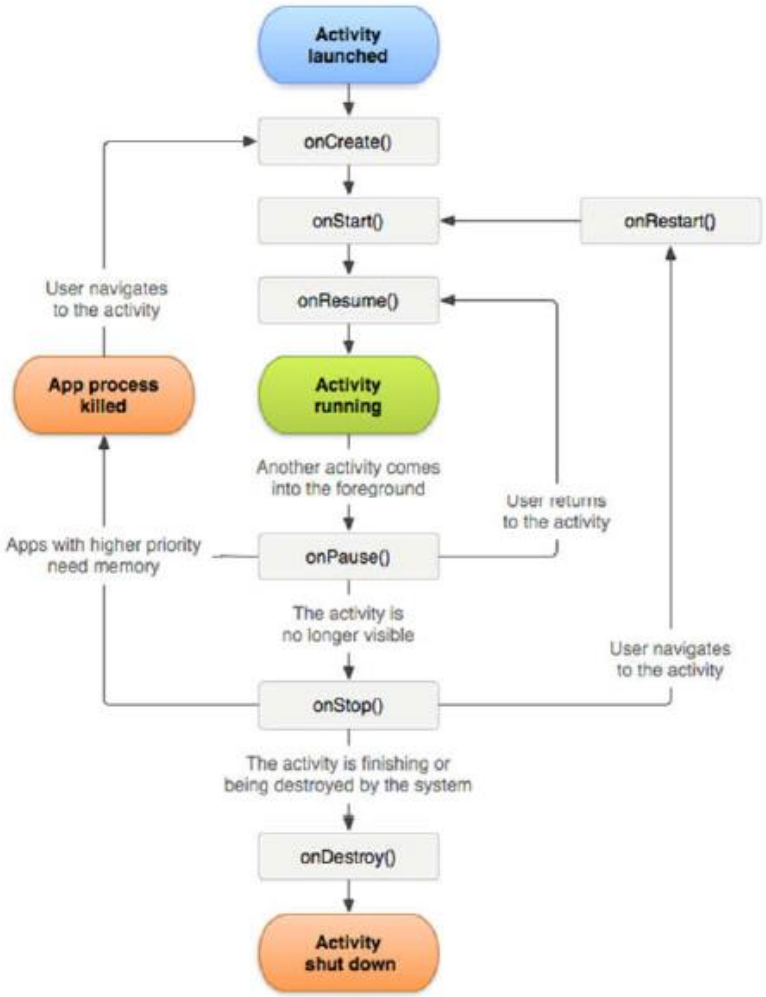
Zbog svoje prilagodljivosti i kompatibilnosti Java je jedan od najpopularniji programskih jezika u svijetu.

4. PROGRAMSKO RJEŠENJE ZADATKA

4.1.Arhitektutra i dizajn aplikacije

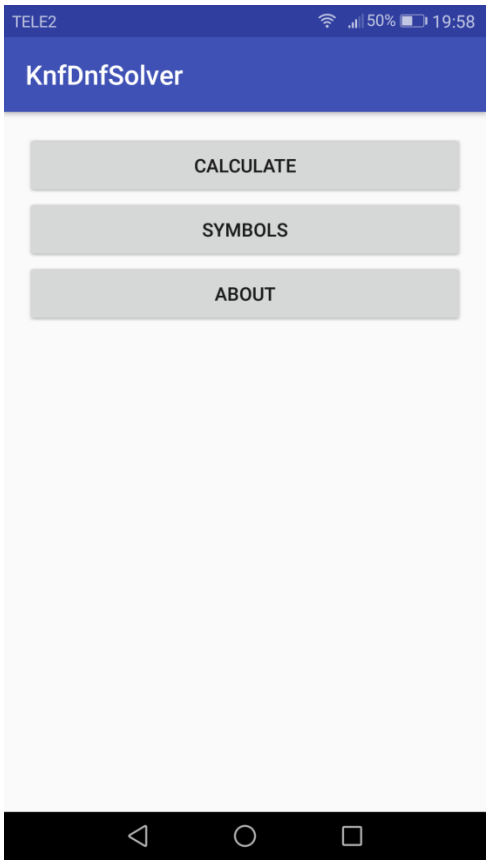
Aplikacija se sastoji od četiri Activitya: *HomeActivity, CalculateActivity, SymbolsActivity, AboutActivity.*

