Web aplikacija za obradu formule logike sudova

Jurković, Damir

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

STRUČNI STUDIJ

WEB APLIKACIJA ZA OBRADU FORMULE LOGIKE SUDOVA

Završni rad

Damir Jurković

Osijek, 2018

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 13.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Damir Jurković
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4474, 22.09.2017.
OIB studenta:	86296151805
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 završnog rada:	Web aplikacija za obradu formule logike sudova
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada	Student će izraditi Web aplikaciju za obradu formule logike sudova. Sumentor: Alfonzo Baumgartner
Prijedlog ocjene pismenog dijela ispita (završnog 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, 28.09.2018.

Ime i prezime studenta:	Damir Jurković
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4474, 22.09.2017.
Ephorus podudaranje [%]:	4%

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za obradu formule logike sudova**

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

Logika sudova je grana matematičke logike. Ona se bavi između ostalog odnosima među rečenicama. Slovima označavamo rečenice ili njihove dijelove, a znakovima: $\wedge, \vee, -, \rightarrow, \leftrightarrow$ odnose među njima, također koristimo zagrade (i)[1]. Sva ta slova, znakovi i zagrade čine alfabet logike sudova. Jedan od problema logike sudova je ispitivanje istinitost formula. Svaka formula može imati četiri ishoda: oboriva je, ispunjiva je, tautologija je i antitautologija je. Primjena logike sudova se pronalazi u digitalnoj elektronici gdje su tri gradivna digitalna sklopa I, ILI, NE ekvivalent veznicima $\wedge$, $\vee$, $-$ u logici sudova.

Aplikacija vrlo brzo i točno ispituje formule koje se zadaju. Bez problema rješava formule s više operacija i varijabli u prihvatljivom vremenu.

Rad se dijeli na dva dijela: izrada aplikacije i provjera ispravnosti. Izrada web aplikacije sastoji se od sedamnaest potpoglavlja kroz koje se opisuje kompletna struktura aplikacije te njezin izgled. Princip rada kratko opisuje rad aplikacije te provjerava ispravnost rada aplikacije tako što uspoređuje rješenja koja aplikacija nudi i stvarna rješenja.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izrada web aplikacije koja će obrađivati formulu logike sudova te odrediti je li unesena formula oboriva, ispunjiva, tautologija ili antitautologija. Za izradu web aplikacije korištena je tehnologija za razvoj web aplikacija PHP. Također su korištene tehnologije HTML, CSS, Bootstrap 4 i Javascript.

2. IZRADA WEB APLIKACIJE ZA OBRADU FORMULE LOGIKE SUDOVA

U ovome poglavlju je opisana izrada web aplikacije za obradu formule logike sudova, algoritmi, strukture podataka i potrebni stilovi za izgled i interaktivnost stranice.

2.1.Inicijaliziranje varijabli

Prije izvođenja algoritama za obrađivanje i računanje formule, potrebno je najprije deklarirati i inicijalizirati varijable koje će se kasnije kroz aplikaciju koristiti i izmjenjivati. U varijabli `formula` spremljena je formula koju unosimo preko aplikacije. Tekst koji unesemo u formu na web aplikaciji će biti spremljen prvo u ovu varijablu. Nakon inicijalizacije se ova varijabla koristi u gotovo svakome algoritmu. Kako je nužno da ova varijabla postoji prvo se pravi provjera je li je korisnik unio bilo kakav tekst, tek nakon toga se provjerava je li je taj tekst valjana formula. Provjeravanje ispunjenosti forme se provjerava s funkcijom `isset()`[2].

```
$formula = $_POST['formula'];
```

Kôd 2.1.1. *Varijabla koja sadržava formulu unesenu preko aplikacije*

`_SESSION["formula"]` je globalna varijabla koja se može koristiti kroz cijelu skriptu jednom kad se postavi. U njoj je spremljena formula koju unosimo preko aplikacije. Prije nego postavimo vrijednost varijable potrebno je na početak programa, pozvati funkciju `session_start()` kako bi se omogućilo rasprostiranje ove varijable kroz više skripta.

```
$_SESSION["formula"] = $formula;
```

Kôd 2.1.2. *Globalna varijabla koja sadržava formulu*

Sadržaj varijable `operatori_dopusteni` su svi znakovi koji se koriste kao operatori za pisanje formule.

```
$operatori_dopusteni = "QWERTZUIOPASDFGHJKLYXCVBNM";
```

Kôd 2.1.3. *Varijabla koja sadržava sve znakove koji se koriste za operatore*

U varijabli `dopustena_abeceda` su spremljeni svi dopušteni znakovi koji se koriste za pisanje formule, a to uključuje operatore i operacije te također zagrade.

```
$dopustena_abeceda = "QWERTZUIOPASDFGHJKLYXCVBNM^-#%&()";
```

Kôd 2.1.4. *Varijabla koja sadržava sve znakove koji se koriste za pisanje formule*

Ove dvije varijable sadrže određene znakove koji ne smiju biti smješteni jedan do drugoga. Ove varijable koristimo pri provjeri ispravnosti formule.

```
$znak_do_znaka_provjera1 = ")QWERTZUIOPASDFGHJKLYXCVBNM";
$znak_do_znaka_provjera2 = "(QWERTZUIOPASDFGHJKLYXCVBNM";
```

Kôd 2.1.5. *Varijable za provjeru valjanosti znaka pored*

U `nedopusteno_ispred_operacije` su spremljeni određeni znakovi koji ne smiju biti ispred operacije (sve operacije osim negacije). Varijabla se koristi pri provjeri ispravnosti formule.

```
$nedopusteno_ispred_operacije = "(-#%&^";
```