Activity predstavlja jedan zaslon aplikacije. Prilikom kreiranja Activitya stvara se klasa istoga imena unutar koje se piše kod te resurs *layout* pisan XML opisnim jezikom (Extensible Markup Language) u kojem se definira izgled. Navigiranje među Activityima je omogućeno putem Intenta, skupa informacija čiji su osnovni elementi akcija (radnja koju treba izvršiti) i podaci nad kojima se akcija izvršava. Umjesto *main* metode kao ulazne točke programa, životni ciklus android aplikacija sastoji se od određenih *callback* metoda: *onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart() i onDestroy()*. Životni ciklus Activitya prikazan je na slici 4.1.



Slika 4.1 Životni ciklus Activitya

HomeActivity predstavlja prvi ekran nakon pokretanja aplikacije. U njemu se nalaze tri gumba za prelazak na jedan od preostala tri Activitya. Gumbovi se kreiraju pomoću klase *Button* i pritiskom na njih se može obaviti neka radnja. Klasa *Button* je izvedena iz osnovne klase *View* koja predstavlja UI (User Interface) kontrolu. Izgled HomeActivitya prikazan je na slici 4.2.

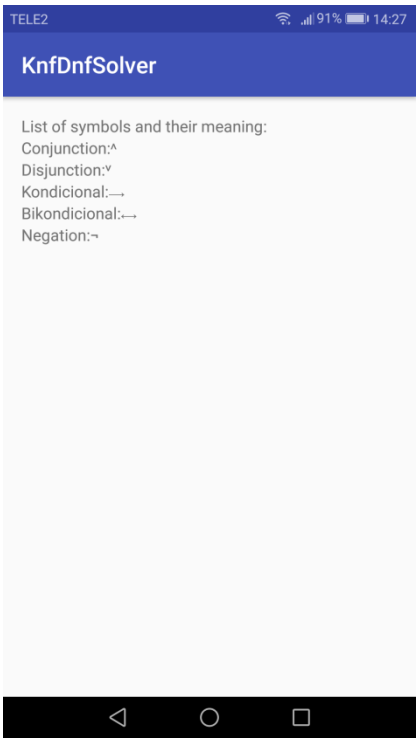


Slika 4.2 Izgled HomeActivitya

CalculateActivity je najvažniji dio aplikacije jer se u njemu nalazi najveći dio funkcionalnosti. U njemu se nalazi *EditText* za unos varijabli te *TextView* za formulu, devet gumbova, od kojih su dva za računanje i brisanje, a preostalih sedam predstavlja malu tipkovnicu koja sadrži sve logičke operacije te zagrade, *Spinner* za unos varijabli u fomuli, jedan *TextView* za prikaz rezultata (KNF i DNF) i tri *TableLayouta* za prikaz semantičke tablice unesene formule. *TextView* je klasa izvedena iz osnovne *View* klase i služi za prikaz tekstualnog sadržaja. *EditText* je klasa koja nasljeđuje *TextView* i služi za unos alfanumeričkih znakova. *TableLayout* je klasa izvedena iz *ViewGroup* klase i ona pozicionira elemente u obliku tablice po redovima i stupcima. *Spinner* omogućava odabir podataka tako što se klikom na njega otvara lista podataka za odabir. Izgled CalculateActivitya prikazan je na slici 4.3.

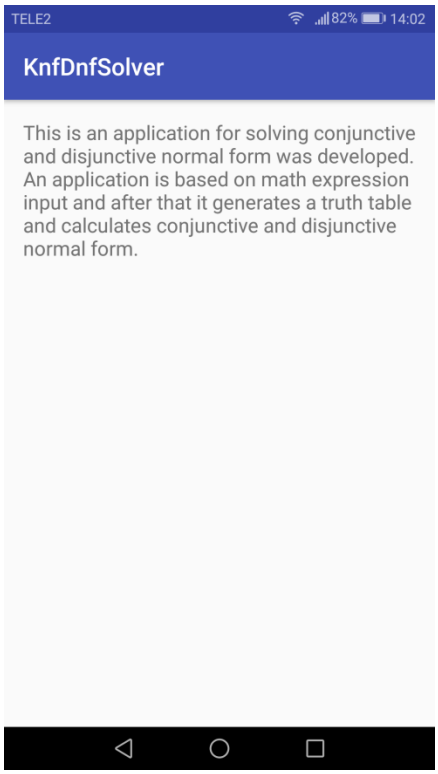
Slika 4.3 Izgled CalculateActivitya

Unutar Symbols Activitya nalazi se samo jedan *TextView* u kojem su svakoj logičkoj operaciji pridruženi određeni znakovi koji ju predstavljaju. Izgled Symbols Activitya prikazan je na slici 4.4.



Slika 4.4 Izgled SymbolsActivitya

AboutActivity također sadrži samo jedan *TextView* u kojem je opisana aplikacija i njene funkcionalnosti. Izgled AboutActivitya prikazan je na slici 4.5.



Slika 4.5 Izgled AboutActivitya

4.2.Program

Prilikom pisanja koda korištena je dodatna biblioteka *mXparser*[8] za izračun istinitosne vrijednosti unesene formule. Autorska prava na korištenje su navedena u kodu.

Kako je već prethodno navedeno, najbitniji dio aplikacije nalazi se u CalculateActivityu.

```
ResultTv = findViewById(R.id.resulttv);
ResultTv.setMovementMethod(new ScrollingMovementMethod());
FormulaTv = findViewById(R.id.formulatv);
VariableEt = findViewById(R.id.variableset);
VariableSpinner = findViewById(R.id.variableSpinner);
ValuesTableLayout = findViewById(R.id.valuesTableLayout);
ResultTableLayout = findViewById(R.id.resultTableLayout);
VariableTableLayout = findViewById(R.id.variableTableLayout);
ConjunctionBtn = findViewById(R.id.conbtn);
DisjunctionBtn = findViewById(R.id.disbtn);
KondicionalBtn = findViewById(R.id.kondbtn);
BikondicionalBtn = findViewById(R.id.bikonbtn);
NegationBtn = findViewById(R.id.negbtn);
LeftBracketBtn = findViewById(R.id.lbtn);
RightBracketBtn = findViewById(R.id.rbtn);
ResultBtn = findViewById(R.id.resultbtn);
ResetBtn = findViewById(R.id.resetbtn);
```

Slika 4.6 Dohvaćanje referenci svih elemenata u layoutu

Za početak se dohvaćaju reference na sve elemente koji su postavljeni u layout XML resursu putem njihovih ID-jeva. Dohvaćanje reference postiže se uporabom *findViewById ()* metode kojoj se predaje ID svakog elementa, a metoda vraća objekt pripadajuće klase kako je prikazano na slici 4.6.

```
ResetBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ResultTv.setText("");
        FormulaTv.setText("");
        ValuesTableLayout.removeAllViews();
        ResultTableLayout.removeAllViews();
        VariableTableLayout.removeAllViews();
    }
});
```

Slika 4.7 Obrada klika na gumb

Na slici 4.7 prikazana je obrada klika na gumb. To se postiže implementacijom *onClickListener()* sučelja i pomoću *setOnClickListener()* metode. Registrira se osluškivač na gumb koji sluša kada

korisnik klikne na gumb i tada pokreće metodu *onClick(View v).* Unutar te metode se pomoću *removeAllViews()* briše sav sadržaj koji se nalazi u elementima.

```
VariableSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        if(position > 0) {
            String s = VariableSpinner.getSelectedItem().toString();
            FormulaTv.append(s);
            VariableSpinner.setSelection(0);
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
    }
});

ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.variableArray, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
VariableSpinner.setAdapter(adapter);
```