Kôd 2.1.6. *Varijabla sa znakovima koji ne smiju biti ispred operacije*

Kod ove varijable su spremljeni određeni znakovi koji ne smiju biti iza operacije (sve operacije). Varijabla se koristi pri provjeri ispravnosti formule.

```
$nedopusteno_iza_operacije = ")#%&^";
```

Kôd 2.1.7. *Varijabla sa znakovima koji ne smiju biti iza operacije*

Dok su kod ove varijable spremljeni određeni znakovi koji smiju biti ispred negacije. Varijabla se koristi pri provjeri ispravnosti formule.

```
$dopusteno_ispred_negacije = "(#%&^";
```

Kôd 2.1.8. *Varijabla sa znakovima koji ne smiju biti ispred negacije*

U varijabli `operacije` su spremljene sve operacije. Sve operacije su poslagane po njihovom prioritetu izvođenja, od najvišeg prioriteta do najnižeg s lijeva na desno.

```
$operacije = "-^%&#";
```

Kôd 2.1.9. *Varijabla sa svim operacijama*

U ovoj varijabli je spremljen broj operacija. Za brojanje operacija se koristi funkcija `strlen()` koja vraća broj znakova jedne varijable.

```
$broj_operacija = strlen($operacije);
```

Kôd 2.1.10. *Varijabla čiji je sadržaj broj operacija*

U ovoj varijabli je spremljena URL adresa stranice na kojoj se nalazi ova skripta. Ova varijabla se koristi u funkciji `header()` koja se u ovome programu izvodi kada dođe do unosa pogrešne formule.

```
$URL = $_SERVER['REQUEST_SCHEME']."://".$_SERVER['SERVER_NAME'].
                    $_SERVER['SCRIPT_NAME']."";
```

Kôd 2.1.11. *Varijabla koja sadrži URL adresu skripte*

2.2.Stvaranje forme za unos

Forma je izrađena uz pomoć HTML-a, koriste se oznake `<form>`, `<input>`, `<small>`. Forma je dio korisničkog sučelja web aplikacije. Sa oznakom `<form>` se stvara prazna forma te joj postavljamo način na koji ćemo proslijediti unesene podatke u nju pomoću `method="post"` gdje `„post"` označava da će podatke poslati preko HTTP zahtjeva. Sa oznakom `<input>` se stvara polje za unos podataka. Atribut `action` označava na koju adresu se šalju podaci iz forme. U ovoj aplikaciji, podaci se šalju na istu skriptu. Atribut `placeholder` postavlja tekst koji će biti prikazan ukoliko je forma prazna. Atribut `required` označava da je obavezno unošenje teksta u formu. Oznaka `<small>` je običan tekst manjeg fonta koji se koristi za prikaz pomoćne informacije korisniku aplikacije. U slučaju ako je unesena formula sa pogreškama prikazati će se različita forma. Biti će obojana crveno s klasom `is-invalid` te će biti ispod forme ispisana poruka o kakvoj je pogrešci riječ.

```
<div class="invalid-feedback">
  <?php echo $error_poruka; ?>
</div>
```

Kôd 2.2.1. *Ispis pogreške u formi*

S oznakom `<button>` stvaramo gumb koji šalje uneseni podatak iz forme na procesiranje. Izgled gumba je određen predefiniranim klasama `btn btn-primary btn-lg` koje nudi Bootstrap 4.

```
<button type="submit" class="d-flex btn btn-primary btn-lg submit-gumb" form="formula-forma" name="submit" value="Izračunaj">Izracunaj</button>
```

Kôd 2.2.2. *Gumb za slanje formule na obradu*

2.3.Provjera valjanosti unesene formule u formu

Nakon što se unese formula u formu i klikne se gumb „Izračunaj“, ta formula prolazi kroz niz algoritama koji provjeravaju valjanost formule.

Algoritam za provjeru valjanosti zagrada je algoritam koji provjerava jesu li zatvorene sve otvorene zagrade. Također se prati pojavljivanje svake zagrade i zapisuje u varijablu `razina_zagrade_polje`. Sa `for` petljom se prolazi kroz svaki znak u formuli, te sa `if` uvjetima se provjerava je li zagrada otvorena ili zatvorena, ukoliko je zagrada otvorena varijabli `razina_zagrade` se vrijednost uvećava za 1, a u suprotnom ako je zagrada zatvorena varijabli umanjujemo vrijednost za 1. Prilikom svake iteracije u petlji provjerava se je li vrijednost varijable `$razina_zagrade` manja od 0, ukoliko je poziva se funkcija `header()` s pripadajućom porukom o pogrešci. Ako se nakon prolaska kroz cijelu petlju ispostavi da je brojač veći od nule to bi značilo da postoji jedna zagrada koja nema svoj par te se poziva `header()` funkcija sa pripadajućom porukom o pogrešci.

```
if($formula[$i] == '(' ){
  $razina_zagrade++;
}elseif ($formula[$i] == ')' ) {
  $razina_zagrade--;
}
```

Kôd 2.3.1. *Provjera valjanosti zagrada*

Algoritam za provjeru valjanosti znakova alfabeta formule je algoritam koji pretražuje cijelu unesenu formulu i traži je li ima u sebi neke nedozvoljene znakove poput malih slova, brojeva ili znakova poput ?, _, ., =, itd.. U `for` petlji se prolazi kroz svaki znak u formuli te ukoliko se naiđe na nedozvoljeni znak odmah se poziva funkcija `header()` sa pripadajućom porukom o pogrešci. Provjera se vrši preko `if` uvjeta. U `if` se stavlja PHP funkcija `strpos()` koja pronalazi poziciju određenog znaka unutar određenog teksta. Ukoliko se ne pronađe nedozvoljen znak program nastavlja dalje s radom.

```
if(strpos($dopustena_abeceda, $formula[$i]) === false){
  header("Location: ".$URL."?error=3");
}
```

Kôd 2.3.2. *Provjera valjanosti alfabeta logike sudova*

Algoritam za provjeru smještaja varijabli je algoritam koji provjerava postoji li operacija između varijable i zagrade, dvije varijable te zagrade i zagrade. S `for` petljom prolazimo kroz svaki znak formule i uz `if` uvjete provjeravamo je li varijable i zagrade imaju između sebe operacije. Nakon lociranja varijable, prvi `if` provjerava je li znak na mjestu `i-1` zatvorena zagrada ili slovo, ako je, provjerava se na drugom `if`-u je li na mjestu ispred od trenutnog `i-1` otvorena zagrada ili slovo, ako je, poziva se funkcija `header()` sa priloženom porukom o pogrešci.

```
if(strpos($znak_do_znaka_provjera1, $formula[$i-1]) > -1){
  if(strpos($znak_do_znaka_provjera2, $formula[$i]) > -1){
    header("Location: ".$URL."?error=2");
  }
}
```

Kôd 2.3.3. *Provjera smještaja varijabli*

Algoritam za provjeru argumenata operacije je algoritam u kojem se provjerava jesu li ispred i iza operacija dopušteni znakovi. S `for` petljom se prolazi kroz sve znakove te se traže ∨, ∧, ↔, →, −. Prvi `if` provjerava je li znak na trenutnom mjestu `i` jedan od operacija (∨, ∧, ↔, →, −). Drugi `if` provjerava je li operacija na prvom mjestu u formuli, ako je, poziva se funkcija `header()` sa odgovarajućom porukom o pogrešci (za sve operacije osim negacije). Treći `if` provjerava je li

operacija na zadnjem mjestu u formuli, ako je poziva se funkcija `header()` sa odgovarajućom porukom o pogrešci. Četvrti `if` provjerava je li ispred operacije se nalazi jedan od nedozvoljenih znakova "(, -, ∨, →, ↔, ∧", ako da, poziva se funkcija `header()` sa odgovarajućom porukom o pogrešci. Peti `if` provjerava da li se iza operacije nalazi jedan od nedozvoljenih znakova "), ∨, →, ↔, ∧", ako da, poziva se funkcija `header()` sa odgovarajućom porukom o pogrešci. Dodane su blage varijacije tih uvjeta posebno za operaciju negacija, jer za razliku od ostalih operacija, negacija za računanje zahtjeva jedan operand. Ukoliko aplikacija prođe kroz sve provjere bez prekida programa i ispisivanja pogreške, onda ima valjanu formulu spremnu za daljnju obradu.

2.4.Preuređivanje formule prije obrade

Ovdje se preuređuju određeni znakovi i zagrade u formuli koji su nepotrebni ili koji će omogućiti lakše programiranje.

Prvo se vrši zamjena znakova određenih operacija koje se sastoje od više znakova ili imaju neko već drugo značenje u programskom jeziku PHP. Znak ekvivalencije se prebacuje iz ↔ u `#`, znak implikacije se prebacuje iz → u `&`, znak disjunkcije se prebacuje iz ∨ u `%`, prazne zagrade se brišu i nepotrebni razmaci se brišu.

```
$formula = str_replace('<->', '#', $formula);
$formula = str_replace('->', '&', $formula);
$formula = str_replace('v', '%', $formula);
$formula = str_replace('()', '', $formula);
$formula = str_replace(' ', '', $formula);
```

Kôd 2.4.1. *Izmjena znakova operacije*

Zagrada koja obavija cijelu formulu nema funkciju u rezultatu provjere istinitosti formule, te otežava programiranje i povećava vrijeme procesiranja formule računalu. Prvo provjeravamo postoji zagrada na prvom mjestu u formuli, ukoliko postoji algoritam se izvršava, a ako ne, preskače se. Nakon toga iz varijable `razina_zagrade_polje` uzimamo prvi element koji je neki cijeli broj koji označava u kojoj se „razini zagrade“ trenutno nalazimo u formuli i spremamo u varijablu `vanjska_zagrada` npr. ako je 0 onda smo izvan svih zagrada, ako je 1 onda smo unutar jedne zagrade itd. . Nadalje varijablu `vanjska_zagrada` proširujemo tako što joj na kraj dodamo

broj „razine zagrade“ prvog mjesta minus jedan, npr. ako je na prvom mjestu u formuli otvorena zagrada, sadržaj varijable `vanjska_zagrada` će biti „10“. To možemo iskoristiti kako bi prepoznali odgovarajuće parove zagrada. Nakon toga provjeravamo je li se prvo pojavljivanje sadržaja varijable `vanjska_zagrada` pojavljuje skroz na kraju formule, ako da te zagrade brišemo te to radimo ispočetka dok taj uvjet ne bude „laž“. Nakon toga se ponovo inicijalizira varijabla `razina_zagrade_polje` koja je poprimila nove vrijednosti nakon izvođenja ovog algoritma.

```
$vanjska_zagrada = $razina_zagrade_polje[0];
$vanjska_zagrada = "$vanjska_zagrada".($vanjska_zagrada-1);
```

Kôd 2.4.2. *Brisanje zagrade koja obavija formulu*

Nakon toga s `for` petljom pronalazimo svaki operator te izvodimo funkciju `str_replace()` koja zamjenjuje svako pojavljivanje određenog znaka s nekim drugim određenim znakom. Poslije petlje se ponovo inicijalizira varijabla `razina_zagrade_polje` koja je poprimila nove vrijednosti nakon izvođenja ovog algoritma.

```
$duljina_operatora = strlen($operatori_dopusteni);
for($i = 0; $i < $duljina_operatora; $i++){
  $formula = str_replace("(".$operatori_dopusteni[$i].")",
$operatori_dopusteni[$i], $formula);
}
```