Slika 4.8 Spinner za varijable

Na slici 4.8 prikazano je kreiranje *Spinnera* i pripadajućeg mu *adaptera*. ArrayAdapter je kreiran pomoću metode *createFromResource* kojoj su kao argumenti predani *variableArray* polje u kojem se nalaze sva slova engleske abecede i osnovni *simple_spinner_item* layout. Nakon toga je pomoću *setDropDownViewResource* metode postavljen izgled liste nakon klika na spinner te je pomoću metode *setAdapter* adapter primjenjen na spinner *VariableSpinner*. Nakon klika na spinner pomoću metode *setOnItemSelectedListener* implementira se *AdapterView* sučelje i pripadajuće *onItemSelected i onNothingSelected callback* metode. Argument *position* označava poziciju podatka u spinneru koji je odabran, a pomoću metode *getSelectedItem* se dohvaća odabrani podatak i prikazuje u *TextViev FormulaTv.*

```
//spremanje varijabli u polje i string
char[] var = VariableEt.getText().toString().toCharArray();
String varijable = VariableEt.getText().toString();

String unesenaFormula = FormulaTv.getText().toString();
String prilagodenaFormula = unesenaFormula.replace('\u2227','\u0026').replace('\u2228','\u007C').
        replace('\u00AC','\u007E').replace("\u27F6","\u002D\u002D\u003E").replace("\u27F7","\u003C\u002D\u003E");
//dodavanje zareza poslije svake znamenke
String rez_var = "";
for (int j = varijable.length() - 1; j >= 0; j--) {
    char ch = varijable.charAt(j);
    rez_var = ch + "," + rez_var;
}
//brisanje zadnjeg zareza
if (rez_var.charAt(rez_var.length() - 1) == ',') {
    rez_var = rez_var.substring(0, rez_var.length() - 1);
}
//formula
String formula = "ft(" + rez_var + ")=" + prilagodenaFormula;
//brojanje varijabli
String input = VariableEt.getText().toString().toLowerCase();
for (int i = 0; i < input.length(); i++) {
    char chr = input.charAt(i);
    int value = (int) chr;
    if (value >= 97 && value <= 122) {
        brojac_varijabli++;
    }
}
double doubleKombinacija = Math.pow(2, brojac_varijabli);
int brojKombinacija = (int) doubleKombinacija;
List<String> stringData = new ArrayList<~>();
Function ft = new Function(formula);
```

Slika 4.9 Spremanje unesenih varijabli i formule

Obrada klika na gumb Calculate ista je kao i za gumb Reset, dok je *onCreate()* metoda različita. Prvo se varijable unesene iz *EditTexta* spremaju u polje *var* tipa *char* te varijablu *varijable* tipa *String*. Također, u varijablu *unesenFormula* tipa *String* se sprema formula iz *TextViewa* i prilagođava se parseru. Nakon toga se u *for* petlji prolazi kroz cijeli string i nakon svakog znaka se dodaje zarez te se sprema u novu varijablu *rez_var* tipa *String*. U slučaju da je zadnji znak u novoj varijabli zarez, on se korištenjem *if* grananju briše. Zatim se u varijablu *formula* tipa *String* sprema prilagođena unesena formula iz *TextViewa* te varijabla *rez_var*. Nakon toga se u *for* petlji broje unesene varijable i spremaju u varijablu *brojac_varijabli* tipa *int*. Na kraju se u varijablu *brojKombinacija* tipa *int* sprema vrijednost koja će se koristiti prilikom izrade semantičke tablice, stvara se *lista polja* tipa *String* te se instancira objekt klase *Function* i predaje mu se string *formula* prikazano na slici 4.9.

```
//spremanje znamenki u string pa polje
for (int i = 0; i < brojKombinacija; i++) {
    String znam = Integer.toBinaryString(i);
    while (znam.length() < brojac_varijabli) {
        znam = '0' + znam;
    }
    znamenke = znamenke + znam;

}
char[] numbers = znamenke.toCharArray();

//dekadski u binarni konverter
for (int i = 0; i < brojKombinacija; i++) {
    String result = "";
    String s = Integer.toBinaryString(i);
    while (s.length() < brojac_varijabli) {
        s = '0' + s;
    }
    //dodavanje zareza poslije svake znamenke
    for (int j = s.length() - 1; j >= 0; j--) {
        char ch = s.charAt(j);
        result = ch + "," + result;
    }
    //brisanje zadnjeg zareza
    if (result.charAt(result.length() - 1) == ',') {
        result = result.substring(0, result.length() - 1);
    }
    stringData.add(result);
}
```