Kôd 2.4.3. *Brisanje zagrade koja obavija samo operator*

2.5.Prepoznavanje svih varijabli

Ovdje se prikazuje algoritam za prepoznavanje i zapisivanje svih varijabli koje se nalaze u formuli. Sa petljom `for` se prolazi kroz sve znakove u formuli te sa funkcijom `ctype_upper()` provjeravamo je li trenutni znak `formula[$i]` veliko slovo, ukoliko je, u varijablu `registar_varijabli` spremamo to slovo. Također prije izvođenja `ctype_upper()` izvršava se `if` koji provjerava je li postoji već to slovo u varijabli `registar_varijabli`.

```
if(ctype_upper($formula[$i])){
  $registar_varijabli.=$formula[$i];
}
```

Kôd 2.5.1. *Prepoznavanje svih varijabli*

2.6.Popunjavanje varijabli inicijalnim sadržajem

Ovaj algoritam popunjava varijable iz formule (A, B, C, …) sa početnim vrijednostima iz tablice istinitosti. U varijablu `broj_varijabli` zapisujemo ukupan broj operatora s funkcijom `strlen()`, varijabla `duljina_tablice` označava broj redova koji će imati tablica istine, za dobivanje toga broja koristi se funkcija `pow()` koja se koristi za eksponencijalne račune. S `for` petljom prolazimo kroz sve operatore (stupce) te sa još jednom `for` petljom unutar trenutne petlje prolazimo kroz sve redove tablice. Koristi se varijabla `stupanj_izmjene` koja označava izmjenu vrijednosti u tablici (0 i 1) te se preko nje određuje u koje polje ide određena vrijednost. Kod svakog stupca se računa nova vrijednost za varijablu `stupanj_izmjene`.

```
for($k = 0; $k < $broj_varijabli; $k++){
  $stupanj_izmjene = pow(2, $k);
  $izmjena = $stupanj_izmjene * (-1);
  for($z = 0; $z < $duljina_tablice; $z++){
    ...
  }
}
```

Kôd 2.6.1. *Popunjavanje varijabli inicijalni sadržajem*

2.7.Pronalaženje najviše i najniže razine zagrade

Jednostavnom `for` petljom koja prolazi kroz sve znakove varijable `razina_zagrade_polje` uspoređujemo i određujemo koja je najviša razina zagrade.

```
for ($i=0; $i < $duljina_formule; $i++) {
  if($razina_zagrade_polje[$i] > $max_razina){
    $max_razina = $razina_zagrade_polje[$i];
  }
}
```

Kôd 2.7.1. *Pronalaženje najviše razine zagrade*

Jednostavnom `for` petljom koja prolazi kroz sve znakove varijable `razina_zagrade_polje` uspoređujemo i određujemo koja je najniža razina zagrade.

```
for ($i=0; $i < $duljina_formule; $i++) {
  if($razina_zagrade_polje[$i] < $min_razina){
    $min_razina = $razina_zagrade_polje[$i];
  }
}
```

Kôd 2.7.2. *Pronalaženje najniže razine zagrade*

2.8.Izvlačenje sadržaja iz zagrade s najvišom razinom

U ovome algoritmu se određuje sadržaj zagrade s najvišom razinom, a to nam je potrebno jer po prioritetu računanja prvo se kreće računati od dijela s najvišom razinom zagrade. Ulazi se u prvu `for` petlju gdje se prolazi kroz svaki znak formule. Prvo se s `if` uvjetom provjerava da li je trenutni znak unutar zagrade s najvišom razinom, ukoliko nije prelazi se na idući znak, a ako je, ide na idući `if` koji provjerava da li je najviša trenutna razina jednaka najnižoj razini, ako je, za varijablu `trenutni_sadrzaj_zagrade` se uzima cijela formula, ako nije, uzima se trenutna pozicija `for` petlje i sprema u varijablu `pocetak_razine_zagrada`. Zatim se pronalazi prva zatvorena zagrada te se pozicija sprema u varijablu `kraj_razine_zagrade`. Koristeći funkciju `substr()` dobivamo `trenutni_sadrzaj_zagrade` koja označava sadržaj najviše razine odnosno dio formule koji se nalazi unutar zagrade s najvišim prioritetom.

2.9.Preskakanje već izračunatih vrijednosti

Zbog razloga što algoritam za računanje pregledava svaki znak u sadržaju najviše razine zagrade i traži operacije u tom sadržaju, potreban nam je ovaj algoritam kako bi preskočio „[,]“ zagrade, jer unutar tih zagrada se nalaze operatori i operacije koje su već izvršene, npr. `A↔B∧C` nakon prvog koraka računanja bi bio zapisan u programu kao `A↔[B∧C]`. Tako uz pomoć ovoga algoritma preskačemo tu zagradu da se ne bi ispočetka računala ista operacija. Algoritam radi tako što unutar dvije `for` petlje ispituje je li na trenutnom mjestu u sadržaju se nalazi „ [„ ukoliko se nalazi, traži se odgovarajući par te zagrade i na mjestu te zagrade se nastavlja iduća iteracija `for` petlje.

```
for($k = 0; $k < $broj_operacija; $k++){
```

```
for($z = 0; $z < $duljina_trenutnog_sadrzaja_zagrade; $z++){
    ...
  $z = $kraj_uglate_zagrade;
  continue;
  }
}
```

Kôd 2.9.1. *Preskakanje već izračunatih vrijednosti*

2.10.Izvođenje operacija i spremanje podataka u rezultate

Ovaj algoritam vrši računanje operacija i spremanje podataka u tablicu istinitosti. Algoritam za računanje konjunkcije, disjunkcije, implikacije i ekvivalencije je u potpunosti isti, dok je algoritam za računanje negacije sličan i jednostavniji od algoritma za ostale operacije. Programski jezik PHP nema ugrađeno u sebi sve operacije koje se koriste, stoga je potrebno napisati funkcije za operacije. U algoritmu prvo se ispituje da li je trenutni znak `trenutni_sadrzaj_zagrade[$z]` jednak trenutnoj operaciji `$operacije[$k]`. Ukoliko je ulazi se u `switch` grananje.

```
if($trenutni_sadrzaj_zagrade[$z] == $operacije[$k]){
   switch ($operacije[$k]) {
```

Kôd 2.10.1. *Provjera operacije*

Algoritam za operaciju negacija prvo provjerava s desne strane je li se nalazi „ [„ ako postoji postavlja se varijabla zastavica_desna_uglata na jedan ukoliko ne postoji ostaje na nula. Zatim se provjerava s `if` je li zastavica „1“ , ako je, uzima se sadržaj iz zagrada i zapisuje u varijablu `tablica` te se kroz `for` petlju koja se ponavlja `duljina_tablice` puta funkcijom `negacija()` dobiva rješenje za svaku iteraciju računanja koja se zapisuje u varijablu `tablica`, ali ako je zastavica „0“ , onda se uzima varijabla koja je desno od operacije i jednako kroz petlju se računa kao i u prošlom slučaju. Nakon toga se operacija i operator ili sadržaj obavijen „ [,] „ obavijaju s „ [„ i „] “ kako bi se znalo da je tu već izvršen izračun.

```
$zastavica_desna_uglata = 0;
```

```
if($trenutni_sadrzaj_zagrade[$z+1] == '['){
  ...
  $zastavica_desna_uglata = 1;
}
```

Kôd 2.10.2. *Računanje negacije*

Algoritam za računanje ostalih operacije kao i kod negacije prvo pregledava da li se desno od operacije nalazi „ [„ nakon toga, za razliku od algoritma kod operacije negacija, provjerava se lijevo od operacije da li se nalazi „] “. Nakon toga s `if` uvjetovanjem izvršavamo slučaj računanja koji je ispravan ovisno o postojanju zagrada ili ne. Nakon što se odredi slučaj, računanje je jednako kao i kod operacije negacija.

```
$zastavica_desna_uglata = 0;
$zastavica_lijeva_uglata = 0;
if($trenutni_sadrzaj_zagrade[$z+1] == '['){
  ...
  $zastavica_desna_uglata = 1;
}
if ($trenutni_sadrzaj_zagrade[$z-1] == ']') {
  ...
  $zastavica_lijeva_uglata = 1;
}
```

Kôd 2.10.3. *Računanje ostalih operacija*

2.11.Preuređivanje početne formule

Nakon što se izvrši računanje jedne operacije, u istoj iteraciji vrijednost varijable `formula` se izmjenjuje. Vrijednost varijable `trenutni_sadrzaj_zagrade` prije računanja se zamjenjuje s trenutnom vrijednošću iste varijable. To se odrađuje zbog toga da program prepozna da li je zagrada već izračunata te da se može nastaviti na iduće zagrade. Također se nakon ovoga taj sadržaj spušta za jednu razinu zagrade. Nakon što se izmjeni vrijednost varijable `formula` sa funkcijom `str_replace()`, ponovo se inicijalizira varijabla `razina_zagrade_polje` koja je poprimila nove vrijednosti nakon izvođenja ovog algoritma.

```
        if($trenutni_sadrzaj_zagrade != ''){
          $formula=str_replace($trenutni_sadrzaj_zagrade_usporedba,
$trenutni_sadrzaj_zagrade, $formula);
          $duljina_formule = strlen($formula);
        }
```

Kôd 2.11.1. *Preuređivanje početne formule*

2.12.Ispisivanje tablice istine

Prije nego se tablica istinitosti ispiše na ekran, prvo se moraju privremeni znakovi operacija vratiti u prvobitne znakove. Nakon toga korištenjem dvije `for` petlje ispisujemo tablicu istine tako što sa vanjskom petljom ispisujemo redove a unutarnjom petljom ispisujemo stupce. Za ispis se koriste HTML oznake `<table>`, `<tr>`, `<th>`, `<td>`. Također su dodane predefinirane Bootstrap 4 klase za tablicu `table-responsive`, `table`, `table-hover`, `table-dark`, `table-bordered` koje osiguravaju ugodan izgled i interaktivnost tablice.

2.13.Prepoznavanje ishoda rezultata

Ovaj algoritam pregledava rezultat i utvrđuje je li formula ispunjiva, oboriva, tautologija ili antitautologija. Algoritam koristeći `for` petlju prvo broji u varijabli `brojac_ishoda` koliko je jedinica u rješenju formule. Zatim sa `if` ispitujemo, ukoliko je `brojac_ishoda` jednak nuli ispisuje se na ekran da je formula antitautologija i oboriva, ukoliko je varijabla veća ili jednaka 1 onda je ispunjiva te se još naknadno provjerava i nadoveže na ovaj tekst ukoliko je `brojac_ishoda` jednak `duljina_tablice` onda je tautologija, a inače je oboriva.

```
for ($i=0; $i <= $duljina_tablice; $i++) {
 if(isset($tablica[$stupci_tablice_racunanje[$broj_stupaca_tablice-1]][$i])){
   if($tablica[$stupci_tablice_racunanje[$broj_stupaca_tablice-1]][$i] == 1){
     $brojac_ishoda++;
   }
 }
}
```