Slika 4.10 Pretvaranje dekadskih brojeva u binarne

U ovom dijelu koda svaki broj od 0 do *brojKombinacija* se pretvara u binarni i sprema u string, a nakon toga i u polje *numbers* tipa *char* pomoću *toCharArray()* metode. Nakon toga se ponovno brojevi pretvaraju u binarni, dodavaju se i brišu zarezi te se svaki broj u binarnom zapisu prema u listu polja *stringData* tipa *String* prikazano na slici 4.10.

```
Row = brojKombinacija;
Col = brojac_varijabli;
Col2 = 1;

// ispis svih kombinacija za n varijable;
int brojac = 0;
for (i = 1; i <= Row; i++) {
    final TableRow row = new TableRow(CalculateActivity.this);
    if (i % 2 == 0) {
        row.setBackgroundColor(Color.WHITE);
    } else {
        row.setBackgroundColor(Color.LTGRAY);
    }

    for (j = 1; j <= Col; j++) {
        if (brojac == numbers.length) break;

        final TextView txt = new TextView(CalculateActivity.this);
        txt.setTextColor(Color.BLACK);
        txt.setTextSize(TypedValue.COMPLEX_UNIT_PT, 8);
        txt.setText(" " + numbers[brojac]);
        row.addView(txt);
        brojac++;

    }
    ValuesTableLayout.addView(row);
}
```

Slika 4.11 Ispis svih kombinacija u tablicu

Sada se u varijeble *Row, Col i Col2* spremaju vrijednosti potrebne za kreiranje tablica. Pomoću dvije for petlje se kreiraju redovi i stupci. Instancira se objekt klase *TableRow* te se u svakom retku dodaje onoliko *TextViewa* koliko ima varijabli u koje su prikazane binarne kombinacije iz polja *numbers* prikazano na slici 4.11. Na isti način se kreiraju i tablice za prikaz varijabli i rezultata.

```
if (brojac_varijabli == 1) {

    String OneVariable = calculator.calculateForOne(ft, var, stringData);
    ResultTv.append(OneVariable);

} else if (brojac_varijabli == 2) {

    String TwoVariables = calculator.calculateForTwo(ft, var, stringData);
    ResultTv.append(TwoVariables);
} else {

    String ThreeVariables = calculator.calculateForThree(ft, var, stringData);
    ResultTv.append(ThreeVariables);
}
```

Slika 4.12 Računanje KNF i DNF

Na slici 4.12 prikazan je izračun KNF i DNF. Prvo se provjerava broj varijabli i na temelju njega se poziva odgovarajuća funkcija. Prvo je stvoren objekt *calculator* klase *CalculationClass*. Nakon toga se pomoću tog objekta poziva jedna od funkcija (*calculateForOne, calculateForTwo, calculateForThree*) i rezultat se sprema u varijablu tipa *String*. Na kraju se rezultat prikaže u *ResultTv TextView*.

```
public String calculateForOne(Function ft , char[] var, List<String> podaci) {
    for (int i = 0; i < podaci.size(); i++) {
        Expression e1 = new Expression("ft(" + podaci.get(i) + ")", ft);
        double rez = e1.calculate();
        int rez1 = (int) rez;
        if (rez1 == 0 && podaci.get(i).equals("0")) {
            jedan = ("(" + var[0] + ")");
            KNF = KNF + jedan;
        }
        if (rez1 == 0 && podaci.get(i).equals("1")) {
            dva = ("(" + '¬' + var[0] + ")");
            KNF = KNF + dva;
        }
    }
    if (KNF.charAt(KNF.length() - 1) == '∧') {
        KNF = KNF.substring(0, KNF.length() - 1);
    }
    for (int i = 0; i < podaci.size(); i++) {
        Expression e1 = new Expression("ft(" + podaci.get(i) + ")", ft);
        double rez = e1.calculate();
        int rez1 = (int) rez;
        if (rez1 == 1 && podaci.get(i).equals("0")) {
            jedan = ("(" + "¬" + var[0] + ")");
            DNF = DNF + jedan;
        }
        if (rez1 == 1 && podaci.get(i).equals("1")) {
            dva = ("(" + var[0] + ")");
            DNF = DNF + dva;
        }
    }
    if (DNF.charAt(DNF.length() - 1) == '∨') {
        DNF = DNF.substring(0, DNF.length() - 1);
    }
    String konacni = "KNF:" + "\n" + KNF + "\n" + "DNF:" + "\n" + DNF;
    return konacni;
}
```

Slika 4.13 Funkcija za računanje KNF i DNF

Na slici 4.13 prikazana je funkcija za izračun KNF i DNF za jednu varijablu. Postupak je sličan i za dvije i tri varijable. Argumenti funkcije su objekt klase *Function, char* polje i lista stringova. Prvo se u for petlji instancira objekt *Expression* klase i preda mu se formula. Pomoću metode

calculate() računa se vrijednost formule i sprema se u varijablu *rez1* tipa *int*. Nakon toga se u if grananju provjeravaju svi mogući slučajevi za KNF i DNF te se razultati spremaju u varijeblu *KNF* odnosno *DNF* tipa *String*. Na kraju se briše zadnji znak stringa te funkcija vraća varijablu *konacni* tipa *String*.