Kôd 2.13.1. *Prepoznavanje ishoda i ispis rezultata*

2.14.Prepoznavanje grešaka

U dijelu gdje se provjerava valjanost formule ukoliko dođe do pogreške poziva se funkcija `header()` na koju se nadodaje određen broj koji se u ovome algoritmu dokučuje preko `$_GET` globalne varijable te ispisuje pripadajuća poruka o pogrešci ovisno o broju koji `header()` nosi sa sobom. Koristeći `switch case` uvjetovanje, brojevi pogrešaka od nula do jedanaest se raspoznaju i spremaju u varijablu `$error_poruka` koja se dalje koristi za ispisivanje na ekran.

```
switch ($error) {
  ...
  case '1':
    $error_poruka = "Ima viska '(' zagrada";
    break;
  ...
}
```

Kôd 2.14.1. *Prepoznavanje grešaka*

2.15.Gumbi za postavljanje operacija na znaku za umetanje

Ovi gumbi se nalaze odmah ispod forme i klikom na jedan od njih pojaviti će se pripadajuća operacija na mjestu gdje je bio znak za umetanje u formi.

Izgled svakog gumba je identičan, jedina razlika je u nazivu operacije koja se nalazi na gumbu. Za razliku od gumba za izračun, ovi gumbi imaju na sebi `onclick` atribut koji govori da gumb izvršava nekakvu dodatnu funkciju klikom, u ovom slučaju poziva se funkcija `operacijaGumb()`, kojoj se predaju dva argumenta, ID od forme i oznaku operacije.

```
<button data-toggle = "tooltip" data-html = "true" data-placement = "bottom"
onclick = "operacijaGumb('formula', '^')" class = "btn btn-info btn-sm
operacija-gumb-zasebno" type = "button" name = "konjunkcija" value="^">

^ (Konjunkcija)

</button>
```

Kôd 2.15.1. *Izgled gumba*

Funkcija postavljanja operacije na znaku za umetanje za razliku od svih ostalih prijašnjih funkcija je pisana u Javascript programskom jeziku. Ova funkcija prvo dohvaća formu preko ID pa zatim provjerava da li postoji označeni tekst u formi, ako da, postavlja vrijednost u varijabli `string_pozicija` broj početnog mjesta označenog teksta. Zatim dohvaća pomoću funkcije `substring()` tekst ispred i iza te ubacuje varijablu `operacija` između i prepravlja varijablu `string_pozicija`.

```
function operacijaGumb(formaID, operacija) {
  var formula_podrucje = document.getElementById(formaID);
  ...
  var ispred = (formula_podrucje.value).substring(0, string_pozicija);
  var iza = (formula_podrucje.value).substring(string_pozicija,
formula_podrucje.value.length);
  formula_podrucje.value = ispred + operacija + iza;
  ...
}
```

Kôd 2.15.2. *Funkcija postavljanja operacije na znaku za umetanje*

Svaki gumb ima svoj opis kada se pređe preko njega pokazivačem. U opisu se nalazi tablica istine za pripadajuću operaciju. Dodaje se kao atribut `title` u `button` oznakama. Prije nego se doda taj atribut mora se uključiti opis koji Bootstrap 4 nudi. Sadržaj opisa je tablica istine pripadajuće operacije unutar `title` atributa.

2.16.Povratak na vrh gumb

Ovaj gumb se pojavljuje tek nakon što spustimo stranicu za 150 piksela. Funkcija gumba je klikom vratiti stranicu na vrh stranice.

```
<button name="top" id="top" onclick="scrollToTop(400)" >
  <i class="fas fa-arrow-alt-circle-up"></i>
</button>
```

Kôd 2.16.1. *Izgled gumba povratak na vrh*

Funkcija `scrollDoVrha(trajanje)` obavlja pomicanje stranice na vrh prihvaća jedan argument i to je vrijeme trajanja pomicanja prema gore. Nakon što se inicijalizira varijabla `korak` poziva se funkcija `setInterval()` koja se koristi za obavljanje funkcije u vremenskom intervalu. Na svako pomicanje prozora gore ili dolje ugrađena funkcionalnost Javascript-a `window.onscroll` će se pokrenuti te će pozvati funkciju unutar sebe `scrollFunkcija()`.

```
window.onscroll = function() {
  scrollFunkcija()
};
```

Kôd 2.16.2. *Osluškivanje događaja pomicanja stranice*

Funkcija `scrollFunkcija()` provjerava da li je stranica spuštena više od 150 piksela, ukoliko je, gumb se dokučuje preko njegovog ID te mu se postavlja njegov atribut `display` na vrijednost `block`, a u suprotnom mu se atribut `display` postavlja na vrijednost `none` zbog kojeg neće biti vidljiv.

```
if (document.documentElement.scrollTop > 150) {
  document.getElementById("top").style.display = "block";
} else {
  document.getElementById("top").style.display = "none";
}
```

Kôd 2.16.3. *Funkcija za prikaz i skrivanje gumba*

2.17.Dodavanje stila na web aplikaciju

Sav kôd do sada je se bavio o funkcionalnosti izvedbe programa, kôd koji nadolazi je CSS te on opisuje kako će izgledati elementi na web aplikaciji.

Font za sav tekst unutar tablice je postavljen na „Courier new“ sa atributom `font-family`. Sa atributom `border` smo definirali rub tablice. Također smo primijenili stil na redove i svaki parni red. Postavljen je dodatan stil na prvi stupac tako da ukoliko se pojavi `horizontal scroll` da prvi stupac uvijek bude vidljiv kao pokazatelj pozicije.

```
table {
```

```
    font-family: "Courier New", Courier, monospace;
    border: 1px solid
}
```

Kôd 2.17.1. *Stil za tablicu*

Naslov koji je prva stvar što se pojavi na stranici ima samo nekoliko jednostavnih atributa. Margine su promijenjene te je tekst centriran. Također su boja i font izmijenjeni.

```
.naslov{
  margin-top: 40px;
  margin-bottom: 40px;
  text-align: center;
  color: #32383e;
  font-family: "Courier New", Courier, monospace;
}
```

Kôd 2.17.2. *Stil za naslov*

Forma ima izmijenjen font, centriran tekst te pomaknute margine.

```
form {
  font-family: "Courier New", Courier, monospace;
}
```

Kôd 2.17.3. *Stil za formu*

Gumbi imaju izmijenjene margine, font i boju pozadine.

```
.operacija-gumb-zasebno{
  margin: 0 5px;
  font-family: "Courier New", Courier, monospace;
  margin-top: 7px;
  margin-bottom: 7px;
}
```

Kôd 2.17.4. *Stil za gumbe za operacije*

Gumb za računanje je centriran sa `auto` marginom te mu je pozadinska boja i boja ruba izmijenjena. Također se koristi Bootstrap 4 predefinirana klasa `btn-primary` [5].

```
.izracunaj-gumb{
  margin: 20px auto;
}
.btn-primary{
  background-color: #34678e;
  border-color: #34678e;
}
```

Kôd 2.17.5. *Stil za gumb za računanje*

Gumb za povratak na vrh ima zadanu postavku da se ne prikazuje, po potrebi ga funkcija `scrollFunkcija()` prikaže. Ima fiksiranu poziciju u donjem desnom kutu. Prikazuje se iznad svakog elementa. Također ima postavljen atribut `cursor` na `pointer` koji prebacuje pokazivač miša u drugačiji oblik kada se pređe mišem preko gumba.

```
#top {
  display: none;
  position: fixed;
  bottom: 20px;
  right: 30px;
  z-index: 99;
  cursor: pointer;
  ...
}
```

Kôd 2.17.6. *Stil za gumb za povratak na vrh*

Responzivni dizajn u ovome projektu je bilo potrebno namjestiti na tri rezolucije „1023px, 767px, 534px“. `@media` oznaka označava da će se stilovi primjenjivati samo na nekim rezolucijama određenim `max-width` atributom. U ovom slučaju prvi `@media` se primjenjuje od 1023 piksela

širine do prvog idućeg nižeg `@media`, a to je 767 piksela širine.

```
@media only screen and (max-width: 1023px) {
  table{
    font-size: 15px;
  }
  ...
}
@media only screen and (max-width: 767px) {
  table{
    font-size: 12px;
  }
  ...
}
```

Kôd 2.17.7. *Stil za responzivan dizajn*

3.PRINCIP RADA

Korisnik unosi formulu u formu te pokreće računanje. Aplikacija provjerava ispravnost unesene formule u pogledu jesu li pravilno upisane operacije, jesu li na dozvoljenom mjestu, jesu li zagrade uparene, itd. . Nakon provjere ispravnosti formule aplikacija prebrojava koliko varijabli ima u formuli te ispunjava tablicu istinitosti na osnovu broja varijabli. Nakon toga aplikacija započinje s računanjem držeći se pravila prioriteta operacija i zagrada. Program kao rezultat izbacuje konačnu tablicu istinitosti te objavljuje je li formula ispunjiva, oboriva, tautologija ili antitautologija.

3.1.Provjera i računanje

Nakon unesene formule u formu dobivamo rezultat o toj formuli. Provjeru vršimo tako što ručno izračunamo unesenu formulu i provjerimo poklapaju li se rješenja.

Primjer 1:

Formula : $A \vee B \wedge (-C \rightarrow B)$

#	A	B	C	-C	(-C)->B	B^((-C)->B)	Av(B^((-C)->B))
1	0	0	0	1	0	0	0
2	1	0	0	1	0	0	1
3	0	1	0	1	1	1	1
4	1	1	0	1	1	1	1
5	0	0	1	0	1	0	0
6	1	0	1	0	1	0	1
7	0	1	1	0	1	1	1
8	1	1	1	0	1	1	1

Interpretacija Formule:
AvB^(-C->B)
Ispunjiva i Oboriva!

Sl. 3.1.1. *Princip rada aplikacije 1*

Provjera:

Provjerom formule $A \vee B \wedge (-C \rightarrow B)$ dolazimo do rezultata:
Y = [0, 1, 1, 1, 0, 1, 1, 1] te zaključujemo da je formula ispunjiva i oboriva što se poklapa sa rješenjem koje je aplikacija ponudila.

Primjer 2:

Formula: $A \wedge (B \vee (A \leftrightarrow C) \wedge D) \wedge -A$

#	A	B	C	D	A<->C	(A<->C) ^D	Bv ((A<->C) ^D)	-A	(Bv ((A<->C) ^D)) ^ (-A)	Av ((Bv ((A<->C) ^D)) ^ (-A))
1	0	0	0	0	1	0	0	1	0	0
2	1	0	0	0	0	0	0	0	0	1
3	0	1	0	0	1	0	1	1	1	1
4	1	1	0	0	0	0	1	0	0	1
5	0	0	1	0	0	0	0	1	0	0
6	1	0	1	0	1	0	0	0	0	1
7	0	1	1	0	0	0	1	1	1	1
8	1	1	1	0	1	0	1	0	0	1
9	0	0	0	1	1	1	1	1	1	1
10	1	0	0	1	0	0	0	0	0	1
11	0	1	0	1	1	1	1	1	1	1
12	1	1	0	1	0	0	1	0	0	1
13	0	0	1	1	0	0	0	1	0	0
14	1	0	1	1	1	1	1	0	0	1
15	0	1	1	1	0	0	1	1	1	1
16	1	1	1	1	1	1	1	0	0	1

```
Interpretacija Formule:
Av(Bv(A<->C)^D)^-A
Ispunjiva i Oboriva!
```

Sl. 3.1.2. *Primjer rada aplikacije 2*

Provjera:

Provjerom formule $A \wedge (B \vee (A \leftrightarrow C) \wedge D) \wedge -A$ dolazimo do rezultata:
Y = [0, 0, 0, 0, 0, 0, 0, 0] te zaključujemo da je formula antitautologija i oboriva što se poklapa sa rješenjem koje je aplikacija ponudila. Ovdje se može vidjeti ograničenost „pameti“ aplikacije, gdje računalo ne predviđa odmah na početku da se `A` i `-A` nalaze u konjunkciji te da će rješenje biti antitautologija gdje bi čovjek to mogao primijetiti i odmah zaključiti rješenje.