5. ZAKLJUČAK

Tema ovog diplomskog rada je bila „Android aplikacija za računanje konjunktivne i disjunktivne normalne forme“. Izrada rada je podijeljena u tri dijela: teorijski dio zadatka, razvojno okruženje i tehnologije te programsko rješenje zadatka. U teorijskom dijelu je opisana matematička logika te logika sudova i sve logičke operacije sa sudovima uz pripadajuće semantičke tablice. U dijelu razvojnog okruženja i tehnologija opisan je Android operacijski sustav, Android Studio kao razvojno okruženje za izradu programskog dijela te Java programski jezik u kojem je programski kod napisan. U dijelu programskog rješenja prikazani su arhitektura i dizajn aplikacije s pripadajućim slikama te je detaljnije opisan programski kod. Aplikacija je uspješno izrađena te ima sve potrebne funkcionalnosti odnosno uspješno računa konjunktivnu i disjunktivnu normalnu formu za zadanu matematičku formulu. Računanje konjunktivne i disjunktivne normalne forme omogućeno je za formule koje se sastoje od jedne, dvije ili najviše tri varijable. U budućnosti bi se ovaj nedostatak mogao ispraviti drugačijim pristupom u pisanju koda za računanje i korištenjem boljeg algoritma.

LITERATURA

[1] M. Vuković, Matematička logika, Element, Zagreb 2009.

[2] M. Vuković, Matematička logika, http://www.mathos.unios.hr/logika/Logika_skripta.pdf, rujan 2018.

[3] E.Mendelson, Introduction to Mathematical Logic, Chapman and Hall, London 1997.

[4] https://hr.wikipedia.org/wiki/Android_(operacijski_sustav), lipanj 2018.

[5] https://developer.android.com/studio/intro/ , lipanj 2018.

[6] https://www.tutorialspoint.com/android/android_architecture.htm , lipanj 2018.

[7] K.Nenadić,S.Bošnjak,T.Pekanov,J.Balen,A.Baumgartner,B.Zorić,M.Hanzer, Razvoj mobilnih aplikacija, Priručnik za edukaciju, Elektrotehnički fakultet Osijek, Osijek 2013.

[8] http://mathparser.org/ , lipanj 2018.

SAŽETAK

Naslov: Android aplikacija za rješavanje konjunktivne i disjunktivne normalne forme

U ovom radu napravljena je aplikacija za rješavanje konjunktivne i disjunktivne normalne forme. Aplikacija na temelju uneseng matematičkog izraza generira tablicu istinitosti te računa konjunktivnu i disjunkitivnu normalnu formu. Tijekom rada naveden je teorisjki dio potreban za izradu te korištene tehnologije i okruženja prilikom izrade. Na kraju je detaljno objašnjena realizacija aplikacije.

Ključne riječi: android, aplikacija, konjunktivna normalna forma, disjunktivna normalna forma

ABSTRACT

Title: An Android application for solving conjunctive and disjunctive normal form

In this work an application for solving conjunctive and disjunctive normal form was developed. An application is based on math expression input and after that it generates a truth table and calculates conjunctive and disjunctive normal form. Firstly, theoretical part and technologies that are used in application development are described. Finally, application development is explained in detail.

Keywords: Android, application, conjunctive normal form, disjunctive normal form

ŽIVOTOPIS

Matej Tomić rođen je 10. studenoga 1994. godine u Našicama. Od svoga rođenja živi u Našicama. U 2009. godini upisuje se u opću gimnaziju u Srednoj školi Isidora Kršnjavog u Našicama. U trećem razredu srednje škole sudjeluje na školskom, a zatim i na županijskom natjecanju iz informatike pod nazivom „Infokup“. Srednju školu završava redovno u 2013. godini te u istoj upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. U 2016. godini redovno završava upisani preddiplomski studij računarstva i stječe akademski naziv sveučilišni prvostupnik inženjer računarstva. U istoj godini upisuje diplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Matej Tomić

PRILOZI

Izvorni kod nalazi se na slijedećem linku:

https://github.com/matej94/LogicCalculator