Primjer 3:

Formula: $(Q \leftrightarrow P) \wedge -(P \wedge Z)$

#	Q	P	Z	Q<->P	P^Z	-(P^Z)	(Q<->P)^(-(P^Z))
1	0	0	0	1	0	1	1
2	1	0	0	0	0	1	0
3	0	1	0	0	0	1	0
4	1	1	0	1	0	1	1
5	0	0	1	1	0	1	1
6	1	0	1	0	0	1	0
7	0	1	1	0	1	0	0
8	1	1	1	1	1	0	0

```
Interpretacija Formule:
(Q<->P)^-(P^Z)
Ispunjiva i Oboriva!
```

Sl. 3.1.3. *Princip rada aplikacije 3*

Provjera:

Provjerom formule $(Q \leftrightarrow P) \wedge -(P \wedge Z)$ dolazimo do rezultata:
Y = [1, 0, 0, 1, 1, 0, 0, 0] te zaključujemo da je formula ispunjiva i oboriva što se poklapa sa rješenjem koje je aplikacija ponudila.

4.ZAKLJUČAK

Izrada web aplikacije je zahtijevala znanje iz programiranja i alata za izradu web aplikacija u ovom slučaju PHP, HTML, CSS, Bootstrap 4. Izrada stilova i izgleda web aplikacije nije predstavljao problem zahvaljujući Bootstrap 4 predefiniranim klasama te odabiru jednostavnoga i lako prilagodljivog dizajna koji nije zahtijevao puno prerađivanja za različite rezolucije. Web aplikacija računa iznimno brzo u usporedbi sa čovjekom. Otprilike, do sedam varijabli i petnaest operacija računa ispod jedne sekunde, što je za čovjeka nemoguće, ali također aplikacija nije dovoljno „pametna“ da predvidi neke pravilnosti koje čovjek može predvidjeti poput nekih teorema Booleove algebre koji se mogu primijeniti na ovu temu.

5.POPIS LITERATURE

1. M.Vuković, Matematička logika 1, PMF-Matematički odjel Sveučilište u Zagrebu, Zagreb, 2007

2. PHP Group(2001), PHP manual, http://php.net/manual/en/funcref.php , Zadnja posjeta 5.9.2018.

3. L.Welling, L.Thomson, PHP and MySQL Web Development 5th edition, Addison-Wesley Professional, Sjedinjene Američke Države, 2017

4. R.Nixon, Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5, O'Reilly Media, Sjedinjene Američke Države, 2015

5. N. Panchal, J. Thornton, Bootstrap 4, https://getbootstrap.com/docs/4.1/components/buttons/ , Zadnja posjeta 5.9.2018.

Sažetak

Zadatak ovog rada je napraviti Internet aplikaciju za obradu formule logike sudova, pri čemu će aplikacija primiti kao unos formulu logike sudova te će kao rezultat izbaciti tablicu istine i interpretaciju formule. Korištene su Internet tehnologije PHP, HTML, CSS, Bootstrap 4. Nakon što korisnik unese formulu, aplikacija dohvaća formulu i provjerava valjanost prije nego krene u samo računanje formule. Na provjeru valjanosti se odnose operacije provjere dopuštenih znakova, ispravne zagrade, ispravan položaj operanda, ispravnih znakova pored operacija. Ukoliko se utvrdi nevaljanost formule traži se ponovni unos. Nakon provjere valjanosti, vrši se računanje nad formulom, gdje se pazi na prioritet računanja. Nakon računanja se analizira rezultat te zaključuje interpretacija formule. Nakon toga se tablica istine i interpretacija ispisuju na ekran. Aplikacija uspješno rješava formule u kratkom vremenu.

Ključne riječi: Matematička logika, formula, tautologija, PHP

Web application for processing propositional logic formulas

Summary

The task of this paper is to create an web application for processing propositional logic formulas, whereby the application will receive propositional logic formula as an input and as a result will print out the truth table and formula outcome. PHP, HTML, CSS, Bootstrap 4 technologies have been used. After the user enters the formula, application fetches the formula and verifies the validity before it goes into computing. Validation refers to the operations of checking the permissible characters, the correct brackets, the correct operand position, the correct characters by the operations. If a formula is invalid, a re-entry is required. After validation, the calculation of the formula is made, taking into account the priority of the calculation. After calculation, the result is analyzed and the outcome of the formula is concluded. Then the truth table and the outcome are printed on the screen. The application successfully solves formulas in short time.

Keywords: Mathematical logic, formulas, tautology, PHP

Životopis

Damir Jurković je rođen 9. siječnja 1997 u Vinkovcima. Osnovnu je školu završio u Starim Mikanovcima. Tehničku školu smjer Tehničar za mehatroniku je završio u Vinkovcima 2015. godine. Godine 2015. upisuje stručni studij Informatike na Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. 2017. godine na fakultetu osvaja priznanje za postignut uspjeh u studiranju.

Damir Jurković

_____